UNIVERSIDAD TECNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ÁREA TÉCNICA

TITULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Análisis de Rendimiento de Librerías de aprendizaje automático a través de lenguaje de programación Python para hacer predicciones, estimaciones y clasificadores

TRABAJO DE TITULACIÓN.

AUTOR: Bastidas Mendieta, Pablo Vicente

DIRECTOR: Elizalde Solano, Rene Rolando, Mgs

LOJA – ECUADOR

2016

Septiembre, 2016

APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN

Magister.
Rene Rolando Elizalde Solano
DOCENTE DE LA TITULACIÓN

De mi consideración:

El presente trabajo de titulación: Análisis de Rendimiento de Librerías de aprendizaje automático a través de lenguaje de programación Python para hacer predicciones, estimaciones y clasificadores" realizado por Pablo Vicente Bastidas Mendieta, ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación del mismo.

Loja, Septiembre de 2016

f) . . . . . .

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

"Yo Pablo Vicente Bastidas Mendieta declaro ser autor (a) del presente trabajo de titulación: Análisis de Rendimiento de Librerías de Aprendizaje Automático a través de lenguaje de programación Python para hacer predicciones, estimaciones y clasificadores", de la Titulación de Sistemas Informáticos y Computación, siendo Rene Rolando Elizalde Solano director (a) del presente trabajo; y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad.

Adicionalmente declaro conocer y aceptar la disposición del Art. 88 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: "Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado o trabajos de titulación que se realicen con el apoyo financiero, académico o institucional (operativo) de la Universidad"

f. .....
Autor: Pablo Vicente Bastidas Mendieta
Cédula: 1104758998

DEDICATORIA

A Dios, por bendecirme cada dia y permitirme llegar hasta este momento tan importante de mi vida.

A mi padres quienes me han apoyado incondicionalmente, mi padre ejemplo de fortaleza y superación quien me ha demostrado que a pesar de las adversidades de la vida todo es posible, mi madre por sus sabios consejos y brindarme todo su amor y paciencia.

A mis hermanos que con su apoyo, insistencia y perseverancia lograron guiarme.

A todos quienes han compartido este camino conmigo y el sueño de cumplir esta meta.

Pablo Vicente Bastidas Mendieta

AGRADECIMIENTO

A Dios, por darme salud, fuerza y brindarme la sabiduría necesaria para seguir adelante en cada paso que doy, permitiéndome seguir con nuevos sueños.

A mis padres y hermanos, parte fundamental de mi vida, quienes han sabido guiarme con paciencia y sabiduría a lo largo de este camino.

A mi director de trabajo de fin de titulación, Mgs Rene Elizalde, por su orientación, motivación y confianza, impartiéndome sus conocimientos y brindándome su apoyo durante todo el desarrollo de este proyecto.

A la Universidad Técnica Particular de Loja por acogerme en sus instalaciones, en donde tuve la dicha de formarme con profesionalismo.

A todos ellos mi gratitud por su valioso aporte durante este proceso.

Pablo Vicente Bastidas Mendieta

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente trabajo de fin de titulación presenta un análisis de rendimiento de librerías de aprendizaje automático en Python. Se realiza este análisis con el fin de demostrar las ventajas que tiene este lenguaje de programación en el campo de la ciencia de los datos. La información base que ha sido analizada con las librerías de aprendizaje automático, son datos del Instituto Nacional de Estadísticas y Censos con la temática matrimonios y divorcios en el Ecuador.

Para obtener una comparación equilibrada del rendimiento de las librerías, se realizó los experimentos sobre el mismo conjunto de entrenamiento (80% de los datos) y pruebas (20% de los datos) en cada una de las librerías. Resulta útil realizar esta comparación para poder determinar entre otros parámetros la velocidad y exactitud que ofrecen algunas librerías de aprendizaje automático en Python. De manera general, en base a varios parámetros de evaluación entre ellos “Exactitud” del modelo en el experimento de clasificación y “MSE” (error cuadrático medio) en el experimento de regresión, la librería Scikit-Learn obtuvo un mejor rendimiento, en estos dos enfoques.

PALABRAS CLAVE: Aprendizaje Automático, Clasificación, Regresión, Python.

ABSTRACT

This work presents a performance analysis of machine learning libraries through Python programming language. This analysis was done to demonstrate the advantages of this programming language in the field of data science. The basic information that has been analyzed with the machine learning libraries, are data from the National Institute of Statistics and Censuses with the theme marriages and divorces in Ecuador.

To obtain a comparison balanced of libraries performance, the experimentation was done on the same set of training (80% Data) and testing (20% Data) in each of the libraries. This comparison was useful in order to determinate among other parameters, speed and accuracy, offered by some Python's machine learning libraries. In general, based on a number of parameters evaluation among them “Accuracy” (accuracy of the model) in the experiment of classification and “MSE” (mean square error) in the experiment of regression, the library Scikit-Learn obtained the best performance, on these two approaches.

Keywords: Machine Learning, Classification, Regression, Python.

INTRODUCCIÓN

En la actualidad existe grandes volúmenes de datos almacenada en archivos planos, base de datos relacionales y no relacionales. El auge de los dispositivos electrónicos, el internet y el hecho de que casi todos los procesos en el mundo utilizan algún tipo de software, nos están dando enormes cantidades de datos cada minuto.

A todos estos datos se los puede aprovechar de muchas maneras: examinarlos, clasificarlos, e incluso poder visualizarlos de varias formas. Este trabajo parte justamente de dicha información que está a la espera de ser analizada y es ahí donde intervienen los conceptos vinculados a aprendizaje automático ya que los datos sirven como una fuente de experiencia para mejorar el rendimiento de los algoritmos utilizados en aprendizaje automático, es decir estos algoritmos pueden aprender a partir de datos anteriores, es así que estos modelos precisos con el paso del tiempo presentan fallos de predicción ante cambios en la evidencia o han de adaptarse a otro contexto.

En la actualidad existen lenguajes de programación que han desarrollado librerías que son el medio a través del cual se puede realizar aplicaciones con información almacenada. Existen herramientas comerciales disponibles que no satisfacen totalmente los requisitos de eficiencia, eficacia y exactitud, lo cual es complejo aun en los ambientes organizacionales más simples.

En el presente trabajo de fin de titulación se considera las ventajas del lenguaje de programación Python para efectivizar y ejemplificar los conceptos de aprendizaje automático y realizar un análisis de rendimiento de librerías de aprendizaje automático en el ámbito de clasificación y regresión.

El propósito de este trabajo fue determinar sobre la base de dos parámetros; velocidad y exactitud, el rendimiento de librerías de aprendizaje automático en Python, demostrando características destacables no solamente en este ámbito sino en el campo de la ciencia de los datos, ya que en este trabajo previo a la aplicación de las técnicas de aprendizaje se utilizaron herramientas para la preparación y visualización de datos que fueron propias de este lenguaje.

El presente trabajo de fin de titulación se compone de cuatro capítulos que describen lo siguiente:

El capitulo uno es un repaso a conceptos de inteligencia artificial, aprendizaje automático, descripción de las técnicas mas conocidas de aprendizaje automático en el campo de ciencias

de los datos, documentación sobre librerías de aprendizaje automático en Python, con la finalidad de poseer una visión general de las conceptualizaciones que el presente trabajo requiere.

El capitulo dos contiene el proceso de preparación de datos necesario previo a la aplicación de técnicas de aprendizaje donde se detalla la selección de atributos, transformación y análisis estadístico de los datos con el fin de minimizar los errores de predicción tanto para el problema de clasificación como de regresión.

El capitulo tres contempla la implementación de las librerías de aprendizaje automático en Python, la cual se aplica sobre el conjunto de datos preparado en el capitulo dos. Para obtener una comparación equilibrada de rendimiento de las librerías los experimentos se han realizado sobre el mismo conjunto de entrenamiento (80% de los datos) y pruebas (20% de los datos) en cada una de las librerías, además se ha configurado a los algoritmos con el fin de aprovechar su máximo rendimiento a los problemas dados.

El capitulo cuatro presenta la evaluación de resultados obtenidos en el capitulo tres para el problema de clasificación con los parámetros de evaluación: Exactitud, Error, Matriz de Confusión, Tiempo de entrenamiento mas predicción y para el problema de regresión: R2, MSE, Tiempo de entrenamiento mas predicción la cual ha permitido contrastar los resultados entre cada una de las librerías.

OBJETIVOS

Objetivo General

- Analizar el rendimiento de librerías de aprendizaje automático a través de lenguaje de programación Python.

Objetivos Específicos

- Describir las técnicas de aprendizaje automático en el campo de ciencias de los datos.
- Ejecutar procedimientos de automatización que permitan preparar conjuntos de datos para su posterior aplicación de técnicas de aprendizaje.
- Aplicar librerías de aprendizaje automático en lenguaje de programación Python a conjuntos de datos en el ámbito de predicciones, estimaciones o clasificadores.
- Realizar comparación de rendimiento de las librerías de aprendizaje automático en el lenguaje Python.

GLOSARIO

IA: Inteligencia Artificial.

Patrones: es información que permite establecer propiedades de entre conjuntos de objetos.

Datos: son una representación simbólica (numérica, alfabética, algorítmica, etc.) de un atributo o variable cuantitativa o cualitativa.

Algoritmo: es un conjunto de instrucciones bien definidas, ordenadas que permite realizar una actividad mediante pasos sucesivos.

Hipótesis: Suposición hecha a partir de unos datos que sirve de base para iniciar una investigación o una argumentación.

Atributo: especifica o establece una propiedad o valor de un objeto.

Exactitud: es la proporción de ejemplos clasificados correctamente.

Precisión: es la proporción de verdaderos positivos entre los casos clasificados como positivos.

Recall: es la proporción de verdaderos positivos entre todos los casos positivos en los datos

F-1: es una media armónica ponderada de precisión y recall.

ROC AUC: Área bajo la curva.

Pseudo-codigo: es una descripción de alto nivel compacta e informal del principio operativo de un programa informático u otro algoritmo.

Python scripting: significa escribir código de Python desde la consola.

Interfaz: es el medio con que el usuario puede comunicarse con una máquina.

Código fuente: es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa.

Kernel: función que proyecta la información a un espacio de características de mayor dimensión.

Framework: marco de trabajo.

R2: Coeficiente de determinación.

MSE: Error cuadrático medio.

CAPÍTULO I

MARCO TEÓRICO

1.1 Inteligencia Artificial.

Los hombres se han denominado a sí mismos como Homo Sapiens (hombre sabio) por que nuestras capacidades mentales son muy importantes para nosotros. Durante miles de años hemos tratado de entender como pensamos; es decir, entender como un simple puñado de materia puede percibir, entender, predecir y manipular un mundo mucho mas grande y complicado que ella misma. (Russell & Norvig, 2004)

La IA (Inteligencia Artificial) es una de las ciencias mas recientes. El trabajo comenzó poco después de la Segunda Guerra Mundial, y el nombre se acuño en 1956. La IA sintetiza y automatiza tareas intelectuales, y es por lo tanto, potencialmente relevante para cualquier ámbito de la actividad intelectual humana, en este sentido es un campo genuinamente universal. (Russell & Norvig, 2004)

Para otros autores, el origen de la IA se remonta a los intentos del hombre desde la antigüedad por incrementar sus potencialidades físicas e intelectuales, creando artefactos automatizados, simulando la forma y las habilidades de los seres humanos, en los últimos años su avance se ha hecho más notable y se ha extendido a numerosas áreas, por lo que ha sido llevada a intentar simular comportamientos humanos, no sólo en razonamiento, sino también aspectos más difíciles de medir como es el caso de la creatividad o arte artificial. (Romero, Dafonte, Gòmez, & Penousal, 2007)

Según (Díez, Gómez, & de Abajo Martínez, 2001), dada la indefinición del propio concepto de "inteligencia", prácticamente existe una definición de Inteligencia Artificial por cada autor que escribe sobre el tema. Tal vez una de las definiciones que se puede considerar mas ajustada a la realidad es la reflejada en la Encyclopedia of Artificial Intelligence:

> "La IA es un campo de la ciencia y la ingeniería que se ocupa de la comprensión, desde el punto de vista informático, de lo que se denomina comúnmente comportamiento inteligente. También se ocupa de la creación de artefactos que exhiben este comportamiento".

Según (Romero et al., 2007), la Inteligencia Artificial se puede definir como aquella "inteligencia" exhibida por artefactos científicos construidos por humanos, es decir, un sistema artificial que posee inteligencia cuando es capaz de llevar a cabo tareas que, si fuesen realizadas por un humano, se diría de este que es inteligente.

En la tabla 1, se expone definiciones de IA bajo cuatro enfoques: sistemas que actúan/piensan como humanos y sistemas que actúan/piensan racionalmente.

Tabla 1: Algunas Definiciones de Inteligencia Artificial, organizadas

Sistemas que piensan como humanos	**Sistemas que piensan racionalmente**
≪El nuevo y excitante esfuerzo de hacer que los computadores piensen… máquinas con mentes, en el mas amplio sentido literal≫. (Haugeland, 1985) ≪[La automatización de] actividades que vinculamos con procesos de pensamiento humano, actividades como la toma de decisiones, resolución de problemas, aprendizaje.... ≫. (Bellman, 1978)	≪El estudio de las facultades mentales mediante el uso de modelos computacionales≫. (Charniak y McDermott, 1985) ≪El estudio de los cálculos que hacen posible percibir, razonar y actuar≫. (Winston, 1992)
Sistemas que actúan como humanos	**Sistemas que actúan racionalmente**
≪El arte de desarrollar máquinas con capacidad para realizar funciones que cuando son realizadas por personas requieren inteligencia≫. (Kurzweil, 1990) ≪El estudio de cómo lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor≫. (Rich y Knight, 1991)	≪La Inteligencia Computacional es el estudio del diseño de agentes inteligentes≫. (Poole *et al*..1998) ≪IA… esta relacionada con conductas inteligentes en artefactos≫. (Nilsson, 1998)

Fuente: (Russell & Norvig, 2004)
Elaboración. El Autor.

(Russell & Norvig, 2004), en su libro "*Inteligencia Artificial: Un enfoque moderno*", presentan las disciplinas que han contribuido con ideas, puntos de vista y técnicas al desarrollo de la IA, mostradas a continuación:

- ✓ **Filosofía:** Aristóteles (384-322 a.c.) fue el primero en formular un conjunto preciso de leyes que gobernaban la parte racional de la inteligencia. Él desarrolló un sistema informal para razonar adecuadamente con silogismos, que en principio permitía extraer conclusiones mecánicamente, a partir de premisas iniciales.

- ✓ **Matemáticas:** los filósofos delimitaron las ideas ideas mas importantes la IA, para pasar de ahí a una ciencia formal es necesario contar con una formulación matemática en tres áreas fundamentales: lógica, computación y probabilidad.

- ✓ **Economía:** el trabajo de la economía y la investigación operativa ha contribuido en gran medida a la noción de agente racional. La teoría de la decisión, que combina la teoría de la probabilidad con la teoría de la utilidad, proporciona un marco completo y formal para la toma de decisiones.

- ✓ **Neurociencia:** la forma exacta en la que en un cerebro genera el pensamiento es uno de los grandes misterios de la ciencia. Se ha observado durante miles de años que el cerebro está de alguna manera involucrado en los procesos de pensamiento, ya que fuertes golpes en la cabeza puede ocasionar minusvalía mental.

- ✓ **Psicología:** la conceptualización del cerebro como un dispositivo de procesamiento de información, es la característica principal de la psicología cognitiva.

- ✓ **Ingeniería computacional:** la investigación en IA ha generado numerosas ideas novedosas de las que se ha beneficiado la informática en general. Como por ejemplo, los computadores personales con interfaces graficas y ratones, entornos de desarrollo rápido, administración automática de memoria, etc.

- ✓ **Lingüística:** la lingüista moderna y la IA "nacieron", al mismo tiempo, en un campo hibrido llamado procesamiento del lenguaje natural. El entendimiento del lenguaje requiere la comprensión de la materia bajo estudio y de su contexto, y no solamente el entendimiento de la estructura de las sentencias.

1.2 Aprendizaje Automático.

Primeramente es necesario partir del concepto de aprendizaje y para esto (Gramajo et al., 1999), manifiestan que es la habilidad innata de adquirir hechos, habilidades y conceptos más abstractos. Según el autor en mención una meta científica valida: "es comprender el aprendizaje humano para reproducir aspectos en un sistema informático y que mediante la construcción de modelos del aprendizaje humano es probable llegar a construir técnicas educativas más eficaces".
(Mitchell, 1997), lo define como: "Un programa aprende de la experiencia E, con respecto a un conjunto de tareas T si su performance en las tareas, medido por P, mejora con la experiencia E".

(Nilsson, 2005), en cambio habla de que: “Una máquina aprende cada vez que cambia su estructura, programa o datos (basado en sus entradas o en respuesta a información externa) de una manera tal que su rendimiento futuro esperado mejora”.

Según (Daume, 2012), el objetivo de aprendizaje automático inductivo es tomar algunos datos de entrenamiento y utilizarlos para inducir una función f. Esta función f se evaluará en los datos de prueba. El algoritmo de aprendizaje automático tiene éxito si su desempeño en los datos de prueba es alta. En la figura 1 se muestra el marco general de la inducción, donde basándose en datos de entrenamiento, el algoritmo de aprendizaje induce una función f que asigna un nuevo ejemplo para una predicción correspondiente.

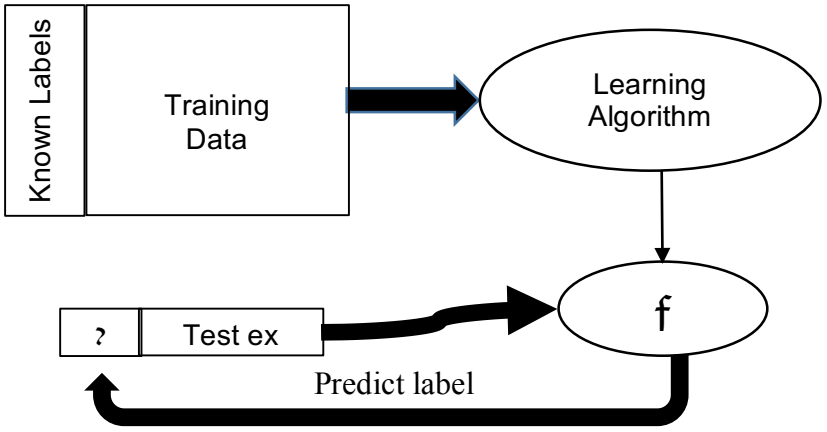

Figura 1. Enfoque general de aprendizaje automático supervisado.

Fuente: (Daume, 2012)

Elaboración. El Autor.

1.2.1 Tipos de Aprendizaje Automático.

El Aprendizaje automático suele dividirse en tres tipos principales: Aprendizaje supervisado, Aprendizaje no supervisado y Aprendizaje por refuerzo.

1.2.1.1 Aprendizaje supervisado.

Se proporciona un conjunto de entrenamiento de ejemplos con las respuestas correctas (objetivos) y, basándose en este conjunto de entrenamiento, el algoritmo generaliza para responder correctamente a todas las entradas posibles. (Marsland, 2015)

Para (Alpaydın, 2014), el objetivo es aprender un mapeo de la entrada a una salida cuyos valores correctos son proporcionados por un supervisor, donde hay una entrada X, una salida Y, y la tarea es aprender el mapeo de la entrada a la salida.

1.2.1.1.1 Clasificación.

El problema de clasificación consiste en tomar vectores de entrada y decidir quienes de ellos pertenecen a N clases, basado en el entrenamiento de ejemplos de cada clase. El punto más importante sobre el problema de clasificación es que cada ejemplo pertenece a exactamente una clase, y el conjunto de clases abarca todo el espacio de salida posible (Marsland, 2015). Para comprender de mejor manera esto se tomara el siguiente ejemplo del mismo autor, donde se considera una máquina expendedora, que utiliza una red neuronal para aprender a reconocer diferentes monedas de Nueva Zelanda, uno de los problemas es ¿Que pasaría si se pone una moneda británica en la máquina?, en ese caso, el clasificador la identifica como la moneda de Nueva Zelanda, que es más cercana en apariencia, pero más bien, el clasificador debe identificar que no es una de las monedas que se ha capacitado. Cuando la moneda es empujada en la ranura, la máquina toma las mediciones de la misma. Estas podrían incluir el diámetro, el peso, y, posiblemente, la forma, y son las características que van a generar el vector de entrada. En este caso, el vector de entrada tendrá tres elementos, cada uno de los cuales será un número que muestra la medición de esa característica (por ejemplo, que 1 = círculo, 2 = hexágono, etc.)

En la figura 2, en la parte superior se muestra las monedas de Nueva Zelanda en la parte de abajo se muestra un conjunto de entradas en 2D con tres clases diferentes de muestra, y dos límites de decisión diferentes; a la izquierda son líneas rectas, y por lo tanto son simples, pero no se categorizan así como la curva no lineal a la derecha.

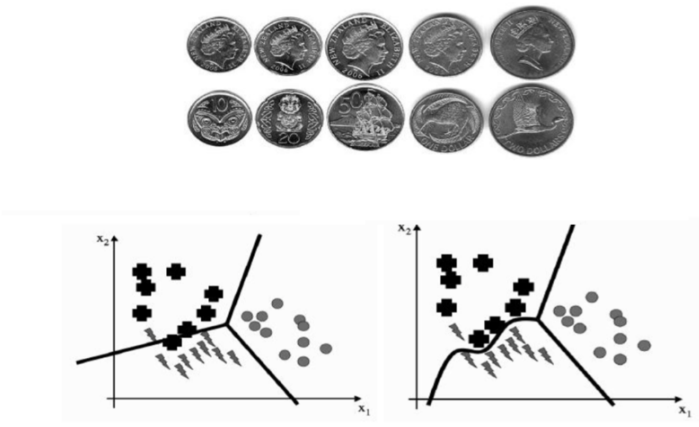

Figura 2. Ejemplo de problema de clasificación.

Fuente: (Marsland, 2015)

1.2.1.1.2 Regresión.

(Drew & White, s. f.), introducen el concepto de regresión de una manera simple ellos mencionan que: “es predecir un conjunto de números, dados otro conjunto de números”.

En general, los números que nos dan se denominan como entradas y los números que queremos predecir como salidas. Lo que hace la regresión diferente de la clasificación es que las salidas son realmente números. En problemas de clasificación, es posible utilizar los números como un código ficticio para una distinción categórica de manera que 0 representa algo y el 1 representa lo contrario. Pero estas cifras son sólo símbolos; no se explota los números mismo de 0 o 1, mas bien se usan como variables ficticias. En la regresión, el hecho esencial acerca de las salidas es que son realmente números: por ejemplo si se quiere predecir cosas como temperaturas, las salidas podrían ser de 50 grados o 71 grados. (Drew & White, n.d.)

Como ejemplos de problemas de regresión podemos considerar los siguientes:

- Predecir cuánto tiempo una persona vivirá dado sus hábitos de fumar.
- Predecir la temperatura del día siguiente dada la temperatura del día anterior.

1.2.1.2 Aprendizaje no supervisado.

El objetivo del aprendizaje no supervisado es encontrar "patrones interesantes" en los datos, esto se puede llamar descubrimiento de conocimiento y es un problema mucho menos bien definido, ya que no se nos dice qué tipo de patrones buscar (Murphy, 2012).
Para (Bell, 2014), el aprendizaje no supervisado, es el extremo opuesto aprendizaje supervisado, donde se deja que el algoritmo encuentre un patrón oculto en una carga de datos. Con el aprendizaje no supervisado no hay respuesta correcta o incorrecta; es sólo un caso de ejecutar el algoritmo de aprendizaje automático y ver qué se producen los patrones y los resultados.

En la figura 3 se puede observar diversos conjuntos de puntos en un espacio de dos dimensiones. El primer grupo (a) parece naturalmente divisible en dos clases, mientras que la segunda (b) parece difícil de dividir en absoluto, y la tercera (c) es problemática.

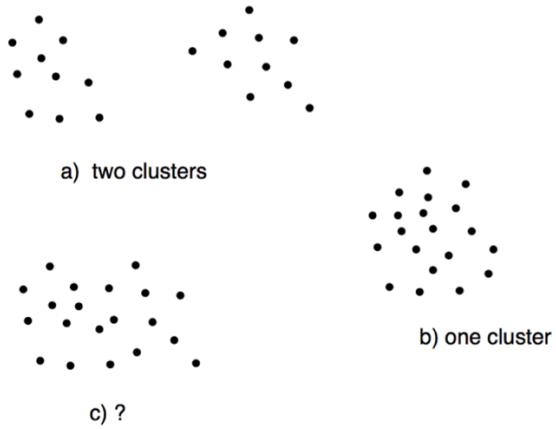

Figura 3. Patrones sin etiqueta.

Fuente: (Nilsson, 2005)

A partir de estos conceptos se puede decir que aprendizaje no supervisado utiliza procedimientos que tratan de encontrar particiones naturales en los patrones, por ejemplo agrupar en un grupo a datos con un gran alto grado de semejanza y en el otro grupo datos que son poco semejantes entre si.

1.2.1.2.1 Clustering.

Para (Bell, 2014), si se reducen todas las definiciones de clustering que existen se puede decir que es "la organización de un grupo de objetos que comparten características similares."

Clustering es utilizado en múltiples campos así por ejemplo:

- ✓ **Internet:** Las redes sociales utiliza la agrupación para determinar las comunidades de usuarios.
- ✓ **Negocios:** investigación de mercado usan mucho agrupación para definir grupos de clientes.
- ✓ **Cumplimiento de la ley:** se ejecuta clustering y otros algoritmos de aprendizaje automático para predecir cuándo y dónde crímenes futuros sucederán.
- ✓ **Informática:** para agrupar los resultados de sensores. Por ejemplo, pensando en un sensor de temperatura, es posible agrupar fecha y la hora en contra de la temperatura.

En la figura 4 se ve que hay tres grupos distintos de los datos; cada uno de esos grupos es un clúster.

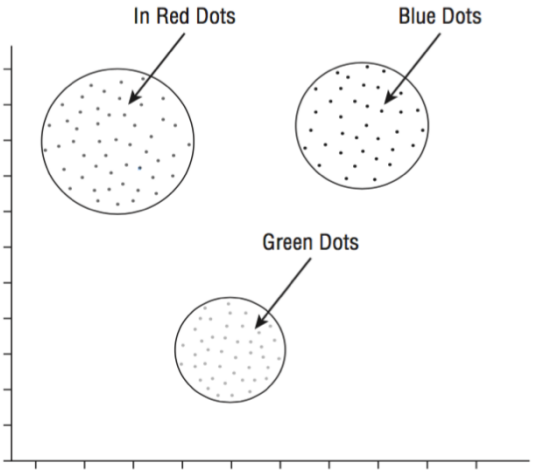

Figura 4. Representación grafica de un clúster

Fuente: (Bell, 2014)

1.2.1.3 Aprendizaje por refuerzo.

Aquí el problema es que sin algunos comentarios sobre lo que es bueno y qué es malo, el agente no tendrá motivos para decidir que hacer. El agente tiene que saber que algo bueno ha sucedido cuando se gana y que algo malo ha sucedido cuando pierde. Este tipo de comentarios se llama una recompensa o refuerzo. Con el aprendizaje por refuerzo los agentes pueden aprender qué hacer, sobre todo cuando no hay maestro diciéndole al agente qué acción tomar en cada circunstancia. (Russell & Norvig, 2004)

1.2.2 Modelos mas utilizados.

Existen varios modelos para aplicación de aprendizaje automático a continuación se presenta un resumen de algunos de ellos.

1.2.2.1 Arboles de Decisión.

(Mitchell, 1997), explica que es un método, en la que la función aprendida está representado por un árbol de decisión que pueden también ser representados como conjuntos de reglas if-then para mejorar la legibilidad humana. Estos métodos de aprendizaje se encuentran entre los más populares algoritmos y se han aplicado con éxito a una amplia gama de tareas, desde aprender a diagnosticar casos médicos hasta aprender a evaluar el riesgo de créditos de los solicitantes de préstamos.

(Bell, 2014), también hace hincapié a los arboles de decisión en el campo medico, donde dice que estos se han diseñado para diagnosticar infecciones de la sangre o incluso predecir los

resultados de ataque al corazón en pacientes con dolor torácico. El autor también manifiesta que los arboles de decisión se utilizan en la industria del juego en reconocimiento de movimientos y reconocimiento facial, donde explica que la plataforma Microsoft Kinect[1] utilizo un millón de imágenes para realizar un seguimiento de los movimientos del cuerpo y entrenó tres árboles dentro de un día, luego usando un clúster de 1.000 núcleos, los árboles de decisión fueron clasificando partes del cuerpo específicas a través de la pantalla.

En la figura 5 se muestra un ejemplo de árbol de decisión, el cual clasifica si es apropiado o no jugar tennis, ordenándolas desde la raíz hasta cierto nodo hoja. Cada nodo en el árbol especifica una prueba de un atributo de la instancia, y cada rama descendente de ese nodo corresponde a uno de los valores posibles para este atributo.

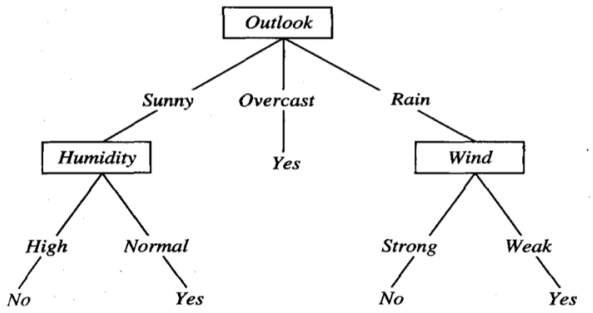

Figura 5. Un árbol de decisión para el concepto jugar tennis

Fuente: (Mitchell, 1997)

Ventajas de arboles de decisión

Se pueden enumerar algunas ventajas para utilizar árboles de decisión:

- ✓ Ayuda a realizar las mejores decisiones con base a la información existente y a las mejores suposiciones.
- ✓ Su estructura permite analizar las alternativas, los eventos, las probabilidades y los resultados.
- ✓ Facilita la interpretación de la decisión adoptada
- ✓ Reduce el numero de variables independientes
- ✓ Funcionan bien con una cantidad razonable de poder de cómputo. Si se tiene un gran conjunto de datos, entonces el aprendizaje de árbol de decisión se manejara muy bien.

[1] Microsoft Kinect: http://www.xbox.com/kinect

Desventajas de arboles de decisión

Existen algunos problemas con los arboles de decisión:

- ✓ Pueden crear modelos excesivamente complejos, en función de los datos presentados en el conjunto de entrenamiento.
- ✓ Sólo es recomendable para cuando el número de acciones es pequeño y no son posibles todas las combinaciones.
- ✓ Presenta inconvenientes cuando la cantidad de alternativas es grande y cuanto las decisiones no son racionales.

1.2.2.1.1 Algoritmo ID3.

En el articulo "*ID3 algorithm based object discrimination for multi object tracking*", escrito por (Kim, Yu, & Kim, 2014), explica que este algoritmo es utilizado para generar una árbol de decisión de un conjunto de datos, donde el árbol de decisión se construye desde un conjunto de datos de ejemplos, cada uno de los cuales tiene varios atributos y pertenece a una clase como (sí o no). Los nodos de decisión del árbol son elegidos por el uso de una medida basada en la entropía conocido como ganancia de información. La entropía significa el grado de congestión de un determinado conjunto de datos . En otras palabras, si los tipos de registros en el conjunto de datos son diversos, la entropía es alta. Por el contrario, cuando los registros tienen un montón de un mismo tipo, la entropía es baja.

En la figura 6, se observa un grafico de entropía que muestra la cantidad de información que está disponible de descubrir teniendo en cuenta lo que ya sabes.

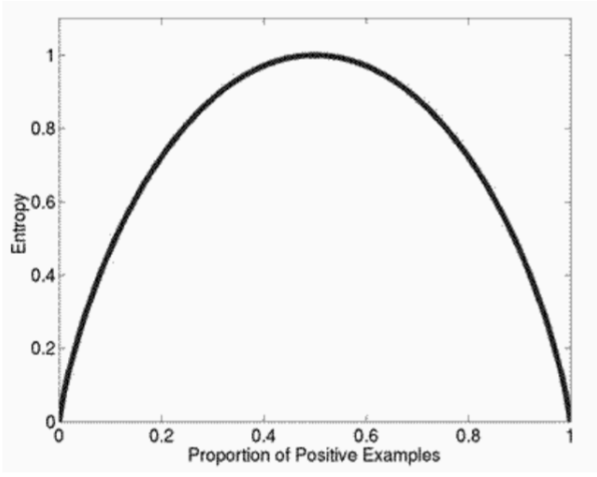

Figura 6. Un grafico de entropía

Fuente: (Marsland, 2015)

1.2.2.1.2 Algoritmo C4.5.

También se basa en el método de ganancia de información, pero permite que los árboles puedan ser usados para clasificación. Este es un algoritmo ampliamente utilizado en usuarios de Weka en la que ejecutan la versión de código abierto de C4.5, el algoritmo J48. Hay mejoras notables en C4.5 sobre el algoritmo ID3 originales. Con la capacidad de trabajar en atributos continuos, el método C4.5 calcula un punto de umbral para la división que se produzca. (Bell, 2014)

Para comprender de mejor manera esto se tomara el siguiente ejemplo del mismo autor en mención:

Dada una lista de valores como la siguiente:

$$85, 80, 83, 70, 68, 65, 64, 72, 69, 75, 75, 72, 81, 71$$

El algoritmo crea un punto de división para el atributo (a) y da un criterio de decisión simple:

$$a \leq 80 \ \mathcal{o} \ a > 80$$

C4.5 tiene la capacidad de trabajar a pesar de perderse los valores de atributos. Los cálculos de ganancia y entropía son simplemente omitidos cuando no hay datos disponibles. Los árboles creados con C4.5 se podan después de la creación ; el algoritmo volverá a examinar los nodos y decidir si un nodo está contribuyendo al resultado en el árbol. Si no es así , entonces se sustituye con un nodo hoja.

1.2.2.2 Maquinas de Vectores de Soporte.

El objetivo de la mayoría de las tareas de aprendizaje automático, es por lo general clasificar algo en un grupo que luego se puede inspeccionar más tarde. Mientras mas tipos de clase se esta intentando clasificar, el proceso se vuelve más complicado. Las Máquinas de vectores soporte sirven para precisamente eso trabajar en clasificaciones desafiantes, estas se pueden utilizar en una variedad de escenarios clasificación, como reconocimiento imagen y el reconocimiento de patrón de escritura a mano.

Este es un método de aprendizaje supervisado, por lo que se necesita tener algunos datos de entrenamiento y algunos datos para probar el algoritmo.

Las máquinas de vectores soporte, se pueden clasificar en lineal y no lineal (Bell, 2014), explica más en su libro *"Machine Learning: Hands-On for Developers and Technical Professionals*", de que tratan cada una de ellas, detallado a continuación:

1.2.2.2.1 Clasificación Lineal.

Se utiliza para determinar a que grupo pertenece un objeto, es decir para establecer la ubicación de los objetos y ver si hay una línea que los divide ordenadamente llamada un hiperplano, en un lado de la línea debe haber claramente un grupo de objetos y en el lado opuesto otro grupo de objetos con la misma claridad.

En la figura 7, se muestra un ejemplo de clasificación lineal, cada objeto que se clasifica se denomina un punto, y cada punto tiene un conjunto de características.

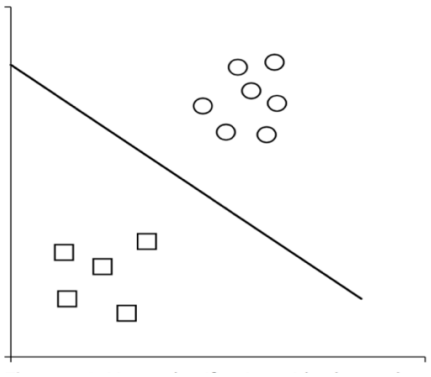

Figura 7. Clasificación Lineal con un Hiperplano.
Fuente: (Bell, 2014)

1.2.2.2.2 Clasificación no Lineal.

En un caso ideal, los objetos se encuentran de un lado u otro del hiperplano. En la clasificación no lineal, se ve objetos apartarse del hiperplano, como se muestra en la figura 8, los objetos rara ves van donde nosotros queramos que vayan.

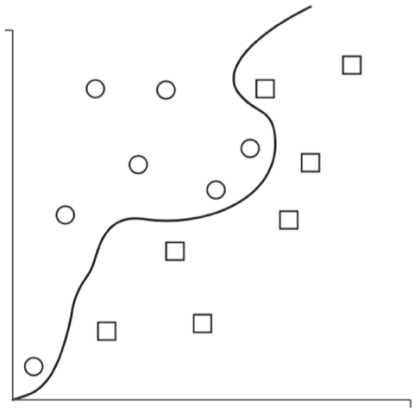

Figura 8. Clasificación no Lineal
Fuente: (Bell, 2014)

1.2.2.2.3 Algoritmo SMO.

Significa Optimización Mínimo Secuencial por sus siglas en español, lo que realiza este algoritmo es; dado un problema de optimización grande, dividirlo en muchos problemas pequeños. Estos pequeños problemas pueden resolverse fácilmente, y de forma secuencial, la respuesta es la misma que se daría si se trataría de resolver todo junto. Además de que la cantidad de tiempo se reduce considerablemente. (Harrington, 2012)

1.2.2.3 Redes Neuronales.

Las redes se basa en una forma simple de entradas y salidas. (Mitchell, 1997), menciona que el estudio de las redes neuronales artificiales (RNA) se ha inspirado en parte por la observación de que los sistemas de aprendizaje biológicos están construidas con muy complejas redes de neuronas interconectadas. En analogía aproximada, las redes neuronales artificiales se construyen a partir de un conjunto densamente interconectado de unidades simples, donde cada unidad tiene un número de entradas de valor real posiblemente las salidas de otras unidades y produce una sola salida de valor real que puede convertirse la entrada a muchas otras unidades.

En la figura 9, se puede observar la estructura de una neurona real donde se tiene una entrada (dendrita), y una salida (axón).

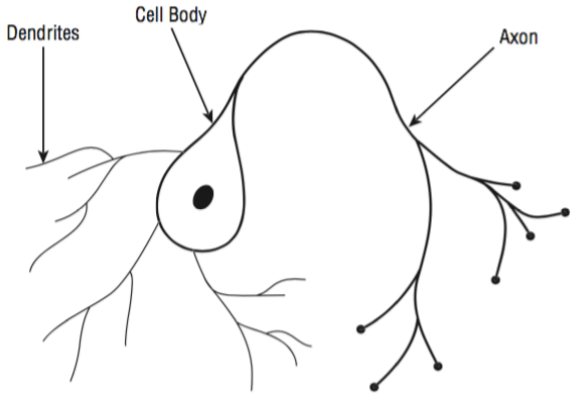

Figura 9. Estructura de una neurona.
Fuente: (Bell, 2014)

1.2.2.3.1 Algoritmo perceptrón.

El perceptrón es una colección de neuronas de McCulloch y Pitts, junto con un conjunto de entradas y algunos pesos para sujetar las entradas a las neuronas. (Marsland, 2015), explica el algoritmo mediante un ejemplo donde supone que presentamos un vector de entrada a la

red y una de las neuronas obtiene la respuesta incorrecta (su salida no coincide con el de destino). Hay *m* pesos que están conectados a esa neurona, uno para cada uno de los nodos de entrada. Lo primero que necesitamos saber es si cada peso es demasiado grande o demasiado pequeño. Algunos de los pesos serán demasiado grandes si la neurona disparo cuando no debería haberlo hecho, y demasiado pequeño si no dispara cuando debería.
Esto es exactamente lo que hace el algoritmo calcula la diferencia entre lo que hizo la neurona, y el objetivo de la neurona, en otras palabras lo que la neurona debería haber hecho.

En la figura 10 se muestra la estructura de una red, los nodos a la izquierda de la figura, sombreados en gris claro, son los nodos de entrada conectados a neuronas McCulloch y Pitts usando conexiones ponderadas.

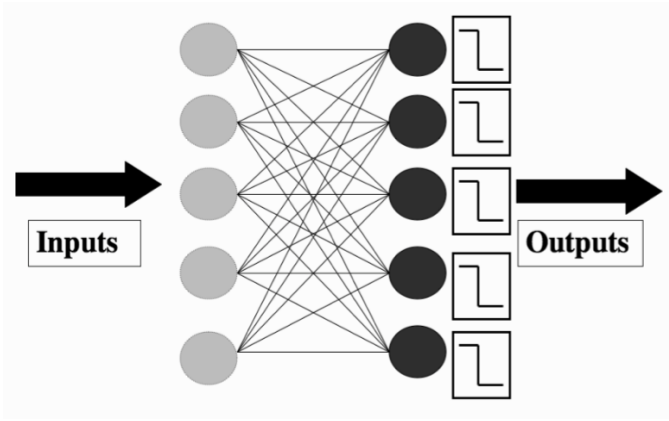

Figura 10. Estructura de una red

Fuente: (Bell, 2014)

1.2.2.4 Aprendizaje Bayesiano.

Razonamiento bayesiano proporciona un enfoque probabilístico a la inferencia. Se basa en la suposición de que las cantidades de interés se rigen por distribuciones de probabilidad y que las decisiones óptimas se pueden hacer por el razonamiento sobre estas probabilidades junto con los datos observados. (Mitchell, 1997)
Un conocimiento básico de los métodos bayesianos es importante para la comprensión y caracterización del funcionamiento de muchos algoritmos en el aprendizaje automático.
(Mitchell, 1997), enumera algunas características de los métodos de aprendizaje bayesiano:

- Cada ejemplo de entrenamiento observado de forma incremental puede disminuir o aumentar la probabilidad estimada que una hipótesis sea correcta.
- El conocimiento previo se puede combinar con los datos observados para determinar la probabilidad de la hipótesis final.

- Métodos bayesianos pueden acomodar hipótesis que hacen predicciones probabilísticas (por ejemplo, las hipótesis tales como "este paciente con neumonía tiene el 93% de probabilidad de la recuperación completa").
- Nuevos casos se pueden clasificar mediante la combinación de las predicciones de múltiples hipótesis, ponderados por sus probabilidades.
- Incluso en los casos en que los métodos bayesianos se demuestran computacionalmente intratables, estos pueden proporcionar un nivel de toma de decisiones óptima contra el que otros métodos prácticos se pueden medir.

1.2.2.4.1 Algoritmo EM.

El algoritmo EM es ampliamente utilizado para el aprendizaje en la presencia de variables no observadas, se puede utilizar incluso para variables cuyo valor nunca se observa directamente, siempre que la forma general de la distribución de probabilidad que rige estas variables se conozca. El algoritmo EM es también la base para muchos algoritmos de agrupamiento no supervisado. (Mitchell, 1997)

Para comprender esto de mejor manera se tomara el siguiente ejemplo del mismo autor en mención:

Si se considera un conjunto de instancias D, generadas por una distribución de probabilidad que es una mezcla de k distintas distribuciones normales, para el caso en que $k = 2$, y donde los casos son los puntos que se muestran a lo largo del eje x. Cada instancia se genera mediante un proceso de dos pasos. En primer lugar, una de las k distribuciones normales se selecciona al azar. En segundo lugar, una única instancia x_i aleatoria se genera de acuerdo con esta distribución seleccionada. Este proceso se repite para generar un conjunto de puntos de datos como se muestra en la figura 11. Para simplificar el análisis, se considera el caso especial de que la selección de la única distribución normal en cada paso se basa en la elección de cada uno con probabilidad uniforme, en donde cada una de las k distribuciones normales tiene la misma varianza σ^2, y donde se sabe σ^2. La tarea de aprendizaje es a la salida de una hipótesis $h = \{\mu_1 \dots \mu_k\}$ que describe los medios de cada una de las distribuciones k.

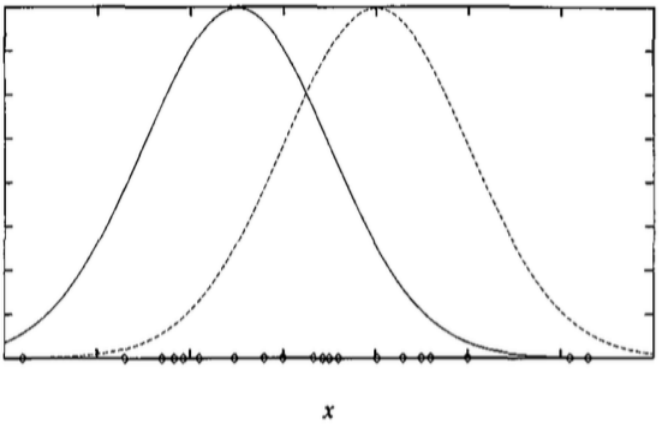

Figura 11. Instancias generadas por una mezcla de dos distribuciones normales con instancias σ.

Fuente: (Mitchell, 1997)

1.2.2.5 Vecinos mas cercanos.

Es un método inductivo muy eficaz para muchos problemas prácticos. Es robusto para datos de entrenamiento ruidosos y bastante eficaz cuando se proporciona un suficientemente amplio conjunto de datos de entrenamiento. (Mitchell, 1997)

En este algoritmo la clasificación de nuevos ejemplos se realiza buscando el conjunto de los k ejemplos más cercanos de entre un conjunto de ejemplos etiquetados previamente guardados y seleccionando la clase más frecuente de entre sus etiquetas. La generalización se pospone hasta el momento de la clasificación de nuevos ejemplos. Una parte muy importante de este método es la definición de la medida de distancia (o similitud) apropiada para el problema a tratar. Esta debería tener en cuenta la importancia relativa de cada atributo y ser eficiente computacionalmente. El tipo de combinación para escoger el resultado de entre los k ejemplos más cercanos y el valor de la propia k también son cuestiones a decidir de entre varias alternativas. (Benítez, Escudero, & Kanaan, 2013)

1.2.2.5.1 K-Nearest Neighbor Algoritmo.

Este algoritmo, en su forma más simple, guarda en memoria todos los ejemplos durante el proceso de entrenamiento y su clasificación se basa en las clases de los k ejemplos más cercanos. Para obtener el conjunto de los k vecinos más cercanos, se calcula la distancia entre el ejemplo a clasificar $x = (x_1, \dots x_m)$ y todos los ejemplos guardados $x_i = (x_{i1}, \dots x_{im})$. Una de las distancias más utilizadas es la euclídea:

$$de(x, x_i) = \sqrt{\sum_{j=1}^{m} (x_j - x_{ij})^2}$$

(Harrington, 2012), explica esto de mejor manera mediante un ejemplo donde considera que si se tiene un conjunto de datos de entrenamiento, donde se tiene etiquetas de todos estos datos, sabemos qué clase de cada pieza de los datos debe caer. Cuando se da una nueva pieza de datos sin una etiqueta, se compara esa nueva pieza de datos a cada pieza de los datos existentes. Entonces se toma las piezas más similares de datos (de los vecinos más cercanos) y se observa sus etiquetas; aquí es donde el k viene (k es un número entero y es por lo general inferior a 20.), se da un voto de mayoría de el k con mas piezas de datos similares, y la mayoría es la nueva clase que asignamos a los datos que se pidieron para clasificar.

1.2.3 Librerías de Aprendizaje Automático.

Existen numerosas librerías relacionadas al aprendizaje automático a continuación se hacen un repaso a algunas de ellas sin tomar en cuenta el lenguaje de programación Python

1.2.3.1 Java.

- **Mallet[2]:** incluye rutinas eficientes para la conversión de texto a "características", dispone de una amplia variedad de algoritmos y el código para evaluar el desempeño del clasificador usando varias métricas de uso común. (Mallet, 2015)

- **Weka[3]:** es una colección de algoritmos de aprendizaje automático para tareas de minería de datos. Los algoritmos se pueden aplicar directamente a un conjunto de datos o pueden ser llamados desde su propio código Java. Weka contiene herramientas para los datos pre-procesamiento, clasificación, regresión, clustering, reglas de asociación, y la visualización. También es muy adecuado para el desarrollo de nuevos sistemas de aprendizaje de máquina. (Weka, 2015)

[2] Mallet: http://mallet.cs.umass.edu/
[3] Weka: http://www.cs.waikato.ac.nz/ml/weka/

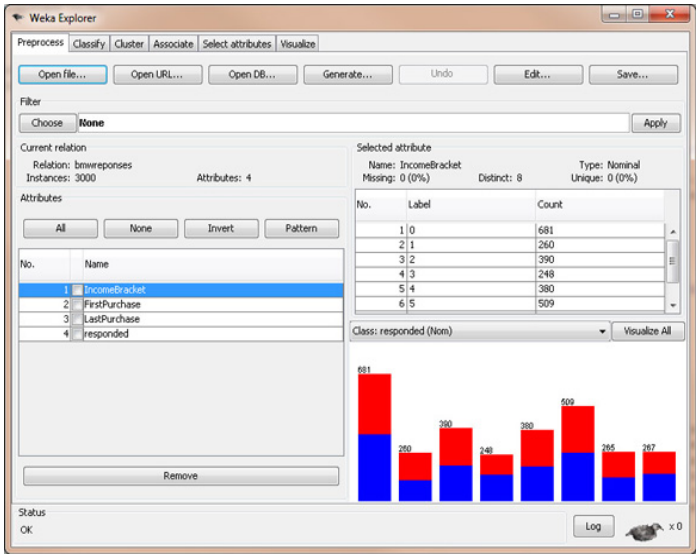

Figura 12. Interfaz de Weka

Fuente: http://www.ibm.com/developerworks/library/os-weka2/

- **Spark MLlib**[4]**:** es utilizable en Java, Scala, Python y Spark[5] además destaca en el cálculo iterativo, contiene algoritmos de alta calidad, es fácil de implementar se ejecuta en clusters y datos Hadoop existentes (Spark, 2015).

- **Java-ML**[6]**:** proporciona una colección de algoritmos de aprendizaje automático, interfaz común para cada tipo de algoritmos, interfaces de la librería destinada a los ingenieros de software y programadores, implementaciones de referencia para los algoritmos descritos en la literatura científica. Además que su código fuente se encuentra bien documentado.

- **Mahout**[7]**:** es un proyecto de código abierto que es parte del proyecto Apache. La característica clave de Mahout es su escalabilidad; funciona ya sea en un solo nodo o un grupo de máquinas. Tiene una estrecha integración con el paradigma Hadoop Map/Reduce para permitir el procesamiento a gran escala. (Bell, 2014)

[4] Spark MLlib: http://spark.apache.org/mllib/
[5] Spark: http://www.adacore.com/sparkpro/
[6] Java-ML: http://java-ml.sourceforge.net/
[7] Mahout: http://mahout.apache.org/

1.2.3.2 C++.

- **Mlpack[8]:** hacen énfasis en la escalabilidad, velocidad y facilidad de uso. Su objetivo es hacer que el aprendizaje de máquina sea posible para los usuarios novatos a través de una API sencilla, consistente, mientras simultáneamente explota las características de C ++ para ofrecer el máximo rendimiento y la máxima flexibilidad para los usuarios expertos.(Mlpack, 2015)

- **Shogun[9]**: permite combinar fácilmente varias representaciones de datos, clases de algoritmos y herramientas de uso general. Permite resolver los problemas de aprendizaje de máquina a gran escala en las máquinas individuales. Una de las características más interesantes de Shogun es que esta escrito en C++ y se puede utilizar de forma transparente en diversos lenguajes y entornos como Python, Octave[10], R, Java, Lua[11], C#, etc. Lo que permite utilizar Shogun como vehículo para exponer su algoritmo para múltiples comunidades. (Shogun, 2015)

- **MLC++[12]:** proporciona algoritmos de aprendizaje automático generales que pueden ser utilizados por los usuarios finales, analistas, profesionales e investigadores. El objetivo principal es proporcionar a los usuarios una amplia variedad de herramientas que pueden ayudar a extraer datos, acelerar el desarrollo de nuevos algoritmos de minería, aumentar la fiabilidad del software, proporcionar herramientas de comparación, y mostrar la información visual.

1.2.3.3 .NET.

- **Accord.NET[13]:** El marco está compuesto de múltiples librerías que abarcan una amplia gama de aplicaciones de computación científica, tales como procesamiento de datos estadísticos, aprendizaje automático, reconocimiento de patrones, visión por ordenador y la audición por ordenador. (Accord, 2015)

[8] Mlpack: http://www.mlpack.org/about.html

[9] Shogun: http://www.shogun-toolbox.org/page/about/project_description

[10] Octave: http://www.gnu.org/software/octave/

[11] Lua: http://www.lua.org/

[12] MLC++: http://www.sgi.com/tech/mlc/

[13] Accord.NET: http://accord-framework.net/intro.html

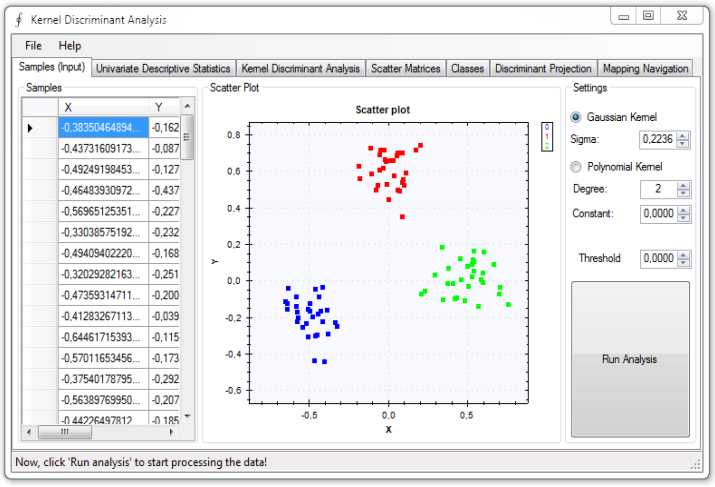

Figura 13. Interfaz de Accord.NET.

Fuente: http://accord-framework.net/

1.2.3.4 Go[14].

- **GoLearn[15]:** fue creado para hacerle frente a la falta de una librería de aprendizaje de máquina todo-en-uno para Go; su meta es obtener simplicidad emparejado con personalización. La simplicidad proviene de la forma como los datos han sido cargados y manipulados en la librería. (GoLearn, 2015)

1.2.3.5 JavaScript.

- **ConvNetJS[16]:** proporciona modelos de aprendizaje profundo en su totalidad en un navegador. No hay requisitos de software, ni los compiladores, ni instalaciones, ni GPU. Incluye módulos comunes de clasificación y regresión, además de un módulo experimental de aprendizaje por refuerzo, basado en Q-learning[17]. (ConvNetJS, 2015)

[14] Go: https://golang.org/
[15] GoLearn: https://github.com/sjwhitworth/golearn
[16] ConvNetJS: http://cs.stanford.edu/people/karpathy/convnetjs/
[17] Q-learning: es un modelo libre de técnica de aprendizaje por refuerzo

1.2.3.6 R.

- **Azure**[18]**:** proporciona herramientas para crear completas soluciones de análisis predictivos en la nube: crear, probar, poner operativos y administrar modelos predictivos rápidamente. (Azure, 2015)

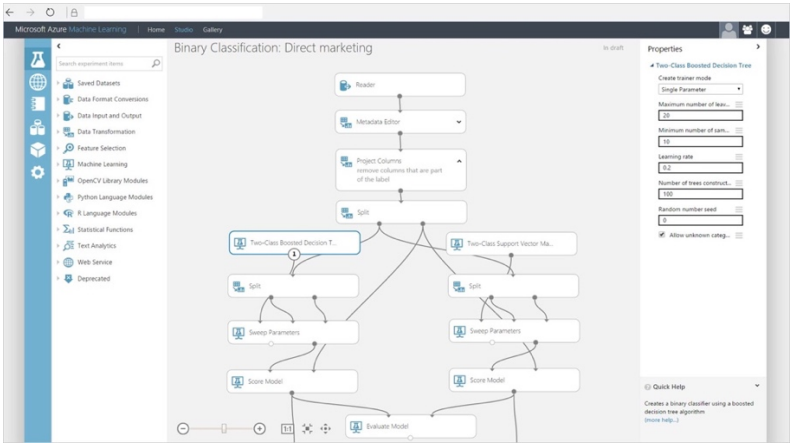

Figura 14. Interfaz de Microsoft Azure.

Fuente: https://azure.microsoft.com/en-us/services/machine-learning/

1.2.3.7 Wolfram.

- **Mathematica**[19]**:** introduce un amplio rango de capacidades integradas de aprendizaje automático, desde funciones altamente automatizadas como predicción y clasificación, hasta funciones basadas en métodos específicos y diagnósticos. Las funciones trabajan en muchos tipos de datos, incluyendo numéricos, categóricos, texto e imagen, permitiéndole realizar a todos aprendizaje automático de avanzada, de manera sencilla. Un amplio rango de tareas pueden ser realizadas, tales como clasificación, reconocimiento de imágenes o clasificación de datos genéricos. (Mathematica, 2015)

[18] Azure: https://azure.microsoft.com/es-es/documentation/articles/machine-learning-what-is-machine-learning/
[19] Mathematica: http://www.wolfram.com/mathematica/new-in-10/highly-automated-machine-learning/

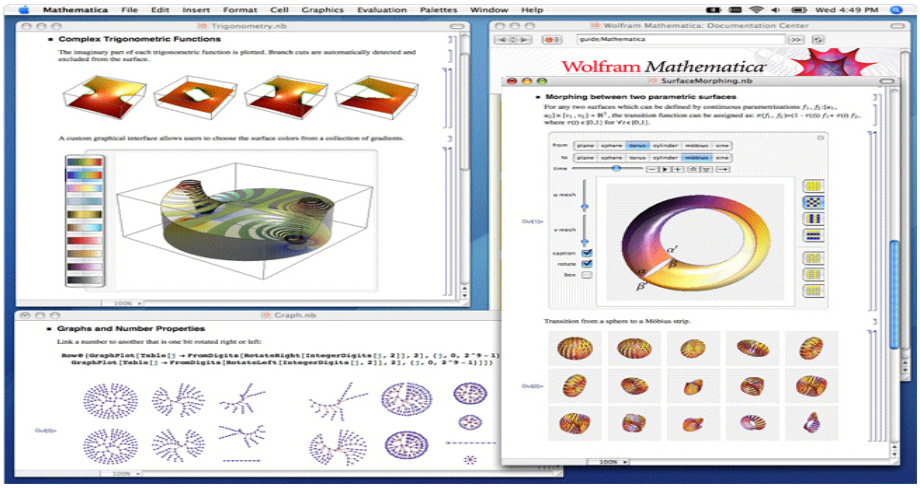

Figura 15. Interfaz de Mathematica.

Fuente: http://www.wolfram.com/products/mathematica//overview/deploy.es.html?footer=lang

1.2.4 ¿Por qué usar Python?.

Python es un gran lenguaje para el aprendizaje automático por un gran número de razones. En primer lugar, Python tiene una sintaxis clara, es por eso que se ha ganado el nombre de pseudo-código ejecutable, además que se puede programar en cualquier estilo: orientado a objetos, procedimental, funcional. En segundo lugar, hace que la manipulación de texto sea muy fácil. Hay una serie de librerías para acceder a las páginas web, y la manipulación intuitiva de texto. Un gran número de personas y organizaciones utilizan Python, por lo que una gran cantidad de ejemplos están disponibles, lo que hace que el aprendizaje sea rápido, además existen un gran numero de módulos disponibles para muchas aplicaciones, así que hay un amplio desarrollo y documentación para Python (Harrington, 2012)

Python además tiene librerías especialmente para operaciones con matriz matemáticas, pero ¿Porque seleccionar Python para aplicaciones de aprendizaje automático y no otras librerías?. (Harrington, 2012), sostiene que si bien hay librerías de matriz matemáticas para lenguajes de bajo nivel como Java y C. El problema de estos lenguajes de programación es que se necesita una gran cantidad de código para hacer cosas simples. Sostiene que en primer lugar, se tiene que encasillar las variables, y luego con Java se tiene que escribir cada vez setters y getters. Además de subclases, existe sub-métodos de clase, incluso si no se va a utilizar. Al final, menciona que se ha escrito una gran cantidad de código y a veces este tedioso código hace cosas simples.

Existen una ventaja no solo en el numero de librerías de aprendizaje automático en Python sino también en rendimiento frente a otras librerías, un ejemplo es la librería Scikit-Learn[20] que es quizá la mas utilizada en el área.

Junto al lenguaje de programación R[21] son los son los mas populares en el campo de la ciencia de los datos, aunque la falta de versatilidad fuera de la manipulación de datos de este ultimo hace que Python tome ventaja al ser además útil para el desarrollo de otro tipo de aplicaciones. Otra ventaja que presenta frente a otros lenguajes de programación es que esta se usa en todas las fases de trabajo que se presentan en el modelado de aprendizaje automático. En este trabajo para las fases previas a la aplicación de las técnicas de aprendizaje automático se utilizo herramientas en Python como pandas[22], NumPy[23] las cuales sirvieron para el análisis estadístico de los datos, además se uso la librería seaborn[24] la misma que esta basada en matplotlib[25] para la visualización de los mismos. Por ultimo herramientas que van desde el procesamiento de datos pasando por la selección de las características mas representativas para la solución de nuestro problema de clasificación y regresión la validación y el uso de métricas para contrastar los resultados fueron propias de este lenguaje de programación.

1.2.5 El ciclo de aprendizaje automático.

Para (Bell, 2014), un proyecto de aprendizaje automático es básicamente un ciclo de acciones que deban llevarse a cabo, mostrados en la figura 16.

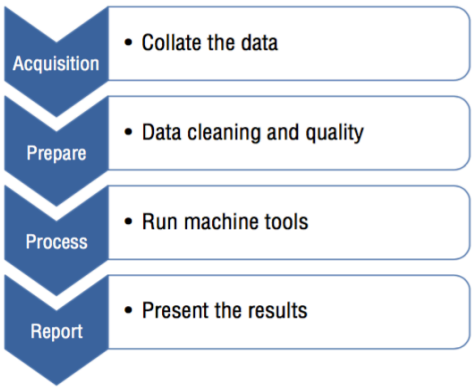

Figura 16. Ciclo de aprendizaje automático.

Fuente: (Bell, 2014)

[20] Scikit-learn: http://scikit-learn.org/
[21] R: https://www.r-project.org/
[22] Pandas: http://pandas.pydata.org/
[23] NumPy: http://www.numpy.org/
[24] seaborn: https://stanford.edu/~mwaskom/software/seaborn/
[25] matplotlib: http://matplotlib.org/

Según (Bell, 2014), primero se debe adquirir datos de muchas fuentes, luego a estos datos se debe ser limpiar y comprobar su calidad antes de cualquier procesamiento, todo esto se produce en la fase de preparación. La fase de procesamiento es donde se realiza el trabajo. Las rutinas de aprendizaje automático que se hayan creado realizan esta fase. Finalmente, se presentan los resultados que puede suceder en una variedad de formas, tales como la reinversión de los datos en un almacén de datos o informar los resultados como una hoja de cálculo. A continuación se detalla más de estos 4 pasos.

1.2.5.1 Adquisición.

Este proceso es muy importante debido a que el algoritmo aprende y descubre a partir de los datos disponibles. Lo primero que se tiene que tener en cuenta antes de comenzar con el proceso, es saber qué es lo que se quiere obtener y cuáles son los datos que nos permitirán realizar esta tarea.

Para que los datos sean buenos (Cuesta, 2010), manifiesta que deben cumplir las siguientes características:

- Completa
- Coherente
- No ambigua
- Contable
- correcta
- Estandarizada
- No redundante

Existen muchos tipos de variables, en este trabajo de fin de titulación se va a utilizar dos: discretas y continuas.

- **Variables discretas:** es una variable que solo puede tomar cierto numero de valores, es decir únicamente aquellos que pertenecen al conjunto.
- **Variables continuas:** pueden tomar cualquier valor a lo largo de un continuo o de una escala numérica continua por ejemplo: números reales, como el 1, 2.3, 3.1416 etc.

Hay una parte muy importante en esto y es la selección de las variables correctas ya puede mejorar el proceso de aprendizaje, mediante la reducción de la cantidad de ruido (información inútil) que pueden influir en el resultado que se quiere obtener. La selección de variables, por

lo tanto, puede reducir eficazmente la varianza de las predicciones. Con el fin de involucrar a sólo las variables útiles en la formación y dejar de lado los redundantes, se puede utilizar varias técnicas entre ellas están las de enfoque univariante y de enfoque retrospectivo.

- **Enfoque univariante:** Selecciona las variables más relacionadas con el resultado de destino.
- **Enfoque retrospectivo:** Mantiene sólo las variables que se pueden eliminar del proceso de aprendizaje sin dañar su rendimiento.

Para realizar este análisis se ha utilizado la librería Scikit-Learn, para lo cual se ha utilizado chi2 para el enfoque de clasificación y f_regression para el enfoque de regresión.

- **chi2:** realiza la estadística de chi2 para objetivos categóricos, que es menos sensible a la relación no lineal entre la variable predictiva y su objetivo.
- **f_regression:** usado sólo para objetivos numéricos y con base en el desempeño de regresión lineal.

1.2.5.2 Preparación.

Esto incluye limpieza de datos y transformación de datos, tales como el manejo de los valores perdidos y la eliminación de ruido, es decir, valores incorrectos o inesperados. Si algunos atributos importantes se pierden, entonces todo el estudio puede fallar. Para el éxito del proceso, es bueno tener en cuenta el mayor número posible de atributos en esta etapa.

Una herramienta muy útil para la limpieza y transformación de datos es OpenRefine[26] anteriormente conocido como Google Refine.

1.2.5.2.1 OpenRefine.

Es una herramienta muy útil en la limpieza de datos, exploración de datos, y la transformación de datos. Es una aplicación de código abierto que se ejecuta directamente en su ordenador, pasando por alto el problema de cargar su información delicada a un servidor externo. (Cuesta, 2010)
Además (Cuesta, 2010) en su libro "*Practical Data Analysis"* hace un repaso a algunas de las características mas importantes de OpenRefine resumidas a continuación:

[26] OpenRefine: http://openrefine.org/

- **Text facet:** es similar a un filtro en una hoja de calculo, esto nos permite combinar una misma información que puede ser escrita de diferentes maneras.
- **Text filters:** Podemos filtrar una columna mediante el uso de una cadena de texto específico o el uso de una expresión regular (expresiones regulares de Java).
- **Transforming data:** se puede transformar por ejemplo formatos de fecha o simplemente cadenas de texto a un valor numérico que la represente.
- **Exporting data:** se puede exportar desde un proyecto OpenRefine a formatos como TSV, CSV, Excel, HTML table, JSON.

1.2.5.3 Procesamiento.

En esta etapa se decide sobre cuál tipo de aprendizaje a utilizar, por ejemplo, clasificación, regresión, o clustering. Hay dos objetivos la predicción y descripción. La predicción se refiere a menudo como supervisada. mientras que la descriptiva incluye los aspectos no supervisados y de visualización.

Teniendo la estrategia se procede a la selección del algoritmo, es decir, ahora se decide sobre las tácticas. Esta etapa incluye seleccionar el método específico que se utilizará para resolver el problema. Por ejemplo, considerando la precisión frente a la comprensión, la primera es mejor con redes neuronales, mientras que el segundo es mejor con los árboles de decisión. Finalmente se llega a la implementación del algoritmo.

En el presente trabajo de fin de titulación se enfocará sobre algoritmos de clasificación y regresión la diferencia entre estas dos es que la primera tiene la tarea de asignar una clase, es decir predecir a que clase pertenece un conjunto de datos y la segunda tiene el objetivo de predecir valores continuos, aquí es muy importante entender que en los problemas de clasificación los valores a predecir son discretos.

La herramienta utilizada para implementar los algoritmos en esta etapa además del análisis estadístico de los datos fue Jupyter Notebook[27].

1.2.5.3.1 Jupyter Notebook.

Es una aplicación web interactiva que permite crear y compartir código, combina dos componentes:

[27] Jupyter Notebook: http://jupyter.readthedocs.io/en/latest/install.html

- **Una aplicación web** : basada en navegador para la creación interactiva de documentos que combinan texto explicativo, matemáticas, cálculos y su producción multimedia.
- **Documentos portátiles** : una representación de todo el contenido visible en la aplicación web, incluyendo entradas y salidas de los cálculos, texto explicativo, matemáticas, imágenes y representaciones multimedia.

Además es una herramienta muy fácil de instalar, el proceso del mismo se encuentra en el Anexo 1.

1.2.5.4 Resultados.

En esta etapa, se evalúan e interpretan los valores obtenidos, con respecto a los objetivos definidos en el primer paso.
Usando el conocimiento descubierto. En esta etapa se incorpora el conocimiento a otro sistema para acciones futuras. El conocimiento se vuelve activo en el sentido de que es posible realizar cambios en el sistema y medir los efectos.

1.2.6 Aplicaciones de Aprendizaje Automático.

Hoy en día existen aplicaciones que utilizan agentes basados en aprendizaje en numerosas ramas de la industria y de la ciencia. Por ejemplo:

NLP: (Procesamiento de Lenguaje Natural en sus siglas en español), "(....) es construir sistemas y algunos mecanismos que permitan la comunicación entre personas y maquinas por medio de lenguajes naturales". (Benavides Cañon & Rodriguez Correa, 2007)

Sistemas de Recuperación de Información: (Salton, 1983), piensa que: "cualquier SRI puede ser descrito como un conjunto de ítems de información (DOCS), un conjunto de peticiones (REQS) y algún mecanismo (SIMILAR) que determine qué ítem satisfacen las necesidades de información expresadas por el usuario en la petición".

Diagnóstico Médico: Según (Montoya, Jairo, Chávez, Jesús, & Mora, 2013), en su escrito manifiestan que el aprendizaje automático servirá como asistente a la hora de emitir un diagnóstico médico como un caso de clasificación.
Con el número de consumidores que utilizan los teléfonos inteligentes y los dispositivos relacionados para recopilar una serie de datos de salud como el peso, el ritmo cardíaco, el

pulso, presión arterial, e incluso de niveles de glucosa en sangre, ahora es posible hacer un seguimiento y rastreo de la salud del usuario con regularidad y ver patrones en las fechas y horas. Sistemas de aprendizaje automático puede recomendar alternativas más saludables para el usuario a través del dispositivo (Bell, 2014).

Finanzas e Industria bancaria: desde el punto de vista de los negocios, lo que interesa es construir sistemas que incorporen conocimiento y, de esta manera, sirvan de ayuda a los procesos de toma de decisiones en el ámbito de la gestión empresarial. (de Andrés Suárez, 2000)

Análisis de imágenes: Para reconocer entre otros personas, rostros es utilizado actualmente en algunos aeropuertos.

Juegos: Hoy en día existen muchos juegos que utilizan aprendizaje automático entre los mas populares los juegos de ajedrez.
Según (Vila & Penín, 2007), en el ámbito comercial y de entretenimiento, existen múltiples juegos que pueden utilizarse con fines educativos. Uno de los más populares es Brain Training[28], de la empresa Nintendo. El juego se concibe con el objetivo de mejorar la capacidad mental del usuario, proponiendo ejercicios de memoria, de razonamiento, etc para que el jugador mejore su habilidad y consiga el reto de llegar a la edad cerebral de veinte años, edad óptima según la escala propia del juego.

Robótica: en la Robótica se utilizan modelos de aprendizaje automático para controlar el desplazamiento de robots, planificación de tareas entre otras tareas. Esto facilita que una máquina adquiera habilidades sensoriales y motoras, lingüísticas e interactivas.

Sistemas de Recomendación: (Velez-Langs & Santos, 2006), manifiesta que: “son sistemas que se encargan de recomendar o sugerir a los usuarios ítems o productos concretos basándose en sus preferencias”.

[28] Brain Training: https://www.nintendo.es/Juegos/Nintendo-DS/Brain-Training-del-Dr-Kawashima-Cuantos-anios-tiene-tu-cerebro--270627.html

1.3 Librerías de Aprendizaje Automático en Python.

1.3.1 Scikit-learn.

Desde su lanzamiento en 2007, scikit-learn se ha convertido en una de las mas populares librerías de aprendizaje automático de código abierto para Python. Proporciona algoritmos para tareas de aprendizaje automático incluidos clasificación, regresión y agrupación. También proporciona módulos de extracción de características, procesamiento de datos, y evaluación de modelos. Es también popular para la investigación académica, ya que tiene una API bien documentada, fácil de usar y versátil. Los desarrolladores pueden utilizar scikit-learn para experimentar con diferentes algoritmos cambiando sólo unas pocas líneas de código. Esta licenciado bajo la licencia BSD[29] permisiva, por ende se puede utilizar en aplicaciones comerciales sin restricciones. (Hackeling, 2014)

(Abraham et al., 2014), dicen que: "en scikit-learn, todos los objetos y algoritmos aceptan los datos de entrada en forma de matrices de 2 dimensiones de tamaño: muestras × características. Esta convención hace que sea genérico e independiente del dominio".

En scikit-learn existen estimadores, que predicen un valor basado en los datos observados, todos los estimadores implementan: ajuste y predicen métodos. El primer método se utiliza para aprender los parámetros de un modelo, y el segundo método se utiliza para predecir el valor de una variable de respuesta para una variable explicativa utilizando los parámetros aprendidos. Es fácil experimentar con diferentes modelos porque todos los estimadores aplican el ajuste y predicen métodos. (Hackeling, 2014)

(Abraham et al., 2014) en su escrito profundiza mas sobre los conceptos de estimador y predictor, además añade un termino "transformador" explicados a continuación:

- **Estimador:** la interfaz estimador, es el núcleo de la librería, expone un método de ajuste para el aprendizaje de los parámetros del modelo de datos de entrenamiento. Todos los algoritmos de aprendizaje supervisado y no supervisado (clasificación, regresión o agrupación) están disponibles como objetos de aplicación de esta interfaz.

- **Predictor:** un predictor es un estimador con un método de predecir, que toma una matriz de entrada *X_test* y hace predicciones para cada muestra en ella. En el caso

[29] BSD: Berkeley Software Distribution

de los estimadores de aprendizaje supervisado, este método típicamente devuelve las etiquetas predichas o valores calculados a partir del modelo estimado.

- **Transformador:** es común para modificar o filtrar los datos antes de alimentar a un algoritmo de aprendizaje, algunos estimadores, llamados transformadores, implementan un método de transformación. Algoritmos de pre procesamiento, para selección y reducción de dimensionalidad están incluidos como transformadores dentro de la librería.

1.3.1.1 Regresión Lineal.

La clase de regresión lineal en scikit-learn es la siguiente:

class sklearn.linear_model.LinearRegression(*fit_intercept=True*, *normalize=False*, *copy_X=True*, *n_jobs=1*)

Parámetros:

- ***fit_intercept : boolean, optional***

Para calcular la intersección en este modelo.

- ***normalize : boolean, optional, default False***

Si es true, los regresores X se normalizarán antes de regresión.

- ***copy_X : boolean, optional, default True***

Si es true, se copiará X; lo demás, se puede sobrescribir.

- ***n_jobs : int, optional, default 1***

El número de puestos de trabajo a utilizar para el cálculo. Si es -1 se utilizan todas las CPU.

Atributos:

- ***coef_ : array, shape (n_features,) or (n_targets, n_features)***

Coeficientes estimados para el problema de regresión lineal

- ***intercept_ : array***

Término independiente en el modelo lineal.

(Hackeling, 2014), explica como scikit-learn implementa regresión lineal, la cual clasifica como regresión lineal simple y múltiple cabe mencionar que la clase utilizada para estas dos es la misma detallada anteriormente. A continuación se presenta lo expuesto por el autor:

1.3.1.1.1 Regresión lineal simple.

En scikit-learn el método fit de LinearRegression aprende los parámetros del siguiente modelo para la regresión lineal simple:

$$y = \alpha + \beta x$$

Donde:

y es el valor predicho de la variable de respuesta,

x es la variable independiente,

α termino de intercepción y el coeficiente,

β son parámetros del modelo que se aprendieron por el algoritmo de aprendizaje.

Varias medidas pueden utilizarse para evaluar la capacidad de predicción de un modelo. Scikit-learn utiliza un método para calcular *r-squared*. Una puntuación de *r-squared* de uno indica que la variable de respuesta puede predecirse sin ningún error utilizando el modelo. Otros métodos, incluido el método utilizado por scikit-learn, no calculan *r-squared* como el cuadrado de Pearson's, y puede devolver un *r-squared* negativo si el modelo funciona muy mal. El método utilizado por scikit-learn para calcular r-squared es con la siguiente formula:

$$SS_{tot} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

En primer lugar se debe medir la suma total de cuadrados, donde:

y_i es el valor observado de la variable de respuesta para la instancia de prueba ith,

$\bar{y}$ es la media de los valores observados de la variable de respuesta.

Después se debe encontrar la suma residual de los cuadrados. Esta es la función de costo.

$$SS_{tot} = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Finalmente se puede encontrar *r-squared* usando la siguiente formula

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

1.3.1.1.2 Regresión lineal múltiple.

Es una generalización de regresión lineal simple que puede utilizar múltiples variables explicativas. Formalmente, la regresión lineal múltiple es el siguiente modelo:

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

Regresión lineal simple utiliza una sola variable explicativa con un coeficiente único, regresión lineal múltiple utiliza un coeficiente para cada uno de un número arbitrario de variables explicativas.

$$Y = X\,\beta$$

Para la regresión lineal simple, esto es equivalente a lo siguiente:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} \alpha & + & \beta X_1 \\ \alpha & + & \beta X_2 \\ & \vdots & \\ \alpha & + & \beta X_n \end{bmatrix} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix} \times \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

Donde:

Y es un vector columna de los valores de las variables de respuesta para los ejemplos de entrenamiento,
β es un vector columna de los valores de los parámetros del modelo,
X, es una matriz $m\ x\ n$ dimensional de los valores de las variables explicativas para los ejemplos de entrenamiento,
m es el número de ejemplos de entrenamiento y,
n es el número de variables explicativas.

Si se sabe los valores de Y y X de los datos de entrenamiento. Se puede encontrar el valor de β, que minimiza la función de coste. Se puede resolver β como sigue:

$$\beta = (X^T X)^{-1} X^T Y$$

Se puede resolver β usando NumPy.

1.3.1.1.3 Regularización.

“Regularización es una colección de técnicas que se pueden utilizar para evitar sobreajuste. Regularización añade información en forma de una sanción contra la complejidad para un problema” (Hackeling, 2014).

Al probar un estimador o establecer hiperparámetros, se necesita un indicador fiable para evaluar su desempeño. No es aceptable usar los mismos datos para el entrenamiento como para la prueba, porque conduce a un rendimiento del modelo demasiado confiado, fenómeno también conocido como sobreajuste. *Cross-validation* es una técnica que permite evaluar de manera fiable un estimador en un determinado conjunto de datos. Consiste en forma iterativa montar el estimador en una fracción de los datos, llamado datos de entrenamiento, y probar esto, en los datos omitidos, llamados datos de prueba. Una de las estrategias mas simple y mas utilizada para dividir los datos es *k-fold cross-validation* consiste en dividir (al azar o no) las muestras en k subconjuntos: cada subconjunto es usado como datos de prueba, mientras que los otros k - 1 subconjuntos se utilizan para entrenar el estimador. (Abraham et al., 2014)

La figura 17 representa *cross validation* con cinco particiones donde el conjunto de datos original, se divide en cinco subconjuntos de igual tamaño, con etiquetas de la A a la E. Las particiones se giran hasta que los modelos han sido entrenados y probados en todas las particiones.

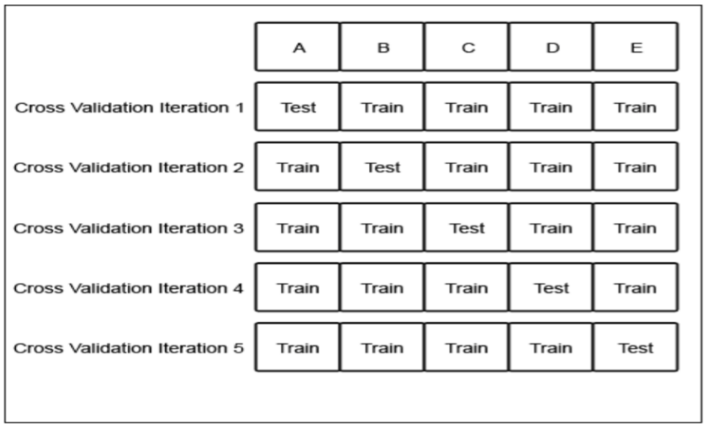

Figura 17. Ejemplo de cross-validation con 5 particiones.

Fuente: (Hackeling, 2014)

(Hackeling, 2014), explica que scikit-learn ofrece varios modelos de regresión lineal regularizados uno de ellos es regresión Ridge, también conocida como la regularización de Tikhonov, quien penaliza los parámetros que se vuelven demasiado grandes.

La clase de regresión Ridge de scikit-learn es la siguiente:
class sklearn.linear_model.Ridge(*alpha=1.0*, *fit_intercept=True*, *normalize=False*, *copy_X=True*, *max_iter=None*,*tol=0.001*, *solver='auto'*, *random_state=None*)

Parámetros:

- ***alpha : {float, array-like}, shape (n_targets)***

Valores pequeños positivos de alfa mejoran el acondicionamiento del problema y reducen la varianza de las estimaciones.

- ***copy_X : boolean, optional, default True***

Si es True, se copiará X; lo demás, se puede sobrescribir.

- ***fit_intercept : boolean***

Para calcular la intersección en este modelo.

- ***max_iter : int, optional***

El número máximo de iteraciones para el solucionador de gradiente conjugado

- ***normalize : boolean, optional, default False***

Si es true, los regresores X se normalizarán antes de regresión.

- ***solver : {'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag'}***

Solucionador para su uso en las rutinas de cálculo

- ***tol : float***

La precisión de la solución.

- ***random_state : int seed, RandomState instance, or None (default)***

La semilla del generador de números pseudo aleatorio a usar cuando se barajan los datos.

Regresión Ridge modifica la suma residual de la función de coste cuadrático mediante la adición de la norma L2 de los coeficientes, como sigue:

$$RSS_{ridge} = \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

λ es un hiperparámetro que controla la fuerza de la penalización. Los hiperparámetros son parámetros del modelo que no se aprenden de forma automática y se debe configurar manualmente. A medida que aumenta λ (lambda), la penalización sube, y el valor de la función costo incrementa. Cuando λ es igual a cero, la regresión Ridge es igual a regresión lineal.

(Hackeling, 2014), explica que scikit-learn además proporciona una implementación de contracción menos absoluta y operador de selección (LASSO) quien penaliza los coeficientes mediante la adición de su norma $L1$ a la función costo, como sigue:

$$RSS_{lasso} = \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda \sum_{j=1}^{p} \beta_j$$

LASSO produce parámetros dispersos; la mayor parte de los coeficientes se convertirá en cero, y el modelo dependerá de un pequeño subconjunto de características. En contraste, la regresión ridge produce modelos en los que la mayoría de los parámetros son pequeños pero no cero. Scikit-learn proporciona una implementación de regularización red elástica, que combina linealmente las sanciones $L1$ y $L2$ utilizados por la regresión lasso y ridge, es decir, lasso y regresión ridge son ambos casos especiales del método red elástica en el que el hiperparámetro ya sea para la penalización $L1$ o $L2$ es igual a cero.

1.3.1.2 Regresión logística.

(Hackeling, 2014), explica que la regresión logística se utiliza para tareas de clasificación. En clasificación binaria, el clasificador debe asignar casos a una de dos clases. En clasificación multiclase, el clasificador debe asignar una de varias etiquetas a cada instancia. En la regresión logística, la variable de respuesta describe la probabilidad de que el resultado es el caso positivo. Si la variable de respuesta es igual o supera un umbral de discriminación, la clase positiva se predijo; de lo contrario, la clase negativo se predijo. La variable de respuesta se modela como una función de una combinación lineal de las variables explicativas utilizando la función logística. Teniendo en cuenta por la siguiente ecuación, la función logística siempre devuelve un valor entre cero y uno:

$$F(t) = \frac{1}{1 + e^{-t}}$$

La clase que utiliza scikit-learn para regresión logística es la siguiente:

class sklearn.linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0,fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100,multi_class='ovr', verbose=0, warm_start=False, n_jobs=1)
Parámetros:

- ***penalty : str, 'l1' or 'l2'***

Se utiliza para especificar la norma utilizada en la penalización

- ***dual: bool***

formulación dual o primaria. Formulación dual sólo se implementa para la pena de L2 con solucionador LIBLINEAR

- ***C : float, optional (default=1.0)***

Inverso de la fuerza de regularización; debe ser un float positivo. Al igual que en las máquinas de vectores de soporte, los valores más pequeños especifican regularización más fuerte.

- ***fit_intercept : bool, default: True***

Especifica si una constante (también conocido como el sesgo o en el origen) debe añadirse a la función de decisión.

- ***intercept_scaling : float, default: 1***

Útil sólo si es solucionador LIBLINEAR

- ***class_weight : dict or 'balanced', optional***

Pesos asociados con las clases en forma . Si no se da, todas las clases se supone que tiene un peso.{class_label: weight}

- ***max_iter : int***

Útil sólo para los newton-cg, SAG y lbfgs solucionadores. El número máximo de iteraciones tomadas por los solucionadores de converger.

- ***random_state : int seed, RandomState instance, or None (default)***

La semilla del generador de números pseudo aleatorio a usar cuando se barajan los datos.

- ***solver : {'newton-cg', 'lbfgs', 'liblinear', 'sag'}***

Algoritmo para el uso en el problema de optimización.

- ***tol : float, optional***

Tolerancia para detener criterios.

- ***multi_class : str, {'ovr', 'multinomial'}***

opción multiclase puede ser 'SOB' o 'multinomial'. Si la opción elegida es 'SOB', entonces un problema binario es apto para cada etiqueta.

- ***verbose : int***

Para los solucionadores LIBLINEAR y lbfgs establecidos detallado a cualquier número positivo para la verbosidad.

- ***warm_start : bool, optional***

Cuando se establece en True, reutilizar la solución de la llamada anterior para encajar como inicialización, de lo contrario, simplemente borrar la solución anterior.

- ***n_jobs : int, optional***

Número de núcleos de CPU utilizados durante la validación cruzada bucle. Si se les da un valor de -1, se utilizan todos los núcleos.

(Hackeling, 2014), profundiza mas en su libro sobre clasificación binaria y multiclase ambos detallados a continuación:

1.3.1.2.1 Clasificación Binaria.

Existen varias métricas para evaluar el desempeño de clasificadores binarios, entre los más conocidos están *exactitud, precisión, recall, F-1 y ROC AUC*. Todas estas medidas dependen de los conceptos de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. Positivo y negativo se refieren a las clases. Mientras que verdadero y falso indican si la clase predicha es la misma que la clase verdadera.

- ***Exactitud*:** es la fracción de casos que fueron clasificados correctamente, es una medida intuitiva del desempeño del modelo.

Para entender esto de mejor manera se toma el siguiente ejemplo:

Se considera una tarea de clasificación en el que un sistema de aprendizaje automático observa tumores y debe predecir si estos tumores son malignos o benignos.
Cuando el sistema clasifica correctamente un tumor maligno como maligno, la predicción se llama un verdadero positivo. Cuando el sistema clasifica incorrectamente un tumor benigno

como maligno, la predicción es un falso positivo. Del mismo modo, un falso negativo es una predicción incorrecta de que el tumor es benigno, y un verdadero negativo es una predicción correcta de que el tumor es benigno. Estos cuatro resultados pueden utilizarse para calcular varias medidas comunes de clasificación de rendimiento.
La exactitud se calcula con la siguiente fórmula:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

Donde:
TP es el número de verdaderos positivos,
TN es el número de verdaderos negativos,
FP es el número de falsos positivos,
y FN es el número de falsos negativos.
La precisión es la fracción de los tumores que se prevé que son malignos y que son realmente malignos. Precisión se calcula con la siguiente fórmula:

$$P = \frac{TP}{TP + FP}$$

- ***Recall*:** en este ejemplo es la fracción de los tumores malignos que el sistema identifico. *Recall* se calcula con la siguiente fórmula:

$$R = \frac{TP}{TP + FN}$$

Scikit-learn proporciona una función para calcular la *Exactitud y Recall* de un clasificador de un conjunto de predicciones.

1.3.1.2.2 Clasificación multiclase.

Scikit-learn utiliza una estrategia llamada *one-vs.-all* o *one-vs.-the-rest*, para la clasificación multi-clase. O*ne-vs.-all* utiliza clasificación binaria para cada una de las clases posibles. La clase que se predice con la mayor confianza se asigna a la instancia. Regresión logística soporta clasificación multiclase utilizando la estrategia *one-vs.-the-rest.*
Al igual que clasificación binaria, las matrices de confusión son útiles para visualizar los tipos de errores cometidos por la clasificación. Precisión, recall, y F1 measure se pueden calcular

para cada una de las clases, y la exactitud para todas las predicciones también pueden ser calculadas.

1.3.2 Orange[30].

Orange es una suite de aprendizaje automático y minería de texto para el análisis de datos a través de Python scripting y programación visual. Orange fue concebido a finales de 1990 y es uno de los más antiguos de este tipo de herramientas. Se centra en la simplicidad, interactividad a través de secuencias de comandos, y el diseño basado en componentes. (Demšar et al., 2013)

Se puede utilizar ya sea a través de secuencias de comandos de Python como un plug-in de Python, o por medio de programación visual. Su interfaz de programación visual Orange Canvas, ofrece una visión estructurada de las funcionalidades soportadas agrupados en nueve categorías: operaciones de datos, visualización, clasificación, regresión, evaluación, aprendizaje no supervisado, asociación, visualización utilizando Qt[31], e implementaciones prototipo. (Jović, Brkić, & Bogunović, 2014)

Orange contiene widgets de gran alcance para la visualización y la exploración de redes, enfocado en la interacción y flexibilidad.

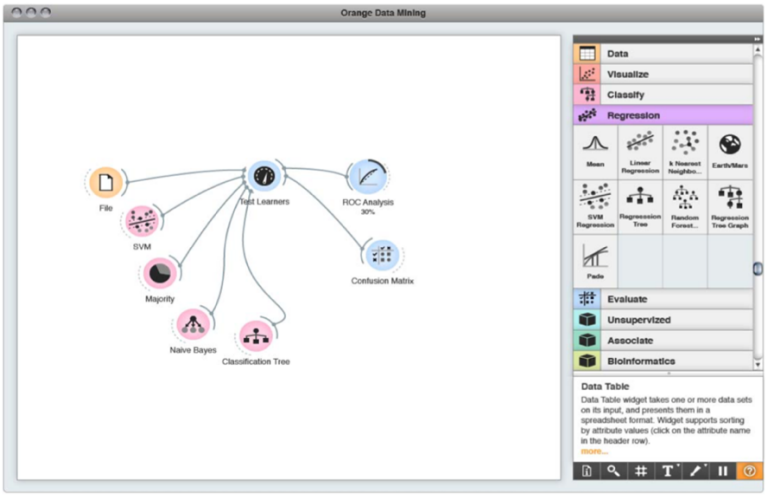

Figura 18. Interfaz grafica de Orange.

Fuente: (Demšar & Zupan, 2012)

Según (Demšar et al., 2013), Orange se diferencia de la mayoría de otras librerías de aprendizaje automático basado en Python por su madurez (más de 15 años de desarrollo activo y uso) ya que existe una gran comunidad de usuarios, y una extensa documentación.

[30] Orange: http://orange.biolab.si/
[31] Qt: http://www.qt.io/

Los autores en mención hacen un repaso a las características de la librería, algunas de ellas detalladas a continuación:

- **Gestión de datos y pre-procesamiento**: para la entrada y salida de datos, filtrado de datos y toma de muestras, imputación, manipulación de características y selección de características.
- **Clasificación**: con las implementaciones de diferentes algoritmos de aprendizaje automático supervisado.
- **Regresión**: incluye entre otros regression linear y lasso, partial least square regression, regression trees y forests.
- **Asociación**: para reglas de asociación y minería de conjuntos de elementos frecuentes.
- **Evaluación**: cross-validation y otros procedimientos para calificar la calidad de los métodos de predicción, y los procedimientos para la estimación de la fiabilidad.

Orange implementa varios algoritmos de clasificación. (Moriana Becerra, 2015), hace un repaso a varios algoritmos de clasificación que podrían resultar de útil aplicación, tanto a nivel alternativo como complementarios:

- ✓ Rule induction (rules)
- ✓ Support Vector Machines (svm)
- ✓ k-nearest neighbors (knn)
- ✓ Naive Bayes classifier (bayes)
- ✓ Classification trees (tree)
- ✓ Logistic regression (logreg)
- ✓ Neural Network Learner (neural)

1.3.2.1 Modelo de datos.

Orange almacena los datos en ***Orange.data.Storage classes***. El almacenamiento más comúnmente utilizado es ***Orange.data.Table***, que almacena todos los datos en arreglos bidimensionales Numpy. Cada fila de los datos representa una instancia de datos.

Instancias de datos individuales se representan como instancias de ***Orange.data.Instance***. Diferentes clases de almacenamiento pueden derivar subclases de instancia para representar las filas recuperadas en los datos de manera más eficiente y para permitir la modificación de los datos a través de la modificación de instancia de datos.

Los datos se divide en atributos (características, variables independientes), las variables de clase (clases, objetivos, resultados, variables dependientes) y atributos meta. Esta división se aplica a las descripciones de dominio, almacenes de datos que contienen matrices independientes para cada una de las tres partes de las instancias de datos.

Los atributos y las clases se representan con valores numéricos y se utilizan en el modelado. Atributos meta contienen datos adicionales que pueden ser de cualquier tipo.

Esta información fue recuperada de la documentación oficial de Orange. (Orange, 2016).

1.3.2.2 División splitter.

(Moriana Becerra, 2015), hace un repaso a varios criterios de división que Orange implementa, a los que se pueden acceder a través del parámetro *splitter*, algunos de ellos se detallan a continuación:

- ***Splitter_IgnoreUnknowns*:**

Ignora las observaciones para las que no puede ser determinada una rama individual.

- ***Splitter_UnknownsAsBranchSizes:***

Divide los casos con valor desconocido en función de la proporción de casos en cada rama.

- ***Splitter_UnknownsToAll:***

Divide los casos con un valor desconocido de la característica en todas las ramas posibles.

- ***Splitter_UnknownsToBranch:***

Construye una rama adicional para casos ambiguos. La descripción de la rama es 'unknown'

- ***Splitter_UnknownsAsSelector:***

Divide los casos con valor desconocido en función de la distribución propuesta por el selector de ramas (normalmente la proporción de casos en cada rama).

1.3.2.3 Regresión logística.

La regresión logística es un método de clasificación estadística que se ajuste a los datos a una función logística. Orange proporciona varias mejoras del método, tales como la selección de variables por pasos y manipulación de variables constantes y singularidades.

En el caso multiclase, el algoritmo de entrenamiento puede utilizar el esquema one-vs-rest o usar pérdida de entropía cruzada.

Orange implementa la regresión logística regularizada mediante la librería liblinear, newton-cg y lbfgs solucionadores. El solucionador liblinear soporta tanto regularización L1 y L2.

La clase de regresión logística de Orange es la siguiente:

class Orange.classification.LogisticRegressionLearner(penalty='l2', dual=False,tol=0.0001, C =1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None,preprocessors=None)

Parámetros:

- ***penalty: str, 'L1' o 'L2'***

Se utiliza para especificar la norma utilizada en la penalización.

- ***dual: bool***

Formulación dual o primitiva.

- ***C: float, optional (default = 1,0)***

Inverso de la fuerza de regularización; debe ser un float positivo. Al igual que en las máquinas de vectores soporte, los valores más pequeños especifican regularización más fuerte.

- ***fit_intercept: bool, default: True***

Especifica si una constante debe añadirse a la función de decisión.

- ***intercept_scaling: float, default: 1***

Útil sólo si el solucionador es liblinear.

- ***class_weight: dict o "balanced", optional***

Los pesos asociados a las clases.

- ***max_iter: int***

Útil sólo para solucionadores newton-cg, sag y lbfgs.

- ***random_state: int seed***

La semilla del generador de números aleatorios para usar cuando se barajan los datos.

- ***solver: {'newton-cg', 'lbfgs', 'liblinear', 'sag'}***

Algoritmo para utilizar en el problema de optimización.

- ***tol: float, opcional***

Tolerancia para detener criterios.

- ***multi_class: str, {'ovr', 'multinomial'}***

Opción multiclase puede ser 'ovr' o 'multinomial'.

- ***verbose: int***

Para los solucionadores liblinear y lbfgs establecidos detallado a cualquier número positivo para la verbosidad.

- ***warm_start: bool, optional***

Inútil para el solucionador liblinear.

- ***n_jobs: int, optional***

Número de núcleos de CPU utilizados durante el bucle de cross-validation.

Esta información fue recuperada de la documentación oficial de Orange. (Orange, 2016).

1.3.2.4 Regresión lineal.

La regresión lineal es un método de regresión estadística que intenta predecir un valor de una variable de respuesta continua (clase) basado en los valores de varios predictores. El modelo asume que la variable de respuesta es una combinación lineal de los predictores, la tarea de regresión lineal por lo tanto, es ajustar los coeficientes desconocidos.

La clase de regresión ridge es la siguiente:

class Orange.regression.linear. RidgeRegressionLearner (*alpha=1.0* ,*fit_intercept=True* , *normalize=False* , *copy_X=True* , *max_iter=None* , *tol=0.001* , *solver='auto'* ,*preprocessors=None*)

Parámetros:

- ***fit_intercept: boolean, optional***

True para calcular el punto de intersección para este modelo.

- ***normalize: boolean, optional, default False***

Si es True, los regresores X se normalizarán antes de regresión.

- ***copy_X: boolean, optional, default True***

Si es True, X se copiará; otra cosa, se puede sobrescribir.

- ***n_jobs: int, optional, default 1***

El número de puestos de trabajo que se utilizará para el cálculo. Si es -1 se utilizan todas las CPUs.

Esta información fue recuperada de la documentación oficial de Orange. (Orange, 2016)

1.3.2.5 Cross validation.

(Moriana Becerra, 2015), explica que Orange implementa la técnica de validación de modelos conocida como *cross validation*, la cual al aplicar cualquiera de los algoritmos disponibles se calculan los cuatro valores más representativos de la técnica:

- AP. Probabilidad media de ser asignado en la fuente real.
- CA. Porcentaje de coincidencias entre la fuente predicha y la real.
- Sensitivity. Proporción de positivos verdaderos que se identificaron correctamente como tal. Resulta útil en pruebas de clasificación binaria.
- Specificity. Proporción de negativos que se identificaron correctamente como tal. Resulta útil en pruebas de clasificación binaria.

1.3.3 PyML[32].

La información que aquí se presenta fue recuperada de la documentación oficial de PyML. (PyML, 2016)

PyML es un marco orientado a objetos interactivo para aprendizaje automático escrito en Python. PyML se centra en métodos-kernel de clasificación y regresión, incluyendo Máquinas de Vectores de Soporte. Proporciona herramientas para la selección de características, selección de modelos, sintaxis para combinar clasificadores y métodos para evaluar el desempeño del clasificador.

Algunas características de PyML son:

Clasificadores:

- ✓ Support vector machines, k-Nearest Neighbor, regresión ridge
- ✓ Métodos Mult-Clase (one-against-rest y one-against-one)
- ✓ Selección de características

[32] PyML: http://pyml.sourceforge.net/

- ✓ Selección del modelo
- ✓ Pre procesamiento y normalización
- ✓ Sintaxis para combinación de clasificadores
- ✓ Pruebas de clasificador (cross-validation, error rates, curvas ROC)

1.3.3.1 Contenedores de Datos.

Un *dataset* es una colección de patrones y sus etiquetas de clase, PyML tiene varios contenedores de *datasets* a continuación se detallan algunos de ellos:

- ***VectorDataSet***

Un contenedor de datos vectoriales

- ***SparseDataSet***

Un contenedor para datos vectoriales escasos

- ***KernelData***

Un contenedor que almacena una matriz kernel pre computarizada

- ***SequenceData***

Un contenedor para cadenas

- ***PairDataSet***

Un contenedor para los patrones que se componen de pares de objetos

- ***Aggregate***

Un contenedor que recoge una variedad de conjuntos de datos en un único contenedor:

PyML además ofrece varios métodos para construir una instancia de conjunto de datos:

1. La lectura de datos de un archivo es compatible con el formato utilizado por libsvm y SVMLight, además archivos delimitados también son soportados.
2. La construcción de una matriz de 2 dimensiones proporcionada como una matriz Numpy o una lista de Python.

1.3.3.1.1 La lectura de datos de un archivo.

Los datos se leen desde un archivo mediante una llamada al constructor de una clase de contenedor de datos con un argumento de nombre de archivo. PyML admite dos formatos de archivo para datos vectoriales:

- Formato delimitado (coma, tabulador o espacio delimitado).
- Formato disperso.

El formato de archivo disperso soportado por PyML es similar a la utilizada por libsvm (formato-libsvm es reconocido por PyML). Cada patrón se representa mediante una línea de la forma:

- ***[id,]label fid1:fval1 fid2:fval2 ...***

Parámetros:

- ***Id:*** es una identificación de patrón (opcional).
- ***label:*** etiqueta de clase asociado con el patrón
- los datos se proporciona como pares ***fid: fval*** donde ***fid*** es la identificación de características y ***fval*** es su valor.

1.3.3.2 Entrenamiento y prueba a un clasificador.

Todos los clasificadores en PyML ofrecen la misma interfaz:

- **constructor:** ofrece la construcción desde una copia, y la construcción "desde cero".
- **train(data):** entrenar el clasificador en el conjunto de datos dado
- **test(data)**
- **trainTest(data, trainingPatterns, testingPatters)**
- **stratifiedCV(data)**

Supongamos que se quiere construir un clasificador SVM se utiliza la siguiente línea de código:

- >>> s=SVM()

Si se quiere cambiar valor del parámetro C, se escribe simplemente:

- s.C = some_other_value

o dar el valor de c en el constructor

- s= svm.SVM(C = someValue)

El método de entrenar SVM es el siguiente:

- >>> s.train (data)

Por defecto, libsvm se utiliza en el entrenamiento. También existe algunas otras opciones:

- Para grandes conjuntos de datos, y si se necesita un SVM lineal, se puede utilizar liblinear, que puede ser más rápido que libsvm.

- Cuando el conjunto de datos es un contenedor no vectorial, es necesario utilizar el PyML optimizador nativo. Se elige de forma automática en este caso.

Para evaluar el rendimiento de un clasificador, se usa el método ***cv (cross validation):***

- >>> r = s.cv (data, 5)

Esto realiza 5 veces *cross-validation* y almacena los resultados en un objeto ***Results.***

1.3.3.2.1 Clasificación multiclase.

Clasificadores multiclase se encuentran en el módulo múltiple. PyML soporta clasificación de *one-against-one* y *one-against-the-rest.*

Para construir un clasificador one-against-the-rest que utiliza una SVM lineal como un clasificador de base se hace de la siguiente manera:

- >>> mc = multi.OneAgainsRest (SVM())

Para evaluar el rendimiento del clasificador:

- >>> R = mc.Cv (data)

Un clasificador *one-against-one* es proporcionado por la clase ***OneAgainstOne.***

1.3.3.2.2 Otros clasificadores.

Otros clasificadores adicionales son:

- clasificador k-vecino más cercano
- clasificador Regresión Ridge

1.3.3.3 Regresión SVM.

La lectura de los datos de un problema de regresión es diferente que para la clasificación: se interpreta las etiquetas como números en lugar de etiquetas de clase. Los formatos de archivo son los mismos, simplemente se reemplaza la etiqueta de clase por el valor numérico que desea predecir. Para leer los datos se usa:

- >>> data = SparseDataSet(fileName, numericLabels = True)

Para construir un objeto de vectores de soporte de regresión (SVR):

- >>> s = SVR()

Este objeto es compatible con la interfaz de clasificador, a excepción de *stratifiedCV*; la función de clasificar retorna el valor predicho, realizando la misma función que el método *decisionFunc*. El resultado de cualquiera de los métodos de prueba (cv, test, etc.) contiene atributos similares a los objetos *Results* utilizado para problemas de clasificación.

1.3.4 Mlpy[33].

(Albanese et al., 2012), mencionan que "es una librería de aprendizaje automático de código abierto de Python, que esta construido sobre NumPy y SciPy[34] y librerías científicas GNU". (Albanese et al., 2012), también mencionan que mlpy ofrece una amplia gama de métodos para problemas de aprendizaje supervisado y no supervisados y que está dirigida a la búsqueda de un compromiso razonable entre la modularidad, facilidad de mantenimiento, la reproducibilidad, facilidad de uso y eficiencia. Además dicen que mlpy está dirigido a alcanzar un buen compromiso entre código modularidad, facilidad de uso y eficiencia.

La librería consta de una serie de algoritmos en lo que tiene que ver con clasificación, regresión y reducción de dimensionalidad.

(Albanese et al., 2012), hacen un repaso en particular a los algoritmos de clasificación y regresión se detalla algunos de ellos a continuación:

En cuanto a clasificación:

- Linear Discriminant Analysis (LDA)
- Basic Perceptron
- Logistic Regression
- Elastic Net
- k-nearest neighbor (KNN)
- classification tree
- Maximum Likelihood.

En cuanto a regresión:

- Ordinary (Linear) Least Squares
- Linear and Kernel Ridge
- Partial Least Squares

[33] Mlpy: http://mlpy.sourceforge.net/
[34] SciPy: http://www.scipy.org/

- LARS
- Elastic Net
- Linear and Kernel SVM.

1.3.4.1 Regresión logística.

Mlpy implementa LibLinear, la cual es una clase simple para resolver clasificación lineal regularizado a gran escala, soporta:

- L2- regularized logistic regression
- L2-loss support vector classification
- L1-loss support vector classification
- L1-regularized L2-loss support vector classification
- Logistic regression.

Soluciona:

- l2r_lr: L2-regularized logistic regression (primal)
- l2r_l2loss_svc_dual: L2-regularized L2-loss support vector classification (dual)
- l2r_l2loss_svc: L2-regularized L2-loss support vector classification (primal)
- l2r_l1loss_svc_dual: L2-regularized L1-loss support vector classification (dual)
- mcsvm_cs: multi-class support vector classification by Crammer and Singer
- l1r_l2loss_svc: L1-regularized L2-loss support vector classification
- l1r_lr: L1-regularized logistic regression
- l2r_lr_dual: L2-regularized logistic regression (dual)

La clase es la siguiente:

class mlpy.LibLinear(solver_type='l2r_lr', C=1, eps=0.01, weight={})

Parámetros:

- ***solver_type [string] solver,***

puede ser uno de: 'l2r_lr', 'l2r_l2loss_svc_dual', 'l2r_l2loss_svc', 'l2r_l1loss_svc_dual', 'mcsvm_cs', 'l1r_l2loss_svc', 'l1r_lr', 'l2r_lr_dual'

- ***C [float]***

Costo de la violación de restricción

- ***eps [float]***

stopping criterion

- ***weight [dict]***

cambia la pena para algunas clases (si no se cambia el peso para una clase, esta se establece en 1).

Esta información fue recuperada de la documentación oficial de mlpy. (Mlpy, 2016).

1.3.4.2 Regresión Ridge.

La clase que implementa para regresión ridge mlpy es la siguiente:

class mlpy.Ridge(lmb=1.0)

Parámetros:

- **lmb [float (>= 0.0)]**

parámetro de regularización

- ***learn(x, y)***

Calcula los coeficientes de regresión donde:

- **x [2d array_like object]**

datos de entrenamiento (N, P)

- **y [1d array_like object (N)]**

valores objetivo

- ***pred(t)***

calcula la respuesta predicha donde:

- **t [1d or 2d array_like object ([M,] P)]**

datos de prueba

Retorna:

- **p [integer or 1d numpy darray]**

respuesta predicha

Esta información fue recuperada de la documentación oficial de mlpy. (Mlpy, 2016).

1.3.5 PyMVPA[35].

(Hanke, Halchenko, Sederberg, Olivetti, et al., 2009), mencionan que en PyMVPA , cada bloque de construcción sigue una sencilla y estandarizada interfaz. Esto permite utilizar varios tipos de clasificadores indistintamente, sin cambios adicionales en el código fuente, y hace que sea fácil para probar el rendimiento de los algoritmos desarrollados.

Además (Hanke, Halchenko, Haxby, & Pollmann, 2010), mencionan en su escrito que todas las unidades de procesamiento en PyMVPA , como clasificadores u otras medidas de conjunto de datos, están diseñados para tener interfaces compatibles que permitan ajustes modulares a un análisis de tuberías. Esta propiedad hace que sea muy fácil de comparar el desempeño de varios algoritmos en un conjunto de datos en particular.

PyMVPA es abierto a contribuciones, (Hanke, Halchenko, Sederberg, Olivetti, et al., 2009), hacen un repaso a este punto y mencionan algunas de las características en este campo:

- **Accesibilidad de código fuente y documentación:** todo el código fuente, junto con el historial completo de desarrollo está disponible al público a través de un sistema distribuido de control de versiones.

- **Documentación de código inplace:** Gran parte del código fuente están bien documentados usando un lenguaje de marcado ligero que es muy fácil de leer en formato de origen, además de ser adecuado para la conversión automática en la documentación de referencia HTML o PDF de referencia.

- **Directrices de Desarrollador:** un breve resumen define un conjunto de convenciones de codificación para facilitar código y documentación uniforme.

En la figura 19 se puede observar el diseño y flujo de trabajo de la librería PyMVPA

[35] PyMVPA: http://www.pymvpa.org/

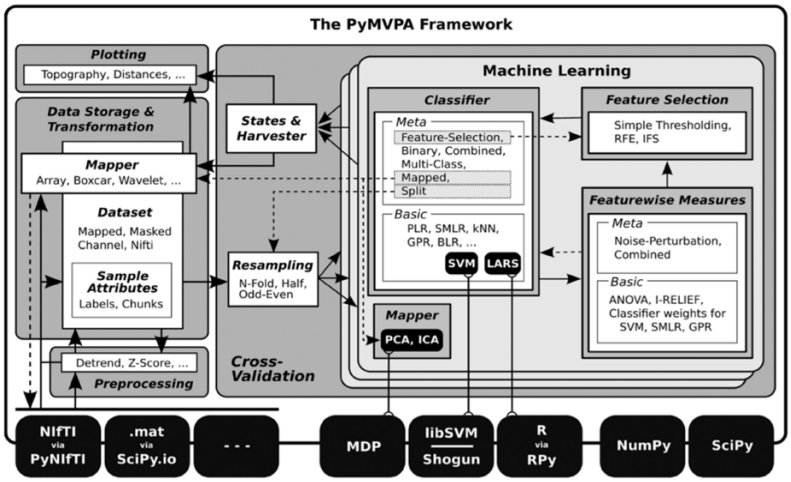

Figura 19. Diseño y Flujo de trabajo de PyMVPA.

Fuente: (Hanke, Halchenko, Sederberg, Olivetti, et al., 2009)

PyMVPA en lo que respecta a clasificadores incluyen entre otros implementaciones de:

- k-nearest- neighbor classifier
- Bayesian linear
- Gaussian process (GPR)
- Penalized logistic
- Sparse multinomial logistic regressions
- Support vector machine algorithms

(Hanke, Halchenko, Sederberg, José, et al., 2009), explican que los análisis basados en clasificadores típicamente consisten en algunos procedimientos básicos que son independientes del algoritmo de clasificación o proceso de decisión que se utiliza realmente, por ejemplo, *error calculation, cross-validation* de predicción de rendimiento, y selección de características.

1.3.5.1 Cargar un Dataset.

(Hanke, Halchenko, Sederberg, José, et al., 2009), mencionan que la representación del conjunto de datos en PyMVPA se basa en matrices NumPy, cualquier cosa que se puede convertir en una matriz también puede ser utilizado como una fuente de conjunto de datos para PyMVPA.

(Hanke, Halchenko, Sederberg, José, et al., 2009), también dicen que: “En PyMVPA un conjunto de datos se compone de tres partes: las muestras de datos, los atributos de la muestra y los atributos del conjunto de datos”.

1.3.5.2 Clasificador k-Nearest-Neighbour.

Se trata de un clasificador simple que basa su decisión en las distancias entre las muestras de la formación de conjuntos de datos y la muestra de prueba. Las distancias se calculan utilizando una función de distancia personalizable. Un cierto número (k) de los vecinos más cercanos se selecciona basándose en las distancias más pequeñas y las etiquetas de esta muestras vecinos se introducen en una función de votación para determinar las etiquetas de la muestra de prueba.

El entrenamiento de un clasificador kNN es extremadamente rápida , ya que no se realiza ninguna formación real como la formación de datos simplemente se almacena en el clasificador. Todos los cálculos se hacen durante la predicción clasificador.

La clase es la siguiente:

class mvpa2.clfs.knn.kNN(k=2, dfx=<function squared_euclidean_distance>, voting='weighted', **kwargs)

Parámetros:

- ***k : entero sin signo***

Número de vecinos más cercanos que se utilizará para la votación.

- ***dfx : functor***

Función para calcular las distancias entre muestras de entrenamiento y de prueba. Por defecto: Distancia euclídeana al cuadrado

- ***voting : str***

método votación utilizado para derivar predicciones de los vecinos más cercanos.

- ***enable_ca : None or list of str***

nombres de los atributos condicionales

- ***disable_ca : None or list of str***

Los nombres de los atributos condicionales que deben deshabilitarse

- ***auto_train : bool***

Bandera si el aprendiz será entrenado automáticamente a sí mismo en el conjunto de datos de entrada.

- ***force_train : bool***

Bandera si el aprendiz aplica el entrenamiento en el conjunto de datos de entrada a cada llamada.

- ***space : str, optional***

En general, este es un disparador que indica el nodo para computar y almacenar información sobre los datos de entrada que es "interesante" en el contexto del procesamiento correspondiente en el conjunto de datos de salida.

- ***pass_attr : str, list of str|tuple, optional***

Atributos adicionales para pasar a un conjunto de datos de salida.

- ***postproc : Node instance, optional***

Nodo para realizar el procesamiento posterior de los resultados.

- ***descr : str***

Descripción de la instancia

Esta información fue recuperada de la documentación oficial de PyMVPA. (PyMVPA, 2016)

1.3.5.3 Support Vector Machines Regression.

Múltiples librerías externas implementan Support Vector Machines para clasificación y regresiones, en PyMVPA están disponibles: LIBSVM y shogun. La librería LIBSVM es elegido por defecto, pero en cualquier caso, ambas librerías están disponibles a través de la importación de este módulo :

La clase para la implementación de SVM usando kernel lineal es:

class mvpa2.clfs.svm.LinearCSVMC(C=-1.0, **kwargs)

Parámetros:

- ***kernel :***

Kernel object. [Default: None]

- ***enable_ca : None or list of str***

Nombres de los atributos condicionales

- ***disable_ca : None or list of str***

Nombres de los atributos condicionales que deben deshabilitarse

- ***tube_epsilon :***

Epsilon en epsilon-SVM regression (SVR). [Default: 0.01]

- ***C :***

Parámetro de compromiso entre el ancho del margen y el número de vectores de soporte.

- ***weight : list(float), optional***

Pesos personalizados por etiqueta.

- ***probability :***

Bandera de señal de estimación de probabilidad

- ***epsilon :***

Tolerancia de los criterios de terminación.

- ***weight_label : list(int), optional***

Para ser utilizado en conjunción con el peso.

- ***shrinking :***

De cualquier de las dos reducción se lleve a cabo.

- **nu** :

Fracción de puntos de datos dentro del margen.

- ***auto_train : bool***

Bandera si el aprendiz será entrenado automáticamente a sí mismo en el conjunto de datos de entrada.

- ***force_train : bool***

Bandera si el aprendiz aplica el entrenamiento en el conjunto de datos de entrada a cada llamada.

- ***space : str, optional***

Nombre del 'processing space', en general, este es un disparador que indica el nodo para computar y almacenar información sobre los datos de entrada que es "interesante" en el contexto del procesamiento correspondiente en el conjunto de datos de salida.

- ***pass_attr : str, list of str|tuple, optional***

Atributos adicionales para pasar a un conjunto de datos de salida.

- ***postproc : Node instance, optional***

Nodo para realizar el procesamiento posterior de los resultados.

- ***descr : str***

Descripción de la instancia

Esta información fue recuperada de la documentación oficial de PyMVPA. (PyMVPA, 2016)

1.3.6 Mdp[36].

Herramientas modulares para procesamiento de datos por sus siglas en español, es un framework para el procesamiento de datos escrito en Python.

Los autores (Zito, Wilbert, Wiskott, & Berkes, 2008), mencionan que los cálculos se llevan a cabo de manera eficiente en términos de requisitos de velocidad y de memoria, además desde

[36] Mdp: http://mdp-toolkit.sourceforge.net/

la perspectiva del desarrollador, es un framework modular, que se puede ampliar fácilmente ya que la implementación de nuevos algoritmos es fácil e intuitiva.

La implementación de varios algoritmos generalizados ofrece un marco unificado para combinarlos y construir arquitecturas de procesamiento de datos más complejas, a través de esta interfaz común es posible la construcción de software de procesamiento de datos complejos de forma modular. (Wilbert et al., 2013)

(Zito et al., 2008), también hacen hincapié en esto y mencionan que las nuevas unidades implementadas se integran automáticamente con el resto de la librería y que MDP ha sido escrito en el contexto de la investigación teórica en la neurociencia, pero ha sido diseñado para ser útil en cualquier contexto en el que se utilizan algoritmos de procesamiento de datos entrenables.
Los autores en mención detallan algunas características importantes de la librería mostradas a continuación:

- Documentación completa: tutorial que cubre el uso básico y avanzado.
- API estable y diseñado para la incorporación directa
- Dependencias mínimas : Python + NumPy

1.3.6.1 Elementos básicos de MDP: Nodos y Flujos.

Un nodo es el componente básico de una aplicación MDP. Representa un elemento de procesamiento de datos, como por ejemplo un algoritmo de aprendizaje, un filtro de datos, o una etapa de visualización. Cada nodo se caracteriza por una dimensión de entrada, una dimensión de salida, y un dtype, que determina el tipo numérico de las estructuras internas y de la señal de salida. Por defecto, estos atributos se heredan de los datos de entrada. (Zito et al., 2008)
Los autores en mención resumen en su escrito algunos nodos disponibles de la librería:

- PCA (standard, NIPALS)
- ICA (FastICA, CuBICA, JADE, TDSEP)
- Locally Linear Embedding
- Slow Feature Analysis
- Factor Analysis
- Gaussian Classifers

Múltiples nodos se pueden combinar en secuencias de procesamiento de datos, que se denominan flujos.

Un flujo es una secuencia de nodos que están capacitados y ejecutadas entre sí para formar un algoritmo más complejo. Los datos de entrada se envía al primer nodo y se procesa sucesivamente por los nodos subsiguientes a lo largo de la secuencia. (Zito et al., 2008)

(Wilbert et al., 2013), mencionan que la base de algoritmos disponibles en MDP incluyen métodos de procesamiento de señal, diversos métodos de clasificación, y muchos otros. La mayoría de los algoritmos de procesamiento de datos se pueden dividir en dos fases: la primera fase de un conjunto de datos de entrenamiento, se utiliza para ajustar una serie de parámetros internos (la fase de formación); en la segunda fase el algoritmo hace uso de los parámetros aprendidos para procesar los datos de prueba (la fase de ejecución).

(Wilbert et al., 2013), en su escrito hacen un repaso a algunas características de la librería resumidas a continuación:

1.3.6.2 Flujo de datos bidireccional: BiMDP.

El paquete bimdp expande el procesamiento de flujo de alimentación hacia adelante con la capacidad de transferir diversos datos y ejecutar los nodos en orden arbitrario. Esto hace que sea posible utilizar el marco MDP para una clase más amplia de algoritmos, las características más importantes de BiMDP son:

- Los nodos pueden especificar otros nodos como objetivos donde continuará la ejecución.
- Además de la matriz de datos estándar, los nodos pueden transportar datos arbitrarios en un mensaje diccionario.
- Se proporciona una herramienta de inspección basada en HTML interactivo para la formación de flujo y ejecución.

1.3.6.3 Paralelización.

Algunos de los algoritmos implementados en MDP podría, en principio, ser paralelizados de una manera directa. Los cálculos para cada bloque son independientes el uno del otro y por lo tanto se pueden realizar en paralelo.

Paralelización es un tema cada vez más importante, ya que el progreso en la velocidad de un solo núcleo de la CPU se ha ralentizado en los últimos años. El paquete paralelo en MDP agrega funcionalidad para paralelizar la formación y ejecución de flujos de MDP, proporcionando así un aumento de velocidad conveniente que escala con el número de núcleos de CPU.
El paquete paralelo en MDP se divide en dos partes:

- La primera parte consta de los programadores. Un planificador toma "tareas" arbitrarias y las procesa en paralelo (por ejemplo, en múltiples procesos de Python).
- La segunda parte consiste en versiones paralelas de los nodos MDP regulares y flujos. Todos los nodos de MDP, naturalmente, apoyan la ejecución en paralelo, ya que las copias de un nodo se pueden hacer y ejecutar en paralelo.

1.3.6.4 Extensiones de nodo.

El mecanismo de extensión nodo aborda un problema que es bastante común en el diseño de software y está bien ilustrado por la situación en el paquete paralelo: con el fin de agregar la función de formación paralela.

A través de un mecanismo de extensión de nodo y programación orientada a aspectos, se añade métodos y atributos de clase al nodo clases en tiempo de ejecución, con lo que añade nuevas características de forma dinámica. De esta manera, las extensiones pueden activarse exactamente cuando se necesitan, lo que reduce el riesgo de interferencias con otras extensiones. Es posible utilizar múltiples extensiones al mismo tiempo, siempre y cuando no hay colisiones de nombres. Los usuarios pueden fácilmente implementar extensiones personalizadas, lo que añade nuevas funcionalidades a los nodos MDP sin modificar el código de la librería.

1.3.6.5 Desarrollo Futuro.

MDP es mantenido por un equipo básico de tres desarrolladores, pero está abierto a contribuciones de los usuarios. Los usuarios ya han contribuido algunos de los nodos.
MDP también podría actuar eficazmente como un contenedor para la gran cantidad de algoritmos de análisis de datos estadísticos ya disponibles en otras librerías y lenguajes. (Zito et al., 2008)

1.3.7 Statsmodels[37].

Statsmodels es un módulo de Python que permite explorar los datos, estimar modelos estadísticos, y realizar pruebas estadísticas, proporciona un complemento a SciPy para los cálculos estadísticos que incluyen estadística descriptiva y estimación de modelos estadísticos.
Su objetivo es teórica y a la vez aplica a estadística y econometría, así como los usuarios de Python y desarrolladores aplica en todas las disciplinas que utilizan modelos estadísticos. Los principales desarrolladores de statsmodels están capacitados como economistas con experiencia en econometría. Por lo tanto, gran parte del desarrollo se ha centrado en las aplicaciones econométricas. Sin embargo, el diseño de statsmodels sigue un patrón consistente para que sea fácil de usar y fácilmente extensible por los desarrolladores de cualquier disciplina. (Seabold & Perktold, 2010)

Statsmodels proporciona una amplia gama de modelos *cross-sectional*, así como algunos modelos de series de tiempo. Statsmodels utiliza un modelo de lenguaje descriptivo para formular el modelo cuando se trabaja con *DataFrames* de pandas.
Algunos módulos soportados por la librería se detallan a continuación:

- ✓ Linear regression models
- ✓ Generalized linear models
- ✓ Discrete choice models
- ✓ Robust linear models
- ✓ Many models and functions for time series analysis

(Seabold & Perktold, 2010), hacen referencia al desarrollo, diseño y una breve descripción de la estructura de la librería resumidas a continuación.

1.3.7.1 Desarrollo y Diseño.

Dada la importancia de la precisión numérica y la multitud de opciones de software para el análisis estadístico, el desarrollo de statsmodels sigue un proceso para ayudar a asegurar resultados precisos y transparentes. Este proceso se conoce como Test-Driven Development (TDD). En su forma más estricta, TDD significa que las pruebas se escriben antes de la funcionalidad de la que se supone que debe probar.

[37] Statsmodels: http://statsmodels.sourceforge.net/

La idea principal detrás del diseño es que un modelo es en sí mismo es un objeto que se utilizará para la reducción de datos.

1.3.7.2 Módulos.

Se tiene como base cinco módulos de código principal que contiene los modelos estadísticos. Estos son:

- least squares regression models
- glm (generalized linear models)
- rlm (robust linear models)
- discretemod (discrete choice models)
- contrast (contrast analysis)

Además de los modelos y los resultados post-relacionados de estimación y pruebas, statsmodels incluye una serie de clases y funciones de conveniencia para ayudar con las tareas relacionadas con el análisis estadístico.

(McKinney, Perktold, & Seabold, 2011), hacen un repaso a algunos de los módulos nombrados anteriormente, además de algunos adicionales. A continuación se resume brevemente lo antes mencionado.

1.3.7.2.1 Ordinary Least Squares (OLS).

El modelo lineal simple supone que observamos una variable endógena y un conjunto de regresores o variables explicativas x, donde y y x son enlazados a través de una simple relación lineal más un término de ruido o error

$$y_t = x_t\beta + \varepsilon_t$$

En el caso más simple, los errores son de forma independiente e idénticamente distribuidos. Insesgamiento de OLS requiere que los regresores y los errores no estén correlacionados. Si los errores se distribuyen normalmente y los regresores no son aleatorios, entonces los resultados OLS o maximum likelihood estimator de β también se distribuyen normalmente en muestras pequeñas. Obtenemos el mismo resultado, si tenemos en cuenta las distribuciones como condicional de x_t cuando son variables aleatorias exógenas. Hasta el momento esto es independiente de índice de tiempo t o cualquier otro índice de observaciones.

1.3.7.2.2 Modelo lineal con error de autocorrelación (GLSAR).

Este modelo supone que las variables explicativas, regresores, no están correlacionadas con el término de error . Sin embargo, el término de error es un proceso:

$$E(x_t, \varepsilon_t) = 0$$

$$\varepsilon_t = a_1\varepsilon_{t-1} + a_2\varepsilon_{t-1} + \cdots + a_k\varepsilon_{t-k}$$

1.3.7.2.3 Linear Model with lagged dependent variables (OLS, AR, VAR)

Este grupo de modelos asumen que las ultimas variables dependientes, y_{t-i}, se incluyen entre las variables independientes, sino que el término de error no están correlacionados en serie

$$E(\varepsilon_t, \varepsilon_s) = 0, para\ t \neq s$$

$$y_t = a_1 y_{t-1} + a_2 y_{t-1} + \cdots + a_k y_{t-k} + x_t\beta + \varepsilon_t$$

Los procesos dinámicos como los procesos autorregresivos dependen de las observaciones en el pasado.

La forma más sencilla es tratar a la primera observación como fijos, y analizar la muestra a partir de la observación de orden k. Esto lleva a los mínimos cuadrados condicionales o la estimación de máxima verosimilitud condicional.

CAPÍTULO II

PREPARACIÓN DE DATOS

2.1 Preparación de Datos.

La información utilizada tanto para el enfoque de clasificación y regresión se la obtuvo del INEC[38], y corresponde a estadísticas de matrimonios y divorcios en el Ecuador bajo el censo realizado en el año 2014.

Para el enfoque de clasificación la información fue recuperada en dos archivos formato SPSS[39] para posteriormente combinar estos dos archivos en uno solo de formato csv (comma separated values). En tanto que para el enfoque de regresión se utilizo solamente el archivo SPSS correspondiente a divorcios para luego así mismo transformarlo a formato csv. Revisar Anexo 2.

Con el fin de no entrenar y probar el modelo sobre un mismo conjunto de datos se ha particionado los datos en dos conjuntos uno para entrenamiento y otro para pruebas y se ha utilizado el mismo conjunto en cada una de las librerías, con el fin de obtener condiciones mas equilibradas y así poder contrastar los resultados de mejor manera.

Parte del código correspondiente a la preparación de datos se encuentra en el Anexo 3, para el problema de clasificación y en el Anexo 4, para el problema de regresión.

2.1.1 Clasificación.

Para el problema de clasificación se intenta resolver si una pareja se esta casando o divorciando. Para esto se ha divido los datos en 80% (54930 instancias) para el entrenamiento y 20% (13733 instancias) para pruebas. En la tabla 2, se puede apreciar de mejor manera como esta realizada esta división, además de cómo están distribuidos los matrimonios y divorcios en las diferentes particiones.

Tabla 2: División y distribución del archivo correspondiente a clasificación

Partición	**Matrimonio**	**Divorcio**	**Total**
entrenamiento	41354	13576	54930
prueba	10317	3416	13733
completa	51671	16992	68663

Elaboración. El Autor.

[38] INEC: http://www.ecuadorencifras.gob.ec/

[39] SPSS: software que se utiliza mayormente para cálculos estadísticos ofrece una gran cantidad de formatos incluyendo los propios.

La figura 20 muestra como esta distribuida la variable dependiente en todo el conjunto de datos.

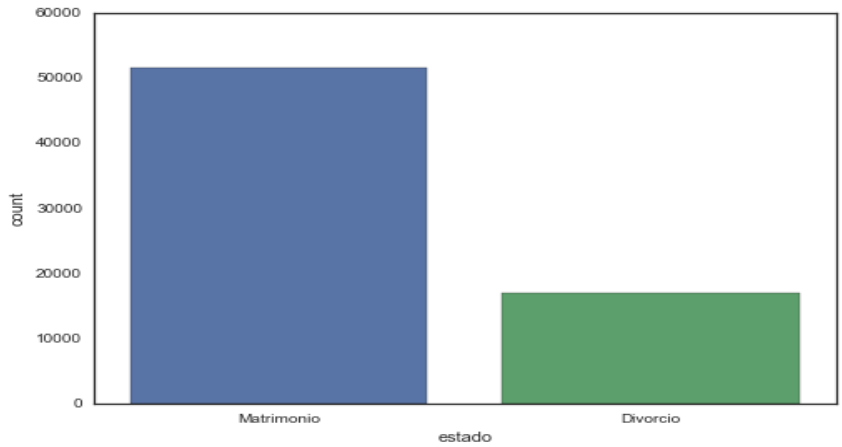

Figura 20. Distribución de Matrimonios y Divorcios en la partición completa

Elaboración. El Autor.

Las variables del archivo de entrenamiento son las mismas que las del archivo prueba y cuenta con 10 variables independientes, (8 discretas y 2 continuas), mas la variable dependiente, a continuación en la tabla 3 se detallan las mismas:

Tabla 3: Descripción de variables del archivo de entrenamiento y prueba en el experimento de clasificación

VARIABLE	DESCRIPCION
prov_insc	Provincia del registro del formulario
mcap_bie	Si el matrimonio es con capitulación de bienes o no
edad_hom	Edad del hombre
p_etnica_hom	Auto identificación étnica del hombre
niv_insth	Nivel de instrucción del hombre
area_hom	Área (urbana o rural) donde reside habitualmente el hombre
edad_muj	Edad de la mujer
p_etnica_muj	Auto identificación étnica de la mujer
niv_instm	Nivel de instrucción de la mujer
area_muj	Área (urbana o rural) donde reside habitualmente de la mujer
estado	Es la clase a predecir cuenta con dos valores [M,D] M=matrimonio, D=divorcio

Elaboración. El Autor.

Con el fin de involucrar a sólo las variables útiles en la formación y dejar de lado los datos redundantes se han seleccionado las variables más relacionadas con el resultado de destino,

para ello se ha utilizado el calculo de chi2 del enfoque univariante correspondiente a la librería Scikit-Learn.

La tabla 4 muestra el resultado de chi2 de las variables seleccionadas.

Tabla 4: Resultado F-score mediante chi2

Variable	**F-score**
prov_insc	165.79
mcap_bie	101.48
edad_hom	45147.22
niv_insth	1561.13
p_etnica_hom	137.22
area_hom	69.95
edad_muj	50032.05
niv_instm	1499.57
p_etnica_muj	128.94
area_muj	58.46

Elaboración. El Autor.

2.1.1.1 Transformación y análisis de los datos.

Para la limpieza y transformación de los datos se utilizo la herramienta OpenRefine, en la cual se elimino valores atípicos y se resolvió incoherencias, además con el uso de facetas se transformaron valores de tipo texto a valores numéricos. También cabe mencionar que ninguna variable conto con valores perdidos por lo que no hubo necesidad de llenar ningún valor en las mismas.

Para entender mejor la distribución de los datos tanto en el enfoque de clasificación y regresión se realizaron gráficos descriptivos para lo cual como se menciono anteriormente se utilizo la librería seaborn. A continuación se presenta el resultado de lo antes mencionado de todas las variables independientes.

2.1.1.1.1 Variable prov_insc.

La variable es de tipo discreta, se transformaron los datos de tipo texto a numéricos, de acuerdo a los criterios mostrados en la tabla 5.

Tabla 5: Transformación de datos de la variable prov_insc

Región	Valores originales de la variable	Nuevos valores de la variable
LITORAL	ESMERALDAS	1
	MANABI	
	LOS RIOS	
	GUAYAS	
	SANTA ELENA	
	EL ORO	
INTERANDINA	CARCHI	2
	IMBABURA	
	PICHINCHA	
	SANTO DOMINGO	
	COTOPAXI	
	TUNGURAHUA	
	CHIMBORAZO	
	BOLIVAR	
	CAÑAR	
	AZUAY	
	LOJA	
AMAZONICA	SUCUMBIOS	3
	ORELLANA	
	NAPO	
	PASTAZA	
	MORONA SANTIAGO	
	ZAMORA CHINCHIPE	
INSULAR	GALAPAGOS	4

Elaboración. El Autor.

La Figura 21, muestra que la región 2 que corresponde a la región interandina como se ha mencionado anteriormente en la tabla 3, es la que cuenta con mas datos tanto para matrimonios como para divorcios.

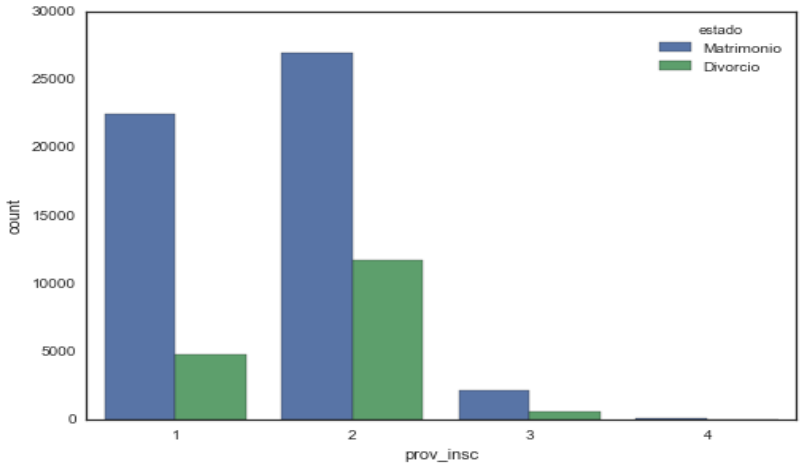

Figura 21. Distribución de la variable prov_insc.

Elaboración. El Autor.

2.1.1.1.2 Variable mcap_bie.

La variable es de tipo discreta, se transformaron los datos de tipo texto a numéricos, de acuerdo a los criterios mostrados en la tabla 5.

Tabla 6: Transformación de datos de la variable mcap_bie

Valores originales de la variable	Nuevos valores de la variable
Si	1
No	0

La Figura 22, muestra que existe una amplia diferencia entre los dos únicos valores de la variable mcap_bie.

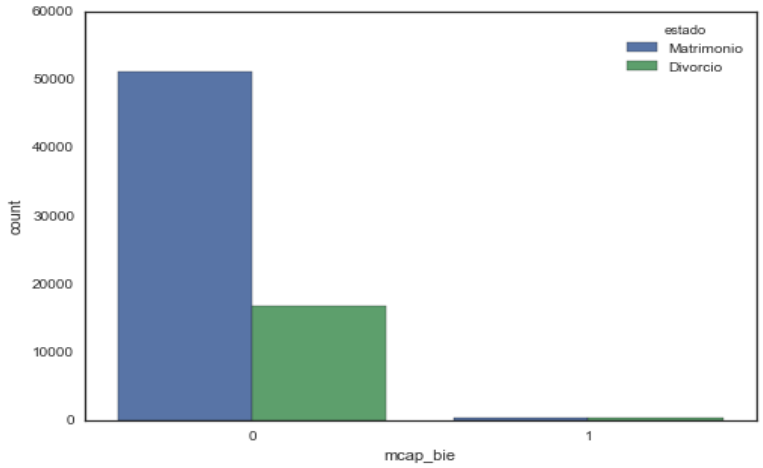

Figura 22. Distribución de la variable mcap_bie

Elaboración. El Autor.

2.1.1.1.3 Variable edad_hom y edad_muj.

Las variables son de tipo continuas, para este caso no hubo ninguna transformación, es decir se dejaron los valores originales, como se muestra en la tabla 3 estos valores corresponden a la edad del hombre y la mujer.

La figura 23 muestra que existe una notoria diferencia de la edad mas frecuente de matrimonio y divorcio tanto para el hombre como para la mujer. Por ejemplo se puede apreciar que en el hombre a la edad aproximada de 25 años, existe un alto numero de matrimonios y a la edad aproximada de 35 años, existe un alto numero de divorcios.

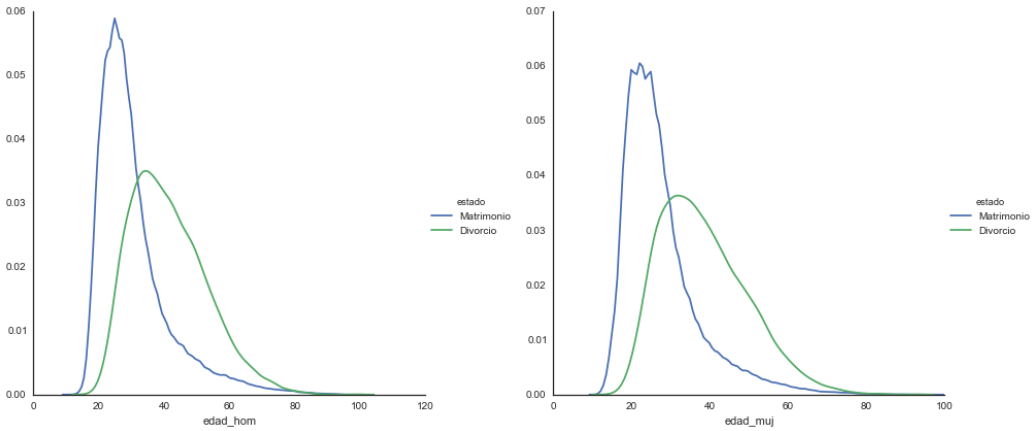

Figura 23. Distribución de densidad de las variables edad_hom y edad_muj

Elaboración. El Autor.

La Figura 24, muestra el diagrama de dispersión de las variables edad_hom y edad_muj divididas por matrimonio y divorcio. Se puede observar que las edades mas bajas de los datos corresponden a matrimonios.

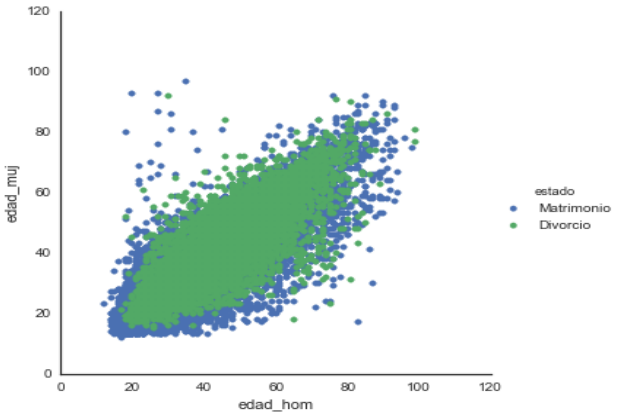

Figura 24. Diagrama de dispersión de las variables edad_muj y edad_hom.

Elaboración. El Autor.

2.1.1.1.4 Variable p_etnica_hom y p_etnica_muj.

Las variables son de tipo discretas, se transformaron los datos de tipo texto a numéricos, de acuerdo a los criterios mostrados en la Tabla 7.

Tabla 7: Transformación de datos de las variables p_etnica_hom y p_etnica_muj

Valores originales de la variable	**Nuevos valores de la variable**
Indígena	1
Afro-Ecuatoriano/ Afrodescendiente	2
Negro	3
Mulato	4
Montubio	5
Mestizo	6
Blanco	7

Elaboración. El Autor.

En la figura 25 se puede observar la distribución de las variables p_etnica_hom y p_etnica_muj divididas por matrimonio y divorcio, no se puede observar un patrón destacable en las mismas.

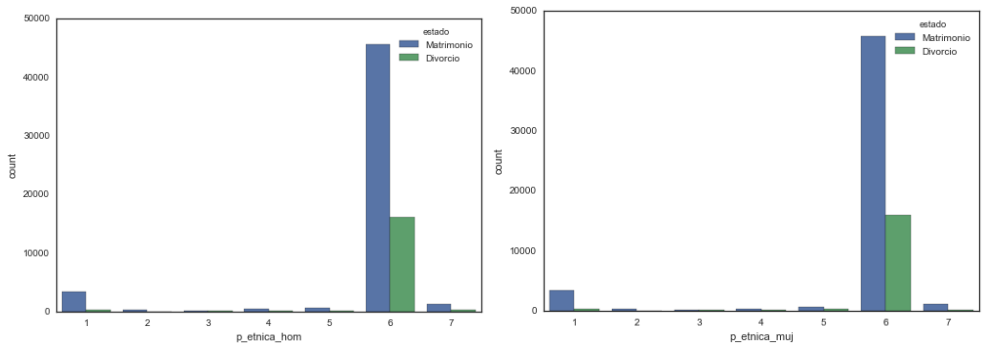

Figura 25. Distribución de las variables p_etnica_hom y p_etnica_muj.

Elaboración. El Autor.

2.1.1.1.5 Variable niv_insth y niv_instm.

Las variables son de tipo discretas, se transformaron los datos de tipo texto a numéricos, de acuerdo a los criterios mostrados en la tabla 8.

Tabla 8: Transformación de datos de la variables niv_insth y niv_instm

Valores originales de la variable	Nuevos valores de la variable
Ninguno	0
Centro de Alfabetización	1
Primaria	2
Secundaria	3
Educación Básica	4
Educación Media/Bachillerato	5
Ciclo Post-Bachillerato	6
Superior	7
Postgrado	8

Elaboración. El Autor.

La figura 26 muestra como están distribuidos los valores, se ha utilizado diagrama de barras y de frecuencias para representar de mejor manera estos datos.

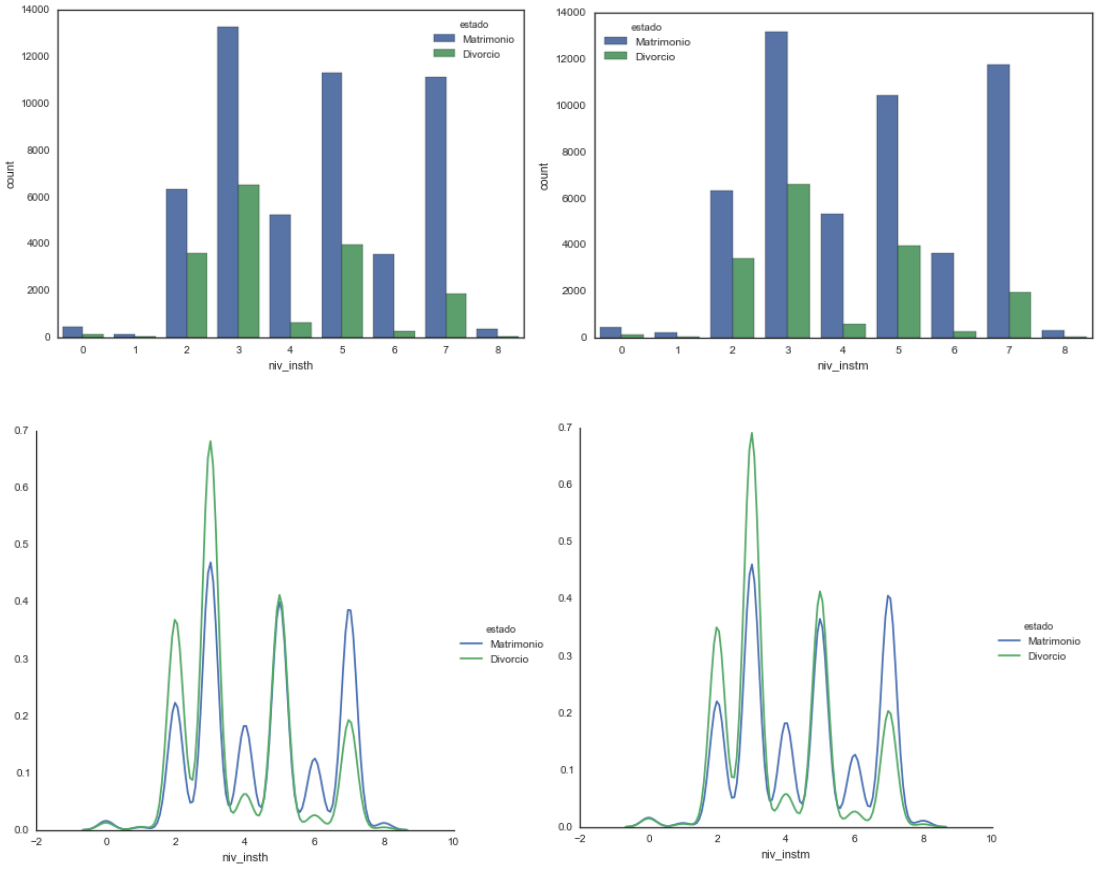

Figura 26. Distribución de las variables niv_insth y niv_instm.

Elaboración. El Autor.

2.1.1.1.6 Variable area_hom y area_muj.

Las variables son de tipo discretas, se transformaron los datos de tipo texto a numéricos, de acuerdo a los criterios mostrados en la tabla 9.

Tabla 9: Transformación de datos de las variables area_hom y area_muj

Valores originales de la variable	Nuevos valores de la variable
Rural	0
Urbana	1

Elaboración. El Autor.

No se observa ningún patrón particular destacable en los datos, respecto al área de residencia tanto del hombre como de la mujer como se puede observar en la figura 27.

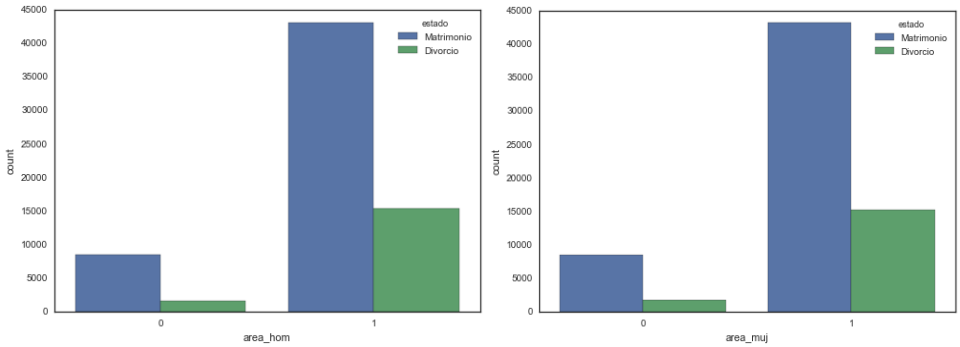

Figura 27. Distribución de las variables area_hom y area_muj.

Elaboración. El Autor.

2.1.1.2 Estadísticas de los datos.

Para realizar los análisis que en este apartado se presentan, se utilizo Python y la librería Orange.

Para tener una idea mas clara del comportamiento de los datos de entrada se ha realizado una tabla estadística con los valores mínimos, máximos, media, desviación estándar y la correlación entre clases. La tabla 10 muestra en resumen lo antes mencionado.

Tabla 10: Resumen estadístico de las variables independientes.

	Min	Max	Mean	SD	Class Correlation
prov_insc	1,00	4,00	1,65	0,56	-0,11
mcap_bie	0,00	1,00	0,01	0,10	-0,04
edad_hom	12,00	99,00	33,34	12,07	-0,39
niv_insth	0,00	8,00	4,30	1,78	0,18
p_etnica_hom	1,00	7,00	5,70	1,19	-0,09
area_hom	0,00	1,00	0,85	0,35	-0,08
edad_muj	12,00	97,00	30,33	11,15	-0,42
niv_instm	0,00	8,00	4,33	1,80	0,17
p_etnica_muj	1,00	7,00	5,69	1,19	-0,09
area_muj	0,00	1,00	0,85	0,35	-0,07

Elaboración. El Autor.

Un valor destacable es el puntaje de correlación de las variables niv_insth y niv_instm que aunque no es alto es un valor positivo.

2.1.1.3 Ranking de diagramas de dispersión.

Mediante este ranking se puede observar de una mejor manera el par de variables que tienen un mejor score de discriminación. La tabla 11 muestra las 10 primeras posiciones de este ranking.

Tabla 11: Primeras 10 posiciones de máximos scores de diagramas de dispersión.

Posición	Score	Variable 1	Variable 2
1	0.757	niv_instm	edad_muj
2	0.749	edad_hom	p_etnica_muj
3	0.749	edad_muj	p_etnica_muj
4	0.748	prov_insc	p_etnica_muj
5	0.748	edad_hom	p_etnica_hom
6	0.748	prov_insc	p_etnica_hom
7	0.748	niv_insth	edad_muj
8	0.748	niv_insth	p_etnica_muj
9	0.747	niv_insth	p_etnica_hom
10	0.747	edad_muj	p_etnica_hom

Elaboración. El Autor.

En la figura 28 se puede observar las primeras 3 posiciones enumeradas en la Tabla 10. Para este grafica se ha utilizado un diagrama de dispersión que mediante un algoritmo evita puntos de solapamiento en el eje categórico.

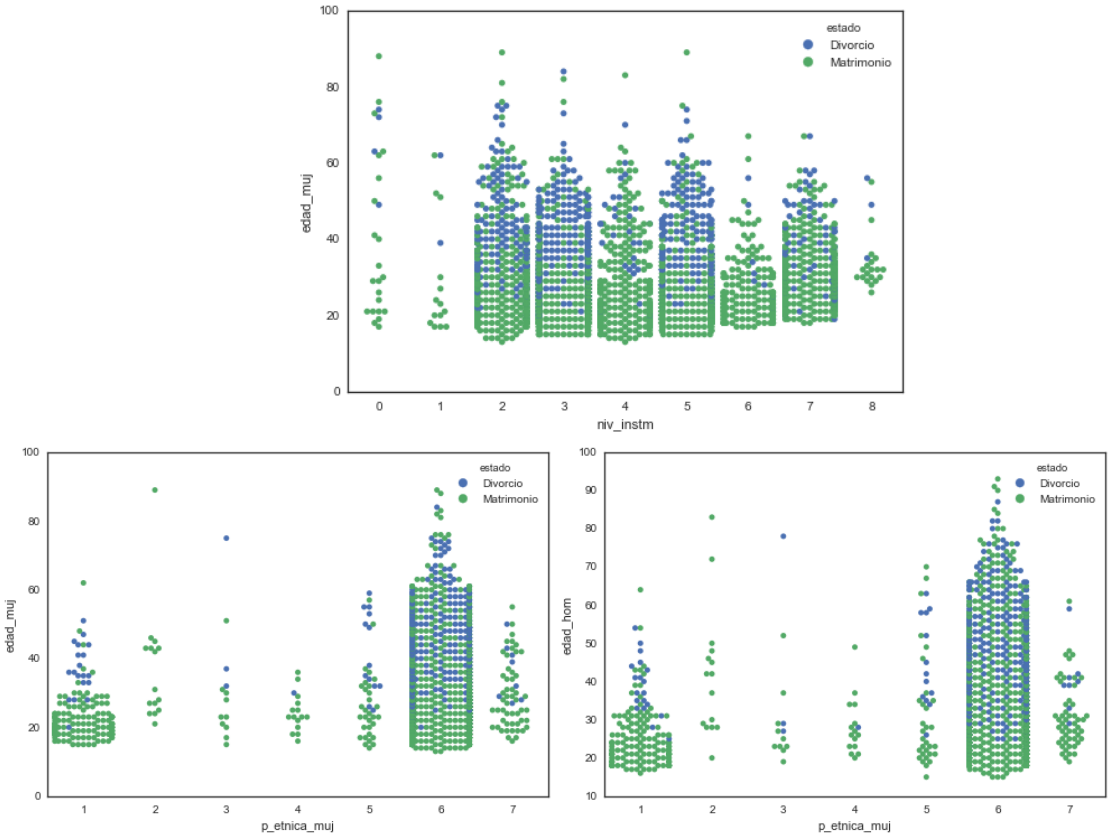

Figura 28. Diagrama de dispersión de las 3 primeras posiciones mostradas en la Tabla 10.

Elaboración. El Autor.

Para comprender este grafico se toma el siguiente ejemplo; para el diagrama de dispersión de las variables edad_muj y niv_instm se observa que existe un patrón destacable cuando el nivel de instrucción de la mujer es 6 (Ciclo Post-Bachillerato) existe menos probabilidades de divorcio.

2.1.2 Regresión.

Para el problema de regresión se intenta predecir el tiempo de duración en años de un matrimonio. Para esto como se ha dicho anteriormente se ha utilizado únicamente el archivo correspondiente a divorcios, la división de los datos es de 80% (14546 instancias) para el entrenamiento y 20% (3637 instancias) para pruebas. La tabla 12 muestra como esta realizada esta división, además de cómo esta distribuida la duración del matrimonio en años en intervalos específicos de tiempo.

Tabla 12: División y distribución del archivo correspondiente a regresión.

Partición	menor a 5	de 5 a 25	de 26 a 50	mayor a 50	Total
entrenamiento	2112	10031	2349	54	14546
prueba	553	2531	537	15	3636
completo	2665	12562	2887	68	18182

Elaboración. El Autor.

Las variables del archivo de entrenamiento son las mismas que las del archivo prueba y cuenta con 9 variables independientes, (5 discretas y 4 continuas), mas la variable dependiente, a continuación en la tabla 13 se detallan las mismas:

Tabla 13: Descripción de variables del archivo de entrenamiento y prueba en el experimento de regresión

VARIABLE	DESCRIPCION
cau_div	Causa del divorcio.
nac_hom	Nacionalidad del hombre.
edad_hom	Edad del hombre.
hijos_hom	Numero de hijos del hombre.
niv_insth	Nivel de instrucción del hombre.
nac_muj	Nacionalidad de la mujer.
edad_muj	Edad de la mujer.
hijos_muj	Numero de hijos de la mujer.
niv_instm	Nivel de instrucción de la mujer.
dur_mat	Es la variable a predecir que corresponde a la duración del matrimonio en años.

Elaboración. El Autor.

Para la selección de las variables mostradas en a tabla 12 previamente se ha utilizado el enfoque univariante; y mas específicamente f_regression que se desempeña para objetivos numéricos y con base de regresión lineal.

La tabla 14 muestra el resultado de las variables ya seleccionadas.

Tabla 14: Resultado F-score mediante f_regression

Variable	F-Score
cau_div	529.25
nac_hom	143.82
edad_hom	23562.00
hijos_hom	1.06
niv_insth	522.70
nac_muj	70.45
edad_muj	29253.98
hijos_muj	181.65
niv_instm	730.06

Elaboración. El Autor.

2.1.2.1 Transformación y análisis de los datos.

Para la limpieza y transformación de los datos, se eliminaron valores atípicos, además con el uso de facetas se transformaron valores de tipo texto a valores numéricos. También cabe mencionar que al igual que el enfoque de clasificación ninguna variable conto con valores perdidos.

2.1.2.1.1 Variable cau_div.

La variable es de tipo discreta, se transformaron los datos de tipo texto a numéricos, de acuerdo a los criterios mostrados en la Tabla 15.

Tabla 15: Transformación de datos de las variables cau_div

Valores originales de la variable	Nuevos valores de la variable
Mutuo acuerdo	1
Injurias graves o actitud hostil	2
Amenazas graves de un conyugue contra la vida del otro	3
Tentativa de uno de los conyugues contra la vida del otro	4
Concepción o alumbramiento ilegitimo	5
Los actos ejecutados por uno de los conyugues con el fin de comprometer al otro	6
El hecho de adolecer uno de los conyugues de enfermedad grave	7
El hecho de que uno de los conyugue sea ebrio consuetudinario	8
La condena ejecutoriada a reclusión mayor	9

El abandono voluntario e injustificado del otro conyugue por mas de un año ininterrumpidamente	10

Elaboración. El Autor.

La Figura 29, muestra como esta distribuidos los nuevos valores de la variable cau_div según los criterios mostrados en la tabla 15.

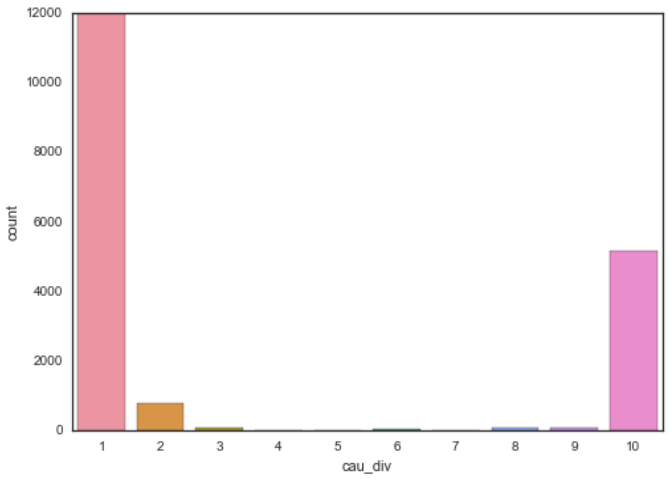

Figura 29. Distribución de la variable cau_div.

Elaboración. El Autor.

2.1.2.1.2 Variable nac_hom y nac_muj.

La variable es de tipo discreta, se transformaron los datos de tipo texto a numéricos, de acuerdo a los criterios mostrados en la Tabla 16.

Tabla 16: Transformación de datos de las variables nac_hom y nac_muj

Valores originales de la variable	**Nuevos valores de la variable**
Ecuatoriana	1
Extranjera	2

Elaboración. El Autor.

La Figura 30, muestra como esta distribuidos los nuevos valores de las variables nac_hom y nac_muj según los criterios mostrados en la tabla 16.

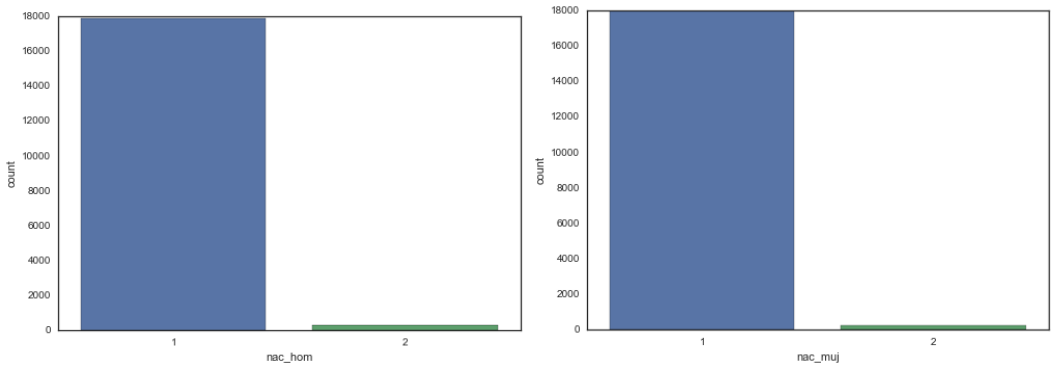

Figura 30. Distribución de las variables nac_hom y nac_muj.

Elaboración. El Autor.

2.1.2.1.3 Variable edad_hom y edad_muj.

Las variables son de tipo continuas, para este caso se dejaron los valores originales es decir no hubo ninguna transformación en sus datos.

La figura 31 muestra el diagrama de dispersión de las dos variables se ha utilizado junto con los histogramas en la misma figura para representar de mejor manera estos datos.

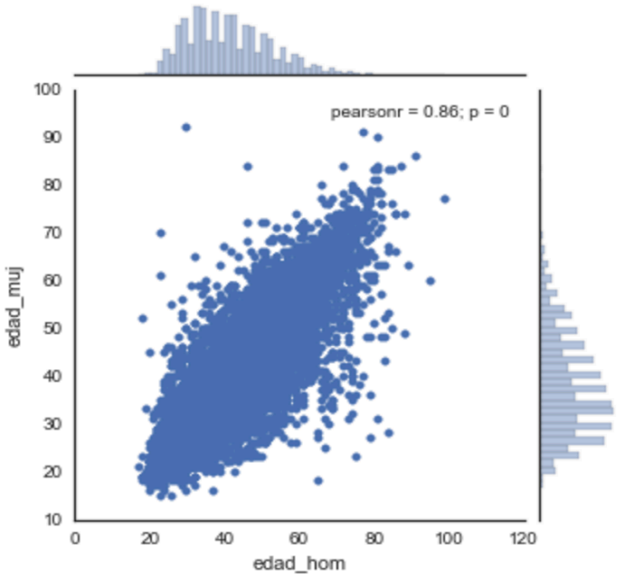

Figura 31. Diagrama de dispersión de las variables edad_muj y edad_hom junto con sus histogramas.

Elaboración. El Autor.

2.1.2.1.4 Variable hijos_hom e hijos_muj.

Las variables son de tipo continuas, para este caso se dejaron los valores originales es decir no hubo ninguna transformación en sus datos.

En la figura 32 se puede observar diferencia entre los valores de las dos variables, donde por ejemplo la cantidad de cero hijos del hombre prácticamente duplica a los de la mujer, que cuenta con mas distribución entre los distintos valores.

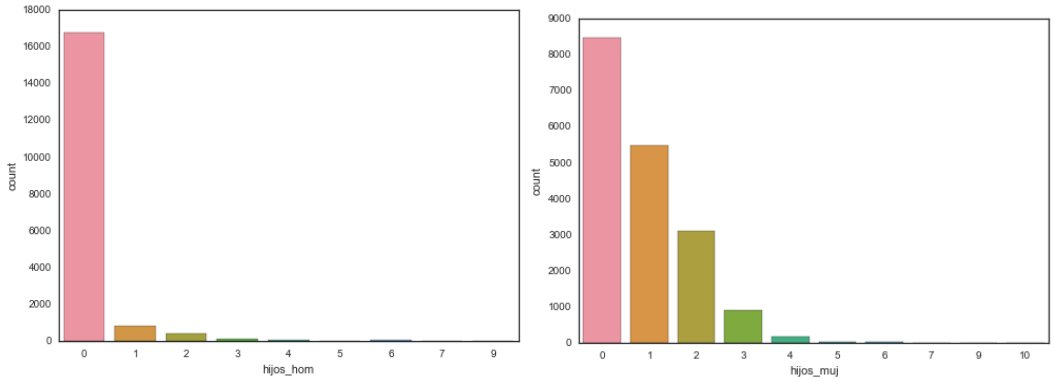

Figura 32. Distribución de las variables hijos_hom e hijos_muj.

Elaboración. El Autor.

2.1.2.1.5 Variable niv_insth y niv_instm.

Las variables son de tipo discretas, se transformaron los datos de tipo texto a numéricos, de acuerdo a los criterios mostrados anteriormente en la tabla 6.

La figura 33 muestra como están distribuidos los valores de las dos variables antes mencionadas.

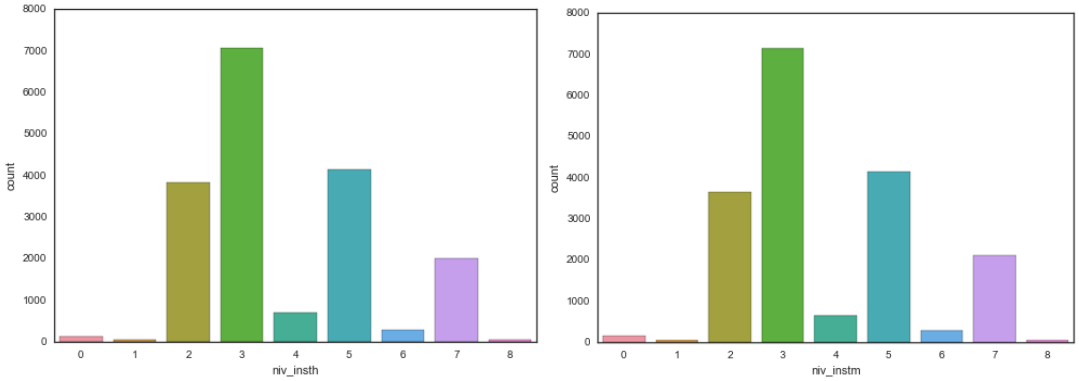

Figura 33. Distribución de las variables niv_insth y niv_instm

Elaboración. El Autor.

2.1.2.2 Estadísticas generales de los datos.

La tabla 17 muestra un resumen general estadístico del archivo completo. Un valor representativo que se muestra es el alto puntaje de correlación de los atributos edad_hom y edad_muj, además de la variable cau_div que aunque no tiene un puntaje alto es un valor positivo que representa ganancia.

Tabla 17: Resumen estadístico de las variables independientes

	Min	Max	Mean	SD	Class Correlation
cau_div	1,00	10,00	3,68	4,05	0,17
nac_hom	1,00	2,00	1,02	0,13	-0,09
edad_hom	17,00	99,00	41,48	11,51	0,75
hijos_hom	0,00	9,00	0,13	0,52	-0,01
niv_insth	0,00	8,00	3,75	1,62	-0,17
nac_muj	1,00	2,00	1,01	0,11	-0,06
edad_muj	15,00	92,00	38,48	11,05	0,79
hijos_muj	0,00	10,00	0,85	0,99	-0,10
niv_instm	0,00	8,00	3,78	1,64	-0,20

Elaboración. El Autor.

CAPÍTULO III

IMPLEMENTACIÓN

3.1 Implementación.

En este capitulo se presenta el proceso de implementación de aprendizaje automático en el ámbito de clasificadores y regresiones al conjunto de datos previstos en el capitulo 2. Se ha considerado utilizar librerías que cumplan con similares características en cuanto a la disponibilidad de algoritmos de aprendizaje supervisado, en el caso de clasificación para predecir la variable “estado” y en el caso de predicción para predecir la variable “dur_mat”.

3.1.1 Clasificación.

El problema a resolver en este enfoque se trata de clasificación binaria donde se intenta etiquetar si una pareja se esta casando o divorciando, con valores de entrada discretos y continuos. Este conjunto de datos corresponde a matrimonios y divorcios detallado en la sección 2.1.1

3.1.1.1 Selección de librerías.

Para la selección de las librerías se ha tomado en cuenta la precisión, y la disponibilidad de algoritmos de clasificación en las mismas. Las que se han considerado mas apropiadas son:

- ✓ Scikit-Learn 0.16.1
- ✓ Orange 2.7.8
- ✓ Mlpy 3.5.0
- ✓ PyMVPA 2.4.2
- ✓ Mdp 3.5

3.1.1.2 Selección de algoritmos.

Para la selección de los algoritmos, se ha tomado en cuenta que la gran mayoría de las librerías cuente con los mismos, a continuación se detallan los mismos:

- ✓ Logistic Regression
- ✓ Naïve Bayes
- ✓ K-nearest neighbors
- ✓ Support Vector Machine (kernel Linear)
- ✓ Support Vector Machine (kernel RBF)
- ✓ Classification Tree

En la tabla 18 se puede observar la disponibilidad de algunos algoritmos bajo enfoques de clasificación entre las diferentes librerías seleccionadas para este trabajo de fin de titulación.

Tabla 18: Comparación de algoritmos de clasificación en librerías de aprendizaje automático en Python

	Orange	mlpy	scikit-learn	PyMVPA	Mdp
Gaussian Naive Bayes	✓	Maximum Likelihood Classifier	✓	✓	✓
Logistic Regression	✓	✓	✓	✓	
Classification Tree	✓	✓	✓		✓
k-nearest neighbors	✓	✓	✓	✓	✓
Support vector machines	✓	✓	✓	✓	✓

Elaboración. El Autor.

A continuación se detallan las configuraciones que se realizaron en cada algoritmo con el fin de que alcance su máximo rendimiento:

- **Logistic Regression**: se estableció el valor de parámetro de regularización de C=1.
- **Naïve Bayes:** este modelo no requiere de configuraciones.
- **K-nearest neighbors:** se realizaron modelos con valores de k entre 1 y 10, el mejor resultado obtenido fue para un valor de k=9. con distancia euclidiana.
- **Support Vector Machine (kernel Linear):** se utilizo el hiperparametro de regularización de C=1.
- **Support Vector Machine (kernel RBF):** se utilizo el hiperparametro de regularización de C=1 y el grado de kernel con un valor de gamma=0.1.
- **Classification Tree:** se utilizo entropía para el criterio de división y con el fin de simplificar la estructura del árbol se utilizo 20 muestras por hoja.

3.1.1.3 Implementación de librerías.

Se ha divido la implementación por librerías y por cada modelo se presenta el tiempo de entrenamiento mas predicción del valor de la variable “estado”, la exactitud del modelo y la matriz de confusión. A continuación se presenta el resultado de lo mencionado anteriormente.

3.1.1.3.1 Scikit-Learn.

Los códigos para el experimento de clasificación de la librería Scikit-Learn se encuentran en el Anexo 5, la tabla 19 muestra el resultado de los mismos.

Tabla 19: Resultados de los experimentos de clasificación con la librería Scikit-Learn

LOGISTIC REGRESSION

```
Time: 0.295587
Exactitud: 0.7877
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION     0      1    All
VALOR REAL
0           1174   2242   3416
1            674   9643  10317
All         1848  11885  13733
```

NAIVE BAYES

```
Time: 0.018079
Exactitud: 0.7823
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION     0      1    All
VALOR REAL
0           2111  1305   3416
1           1685  8632  10317
All         3796  9937  13733
```

K-NEAREST NEIGHBORS

```
Time: 1.201276
Exactitud: 0.8107
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION     0      1    All
VALOR REAL
0           1885   1531   3416
1           1069   9248  10317
All         2954  10779  13733
```

SUPPORT VECTOR MACHINE (KERNEL LINEAR)

```
Time: 342.590069
Exactitud: 0.7820
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION     0      1    All
VALOR REAL
0           1054   2362   3416
1            632   9685  10317
All         1686  12047  13733
```

SUPPORT VECTOR MACHINE (KERNEL RBF)

```
Time: 429.389459
Exactitud: 0.8207
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION     0      1    All
VALOR REAL
0           1641   1775   3416
1            688   9629  10317
All         2329  11404  13733
```

CLASSIFICATION TREE

```
Time: 0.153085
Exactitud: 0.8239
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION   0.0    1.0    All
VALOR REAL
0.0         1867   1549   3416
1.0          870   9447  10317
All         2737  10996  13733
```

Elaboración. El Autor.

En la tabla 19 podemos apreciar que el algoritmo que da mejor resultado es CLASSIFICATION TREE con una Exactitud de 0.8239, y el algoritmo con menor tiempo de entrenamiento mas predicción es NAIVE BAYES con 0.0181 segundos.

3.1.1.3.2 Orange.

Los códigos para el experimento de clasificación de la librería Orange se encuentran en el Anexo 6, la tabla 20 muestra el resultado de los mismos.

Tabla 20: Resultados de los experimentos de clasificación con la librería Orange

LOGISTIC REGRESSION

```
Time: 1.691207
EXACTITUD: 0.8091
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[D]=Divorcio [M]=Matrimonio
PREDICCION  D    M    All
VALOR REAL
D       1449    1967    3416
M       655     9662    10317
All     2104    11629   13733
```

NAIVE BAYES

```
Time: 0.328895
EXACTITUD: 0.8098
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[D]=Divorcio [M]=Matrimonio
PREDICCION  D    M    All
VALOR REAL
D       2252    1164    3416
M       1448    8869    10317
All     3700    10033   13733
```

K-NEAREST NEIGHBORS

```
Time: 702.99862
EXACTITUD: 0.8018
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[D]=Divorcio [M]=Matrimonio
PREDICCION  D    M    All
VALOR REAL
D       1866    1550    3416
M       1172    9145    10317
All     3038    10695   13733
```

SUPPORT VECTOR MACHINE (KERNEL LINEAR)

```
Time: 1802.400914
EXACTITUD: 0.7513
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[D]=Divorcio [M]=Matrimonio
PREDICCION  D    M    All
VALOR REAL
D       0       3416    3416
M       0       10317   10317
All     0       13733   13733
```

SUPPORT VECTOR MACHINE (KERNEL RBF)

```
Time: 1853.578528
EXACTITUD: 0.7513
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[D]=Divorcio [M]=Matrimonio
PREDICCION  D    M    All
VALOR REAL
D       0       3416    3416
M       0       10317   10317
All     0       13733   13733
```

CLASSIFICATION TREE

```
Time: 2.153133
EXACTITUD: 0.8086
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[D]=Divorcio [M]=Matrimonio
PREDICCION  D    M    All
VALOR REAL
D       1924    1492    3416
M       1136    9181    10317
All     3060    10673   13733
```

Elaboración. El Autor.

De la Tabla 20 se resumen a continuación la Exactitud producidos en el 20% de los datos correspondiente a la división para las pruebas:

- Exactitud Logistic Regression: 0.8091
- Exactitud Naive Bayes: 0.8098

- Exactitud K-nearest neighbors: 0.8018
- Exactitud Support Vector Machine (kernel Linear): 0.7513
- Exactitud Support Vector Machine (kernel RBF): 0.7513
- Exactitud Classification Tree: 0.8086

Como se puede observar el algoritmo que da mejor resultado de Exactitud es NAIVE BAYES con 0.8098, así mismo es el algoritmo con menor tiempo de entrenamiento mas predicción.

3.1.1.3.3 Mlpy.

Los códigos para el experimento de clasificación de la librería Mlpy se encuentran en el Anexo 7, la tabla 21 muestra el resultado de los mismos.

Tabla 21: Resultados de los experimentos de clasificación con la librería Mlpy

LOGISTIC REGRESSION

```
Time: 0.53182
Exactitud: 0.7807
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION     0      1    All
VALOR REAL
0           1034   2382   3416
1            629   9688  10317
All         1663  12070  13733
```

MAXIMUM LIKELIHOOD CLASSIFIER

```
Time: 0.05887
Exactitud: 0.7947
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION     0      1    All
VALOR REAL
0           1669   1747   3416
1           1073   9244  10317
All         2742  10991  13733
```

K-NEAREST NEIGHBORS

```
Time: 90.79912
Exactitud: 0.8126
```

```
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION     0      1    All
VALOR REAL
0           1885   1531   3416
1           1042   9275  10317
All         2927  10806  13733
```

SUPPORT VECTOR MACHINE (KERNEL LINEAR)

```
Time: 451.483206
Exactitud: 0.7820
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION   0.0    1.0    All
VALOR REAL
0           1054   2362   3416
1            632   9685  10317
All         1686  12047  13733
```

SUPPORT VECTOR MACHINE (KERNEL RBF)

```
Time: 631.883211
Exactitud: 0.8207
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION   0.0    1.0    All
VALOR REAL
0           1641   1775   3416
1            688   9629  10317
All         2329  11404  13733
```

CLASSIFICATION TREE

```
Time: 0.97854
Exactitud: 0.7518
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION    -1     0     1    All
VALOR REAL
0.0          262  1740  1414   3416
1.0          817   915  8585  10317
All         1079  2655  9999  13733
```

Elaboración. El Autor.

De la tabla 21 se resumen a continuación el total de los tiempos de entrenamiento mas el tiempo de predicción:

- Logistic Regression: 0.53182 seg
- Naive Bayes: 0.0589 seg
- K-nearest neighbors: 90.7991 seg
- Support Vector Machine (kernel Linear): 451.4832 seg
- Support Vector Machine (kernel RBF): 631.8832 seg
- Classification Tree: 0.9785 seg

El algoritmo con mejor tiempo es NAIVE BAYES con 0.0589 segundos y el algoritmo con el mejor puntaje de Exactitud es SUPPORT VECTOR MACHINE (KERNEL RBF) con 0.8207

3.1.1.3.4 PyMVPA.

Los códigos para el experimento de clasificación de la librería PyMVPA se encuentran en el Anexo 8, la tabla 22 muestra el resultado de los mismos.

Tabla 22: Resultados de los experimentos de clasificación con la librería PyMVPA

LOGISTIC REGRESSION

```
Time: 57.017401
Exactitud: 0.7875
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION     0      1    All
VALOR REAL
0           1175   2241   3416
1            677   9640  10317
All         1852  11881  13733
```

NAÏVE BAYES

```
Time: 0.370784
Exactitud: 0.7823
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION     0     1    All
VALOR REAL
0           2111  1305   3416
1           1685  8632  10317
All         3796  9937  13733
```

K-NEAREST NEIGHBORS

```
Time: 158.671912
Exactitud: 0.8120
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION     0      1    All
VALOR REAL
0           1923   1493   3416
1           1089   9228  10317
All         3012  10721  13733
```

SUPPORT VECTOR MACHINE (KERNEL LINEAR)

```
Time: 404.308438
Exactitud: 0.7825
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION   0.0    1.0    All
VALOR REAL
0           1063   2353   3416
1            634   9683  10317
All         1697  12036  13733
```

SUPPORT VECTOR MACHINE (KERNEL RBF)

```
Time: 1149.332825
Exactitud: 0.8207
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION   0.0    1.0    All
VALOR REAL
0           1641   1775   3416
1            687   9630  10317
All         2328  11405  13733
```

Elaboración. El Autor.

Como se puede observar en la tabla 22, el algoritmo con mejor resultado de Exactitud es SUPPORT VECTOR MACHINE (KERNEL RBF) con 0.8207, y el algoritmo con mejor tiempo es NAIVE BAYES con 0.37 segundos.

3.1.1.3.5 Mdp.

Los códigos para el experimento de clasificación de la librería Mdp se encuentran en el Anexo 9, la tabla 23 muestra el resultado de los mismos.

Tabla 23: Resultados de los experimentos de clasificación con la librería Mdp

NAÏVE BAYES

```
Time: 0.039194
Exactitud: 0.7947
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION     0      1    All
VALOR REAL
0           1669   1747   3416
1           1073   9244  10317
All         2742  10991  13733
```

K-NEAREST NEIGHBORS

```
Time: 164.769179
Exactitud: 0.8093
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION   0.0    1.0    All
VALOR REAL
0.0         1871   1545   3416
1.0         1074   9243  10317
All         2945  10788  13733
```

SUPPORT VECTOR MACHINE (KERNEL LINEAR)

```
Time: 2447.671649
Exactitud: 0.7819
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION   0.0    1.0    All
VALOR REAL
0.0         1049   2367   3416
1.0          628   9689  10317
All         1677  12056  13733
```

SUPPORT VECTOR MACHINE (KERNEL RBF)

```
Time: 2790.686687
Exactitud: 0.8207
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION   0.0    1.0    All
VALOR REAL
0.0         1641   1775   3416
1.0          688   9629  10317
All         2329  11404  13733
```

CLASSIFICATION TREE

```
Time: 0.17694
Exactitud: 0.8239
Matriz de Confusión de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio
PREDICCION   0.0    1.0    All
VALOR REAL
0.0         1867   1549   3416
1.0          870   9447  10317
All         2737  10996  13733
```

Elaboración. El Autor.

En la tabla 23, se puede observar que el algoritmo con mejor resultado de Exactitud es CLASSIFICATION TREE con 0.8239, y el algoritmo con mejor tiempo es NAIVE BAYES con 0.039 segundos.

3.1.2 Regresión.

El problema a resolver en el enfoque de regresión es predecir el numero de años de matrimonio de una pareja que se esta divorciando. El conjunto de datos que se ha utilizado para este problema corresponde a divorcios detallado en la sección 2.1.2.

3.1.2.1 Selección de librerías.

Para la selección de librerías se ha tomado en cuenta la disponibilidad de algoritmos de regresión en las mismas. Las librerías seleccionadas fueron las siguientes:

- ✓ Scikit-Learn 0.16.1
- ✓ Orange 2.7.8

- ✓ Mlpy 3.5.0
- ✓ PyMVPA 2.4.2
- ✓ Statsmodels 0.6.1

3.1.2.2 Selección de algoritmos.

Para la selección de los algoritmos al igual que en clasificación, se tomo en cuenta que la gran mayoría de las librerías cuente con los mismos, a continuación se detallan los mismos:

- ✓ Ordinary least squares
- ✓ Ridge Regression
- ✓ Kernel Ridge
- ✓ Lasso Lars
- ✓ Elastic Net
- ✓ Support Vector Regression (kernel RBF)

En la tabla 24 se puede observar la disponibilidad de algoritmos de regresión entre las diferentes librerías seleccionadas para este trabajo.

Tabla 24: Comparación de algoritmos de regresión en librerías de aprendizaje automático en Python

	Orange	**mlpy**	**scikit-learn**	**PyMVPA**	**Statsmodels**
Ridge Regression	✓	✓	✓	✓	✓
Support Vector Regression	✓	✓	✓	✓	
Gaussian Processes			✓	✓	
Kernel Ridge		✓	✓		
Random Forest			✓		✓
Elastic net		✓	✓	✓	✓
Ordinary Least Squares			✓		✓
Lasso Lars	✓		✓	✓	✓

Elaboración. El Autor.

A continuación se detallan las configuraciones que se realizaron en cada algoritmo con el fin de que alcance su máximo rendimiento:

- **Ordinary Least Squares:** no requiere de configuraciones.
- **Ridge Regression:** se estableció el valor del parámetro de regularización de lambda=0.1
- **Kernel Ridge:** se estableció el valor del parámetro de regularización de lambda=0.01 y se utilizo kernel polynomial
- **Lasso Lars:** se estableció el valor del parámetro de regularización de 0.01
- **Elastic Net**: se estableció el valor del parámetro de regularización de 0.001
- **Support Vector Regression (kernel RBF):** se utilizo el hiperparametro de regularización con el valor de C=1e1 y el grado de kernel con un valor de gamma=0.01

3.1.2.3 Implementación de librerías.

Se ha dividido la implementación por librerías y por cada modelo se presenta el tiempo de entrenamiento mas predicción del valor de la variable "dur_mat", coeficiente de determinación, error cuadrático medio y el resultado de la predicción de las primeras 6 instancias en el conjunto de pruebas.

3.1.2.3.1 Scikit-Learn.

Los códigos para el experimento de regresión de la librería Scikit-Learn se encuentran en el Anexo 10, la tabla 25 muestra el resultado de los mismos.

Tabla 25: Resultados de los experimentos de regresión con la librería Scikit-Learn

ORDINARY LEAST SQUARES
Time: 0.012278 Coeficiente de determinación(R^2) : 0.6527 MSE: 36.7666 Predicción de las primeras 6 instancias[VR]=Valor Real [VP]= Valor Predicho VR: 17.0 VP:25.6 VR: 20.0 VP:17.9 VR: 39.0 VP:30.9 VR: 39.0 VP:31.9 VR: 21.0 VP:14.0

```
VR: 15.0 VP:16.0
```

REGRESSION RIDGE

```
Time: 0.004165
Coeficiente de determinación(R²) : 0.6527
MSE: 36.7666

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:25.6
VR: 20.0 VP:17.9
VR: 39.0 VP:30.9
VR: 39.0 VP:31.9
VR: 21.0 VP:14.0
VR: 15.0 VP:16.0
```

KERNEL RIDGE

```
Time: 33.13006
Coeficiente de determinación(R²) : 0.6985
MSE: 31.9107

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:23.7
VR: 20.0 VP:18.6
VR: 39.0 VP:33.1
VR: 39.0 VP:35.5
VR: 21.0 VP:14.4
VR: 15.0 VP:16.0
```

LASSO LARS

```
Time: 0.037743
Coeficiente de determinación(R²) : 0.6521
MSE: 36.8227

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:25.4
VR: 20.0 VP:18.0
VR: 39.0 VP:30.7
VR: 39.0 VP:31.7
VR: 21.0 VP:14.3
VR: 15.0 VP:15.6
```

ELASTIC NET
Time: 0.009527 Coeficiente de determinación(R^2) : 0.6527 MSE: 36.7689 Predicción de las primeras 6 instancias[VR]=Valor Real [VP]= Valor Predicho VR: 17.0 VP:25.6 VR: 20.0 VP:17.9 VR: 39.0 VP:30.9 VR: 39.0 VP:31.9 VR: 21.0 VP:14.0 VR: 15.0 VP:16.0
SUPPORT VECTOR REGRESSION
Time: 11.120379 Coeficiente de determinación(R^2) : 0.6874 MSE: 33.0935 Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=Valor Predicho VR: 17.0 VP:24.0 VR: 20.0 VP:21.2 VR: 39.0 VP:35.3 VR: 39.0 VP:38.1 VR: 21.0 VP:16.4 VR: 15.0 VP:17.5

Elaboración. El Autor.

En la tabla 25, se puede observar que el algoritmo con el mejor puntaje de coeficiente de determinación y error cuadrático medio es SUPPORT VECTOR REGRESSION (KERNEL RBF) con 0.7097 y 33.0935 respectivamente, y el algoritmo con mejor tiempo es REGRESSION RIDGE con 0.004165 seg.

3.1.2.3.2 Orange.

Los códigos para el experimento de regresión de la librería Orange se encuentran en el Anexo 11, la tabla 26 muestra el resultado de los mismos.

Tabla 26: Resultados de los experimentos de regresión con la librería Orange

REGRESSION RIDGE

```
Time: 0.049126
Coeficiente de determinación(R²) : 0.6490
MSE: 37.1506

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:24.3
VR: 20.0 VP:17.8
VR: 39.0 VP:31.5
VR: 39.0 VP:31.7
VR: 21.0 VP:14.6
VR: 15.0 VP:15.6
```

LASSO LARS

```
Time: 0.102948
Coeficiente de determinación(R²) : 0.6490
MSE: 37.1560

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:24.3
VR: 20.0 VP:17.8
VR: 39.0 VP:31.5
VR: 39.0 VP:31.7
VR: 21.0 VP:14.6
VR: 15.0 VP:15.6
```

SUPPORT VECTOR REGRESSION

```
Time: 72.470312
Coeficiente de determinación(R²) : 0.6404
MSE: 38.0677

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:26.4
VR: 20.0 VP:19.0
VR: 39.0 VP:33.5
VR: 39.0 VP:33.5
VR: 21.0 VP:15.7
```

```
VR: 15.0 VP:16.9
```

Elaboración. El Autor.

En la tabla 26, se puede observar que los algoritmos con el mejor puntaje de coeficiente de determinación lo comparten los algoritmos LASSO LARS y REGRESSION RIDGE con 0.6490, el mejor puntaje de error cuadrático medio es para el algoritmo REGRESSION RIDGE con 37.1506 y el algoritmo con mejor tiempo es REGRESSION RIDGE con 0.049126 seg.

3.1.2.3.3 Mlpy.

Los códigos para el experimento de regresión de la librería Mlpy se encuentran en el Anexo 12, la tabla 27 muestra el resultado de los mismos.

Tabla 27: Resultados de los experimentos de regresión con la librería Mlpy

REGRESSION RIDGE

```
Time: 0.002652
Coeficiente de determinación(R²) : 0.6527
MSE: 36.7666

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:25.6
VR: 20.0 VP:17.9
VR: 39.0 VP:30.9
VR: 39.0 VP:31.9
VR: 21.0 VP:14.0
VR: 15.0 VP:16.0
```

KERNEL RIDGE

```
Time: 71.810123
Coeficiente de determinación(R²) : 0.6957
MSE: 32.2133

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:25.1
VR: 20.0 VP:18.4
```

```
VR: 39.0 VP:32.5
VR: 39.0 VP:34.3
VR: 21.0 VP:14.3
VR: 15.0 VP:16.0
```

ELASTIC NET

```
Time: 0.388139
Coeficiente de determinación(R²) : 0.6525
MSE: 36.7820

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:25.5
VR: 20.0 VP:17.9
VR: 39.0 VP:30.7
VR: 39.0 VP:31.7
VR: 21.0 VP:14.2
VR: 15.0 VP:15.8
```

SUPPORT VECTOR REGRESSION

```
Time: 11.357602
Coeficiente de determinación(R²) : 0.6874
MSE: 33.0935

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho


VR: 17.0 VP:24.0
VR: 20.0 VP:21.2
VR: 39.0 VP:35.3
VR: 39.0 VP:38.1
VR: 21.0 VP:16.4
VR: 15.0 VP:17.5
```

Elaboración. El Autor.

En la tabla 27, se puede observar que el algoritmo con el mejor puntaje de coeficiente de determinación es SUPPORT VECTOR REGRESSION (KERNEL RBF) con 0.7097, el mejor puntaje de error cuadrático medio es para el algoritmo KERNEL RIDGE con 32.2133 y el algoritmo con mejor tiempo es REGRESSION RIDGE con 0.002652 seg.

3.1.2.3.4 PyMVPA.

Los códigos para el experimento de regresión de la librería PyMVPA se encuentran en el Anexo 13, la tabla 28 muestra el resultado de los mismos.

Tabla 28: Resultados de los experimentos de regresión con la librería PyMVPA

REGRESSION RIDGE
Time: 0.020603 Coeficiente de determinación(R^2) : 0.6527 MSE: 36.7666 Predicción de las primeras 6 instancias[VR]=Valor Real [VP]= Valor Predicho VR: 17.0 VP:25.6 VR: 20.0 VP:17.9 VR: 39.0 VP:30.9 VR: 39.0 VP:31.9 VR: 21.0 VP:14.0 VR: 15.0 VP:16.0
LASSO LARS
Time: 0.122329 Coeficiente de determinación(R^2) : 0.6513 MSE: 36.9120 Predicción de las primeras 6 instancias[VR]=Valor Real [VP]= Valor Predicho VR: 17.0 VP:25.4 VR: 20.0 VP:18.0 VR: 39.0 VP:30.7 VR: 39.0 VP:31.6 VR: 21.0 VP:14.4 VR: 15.0 VP:15.5
ELASTIC NET
Time: 0.097293 Coeficiente de determinación(R^2) : 0.6513 MSE: 36.9108 Predicción de las primeras 6 instancias[VR]=Valor Real [VP]= Valor Predicho

```
VR: 17.0 VP:25.4
VR: 20.0 VP:18.0
VR: 39.0 VP:30.7
VR: 39.0 VP:31.6
VR: 21.0 VP:14.4
VR: 15.0 VP:15.5
```

SUPPORT VECTOR REGRESSION

```
Time: 11.174803
Coeficiente de determinación(R²) : 0.6875
MSE: 33.0766

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:23.5
VR: 20.0 VP:21.1
VR: 39.0 VP:35.2
VR: 39.0 VP:37.9
VR: 21.0 VP:16.4
VR: 15.0 VP:17.6
```

Elaboración. El Autor.

En la tabla 28, se puede observar que el algoritmo con el mejor puntaje de coeficiente de determinación y error cuadrático medio es SUPPORT VECTOR REGRESSION (KERNEL RBF) con 0.7099 y 33.0766 respectivamente, y el algoritmo con mejor tiempo es REGRESSION RIDGE con 0.020603 seg.

3.1.2.3.5 Statsmodels.

Los códigos para el experimento de regresión de la librería Statsmodels se encuentran en el Anexo 14, la tabla 29 muestra el resultado de los mismos.

Tabla 29: Resultados de los experimentos de regresión con la librería Statsmodels

ORDINARY LEAST SQUARES

```
Time: 0.010433
Coeficiente de determinación(R²) : 0.6524
MSE: 36.7970
```

```
Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:25.5
VR: 20.0 VP:17.9
VR: 39.0 VP:30.9
VR: 39.0 VP:31.9
VR: 21.0 VP:14.1
VR: 15.0 VP:16.1
```

REGRESSION RIDGE

```
Time: 1.432243
Coeficiente de determinación(R²) : 0.6317
MSE: 38.9882

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:21.8
VR: 20.0 VP:17.1
VR: 39.0 VP:30.7
VR: 39.0 VP:32.1
VR: 21.0 VP:14.1
VR: 15.0 VP:16.6
```

LASSO LARS

```
Time: 2.642063
Coeficiente de determinación(R²) : 0.6524
MSE: 36.7974

Predicción de las primeras 6 instancias[VR]=Valor Real [VP]=
Valor Predicho

VR: 17.0 VP:25.4
VR: 20.0 VP:17.9
VR: 39.0 VP:30.9
VR: 39.0 VP:31.9
VR: 21.0 VP:14.1
VR: 15.0 VP:16.1
```

Elaboración. El Autor.

En la tabla 29, se puede observar que el algoritmo que alcanza el mejor puntaje en todo es ORDINARY LEAST SQUARES así el coeficiente de determinación y error cuadrático medio es 0.6526 y 36.7970 respectivamente, con un tiempo de 0.010433 seg.

CAPÍTULO IV

EVALUACIÓN DE RESULTADOS

4.1 Evaluación de Resultados.

En este capitulo se presentan las comparaciones de rendimiento de las diferentes librerías implementadas en el capitulo 3.

Existen enfoques que se pueden utilizar tanto para clasificación y regresión en el presente trabajo de fin de titulación se ha utilizado las maquinas de vectores de soporte tanto para nuestro problema de clasificación como para el de regresión, aprovechado las ventajas que estas tienen con su función kernel, permitiendo proyectar la información a un espacio de características de mayor dimensión, la cual ha dado buenos resultados para la solución de nuestros problemas.

Cabe mencionar que aunque algunos enfoques se utilicen para clasificación y regresión no todos actúan de la misma manera por ejemplo el algoritmo *K-nearest neighbors* para la regresión, las predicciones de los K vecinos más cercanos se basan en los promedios de las salidas de los k vecinos más cercanos; Para la clasificación, en cambio se utiliza una votación mayoritaria. Por ende algunas librerías no necesariamente tienen disponibilidad de un mismo algoritmo bajo estos dos enfoques.

Los algoritmos implementados divididos bajo los enfoques de clasificación y regresión han sido los siguientes:

Clasificación y Regresión

- ✓ Máquinas de vectores de soporte

Clasificación

- ✓ Logistic Regression:
- ✓ Naive Bayes:
- ✓ K-nearest neighbors:
- ✓ Classification Tree

Regresión

- ✓ Ordinary least squares
- ✓ Ridge Regression
- ✓ Kernel Ridge
- ✓ Lasso Lars
- ✓ Elastic Net

Para la selección de los parámetros de evaluación tanto en el problema de clasificación como de regresión, se ha tomando en cuenta como prioridad los mas representativos acerca de la fiabilidad del modelo y la disponibilidad de estas métricas en los mismos, siendo este ultimo muy importante para poder contrastar los resultados entre si. La relación que presentan los parámetros seleccionados con el rendimiento de las librerías representan en general dos puntos muy importantes la velocidad y exactitud de las mismas.

Se ha utilizado la misma partición de datos (entrenamiento y pruebas) en cada una de las librerías, con el fin de obtener condiciones mas equilibradas para contrastar los resultados. Además para realizar este proceso se ha tomado en cuenta algunos parámetros detallados en la siguiente sección.

4.1.1 Resultados del enfoque de clasificación.

Los experimentos se han realizado con el método de evaluación training-test. En este método tenemos todos los datos repartidos en dos conjuntos: entrenamiento y pruebas. Los datos del entrenamiento se han a utilizado para entrenar el modelo, mientras que los datos de pruebas para validar el modelo. Estos experimentos se realizaron sobre la misma distribución de datos de la siguiente manera; la partición de entrenamiento de estos experimentos tiene un total de 54930 instancias las cuales conforman el 80%. La partición pruebas esta compuesto por 13733 instancias que corresponde al 20% del total de los datos.

4.1.1.1 Parámetros de Evaluación.

Para realizar el análisis de rendimiento de las librerías se estableció algunos parámetros de evaluación de los algoritmos con el propósito de contrastar y poder determinar cual es el más adecuado en nuestro experimento para modelos de clasificación.

A continuación se detallan los parámetros seleccionados:

- **Exactitud:** es la exactitud del modelo y representa el numero de instancias clasificadas correctamente para el numero total de instancias.
- **Error:** representa el numero de instancias clasificadas incorrectamente para el numero total de instancias.
- **Tiempo de entrenamiento mas predicción:** es el tiempo en segundos que le toma a cada librería entrenarse sobre el conjunto de entrenamiento (80% de los datos) mas el tiempo de predecir el valor de la variable “estado” en el conjunto de pruebas (20% de los datos)

- **Matriz de Confusión:** sobre esta matriz se selecciono el numero de instancias clasificadas correctamente y el numero de instancias clasificadas incorrectamente.

4.1.1.2 Análisis de los Resultados.

A continuación se presenta un resumen y análisis de los diferentes experimentos

Tabla 30: Resumen de resultados en los algoritmos de clasificación

Nro.	Librería	Algoritmo	Entrenamiento + predicción	Partición Pruebas (20%)			
			Tiempo	Exactitud	Error	Nro. Instancias correctamente clasificadas	Nro. Instancias incorrectamente clasificadas
1	SCIKIT-LEARN	LOGISTIC REGRESSION	0.30	0.7877	0.2123	10817	2916
2	ORANGE	LOGISTIC REGRESSION	1.69	0.8091	0.1909	11111	2622
3	MLPY	LOGISTIC REGRESSION	0.53	0.7807	0.2193	10721	3012
4	PYMVPA	LOGISTIC REGRESSION	57.02	0.7875	0.2125	10815	2918
5	SCIKIT-LEARN	GAUSSIAN NAIVE BAYES	0.02	0.7823	0.2177	10743	2990
6	ORANGE	GAUSSIAN NAIVE BAYES	0.33	0.8098	0.1902	11121	2612
7	MLPY	MAXIMUM LIKELIHOOD CLASSIFIER	0.06	0.7947	0.2053	10914	2819
8	PYMVPA	GAUSSIAN NAIVE BAYES	0.37	0.7823	0.2177	10743	2990
9	MDP	GAUSSIAN NAIVE BAYES	0.04	0.7947	0.2053	10914	2819

10	SCIKIT-LEARN	K-NEAREST NEIGHBORS	1.20	0.8107	0.1893	11133	2600
11	ORANGE	K-NEAREST NEIGHBORS	703.00	0.8018	0.1982	11011	2722
12	MLPY	K-NEAREST NEIGHBORS	90.80	0.8126	0.1874	11160	2573
13	PYMVPA	K-NEAREST NEIGHBORS	158.67	0.8120	0.1880	11151	2582
14	MDP	K-NEAREST NEIGHBORS	164.77	0.8093	0.1907	11114	2619
15	SCIKIT-LEARN	SUPPORT VECTOR MACHINE (LINEAR)	342.59	0.7820	0.2180	10739	2994
16	ORANGE	SUPPORT VECTOR MACHINE (LINEAR)	1802.40	0.7513	0.2487	10318	3415
17	MLPY	SUPPORT VECTOR MACHINE (LINEAR)	451.48	0.7820	0.2180	10739	2994
18	PYMVPA	SUPPORT VECTOR MACHINE (LINEAR)	535.43	0.7825	0.2175	10746	2987
19	MDP	SUPPORT VECTOR MACHINE (LINEAR)	2447.67	0.7819	0.2181	10738	2995
20	SCIKIT-LEARN	SUPPORT VECTOR MACHINE (RBF)	429.39	0.8207	0.1793	11270	2463
21	ORANGE	SUPPORT VECTOR MACHINE (RBF)	1853.58	0.7513	0.2487	10318	3415
22	MLPY	SUPPORT VECTOR MACHINE (RBF)	631.88	0.8207	0.1793	11270	2463

23	PYMVPA	SUPPORT VECTOR MACHINE (RBF)	1149.33	0.8207	0.1793	11271	2462
24	MDP	SUPPORT VECTOR MACHINE (RBF)	2790.69	0.8207	0.1793	11270	2463
25	SCIKIT-LEARN	CLASSIFICATION TREE	0.15	0.8239	0.1761	11314	2419
26	ORANGE	CLASSIFICATION TREE	2.15	0.8086	0.1914	11105	2628
27	MLPY	CLASSIFICATION TREE	0.98	0.7518	0.2482	10324	3409
28	MDP	CLASSIFICATION TREE	0.18	0.8239	0.1761	11314	2419

Elaboración. El Autor.

En la siguientes gráficas podemos ver de una mejor manera los resultados expuestos en la tabla 34. En la figura 27 menos es mejor, mientras que en la figura 35 mas es mejor.

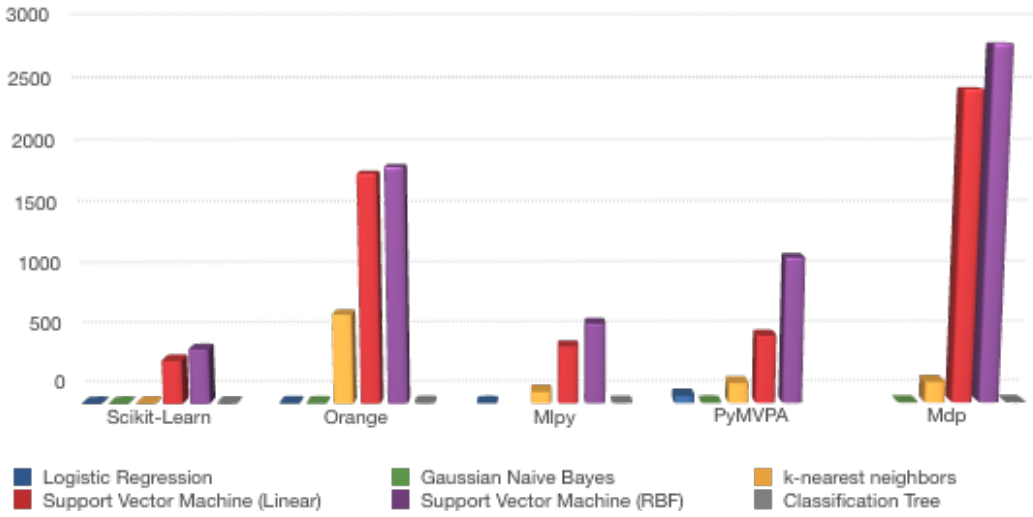

Figura 34. Tiempo de entrenamiento mas predicción de algoritmos de clasificación

Elaboración. El Autor.

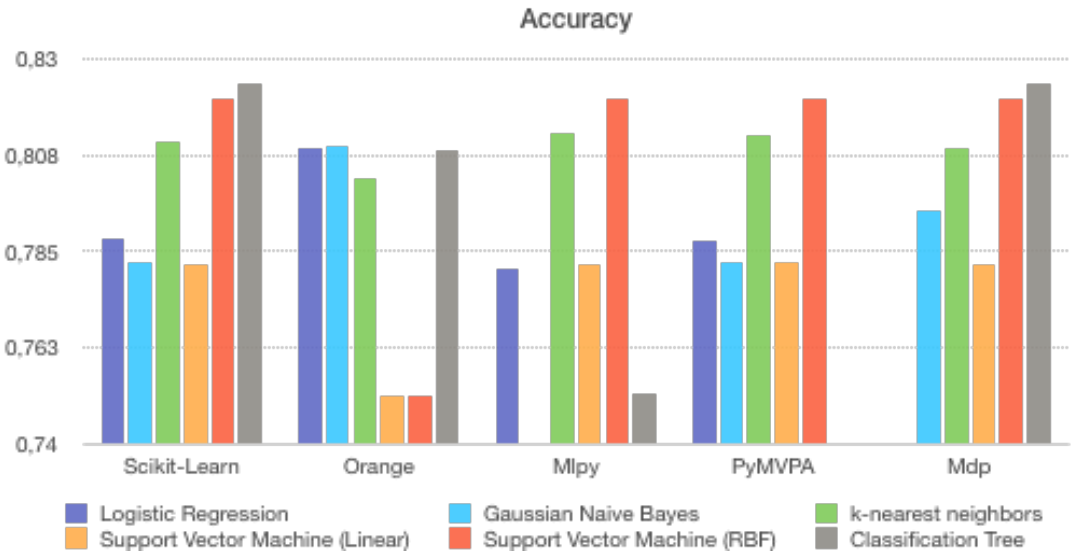

Figura *35*. Puntaje Exactitud de algoritmos de clasificación agrupados por librería

Elaboración. El Autor.

Como apreciaciones generales de los resultados en los experimentos realizados se puede observar que para todas las librerías los algoritmos *Support Vector Machine* ya sea con kernel Linear y Rbf son los que toman mas tiempo de entrenamiento y predicción, mientras que el algoritmo *Gaussian Naive Bayes* es el que requiere menos tiempo como se puede observar en la tabla 16.

En cuanto a las librerías que toman mas tiempo como se puede apreciar en la figura 34 son Orange y Mdp. La librería Mdp con el algoritmo *Support Vector Machine (RBF)* es la que obtiene el peor resultado de todos los experimentos con un tiempo aproximado a los 47 minutos.

Por otro lado en la figura 35 se puede observar que el mejor puntaje de Exactitud[40] le corresponde a la librería Scikit-Learn con el algoritmo *Classification Tree*.

De la tabla 30, se obtiene las tablas 31 y 32 (extraída de la tabla 30) para mostrar los mejores resultados en cuanto a tiempo y Exactitud por cada algoritmo de clasificación.

Tabla 31: Mejores puntaje Exactitud en el experimento de clasificación

Pos.	Librería	Algoritmo	Partición Pruebas (20%)	
			Exactitud	Error
5	ORANGE	LOGISTIC REGRESSION	0.8091	0.1909
4	ORANGE	GAUSSIAN NAIVE BAYES	0.8098	0.1902
3	MLPY	K-NEAREST NEIGHBORS	0.8126	0.1874
6	PYMVPA	SUPPORT VECTOR MACHINE (LINEAR)	0.7825	0.2175
2	PYMVPA	SUPPORT VECTOR MACHINE (RBF)	0.8207	0.1793
1	***SCIKIT-LEARN**	**CLASSIFICATION TREE**	**0.8239**	**0.1761**

Elaboración. El Autor.

[40] Exactitud: representa la tasa de aciertos del modelo mientras se acerca a 1 es mejor.

Tabla 32: Mejores tiempos en el experimento de clasificación

Pos.	Librería	Algoritmo	Entrenamiento + predicción
			Tiempo (seg)
3	SCIKIT-LEARN	LOGISTIC REGRESSION	0.30
1	**SCIKIT-LEARN**	**GAUSSIAN NAIVE BAYES**	**0.02**
4	SCIKIT-LEARN	K-NEAREST NEIGHBORS	1.20
5	SCIKIT-LEARN	SUPPORT VECTOR MACHINE (LINEAR)	342.59
6	SCIKIT-LEARN	SUPPORT VECTOR MACHINE (RBF)	429.39
2	SCIKIT-LEARN	CLASSIFICATION TREE	0.15

Elaboración. El Autor.

La tabla 31 muestra los mejores puntajes de Exactitud, podemos observar variedad de resultados, así; el mejor puntaje corresponde a la librería Scikit-Learn con el algoritmo *Classification Tree* con un puntaje de 0.8239. Cabe mencionar que comparte el mismo puntaje de la librería Mdp, pero esta ultima, para este algoritmo utiliza envolturas de la primera librería antes mencionada.

La librería PyMVPA tiene el segundo puntaje con el algoritmo *Support Vector Machine (RBF)* con un puntaje de 0.8207.

El tercer mejor puntaje corresponde a la librería Mlpy con el algoritmo *K-nearest neighbors* con un puntaje de 0.8126.

El cuarto y quinto mejor puntaje corresponde a la librería Orange con el algoritmo *Gaussian Naive Bayes* y *Logistic Regression* con 0.8098 y 0.8091 respectivamente.

El sexto mejor puntaje corresponde a la librería PyMVPA con el algoritmo *Support Vector* Machine (Linear) con un puntaje de 0.7825.

Un punto importante de los resultados antes mencionados es que el algoritmo *Classification Tree* de la librería Scikit-Learn, además de tener el segundo mejor tiempo con 0.15 segundos

obtiene el mejor puntaje de Exactitud de todos los algoritmos con 0.8239.

Por otro lado la tabla 32, muestra todos los mejores resultados en cuanto a tiempo se puede observar que todos corresponden a la librería Scikit-Learn, aunque resultan ser bastante variados ya que si nos fijamos existe una gran diferencia entre los distintos valores. El mejor resultado corresponde al algoritmo *Gaussian Naive Bayes* con un tiempo muy bajo de 0.02 segundos.

En la gráfica 36 podemos ver de una manera visual cómo se distribuyen los resultados antes mencionados.

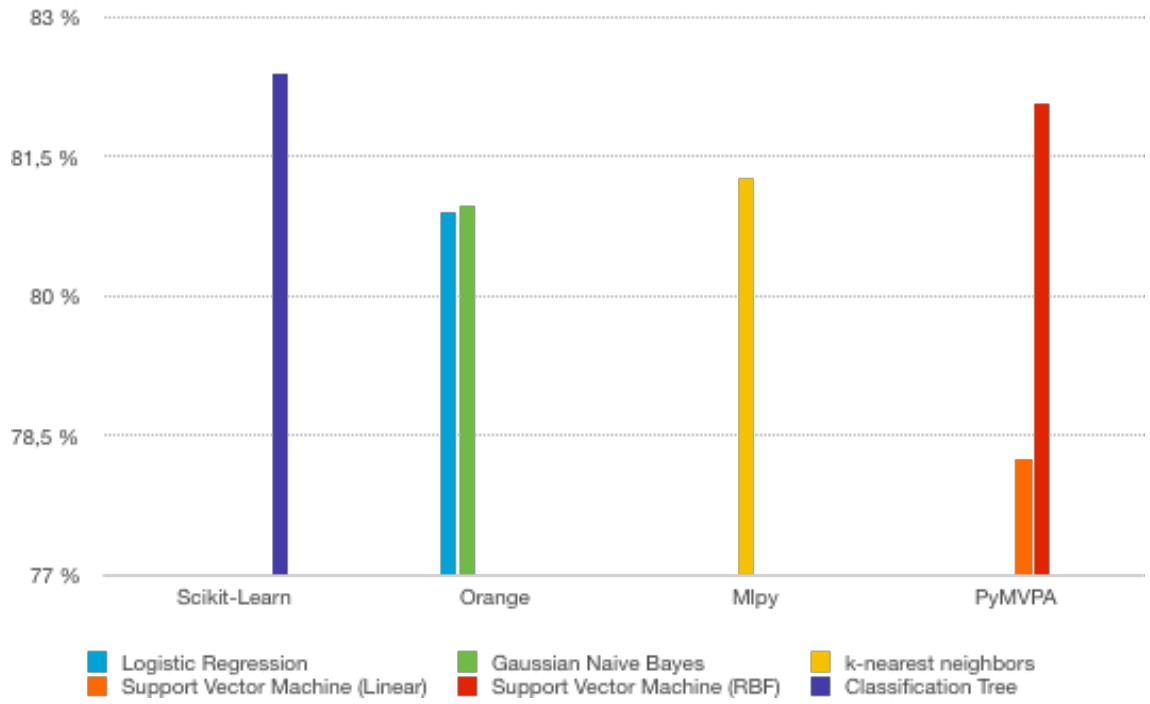

Figura 36. Mejores puntajes Exactitud de algoritmos de clasificación

Elaboración. El Autor.

4.1.2 Resultados del enfoque de regresión.

Los experimentos al igual que el enfoque de clasificación se han realizado con el método de evaluación training-test en donde la partición de entrenamiento tiene un total de 14546 instancias las cuales conforman el 80% y la partición pruebas esta compuesto por 3637 instancias que corresponde al 20% del total de los datos.

4.1.2.1 Parámetros de evaluación.

Los parámetros de evaluación para poder realizar el análisis de rendimiento de las diferentes librerías seleccionadas se detallan a continuación:

- **Coeficiente de determinación (R2):** es la medida mas representativa de la fiabilidad del modelo, siendo 1 el mejor puntaje posible.
- **Error cuadrático medio:** es el promedio de los errores al cuadrado (MSE).
- **Tiempo de entrenamiento mas predicción:** es el tiempo en segundos que le toma a cada librería entrenarse sobre el conjunto de entrenamiento (80% de los datos) y predecir el valor de la variable "dur_mat" en el conjunto de pruebas (20% de los datos)
- **Predicción del valor de la variable dependiente:** se ha tomado la predicción del valor de la variable "dur_mat" de las primeras 6 instancias sobre el conjunto de pruebas.

4.1.2.2 Análisis de los Resultados.

A continuación se presenta un resumen y análisis de los resultados de los diferentes experimentos:

Tabla 33: Resultados de los experimentos en los algoritmos de regresión

Nro.	Librería	Algoritmo	Entrenamiento + predicción	Partición Pruebas (20%)	
			Tiempo	R2	MSE
1	Scikit-Learn	Ordinary Least Squares	0.012	0.6527	36.7666
2	Statsmodels	Ordinary Least Squares	0.010	0.6524	36.7970
3	Scikit-Learn	Ridge Regression	0.004	0.6527	36.7666
4	Mlpy	Ridge Regression	0.003	0.6527	36.7666
5	PyMVPA	Ridge Regression	0.021	0.6527	36.7666
6	Orange	Ridge Regression	0.049	0.6490	37.1506
7	Statsmodels	Ridge Regression	1.432	0.6317	38.9882
8	Scikit-Learn	Kernel Ridge	33.130	0.6985	31.9107
9	Mlpy	Kernel Ridge	71.810	0.6957	32.2133
10	Scikit-Learn	Lasso Lars	0.038	0.6521	36.8227

11	PyMVPA	Lasso Lars	0.122	0.6513	36.9120
12	Orange	Lasso Lars	0.103	0.6490	37.1560
13	Statsmodels	Lasso Lars	2.642	0.6524	36.7974
14	Scikit-Learn	Elastic Net	0.010	0.6527	36.7689
15	Mlpy	Elastic Net	0.388	0.6525	36.7820
16	PyMVPA	Elastic Net	0.097	0.6513	36.9108
17	Scikit-Learn	Support Vector Regression (RBF)	11.120	0.6874	33.0935
18	Mlpy	Support Vector Regression (RBF)	11.358	0.6874	33.0935
19	PyMVPA	Support Vector Regression (RBF)	11.175	0.6875	33.0766
20	Orange	Support Vector Regression (RBF)	72.470	0.6404	38.0677

Elaboración. El Autor.

Tabla 34: Predicción de la variable dur_mat con el algoritmo Ordinary Least Squares

Nro.	Real	Predicho	
		Scikit-Learn	Statsmodels
1	17.0	25.6	25.5
2	20.0	17.9	17.9
3	39.0	30.9	30.9
4	39.0	31.9	31.9
5	21.0	14.0	14.1
6	15.0	16.0	16.1

Elaboración. El Autor.

Tabla 35: Predicción de la variable dur_mat con el algoritmo Regression Ridge

Nro.	Real	Predicho				
		Scikit-Learn	Orange	Mlpy	PyMVPA	Statsmodels
1	17.0	25.6	24.3	25.6	25.6	21.8
2	20.0	17.9	17.8	17.9	17.9	17.1
3	39.0	30.9	31.5	30.9	30.9	30.7
4	39.0	31.9	31.7	31.9	31.9	32.1
5	21.0	14.0	14.6	14.0	14.0	14.1
6	15.0	16.0	15.6	16.0	16.0	16.6

Elaboración. El Autor.

Tabla 36: Predicción de la variable dur_mat con el algoritmo Kernel Ridge

Nro.	Real	Predicho	
		Scikit-Learn	Mlpy
1	17.0	23.7	25.1
2	20.0	18.6	18.4
3	39.0	33.1	32.5
4	39.0	35.5	34.3
5	21.0	14.4	14.3
6	15.0	16.0	16.0

Elaboración. El Autor.

Tabla 37: Predicción de la variable dur_mat con el algoritmo Lasso Lars

Nro.	Real	Predicho			
		Scikit-Learn	Orange	PyMVPA	Statsmodels
1	17.0	25.4	24.3	25.4	25.4
2	20.0	18.0	17.8	18.0	17.9
3	39.0	30.7	31.5	30.7	30.9
4	39.0	31.7	31.7	31.6	31.9
5	21.0	14.3	14.6	14.4	14.1
6	15.0	15.6	15.6	15.5	16.1

Elaboración. El Autor.

Tabla 38: Predicción de la variable dur_mat con el algoritmo Elastic Net

Nro.	V. Real	V. Predicho		
		Scikit-Learn	Mlpy	PyMVPA
1	17.0	25.6	25.5	25.4
2	20.0	17.9	17.9	18.0
3	39.0	30.9	30.7	30.7
4	39.0	31.9	31.7	31.6
5	21.0	14.0	14.2	14.4
6	15.0	16.0	15.8	15.5

Elaboración. El Autor.

Tabla 39: Predicción de la variable dur_mat con el algoritmo Support Vector Regression

Nro.	V. Real	V. Predicho			
		Scikit-Learn	Orange	Mlpy	PyMVPA
1	17.0	24.0	26.4	24.0	23.5
2	20.0	21.2	19.0	21.2	21.1
3	39.0	35.3	33.5	35.3	35.2
4	39.0	38.1	33.5	38.1	37.9
5	21.0	16.4	15.7	16.4	16.4
6	15.0	17.5	16.9	17.5	17.6

Elaboración. El Autor.

Como apreciaciones generales de los experimentos realizados con los algoritmos de regresión se puede observar que todos los tiempos de entrenamiento mas predicción tienen menos de 1 segundo tal como se muestra en la figura 38.

En cuanto a los mejores puntajes y al igual que en nuestro experimento de clasificación se destaca la librería Scikit-Learn, en este caso 5 de los 6 algoritmos implementados fueron los que mejor puntaje de precisión obtuvieron. Dentro de los mejores puntaje de R2[41] y MSE[42] están los algoritmos *Kernel Ridge* y *Support Vector Regression (RBF)*. como se puede apreciar en la tabla 33.

Por otro lado entre los peores resultados de nuestro experimento se encuentra la librería Orange, la cual es la única que no posee un mejor puntaje en algún algoritmo y además obtiene el peor resultado de los experimentos en cuanto al tiempo con el algoritmo *Support Vector Regression (RBF)* con 72.470 segundos.

Para los resultados de predicción del valor de la variable "dur_mat" se han elaborado las tablas 34, 35, 36, 37, 38 y 39, como se ha mencionado anteriormente los algoritmos *Kernel Ridge* y *Support Vector Regression (RBF)* son los de mejor rendimiento y en este apartado son los que mayormente se aproximan al valor real de la variable antes mencionada.

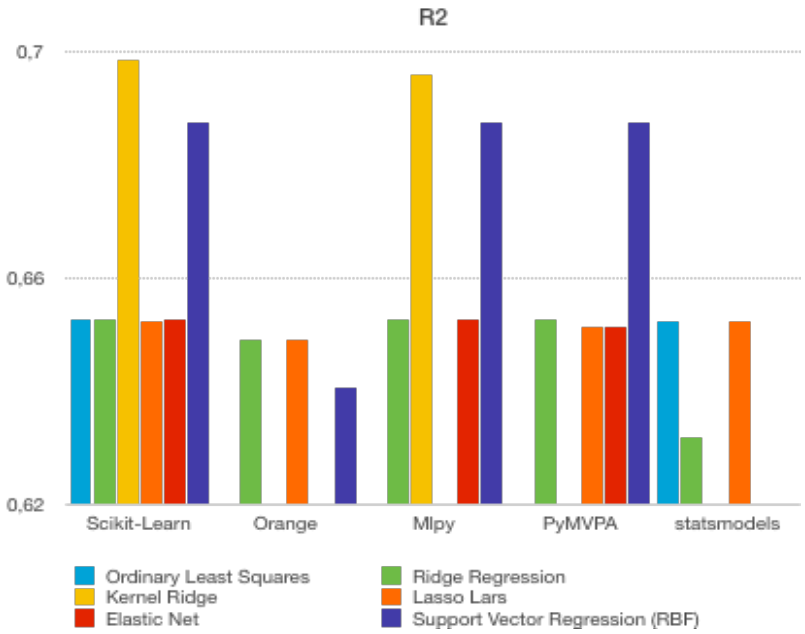

Figura 37. Puntaje R2 de algoritmos de regresión agrupados por librería

Elaboración. El Autor.

[41] R2: determina la calidad del modelo para replicar los resultados.
[42] MSE: representa el error del modelo elevado al cuadrado.

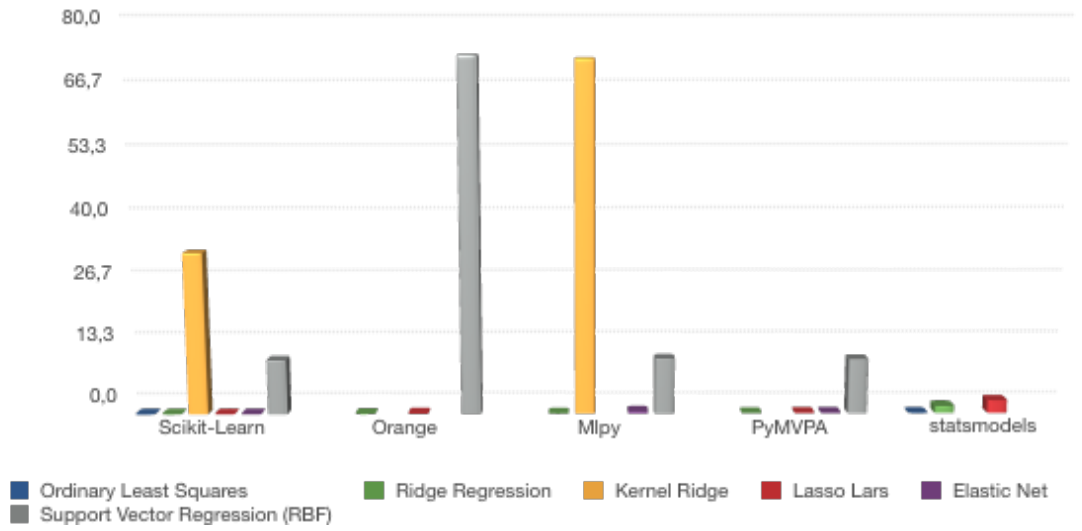

Figura 38. Tiempo de entrenamiento mas predicción de algoritmos de regresión

Elaboración. El Autor.

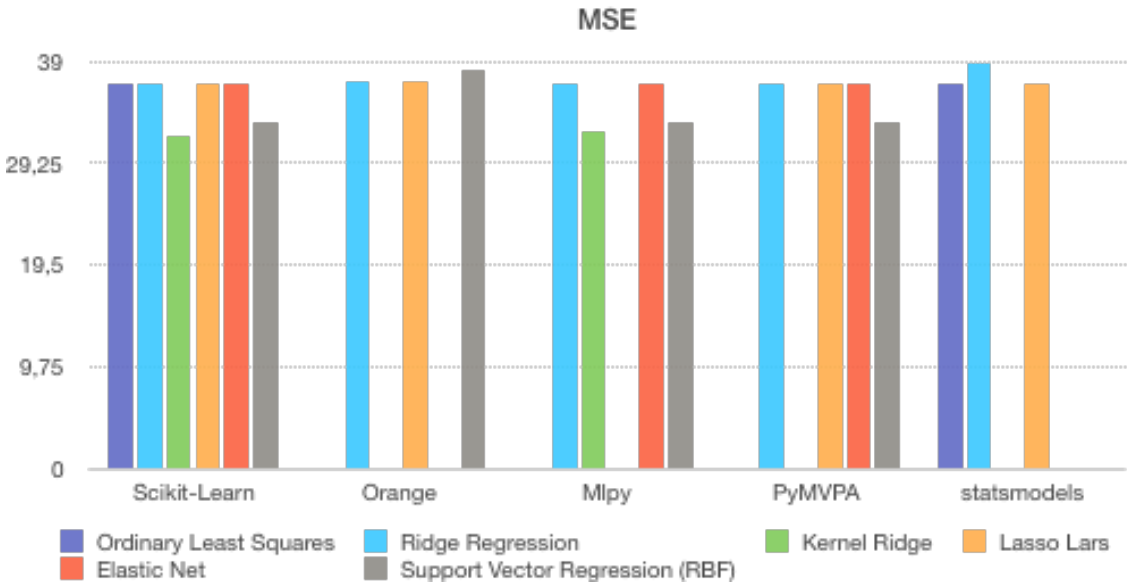

Figura 39. Puntaje MSE de algoritmos de regresión agrupados por librería

Elaboración. El Autor.

De la tabla 33, se obtiene las tablas 40, 41 y 42 (extraída de la tabla 33) para mostrar los mejores resultados en nuestro problema en cuanto a tiempo, R2 y MSE por cada algoritmo de regresión.

Tabla 40: Mejores puntajes R2 en el experimento de regresión

Pos.	Librería	Algoritmo	R2
3	Scikit-Learn	Ordinary Least Squares	0,6527
3	Scikit-Learn, Mlpy, PyMVPA	Ridge Regression	0,6527
1	**Scikit-Learn**	**Kernel Ridge**	**0,6985**
4	Statsmodels	Lasso Lars	0,6524
3	Scikit-Learn	Elastic Net	0,6527
2	PyMVPA	Support Vector Regression (RBF)	0,6875

Elaboración. El Autor.

Tabla 41: Mejores puntajes MSE en el experimento de regresión

Pos.	Librería	Algoritmo	MSE
3	Scikit-Learn	Ordinary Least Squares	36,7666
3	Scikit-Learn, Mlpy, PyMVPA	Ridge Regression	36,7666
1	**Scikit-Learn**	**Kernel Ridge**	**31,9107**
5	Statsmodels	Lasso Lars	36,7974
4	Scikit-Learn	Elastic Net	36,7689
2	PyMVPA	Support Vector Regression (RBF)	33,0766

Elaboración. El Autor.

Tabla 42: Mejores tiempos en el experimento de regresión.

Pos.	Librería	Algoritmo	Entrenamiento + predicción
			Tiempo(seg)
3	Statsmodels	Ordinary Least Squares	0,010
1	**Mlpy**	**Ridge Regression**	**0,003**
6	Scikit-Learn	Kernel Ridge	33,130
4	Scikit-Learn	Lasso Lars	0,038
2	Scikit-Learn	Elastic Net	0,010
5	PyMVPA	Support Vector Regression (RBF)	11,120

Elaboración. El Autor.

La tabla 40 muestra los mejores puntajes de R2 siendo el algoritmo *Kernel Ridge* de la librería Scikit-Learn el de mejor resultado, con un puntaje de R2: 0.6985.

El segundo mejor puntaje de R2 es para la librería PyMVPA con el algoritmo *Support Vector Regression (RBF)* con un puntaje de R2: 0,6875.

En el tercer mejor puntaje de R2 se encuentran 3 algoritmos con el mismo puntaje así; la librería Scikit-Learn, con los algoritmos *Ordinary Least Squares, Elastic Net* y *Ridge Regression,* y las librerías Mlpy, PyMVPA con el algoritmo *Ridge Regression* todas estas con un puntaje de R2: 0,6527.

Finalmente el cuarto mejor puntaje de R2 es para la librería Statsmodels con el algoritmo *Lasso Lars* con R2: 0,6524

La tabla 41 en cambio muestra los mejores puntajes de MSE siendo el algoritmo *Kernel Ridge* de la librería Scikit-Learn el de mejor resultado, con un puntaje de MSE: 31.9107.

El segundo mejor puntaje de MSE es para la librería PyMVPA con el algoritmo *Support Vector Regression (RBF)* con un puntaje de MSE: 33,0766.

En cuanto al tercer mejor puntaje de MSE existen dos algoritmos de la librería Scikit-Learn con el mismo puntaje *Ordinary Least Squares* y *Ridge Regression* para este ultimo al igual que R2 lo comparte con las librerías Mlpy, PyMVPA todos estos algoritmos con un puntaje de MSE: 36,7666.

El cuarto mejor puntaje de MSE es para la librería Scikit-Learn con el algoritmo *Elastic Net* con un puntaje de MSE: 36,7689.

Finalmente el quinto mejor puntaje de MSE es para la librería Statsmodels con el algoritmo *Lasso Lars* con MSE: 36,7974.

Por otro lado en cuanto a las mejores tiempos el mejor resultado es para la librería Mlpy con el algoritmo *Ridge Regression* con un tiempo de 0.003 segundos. En este apartado también se destaca la librería Scikit-Learn ya que en 4 algoritmos obtiene los mejores resultados, en la tabla 42 se puede observar todos los mejores puntajes en cuanto a tiempo.

Las siguientes graficas muestran de manera visual los resultados antes mencionados. En la grafica 40 menos es mejor y en la grafica 41 mas es mejor.

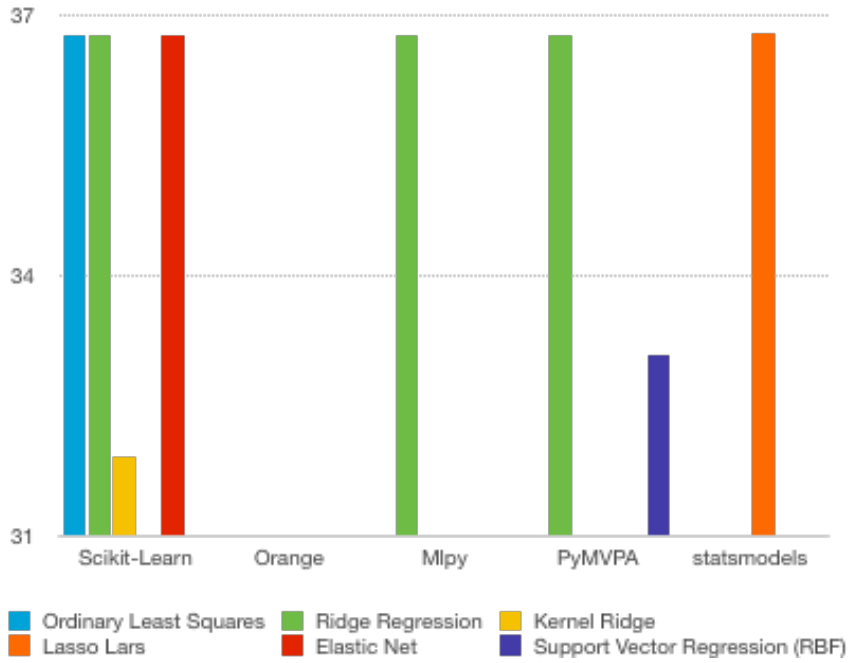

Figura 40. Mejores puntaje MSE de algoritmos de regresión

Elaboración. El Autor.

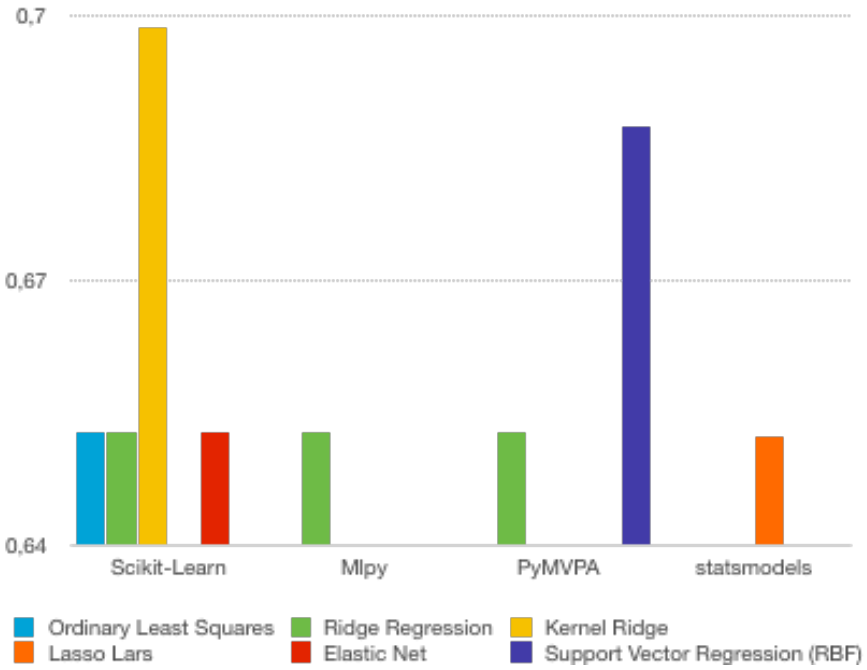

Figura 41. Mejores puntajes R2 de algoritmos de regresión

Elaboración. El Autor.

CONCLUSIONES

Al finalizar el presente trabajo de fin de titulación se puede concluir que:

- Como resultado de la investigación se presentan algunas de las técnicas mas importantes de aprendizaje automático, las cuales representan una base teórica de referencia, además a través de esto se identificó las mas relacionadas con el tipo de información que en este trabajo se utilizó, puesto que cada técnica es más adecuada en algunas situaciones que en otras.

- Mediante la herramienta OpenRefine se pudo ejecutar procedimientos de automatización; así a partir de los datos originales se realizó la transformación de datos de tipo texto a numérico, este proceso sirvió para generar nuevas características, mejorado la predicción de la variable dependiente así como también el tiempo de entrenamiento y predicción de los algoritmos. Además este paso simplificó el proceso de lectura de datos ya que muchas librerías trabajan solamente sobre este tipo de datos y no aceptan caracteres por lo que se podría aplicar a cualquier librería de aprendizaje automático.

- En el proceso de preparación de datos previo a la implementación de los algoritmos, se aplicó técnicas para la selección de atributos mas relacionadas en el problema de clasificación para la variable “estado” y en el caso de regresión para la variable “dur_mat”. Estos nuevos atributos sirvieron para entrenar de mejor manera a los diferentes algoritmos seleccionados. En concreto se aplicó el calculo de *chi2* para clasificación y el calculo de *f_regression* para regresión.

- Se realizó la experimentación dividendo el mismo conjunto de entrenamiento (80% de los datos) y pruebas (20% de los datos) para cada una de las librerías con el fin de tener condiciones mas equilibradas al momento de contrastar los resultados.

- En el problema de clasificación y en base a los parámetros de evaluación (Exactitud, Error, Matriz de Confusión) se puede decir que para nuestro problema el algoritmo con mayor precisión fue Classification Tree de la librería Scikit-Learn que obtuvo un puntaje de Exactitud: 0.8239 y Error: 0.1761.

- En el problema de regresión y en base a los parámetros de evaluación (R2, MSE) se puede decir que para nuestro problema el algoritmo con mayor precisión fue Kernel Ridge de la librería Scikit-Learn con un puntaje de R2=0.6985 y MSE=31.9107.

- Se obtuvo los algoritmos con mejor tiempo de entrenamiento mas predicción de la variable dependiente para nuestros problemas, así en el enfoque de clasificación el algoritmo de menor tiempo fue Gaussian Naive Bayes de la librería Scikit-Learn con 0.02 segundos. En cambio que en el enfoque de regresión el algoritmo de menor tiempo fue Ridge Regression de la librería Mlpy con un tiempo de 0.003 segundos.

- Se obtuvo a la librería Scikit-Learn como la de mejor rendimiento en general para nuestros experimentos de clasificación y regresión ya que redujo los tiempos de entrenamiento mas predicción de los modelos, así como también demostró ser mucho mas precisa al momento de predecir la variable dependiente.

- La Librería PyMVPA puede ser una buena opción para la implementación de Maquinas de Vectores de Soporte ya que en nuestros experimentos obtuvo los mejores resultados de precisión sobre este modelo.

- Las librerías utilizadas en este trabajo que no alcanzaron los mejores puntajes, pueden resultar útil para futuros estudios con diferentes datos y llegar a obtener resultados más convincentes que para nuestros experimentos no lo fueron necesariamente.

- Como resultado del proceso de comparación de rendimiento de librerías de aprendizaje automático en Python, se presentan una serie de pasos los cuales sirven como una guía en la aplicación de algoritmos de clasificación y regresión. Como resultado de esto algunos puntos importantes para una buena implementación bajo los dos enfoques antes mencionados.

RECOMENDACIONES

Al finalizar con el presente trabajo de fin de titulación se recomienda:

- Realizar previamente a la aplicación de los algoritmos un análisis estadístico de los datos ya que esto permitirá saber como están comportándose los mismos, además mediante este proceso se puede conocer si el problema es linealmente separable y por ende poder elegir de mejor manera los algoritmos a implementar. Ya que en este trabajo los dos problemas no han sido linealmente separables y los modelos no lineales precisamente han sido los de mejor resultado.

- Considerar el uso de alguna herramienta que permita ejecutar procedimientos de automatización para el proceso de preparación y transformación del conjunto de datos. En el presente trabajo se utilizó la herramienta OpenRefine que fue de gran ayuda en este apartado.

- Tener en cuenta antes de comenzar con el proceso de implementación qué es lo que se quiere obtener y cuáles son los datos que permitirán realizar esta tarea, una ves realizado esto utilizar el mayor numero de datos posibles ya que los algoritmos de aprendizaje automático aprenden y descubren mientras mas datos disponibles tengan.

- A partir de los errores de clasificación y regresión se podría realizar un análisis para ver en que datos el algoritmo tiene problemas de predicción. Una vez conocidas estas causas, podremos detectar los puntos débiles, ya sea en los datos, o en los modelos de predicción.

BIBLIOGRAFÍA

Abraham, A., Pedregosa, F., Eickenberg, M., Gervais, P., Mueller, A., Kossaifi, J., … Varoquaux, G. (2014). Machine learning for neuroimaging with scikit-learn. *Frontiers in Neuroinformatics*, *8*(February), 14. http://doi.org/10.3389/fninf.2014.00014

Accord. (2015). Accord web site. Obtenido de http://accord-framework.net/intro.html

Albanese, D., Visintainer, R., Merler, S., Riccadonna, S., Jurman, G., & Furlanello, C. (2012). mlpy: Machine Learning PythonXXXX. *arXiv*, (1202.6548). Retrieved from http://mlpy.sourceforge.net/

Alpaydın, E. (2014). *Introduction to machine learning*. *Methods in Molecular Biology* (Vol. 1107). http://doi.org/10.1007/978-1-62703-748-8-7

Azure. (2015). Azure web site Obtenido de https://azure.microsoft.com/es-es/documentation/articles/machine-learning-what-is-machine-learning/

Bell, J. (2014). *Hands-On for Developers and Technical Professionals*. *Machine Learning: Hands-On for Developers and Technical Professionals*.

Benavides Cañon, P. A., & Rodriguez Correa, S. (2007). PLN PROCESAMIENTO DEL LENGUAJE NATURAL EN LA RECUPERACI{Ó}N DE INFORMACI{Ó}N.

Benítez, R., Escudero, G., & Kanaan, S. (2013). Inteligencia Artificial Avanzada, 1–214.

ConvNetJS. (2015). ConvNetJS web site. Obtenido de http://cs.stanford.edu/people/karpathy/convnetjs/

Cuesta, H. (2010). *Practical Data Analysis : An Example*. *Practical Data Analysis for Designed Experiments*. http://doi.org/10.1007/978-1-84882-260-3

Daume, H. (2012). A course in machine learning, 189.

de Andrés Suárez, J. (2000). Técnicas de Inteligencia Artificial aplicadas al análisis de la solvencia empresarial. *Documentos de Trabajo (Universidad de Oviedo. Facultad de Ciencias Econ{ó}micas)*, (206), 1–31.

Demšar, J., Curk, T., Erjavec, A., Gorup, Č., Hočevar, T., Milutinovič, M., … Zupan, B. (2013). Orange: data mining toolbox in python. *The Journal of Machine Learning Research*, *14*(1), 2349–2353. Retrieved from http://dl.acm.org/citation.cfm?id=2567709.2567736

Demšar, J., & Zupan, B. (2012). Orange: Data Mining Fruitful and Fun. *IInformacijska Družba- IS 2012*.

Díez, R. P., Gómez, A. G., & de Abajo Martínez, N. (2001). *Introducción a la inteligencia artificial: sistemas expertos, redes neuronales artificiales y computación evolutiva*. Universidad de Oviedo.

Drew, C., & White, john myles. (n.d.). *Machine learning for hackers*.

GoLearn. (2015). GoLearn web site. Obtenido de https://github.com/sjwhitworth/golearn

Gramajo, E., García-Martínez, R., Rossi, B., Claverie, E., Britos, P., & Totongi, A. (1999). Una visión global del aprendizaje automático. *Revista Del Instituto Tecnol{ó}gico de Buenos Aires*, *22*, 67–75.

Hackeling, G. (2014). *Mastering Machine Learning with scikit-learn*. Retrieved from http://books.google.com/books?id=fZQeBQAAQBAJ&pgis=1

Hanke, M., Halchenko, Y. O., Haxby, J. V., & Pollmann, S. (2010). Statistical learning analysis in neuroscience: Aiming for transparency. *Frontiers in Neuroscience*, *4*(MAY), 38–43. http://doi.org/10.3389/neuro.01.007.2010

Hanke, M., Halchenko, Y. O., Sederberg, P. B., José, S., Haxby, J. V, & Pollmann, S. (2009). NIH Public Access, *7*(1), 37–53. http://doi.org/10.1007/s12021-008-9041-y.PyMVPA

Hanke, M., Halchenko, Y. O., Sederberg, P. B., Olivetti, E., Fründ, I., Garcia, S., & Claude, U. (2009). PyMVPA : a unifying approach to the analysis of neuroscientifi c data. *Neuroinformatics*, *3*(February), 1–13. http://doi.org/10.3389/neuro.11.003.2009

Harrington, P. (2012). *Machine Learning in Action*. http://doi.org/10.1007/978-0-387-77242-4

Jović, A., Brkić, K., & Bogunović, N. (2014). An overview of free software tools for general data mining. *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2014 - Proceedings*, 1112–1117. http://doi.org/10.1109/MIPRO.2014.6859735

Kim, Y.-N., Yu, H.-Y., & Kim, M.-H. (2014). ID3 algorithm based object discrimination for multi object tracking. In *2014 14th International Symposium on Communications and Information Technologies (ISCIT)* (pp. 535–539). IEEE. http://doi.org/10.1109/ISCIT.2014.7011971

Mallet. (201H5). Mallet web site. Obtenido de http://mallet.cs.umass.edu/

Marsland, S. (2015). *Machine Learning: An Algorithmic Perspective*.

Mathematica. (2015). Mathematica web site Obtenido de http://www.wolfram.com/mathematica/new-in-10/highly-automated-machine-learning/

McKinney, W., Perktold, J., & Seabold, S. (2011). Time Series Analysis in Python with statsmodels. *Jarrodmillman.Com*, (July), 96–102. Retrieved from http://jarrodmillman.com/scipy2011-reviews/statsmodels/rokem.pdf

Mitchell, T. M. (1997). *Machine Learning*. *Annual Review Of Computer Science*. http://doi.org/10.1145/242224.242229

Mlpack. (2015). Mlpack web site. Obtenido de http://www.mlpack.org/about.html

Mlpy. (2016). Mlpy Documentación oficial Obtenido de http://mlpy.sourceforge.net/docs

Montoya, R. A., Jairo, J., Chávez, S., Jesús, J. De, & Mora, V. (2013). Aplicación del aprendizaje automático con árboles de decisión en el diagnóstico médico Application of machine learning with decision trees in medical diagnosis, *10*, 63–72.

Moriana Becerra, J. (2015). Desarrollo de software estad{í}stico en Python para el análisis de material.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Nilsson, N. J. (2005). INTRODUCTION TO MACHINE LEARNING AN EARLY DRAFT OF A PROPOSED TEXTBOOK Department of Computer Science.

Orange. (2016). Orange Documentación oficial Obtenido de

http://docs.orange.biolab.si/3/data-mining-library/index.html

PyML. (2016). PyML Documentación oficial Obtenido de http://pyml.sourceforge.net/tutorial.html

PyMVPA. (2016). PyMVPA Documentación oficial Obtenido de http://www.pymvpa.org/modref.html

Romero, J. J., Dafonte, C., Gòmez, Á., & Penousal, F. J. (2007). *Inteligencia Artificial Y Computación Avanzada*. *Inteligencia Artificial …*. Retrieved from http://fmachado.dei.uc.pt/wp-content/papercite-data/pdf/ms07.pdf#page=9

Russell, S., & Norvig, P. (2004). *Inteligencia Artificial. Un enfoque moderno. 2da Edición*. http://doi.org/M-26913-2004

Salton , G. and Mc Gill, M.J. Introduction to Modern Information Retrieval. New York: Mc Graw-Hill Computer Series, 1983.

Seabold, S., & Perktold, J. (2010). Statsmodels: econometric and statistical modeling with python. *… of the 9th Python in Science Conference*, (Scipy), 57–61. Retrieved from https://projects.scipy.org/proceedings/scipy2010/pdfs/seabold.pdf

Shogun. (2005). Shogun web site. Obtenido de http://www.shogun-toolbox.org/page/about/project_description

Spark. (2015). Spark web site. Obtenido de http://spark.apache.org/mllib/

Velez-Langs, O., & Santos, C. (2006). Sistemas Recomendadores: Un enfoque desde los algoritmos genéticos. *Ind. Data*, *9*(1), 23–31.

Vila, E. M. S., & Penín, M. L. (2007). Monografía: Técnicas de la inteligencia artificial aplicadas a la educación. *Inteligencia Artificial*, *11*(33), 7–12.

Wilbert, N., Zito, T., Schuppner, R. B., Jedrzejewski-Szmek, Z., Wiskott, L., & Berkes, P. (2013). Building extensible frameworks for data processing: The case of MDP, Modular toolkit for Data Processing. *Journal of Computational Science*, *4*(5), 345–351. http://doi.org/10.1016/j.jocs.2011.10.005

Weka. (2015). Weka web site. Obtenido de http://www.cs.waikato.ac.nz/ml/weka/

Zito, T., Wilbert, N., Wiskott, L., & Berkes, P. (2008). Modular Toolkit for Data Processing (MDP): A Python Data Processing Framework. *Frontiers in Neuroinformatics*, *2*(January), 8. http://doi.org/10.3389/neuro.11.008.2008

ANEXOS

ANEXO 1. Instalación de Jupyter Notebook

Jupyter Notebook es una aplicación web interactiva que contiene entradas y salidas las cuales permiten combinar la ejecución de código, texto enriquecido, matemáticas, graficas y multimedia. Cada notebook es un documento JSON que pueden ser exportados a varios formatos de archivo como HTML, Python, LaTeX, PDF, etc.

Para la instalación de Jupyter Notebook es recomendable hacerlo desde un entorno virtual a continuación se presenta todos los paso para la instalación de la herramienta:

1. Primeramente vamos a instalar virtualenv para ello es necesario que previamente se tenga instalado Python en caso de así serlo se escribe el siguiente código:
    - `pip install virtualenv`
2. Creamos la carpeta donde se alojara nuestro entorno:
    - `mkdir mi_entorno`
    - `cd mi_entorno`
3. Una ves dentro de la carpeta para crear nuestro entorno escribimos lo siguiente:
    - `virtualenv myproject`
4. Ahora solo hace falta activarlo con la siguiente línea de código:
    - `source myproject/bin/actívate`
5. Finalmente ya podemos instalar Jupyter Notebook dentro de nuestro entorno para ello escribimos:
    - `pip install jupyter`

Para la creación de un proyecto notebook se escribe: `ipython notebook` y luego se abrirá la siguiente ventana

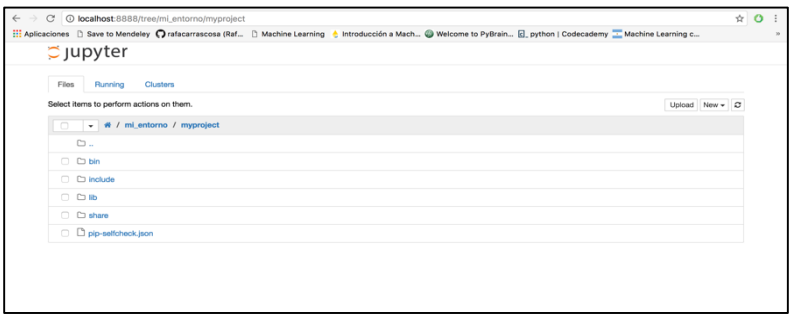

Figura 42. Captura de pantalla de la herramienta Jupyter Notebook

Elaboración. El Autor.

Se debe situar en el icono 

luego en Python 2 y ya se puede comenzar a escribir nuestro código

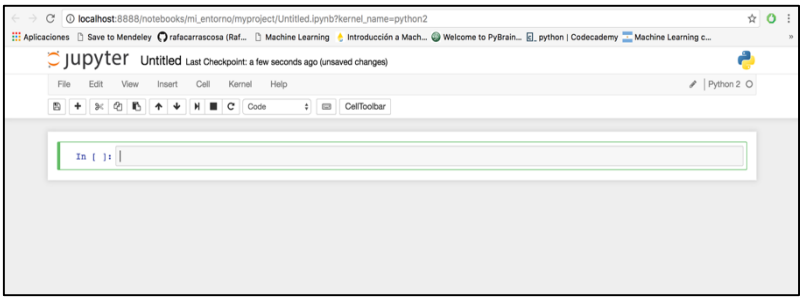

Figura 43. Captura de pantalla de un Notebook

Elaboración. El Autor.

ANEXO 2. Transformación de archivos de formato SPSS a CSV

Para realizar esta transformación se utilizo el lenguaje de programación R. Para la instalación de la consola R en Mac OS X solo hace falta ingresar a la pagina https://cran.r-project.org/bin/macosx/ descargar el paquete “R-3.2.3.pkg” o la versión que se encuentre disponible y seguir los pasos básicos de instalación.

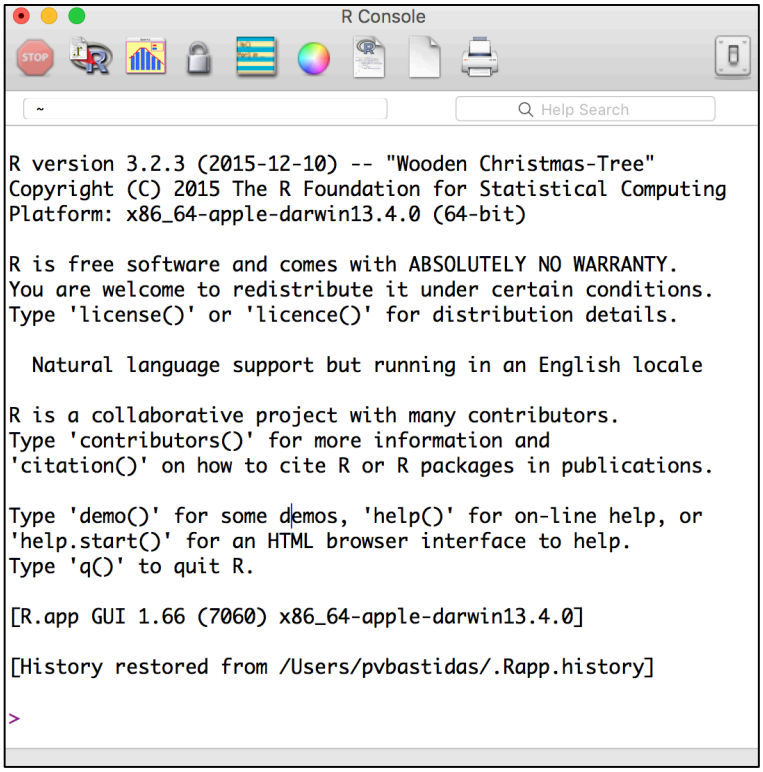

Figura 44. Captura de pantalla de la consola R en Mac OS X

Elaboración. El Autor.

Una ves en la consola solo hace falta escribir las siguientes líneas de código:

```
> library(foreign)
> mydata = read.spss("ruta_del_archivo",to.data.frame=TRUE)
> write.csv(mydata, file='nombre_archivo.csv')
```

ANEXO 3. Código parte de la preparación de datos para el problema de clasificación

CODIGO

```
import pandas as pd
import numpy as np
import sklearn
import warnings
import seaborn as sns
import matplotlib.pyplot as plt

from pandas import DataFrame as df
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectPercentile
from __future__ import division
from collections import Counter
warnings.filterwarnings("ignore")
%matplotlib inline

sns.set(style="white", color_codes=True)
dataset = pd.read_csv("../datos/plotclasificacion/dataset.csv")
train =pd.read_csv("../datos/plotclasificacion/train.csv")
test =pd.read_csv("../datos/plotclasificacion/test.csv")
testreducido=pd.read_csv("../datos/plotclasificacion/testreducido.csv")

# GRAFICA DISTRIBUCION DE MATRIMONIOS Y DIVORCION EN LA PARTICION
COMPLETA
sns.countplot(x="estado", data=dataset);

#CALCULO DE CHI CUADRADO PARA SELECCION DE VARIABLES
dftr = pd.read_csv("../datos/dataset_clasificacion.csv")
values=['prov_insc','mcap_bie','edad_hom','niv_insth','p_etnica_hom','are
a_hom',
        'edad_muj','niv_instm','p_etnica_muj','area_muj','estado']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['estado']
Selector_f = SelectPercentile(chi2)
Selector_f.fit(X,y)
for n,s in zip(dftr.columns.values,Selector_f.scores_):
 print "F-score: %3.2f\t para variable %s "% (s,n)

# GRAFICA DISTRIBUCION DE LA VARIABLE prov_insc
sns.countplot(x="prov_insc", hue="estado", data=dataset);

# GRAFICA DISTRIBUCION DE LA VARIABLE mcap_bie
sns.countplot(x="mcap_bie", hue="estado", data=dataset);

#GRAFICA DISTRIBUCION DE DENSIDAD DE LAS VARIABLES edad_hom y edad_muj
sns.FacetGrid(dataset, hue="estado", size=6) \
   .map(sns.kdeplot, "edad_hom") \
   .add_legend()
sns.FacetGrid(dataset, hue="estado", size=6) \
   .map(sns.kdeplot, "edad_muj") \
   .add_legend()

# GRAFICA DIAGRAMA DE DISPERSION DE LAS VARIABLES edad_muj y edad_hom
sns.FacetGrid(dataset, hue="estado", size=5) \
   .map(plt.scatter, "edad_hom", "edad_muj") \
```

```
   .add_legend()

# GRAFICA DISTRIBUCION DE LAS VARIABLES p_etnica_hom y p_etnica_muj
sns.countplot(x="p_etnica_hom", hue="estado", data=dataset)
sns.countplot(x="p_etnica_muj", hue="estado", data=dataset)

# GRAFICA DISTRIBUCION DE LAS VARIABLES niv_insth y niv_instm
sns.countplot(x="niv_insth", hue="estado", data=dataset);
sns.FacetGrid(dataset, hue="estado", size=6) \
   .map(sns.kdeplot, "niv_insth") \
   .add_legend()
sns.countplot(x="niv_instm", hue="estado", data=dataset);
sns.FacetGrid(dataset, hue="estado", size=6) \
   .map(sns.kdeplot, "niv_instm") \
   .add_legend()

#GRAFICA DISTRIBUCION DE LAS VARIABLES area_hom y area_muj
sns.countplot(x="area_hom", hue="estado", data=dataset);
sns.countplot(x="area_muj", hue="estado", data=dataset);

# RESUMEN ESTADISTICO DE LAS VARIABLES INDEPENDIENTES
dftr = pd.read_csv("../datos/dataset_clasificacion.csv")
values=['prov_insc','mcap_bie','edad_hom','niv_insth','p_etnica_hom','are
a_hom',
        'edad_muj','niv_instm','p_etnica_muj','area_muj','estado']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['estado']
X=np.float64(X)
y=np.float64(y)

def mean(x):
    return sum(x) / float(len(x))
def variance(x):
    """assumes x has at least two elements"""
    n = len(x)
    deviations = de_mean(x)
    return sum_of_squares(deviations) / (n - 1)

def standard_deviation(x):
    return np.sqrt(variance(x))

def covariance(x, y):
    n = len(x)
    return dot(de_mean(x), de_mean(y)) / (n - 1)

def correlation(x, y):
    stdev_x = standard_deviation(x)
    stdev_y = standard_deviation(y)
    if stdev_x > 0 and stdev_y > 0:
        return covariance(x, y) / stdev_x / stdev_y
    else:
        return 0 # if no variation, correlation is zero
def sum_of_squares(v):
    return dot(v, v)

def de_mean(x):
 x_bar = mean(x)
```

```
 return [x_i - x_bar for x_i in x]

def dot(v, w):
    return sum(v_i * w_i
               for v_i, w_i in zip(v, w))

def correlation(x, y):
    stdev_x = standard_deviation(x)
    stdev_y = standard_deviation(y)
    return covariance(x, y) / stdev_x / stdev_y

print "Min", df.min(pd.DataFrame(X))
print "Max", df.max(pd.DataFrame(X))
print "Mean", mean(X)
print "SD", standard_deviation(X)
print "Class Correlation", correlation(X, y)

# GRAFICAS DIAGRAMA DE DISPERSION DE LAS 3 PRIMERAS POSICIONES MOSTRADAS
EN LA LA TABLA 10
sns.swarmplot(x="niv_instm", y="edad_muj",
hue="estado",data=testreducido);
sns.swarmplot(x="p_etnica_muj", y="edad_muj",
hue="estado",data=testreducido);
sns.swarmplot(x="p_etnica_muj", y="edad_hom",
hue="estado",data=testreducido);
```

ANEXO 4. Código parte de la preparación de datos para el problema de regresión

CODIGO

```
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
import matplotlib.pyplot as plt
import warnings

from pandas import DataFrame as df
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import f_regression
from __future__ import division
from collections import Counter

%matplotlib inline
warnings.filterwarnings("ignore")
sns.set(style="white", color_codes=True)
completo = pd.read_csv("../datos/dataset_regresion.csv")
train =pd.read_csv("../datos/train_regresion.csv")
test =pd.read_csv("../datos/test_regresion.csv")

#DIVISION Y DISTRIBUCION DEL ARCHIVO CORRESPONDIENTE A REGRESION
data = pd.read_csv("../datos/dataset_regresion.csv", index_col="Id")
train = pd.read_csv("../datos/train_regresion.csv")
test = pd.read_csv("../datos/test_regresion.csv")

def binning(col, cut_points, labels=None):
  minval = col.min()
  maxval = col.max()
  break_points = [minval] + cut_points + [maxval]
  if not labels:
    labels = range(len(cut_points)+1)
  colBin =
pd.cut(col,bins=break_points,labels=labels,include_lowest=True)
  return colBin

cut_points = [4,25,50]
labels = ["menor de 5","de 5 a 25","de 26 a 50","mayor de 50"]
data["aniosmat"] = binning(data["dur_mat"], cut_points, labels)
train["aniosmat"] = binning(train["dur_mat"], cut_points, labels)
test["aniosmat"] = binning(test["dur_mat"], cut_points, labels)

print "Años de matrimonio en el archivo completo:"
print pd.value_counts(data["aniosmat"], sort=False)
print "Años de matrimonio en la particion de entrenamiento:"
print pd.value_counts(train["aniosmat"], sort=False)
print "Años de matrimonio en la particion de test:"
print pd.value_counts(test["aniosmat"], sort=False)

# CALCULO DE f_regression PARA SELECCION DE VARIABLES
dftr = pd.read_csv("../datos/dataset_regresion.csv")
values=['cau_div','nac_hom','edad_hom','hijos_hom','niv_insth',
       'nac_muj','edad_muj','hijos_muj','niv_instm','dur_mat']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['dur_mat']
```

```
dftrain.describe()
Selector_f = SelectPercentile(f_regression)
Selector_f.fit(X,y)
for n,s in zip(dftr.columns.values,Selector_f.scores_):
 print "F-score: %3.2f\t para variable %s "% (s,n)

# GRAFICA DISTRIBUCION DE LA VARIABLE cau_div
sns.countplot(x="cau_div", data=completo);

#GRAFICA DIAGRAMA DE DISPERSION DE LAS VARIABLES edad_muj y edad_hom
JUNTO CON SUS HISTOGRAMAS
sns.jointplot(x="edad_hom", y="edad_muj", data=completo, size=5)

# GRAFICA DISTRIBUCION DE LAS VARIABLES hijos_hom e hijos_muj
sns.countplot(x="hijos_hom", data=completo);
sns.countplot(x="hijos_muj", data=completo);

# GRAFICA DISTRIBUCION DE LAS VARIABLES niv_insth y niv_instm
sns.countplot(x="niv_insth", data=completo);
sns.countplot(x="niv_instm", data=completo);

# RESUMEN ESTADISTICO DE LAS VARIABLES INDEPENDIENTES
dftr = pd.read_csv("../datos/dataset_regresion.csv")
values=['cau_div','nac_hom','edad_hom','hijos_hom','niv_insth',
       'nac_muj','edad_muj','hijos_muj','niv_instm','dur_mat']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['dur_mat']
X=np.float64(X)
y=np.float64(y)

def mean(x):
    return sum(x) / float(len(x))
def variance(x):
    n = len(x)
    deviations = de_mean(x)
    return sum_of_squares(deviations) / (n - 1)

def standard_deviation(x):
    return np.sqrt(variance(x))

def covariance(x, y):
    n = len(x)
    return dot(de_mean(x), de_mean(y)) / (n - 1)

def correlation(x, y):
    stdev_x = standard_deviation(x)
    stdev_y = standard_deviation(y)
    if stdev_x > 0 and stdev_y > 0:
        return covariance(x, y) / stdev_x / stdev_y
    else:
        return 0 # if no variation, correlation is zero

def dot(v, w):
    return sum(v_i * w_i
               for v_i, w_i in zip(v, w))

def sum_of_squares(v):
```

```
    return dot(v, v)

def de_mean(x):
    x_bar = mean(x)
    return [x_i - x_bar for x_i in x]

def correlation(x, y):
    stdev_x = standard_deviation(x)
    stdev_y = standard_deviation(y)
    return covariance(x, y) / stdev_x / stdev_y

print "Min", df.min(pd.DataFrame(X))
print "Max", df.max(pd.DataFrame(X))
print "Mean", mean(X)
print "SD", standard_deviation(X)
print "Class Correlation", correlation(X, y)
```

ANEXO 5. Código para el experimento de clasificación de la librería Scikit-Learn

CODIGO

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime
%matplotlib inline

# ASIGNACION DE DATOS PARA EL ENTRENAMIENTO
dftr = pd.read_csv("../datos/train_clasificacion.csv")
values=['prov_insc','mcap_bie','edad_hom','niv_insth','p_etnica_hom','are
a_hom',
        'edad_muj','niv_instm','p_etnica_muj','area_muj','estado']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['estado']
X_train=X
y_train=y

# ASIGNACION DE DATOS PARA EL TEST
dftr = pd.read_csv("../datos/test_clasificacion.csv")
values=['prov_insc','mcap_bie','edad_hom','niv_insth','p_etnica_hom','are
a_hom',
        'edad_muj','niv_instm','p_etnica_muj','area_muj','estado']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['estado']
X_test=X
y_test=y
```

LOGISTIC REGRESSION

```
import sklearn
from sklearn.linear_model import LogisticRegression
start = datetime.now()
clf = LogisticRegression(C = 1,          # Parámetro de regularización
                         penalty = 'l2') # Tipo de regularización l1 es
lasso, l2 es ridge
clf.fit(X_train, y_train)
expected = y_test
predicted = clf.predict(X_test)
timeSciLe=datetime.now() - start
timeSciLe=timeSciLe.seconds + (timeSciLe.microseconds * 1e-6)

print ('Time: %s' % timeSciLe)
scoreSciLe = clf.score(X_test,y_test)
print ('Accuracy: %.4f' % scoreSciLe)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

NAIVE BAYES

```
import sklearn
from sklearn.naive_bayes import GaussianNB
start = datetime.now()
clf = GaussianNB()
clf.fit(X_train, y_train)
expected = y_test
predicted = clf.predict(X_test)
timeSciLe=datetime.now() - start
timeSciLe=timeSciLe.seconds + (timeSciLe.microseconds * 1e-6)

print ('Time: %s' % timeSciLe)
scoreSciLe = clf.score(X_test,y_test)
print ('Accuracy: %.4f' % scoreSciLe)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

K-NEAREST NEIGHBORS

```
import sklearn
from sklearn.neighbors import KNeighborsClassifier
start = datetime.now()
clf = KNeighborsClassifier(n_neighbors=9,      # Numero de vecinos a
tener en cuenta
                                    p=2)    # 1:manhattan, 2:euclidea
clf.fit(X_train, y_train)
expected = y_test
predicted = clf.predict(X_test)
timeSciLe=datetime.now() - start
timeSciLe=timeSciLe.seconds + (timeSciLe.microseconds * 1e-6)
print ('Time: %s' % timeSciLe)
scoreSciLe = clf.score(X_test,y_test)
print ('Accuracy: %.4f' % scoreSciLe)

print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

SUPPORT VECTOR MACHINE (KERNEL LINEAR)

```
import sklearn
from sklearn import svm
start = datetime.now()
clf = svm.SVC(C=1,
              kernel='linear')
clf.fit(X_train, y_train)
expected = y_test
predicted = clf.predict(X_test)
timeSciLe=datetime.now() - start
timeSciLe=timeSciLe.seconds + (timeSciLe.microseconds * 1e-6)

print ('Time: %s' % timeSciLe)
scoreSciLe = clf.score(X_test,y_test)
```

```
print ('Accuracy: %.4f' % scoreSciLe)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

SUPPORT VECTOR MACHINE (KERNEL RBF)

```
import sklearn
from sklearn.svm import SVC
start = datetime.now()
clf = SVC( kernel='rbf',      # Kernel gaussiano (función de base radial)
           gamma=0.1,   )      # Grado del kernel
           #probability=False) # Salida no probabilistica
clf.fit(X_train, y_train)
expected = y_test
predicted = clf.predict(X_test)
timeSciLe=datetime.now() - start
timeSciLe=timeSciLe.seconds + (timeSciLe.microseconds * 1e-6)

print ('Time: %s' % timeSciLe)
scoreSciLe = clf.score(X_test,y_test)
print ('Accuracy: %.4f' % scoreSciLe)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

CLASSIFICATION TREE

```
import sklearn
from sklearn.tree import DecisionTreeClassifier
start = datetime.now()
clf = DecisionTreeClassifier(criterion='entropy', # Criterio de división
                             min_samples_leaf=20); # Muestras por hoja
clf.fit(X_train, y_train)
expected = y_test
predicted = clf.predict(X_test)
timeSciLe=datetime.now() - start
timeSciLe=timeSciLe.seconds + (timeSciLe.microseconds * 1e-6)

print ('Time: %s' % timeSciLe)
scoreSciLe = clf.score(X_test,y_test)
print ('Accuracy: %.4f' % scoreSciLe)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

ANEXO 6. Código para el experimento de clasificación de la librería Orange

CODIGO

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime
%matplotlib inline
```

LOGISTIC REGRESSION

```
import Orange
from Orange.evaluation import testing, scoring
train = Orange.data.Table("../datos/plotclasificacion/train.csv")
test = Orange.data.Table("../datos/plotclasificacion/test.csv")
start = datetime.now()
clf =
Orange.classification.logreg.LibLinearLogRegLearner(train,C=1)# l2
c=1
results = Orange.evaluation.testing.test_on_data([clf], test)
timeOrange=datetime.now() - start
timeOrange=timeOrange.seconds + (timeOrange.microseconds * 1e-6)

print ('Time: %s' % timeOrange)
scoreOrange=scoring.CA(results)[0]
print "ACCURACY: %.4f" % scoreOrange
#print "AUC: %.4f" % scoring.AUC(results)[0]
cm = Orange.evaluation.scoring.confusion_matrices(results)[0]
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
print "\tD \tM \tAll"
print "D\t%i \t%i \t%i \nM\t%i \t%i \t%i \nAll\t%i \t%i \t%i" %
(cm.TN, cm.FP,(cm.TN+ cm.FP), cm.FN,cm.TP,(cm.FN+cm.TP),
(cm.TN+cm.FN),(cm.FP+cm.TP),(cm.TN+ cm.FP+cm.FN+cm.TP) )
```

NAIVE BAYES

```
import Orange
from Orange.evaluation import testing, scoring
train = Orange.data.Table("../datos/plotclasificacion/train.csv")
test = Orange.data.Table("../datos/plotclasificacion/test.csv")
start = datetime.now()
clf = Orange.classification.bayes.NaiveLearner(train)
results = Orange.evaluation.testing.test_on_data([clf], test)
timeOrange=datetime.now() - start
timeOrange=timeOrange.seconds + (timeOrange.microseconds * 1e-6)

print ('Time: %s' % timeOrange)
scoreOrange=scoring.CA(results)[0]
print "ACCURACY: %.4f" % scoreOrange
#print "AUC: %.4f" % scoring.AUC(results)[0]
cm = Orange.evaluation.scoring.confusion_matrices(results)[0]
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
print "\tD \tM \tAll"
```

```
print "D\t%i \t%i \t%i \nM\t%i \t%i \t%i \nAll\t%i \t%i \t%i" %
(cm.TN, cm.FP,(cm.TN+ cm.FP), cm.FN,cm.TP,(cm.FN+cm.TP),
(cm.TN+cm.FN),(cm.FP+cm.TP),(cm.TN+ cm.FP+cm.FN+cm.TP) )
```

K-NEAREST NEIGHBORS

```
import Orange
from Orange.evaluation import testing, scoring
train = Orange.data.Table("../datos/plotclasificacion/train.csv")
test = Orange.data.Table("../datos/plotclasificacion/test.csv")
start = datetime.now()
clf = Orange.classification.knn.kNNLearner(train, k=9)#Defaults to
Euclidean.
results = Orange.evaluation.testing.test_on_data([clf], test)
timeOrange=datetime.now() - start
timeOrange=timeOrange.seconds + (timeOrange.microseconds * 1e-6)

print ('Time: %s' % timeOrange)
scoreOrange=scoring.CA(results)[0]
print "ACCURACY: %.4f" % scoreOrange
#print "AUC: %.4f" % scoring.AUC(results)[0]
cm = Orange.evaluation.scoring.confusion_matrices(results)[0]
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
print "\tD \tM \tAll"
print "D\t%i \t%i \t%i \nM\t%i \t%i \t%i \nAll\t%i \t%i \t%i" %
(cm.TN, cm.FP,(cm.TN+ cm.FP), cm.FN,cm.TP,(cm.FN+cm.TP),
(cm.TN+cm.FN),(cm.FP+cm.TP),(cm.TN+ cm.FP+cm.FN+cm.TP) )
```

SUPPORT VECTOR MACHINE (KERNEL LINEAR)

```
import Orange
from Orange.evaluation import testing, scoring
from Orange.classification import svm
train = Orange.data.Table("../datos/plotclasificacion/train.csv")
test = Orange.data.Table("../datos/plotclasificacion/test.csv")
start = datetime.now()
clf = svm.SVMLearner(train,
                             kernel_type=svm.kernels.Linear,
                             C=1)
results = Orange.evaluation.testing.test_on_data([clf], test)
timeOrange=datetime.now() - start
timeOrange=timeOrange.seconds + (timeOrange.microseconds * 1e-6)

print ('Time: %s' % timeOrange)
scoreOrange=scoring.CA(results)[0]
print "ACCURACY: %.4f" % scoreOrange
#print "AUC: %.4f" % scoring.AUC(results)[0]
cm = Orange.evaluation.scoring.confusion_matrices(results)[0]
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
print "\tD \tM \tAll"
print "D\t%i \t%i \t%i \nM\t%i \t%i \t%i \nAll\t%i \t%i \t%i" %
(cm.TN, cm.FP,(cm.TN+ cm.FP), cm.FN,cm.TP,(cm.FN+cm.TP),
(cm.TN+cm.FN),(cm.FP+cm.TP),(cm.TN+ cm.FP+cm.FN+cm.TP) )
```

SUPPORT VECTOR MACHINE (KERNEL RBF)

```
import Orange
from Orange.evaluation import testing, scoring
from Orange.classification import svm
train = Orange.data.Table("../datos/plotclasificacion/train.csv")
test = Orange.data.Table("../datos/plotclasificacion/test.csv")
start = datetime.now()
clf = svm.SVMLearner(train,
                             kernel_type=svm.kernels.RBF,
                             gamma=0.1)
results = Orange.evaluation.testing.test_on_data([clf], test)
timeOrange=datetime.now() - start
timeOrange=timeOrange.seconds + (timeOrange.microseconds * 1e-6)

print ('Time: %s' % timeOrange)
scoreOrange=scoring.CA(results)[0]
print "ACCURACY: %.4f" % scoreOrange
#print "AUC: %.4f" % scoring.AUC(results)[0]
cm = Orange.evaluation.scoring.confusion_matrices(results)[0]
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
print "\tD \tM \tAll"
print "D\t%i \t%i \t%i \nM\t%i \t%i \t%i \nAll\t%i \t%i \t%i" %
(cm.TN, cm.FP,(cm.TN+ cm.FP), cm.FN,cm.TP,(cm.FN+cm.TP),
(cm.TN+cm.FN),(cm.FP+cm.TP),(cm.TN+ cm.FP+cm.FN+cm.TP) )
```

CLASSIFICATION TREE

```
import Orange
from Orange.evaluation import testing, scoring
train = Orange.data.Table("../datos/plotclasificacion/train.csv")
test = Orange.data.Table("../datos/plotclasificacion/test.csv")
start = datetime.now()
clf = Orange.regression.tree.TreeLearner(train, min_instances=20)
results = Orange.evaluation.testing.test_on_data([clf], test)
timeOrange=datetime.now() - start
timeOrange=timeOrange.seconds + (timeOrange.microseconds * 1e-6)

print ('Time: %s' % timeOrange)
scoreOrange=scoring.CA(results)[0]
print "ACCURACY: %.4f" % scoreOrange
#print "AUC: %.4f" % scoring.AUC(results)[0]
cm = Orange.evaluation.scoring.confusion_matrices(results)[0]
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
print "\tD \tM \tAll"
print "D\t%i \t%i \t%i \nM\t%i \t%i \t%i \nAll\t%i \t%i \t%i" %
(cm.TN, cm.FP,(cm.TN+ cm.FP), cm.FN,cm.TP,(cm.FN+cm.TP),

(cm.TN+cm.FN),(cm.FP+cm.TP),(cm.TN+ cm.FP+cm.FN+cm.TP) )
```

ANEXO 7. Código para el experimento de clasificación de la librería Mlpy

CODIGO

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime
%matplotlib inline

# ASIGNACION DE DATOS PARA EL ENTRENAMIENTO
dftr = pd.read_csv("../datos/train_clasificacion.csv")
values=['prov_insc','mcap_bie','edad_hom','niv_insth','p_etnica_hom','are
a_hom',
        'edad_muj','niv_instm','p_etnica_muj','area_muj','estado']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['estado']
X_train=X
y_train=y

# ASIGNACION DE DATOS PARA EL TEST
dftr = pd.read_csv("../datos/test_clasificacion.csv")
values=['prov_insc','mcap_bie','edad_hom','niv_insth','p_etnica_hom','are
a_hom',
        'edad_muj','niv_instm','p_etnica_muj','area_muj','estado']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['estado']
X_test=X
y_test=y
```

LOGISTIC REGRESSION

```
import mlpy
from mlpy import LibLinear
start = datetime.now()
clf = LibLinear(solver_type='l2r_lr', C=1)# L2-regularized logistic
regression (primal)
clf.learn(X_train, y_train)
expected = y_test
predicted = clf.pred(X_test)
timeMlpy=datetime.now() - start
timeMlpy=timeMlpy.seconds + (timeMlpy.microseconds * 1e-6)

print ('Time: %s' % timeMlpy)
scoreMlpy= mlpy.accuracy(expected, predicted)
print ('Accuracy: %.4f' % scoreMlpy)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

MAXIMUM LIKELIHOOD CLASSIFIER

```
import mlpy
from mlpy import MaximumLikelihoodC
start = datetime.now()
clf = MaximumLikelihoodC()
clf.learn(X_train, y_train)
expected = y_test
predicted = clf.pred(X_test)
timeMlpy=datetime.now() - start
timeMlpy=timeMlpy.seconds + (timeMlpy.microseconds * 1e-6)

print ('Time: %s' % timeMlpy)
scoreMlpy= mlpy.accuracy(expected, predicted)
print ('Accuracy: %.4f' % scoreMlpy)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

K-NEAREST NEIGHBORS

```
import mlpy
from mlpy import KNN
start = datetime.now()
clf = KNN(k=9)#k-Nearest Neighbor (euclidean distance)
clf.learn(X_train, y_train)
expected = y_test
predicted = clf.pred(X_test)
timeMlpy=datetime.now() - start
timeMlpy=timeMlpy.seconds + (timeMlpy.microseconds * 1e-6)

print ('Time: %s' % timeMlpy)
scoreMlpy= mlpy.accuracy(expected, predicted)
print ('Accuracy: %.4f' % scoreMlpy)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

SUPPORT VECTOR MACHINE (KERNEL LINEAR)

```
import mlpy
from mlpy import LibSvm
start = datetime.now()
clf = LibSvm(svm_type='c_svc', kernel_type='linear', C=1)
clf.learn(X_train, y_train)
expected = y_test
predicted = clf.pred(X_test)
timeMlpy=datetime.now() - start
timeMlpy=timeMlpy.seconds + (timeMlpy.microseconds * 1e-6)

print ('Time: %s' % timeMlpy)
scoreMlpy= mlpy.accuracy(expected, predicted)
print ('Accuracy: %.4f' % scoreMlpy)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
```

```
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

SUPPORT VECTOR MACHINE (KERNEL RBF)

```
import mlpy
from mlpy import LibSvm
start = datetime.now()
clf = LibSvm(svm_type='c_svc', kernel_type='rbf',gamma=0.1)
clf.learn(X_train, y_train)
expected = y_test
predicted = clf.pred(X_test)
timeMlpy=datetime.now() - start
timeMlpy=timeMlpy.seconds + (timeMlpy.microseconds * 1e-6)

print ('Time: %s' % timeMlpy)
scoreMlpy= mlpy.accuracy(expected, predicted)
print ('Accuracy: %.4f' % scoreMlpy)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

CLASSIFICATION TREE

```
import mlpy
from mlpy import ClassTree
start = datetime.now()
clf = ClassTree(minsize=20)# Muestras por hoja
clf.learn(X_train, y_train)
expected = y_test
predicted = clf.pred(X_test)
timeMlpy=datetime.now() - start
timeMlpy=timeMlpy.seconds + (timeMlpy.microseconds * 1e-6)

print ('Time: %s' % timeMlpy)
scoreMlpy= mlpy.accuracy(expected, predicted)
print ('Accuracy: %.4f' % scoreMlpy)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

ANEXO 8. Código para el experimento de clasificación de la librería PyMVPA

CODIGO

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime
%matplotlib inline

# ASIGNACION DE DATOS PARA EL ENTRENAMIENTO
dftr = pd.read_csv("../datos/train_clasificacion.csv")
values=['prov_insc','mcap_bie','edad_hom','niv_insth','p_etnica_hom','are
a_hom',
        'edad_muj','niv_instm','p_etnica_muj','area_muj','estado']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['estado']
X_train=X
y_train=y

# ASIGNACION DE DATOS PARA EL TEST
dftr = pd.read_csv("../datos/test_clasificacion.csv")
values=['prov_insc','mcap_bie','edad_hom','niv_insth','p_etnica_hom','are
a_hom',
        'edad_muj','niv_instm','p_etnica_muj','area_muj','estado']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['estado']
X_test=X
y_test=y
```

LOGISTIC REGRESSION

```
import mvpa2
from mvpa2.datasets import *
from mvpa2.clfs.plr import PLR
# pymvpa utiliza dataset_wizard para leer los datos
dtrain = dataset_wizard(X_train, y_train)
dtest=dataset_wizard(X_test, y_test)
start = datetime.now()
clf = PLR(criterion=1)
clf.train(dtrain)
expected = y_test
predicted = clf.predict(dtest.samples)
predicted = np.array(predicted) #SE TRANSFORMA A NP.ARRAY
timePymvpa=datetime.now() - start
timePymvpa=timePymvpa.seconds + (timePymvpa.microseconds * 1e-6)

print ('Time: %s' % timePymvpa)
scorePymvpa=np.mean(predicted == y_test)
print ('Accuracy: %.4f' % scorePymvpa)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
```

```
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

NAIVE BAYES

```
import mvpa2
from mvpa2.datasets import *
from mvpa2.clfs.gnb import GNB
# pymvpa utiliza dataset_wizard para leer los datos
dtrain = dataset_wizard(X_train, y_train)
dtest=dataset_wizard(X_test, y_test)
start = datetime.now()
clf = GNB()
clf.train(dtrain)
expected = y_test
predicted = clf.predict(dtest.samples)
predicted = np.array(predicted) #SE TRANSFORMA A NP.ARRAY
timePymvpa=datetime.now() - start
timePymvpa=timePymvpa.seconds + (timePymvpa.microseconds * 1e-6)

print ('Time: %s' % timePymvpa)
scorePymvpa=np.mean(predicted == y_test)
print ('Accuracy: %.4f' % scorePymvpa)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

K-NEAREST NEIGHBORS

```
import mvpa2
from mvpa2.datasets import *
from mvpa2.clfs.knn import kNN
# pymvpa utiliza dataset_wizard para leer los datos
dtrain = dataset_wizard(X_train, y_train)
dtest=dataset_wizard(X_test, y_test)
start = datetime.now()
clf = kNN(k=9)
clf.train(dtrain)
expected = y_test
predicted = clf.predict(dtest.samples)
predicted = np.array(predicted) #SE TRANSFORMA A NP.ARRAY
timePymvpa=datetime.now() - start
timePymvpa=timePymvpa.seconds + (timePymvpa.microseconds * 1e-6)

print ('Time: %s' % timePymvpa)
scorePymvpa=np.mean(predicted == y_test)
print ('Accuracy: %.4f' % scorePymvpa)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

SUPPORT VECTOR MACHINE (KERNEL LINEAR)

```
import mvpa2
from mvpa2.datasets import *
from mvpa2.clfs.svm import  libsvm


# pymvpa utiliza dataset_wizard para leer los datos
dtrain = dataset_wizard(X_train, y_train)
dtest=dataset_wizard(X_test, y_test)
start = datetime.now()
clf = libsvm.SVM(C=1)
clf.train(dtrain)
expected = y_test
predicted = clf.predict(dtest.samples)
predicted = np.array(predicted) #SE TRANSFORMA A NP.ARRAY
timePymvpa=datetime.now() - start
timePymvpa=timePymvpa.seconds + (timePymvpa.microseconds * 1e-6)

print ('Time: %s' % timePymvpa)
scorePymvpa=np.mean(predicted == y_test)
print ('Accuracy: %.4f' % scorePymvpa)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

SUPPORT VECTOR MACHINE (KERNEL RBF)

```
import mvpa2
from mvpa2.datasets import *
from mvpa2.clfs import svm
# pymvpa utiliza dataset_wizard para leer los datos
dtrain = dataset_wizard(X_train, y_train)
dtest=dataset_wizard(X_test, y_test)
start = datetime.now()
kernel = svm.RbfSVMKernel(gamma=0.1)
clf = svm.SVM(kernel=kernel)
clf.train(dtrain)
expected = y_test
predicted = clf.predict(dtest.samples)
predicted = np.array(predicted) #SE TRANSFORMA A NP.ARRAY
timePymvpa=datetime.now() - start
timePymvpa=timePymvpa.seconds + (timePymvpa.microseconds * 1e-6)

print ('Time: %s' % timePymvpa)
scorePymvpa=np.mean(predicted == y_test)
print ('Accuracy: %.4f' % scorePymvpa)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

ANEXO 9. Código para el experimento de clasificación de la librería mdp

CODIGO

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime
%matplotlib inline

# ASIGNACION DE DATOS PARA EL ENTRENAMIENTO
dftr = pd.read_csv("../datos/train_clasificacion.csv")
values=['prov_insc','mcap_bie','edad_hom','niv_insth','p_etnica_hom','are
a_hom',
        'edad_muj','niv_instm','p_etnica_muj','area_muj','estado']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['estado']
X_train=X
y_train=y

# ASIGNACION DE DATOS PARA EL TEST
dftr = pd.read_csv("../datos/test_clasificacion.csv")
values=['prov_insc','mcap_bie','edad_hom','niv_insth','p_etnica_hom','are
a_hom',
        'edad_muj','niv_instm','p_etnica_muj','area_muj','estado']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['estado']
X_test=X
y_test=y
```

NAIVE BAYES

```
import mdp
from mdp.nodes.classifier_nodes import GaussianClassifier

# mdp lee los datos solo en float64 o float32
X_train=np.float64(X_train)
y_train=np.float64(y_train)
X_test=np.float64(X_test)
y_test=np.float64(y_test)
start = datetime.now()
clf = GaussianClassifier()
clf.train(X_train, y_train)
expected = y_test
predicted = clf.label(X_test)
predicted = np.array(predicted)
timeMdp=datetime.now() - start
timeMdp=timeMdp.seconds + (timeMdp.microseconds * 1e-6)

print ('Time: %s' % timeMdp)
scoreMdp=np.mean(predicted == y_test)
print ('Accuracy: %.4f' % scoreMdp)
```

```
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

K-NEAREST NEIGHBORS

```
import mdp
from mdp.nodes.classifier_nodes import KNNClassifier
# mdp lee los datos solo en float64 o float32
X_train=np.float64(X_train)
y_train=np.float64(y_train)
X_test=np.float64(X_test)
y_test=np.float64(y_test)
start = datetime.now()
clf = KNNClassifier(k=9)
clf.train(X_train, y_train)
expected = y_test
predicted = clf.label(X_test)
predicted = np.array(predicted)
timeMdp=datetime.now() - start
timeMdp=timeMdp.seconds + (timeMdp.microseconds * 1e-6)

print ('Time: %s' % timeMdp)
scoreMdp=np.mean(predicted == y_test)
print ('Accuracy: %.4f' % scoreMdp)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

SUPPORT VECTOR MACHINE (KERNEL LINEAR)

```
import mdp
from mdp.nodes.libsvm_classifier import LibSVMClassifier

# mdp lee los datos solo en float64 o float32
X_train=np.float64(X_train)
y_train=np.float64(y_train)
X_test=np.float64(X_test)
y_test=np.float64(y_test)
start = datetime.now()
clf = LibSVMClassifier(kernel='Linear')
clf.train(X_train, y_train)
expected = y_test
predicted = clf.label(X_test)
predicted = np.array(predicted)
timeMdp=datetime.now() - start
timeMdp=timeMdp.seconds + (timeMdp.microseconds * 1e-6)

print ('Time: %s' % timeMdp)
scoreMdp=np.mean(predicted == y_test)
print ('Accuracy: %.4f' % scoreMdp)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

SUPPORT VECTOR MACHINE (KERNEL RBF)

```
import mdp
from mdp.nodes.libsvm_classifier import LibSVMClassifier

# mdp lee los datos solo en float64 o float32
X_train=np.float64(X_train)
y_train=np.float64(y_train)
X_test=np.float64(X_test)
y_test=np.float64(y_test)
start = datetime.now()
clf =  LibSVMClassifier(kernel='RBF')
clf.train(X_train, y_train)
expected = y_test
predicted = clf.label(X_test)
predicted = np.array(predicted)
timeMdp=datetime.now() - start
timeMdp=timeMdp.seconds + (timeMdp.microseconds * 1e-6)

print ('Time: %s' % timeMdp)
scoreMdp=np.mean(predicted == y_test)
print ('Accuracy: %.4f' % scoreMdp)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

CLASSIFICATION TREE

```
import mdp
from mdp.nodes import DecisionTreeClassifierScikitsLearnNode
# mdp lee los datos solo en float64 o float32
X_train=np.float64(X_train)
y_train=np.float64(y_train)
X_test=np.float64(X_test)
y_test=np.float64(y_test)
start = datetime.now()
clf =
DecisionTreeClassifierScikitsLearnNode(criterion='entropy',min_samples_le
af=20)
clf.train(X_train, y_train)
expected = y_test
predicted = clf.label(X_test)
predicted =np.reshape(predicted,13733) #transforma a 1-dimensional
timeMdp=datetime.now() - start
timeMdp=timeMdp.seconds + (timeMdp.microseconds * 1e-6)

print ('Time: %s' % timeMdp)
scoreMdp=np.mean(predicted == y_test)
print ('Accuracy: %.4f' % scoreMdp)
print("Matriz de Confusion de los Datos de Prueba (20% de los datos)
[0]=Divorcio [1]=Matrimonio")
cross=pd.crosstab(expected, predicted, rownames=['VALOR REAL'],
colnames=['PREDICCION'], margins=True)
print(cross)
```

ANEXO 10. Código para el experimento de regresión de la librería Scikit-Learn

CODIGO

```
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime
from scipy import stats
import pandas as pd
%matplotlib inline

# ASIGNACION DE DATOS PARA EL ENTRENAMIENTO
dftr = pd.read_csv("../datos/train_regresion.csv")
values=['cau_div','nac_hom','edad_hom','hijos_hom','niv_insth','nac_muj',
'edad_muj','hijos_muj','niv_instm','dur_mat']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['dur_mat']
X_train=X
y_train=y
# ASIGNACION DE DATOS PARA EL TEST
dftr = pd.read_csv("../datos/test_regresion.csv")
values=['cau_div','nac_hom','edad_hom','hijos_hom','niv_insth','nac_muj',
'edad_muj','hijos_muj','niv_instm','dur_mat']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['dur_mat']
X_test=X
y_test=y
```

ORDINARY LEAST SQUARES

```
from sklearn import linear_model
start = datetime.now()
ridgere=linear_model.LinearRegression()
ridgere.fit(X_train, y_train)
timeridgere=datetime.now() - start
timeridgere=timeridgere.seconds + (timeridgere.microseconds * 1e-6)

print ('Time: %s' % timeridgere)
expected = y_test
predicted = ridgere.predict(X_test)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

REGRESSION RIDGE

```
from sklearn import linear_model
start = datetime.now()
ridgere=linear_model.Ridge(alpha=0.1)
ridgere.fit(X_train, y_train)
timeridgere=datetime.now() - start
timeridgere=timeridgere.seconds + (timeridgere.microseconds * 1e-6)

print ('Time: %s' % timeridgere)
expected = y_test
predicted = ridgere.predict(X_test)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

KERNEL RIDGE

```
from sklearn.kernel_ridge import KernelRidge
from sklearn import metrics
start = datetime.now()
kridge=KernelRidge(alpha=0.01, kernel='polynomial')
kridge.fit(X_train, y_train)
timekridge=datetime.now() - start
timekridge=timekridge.seconds + (timekridge.microseconds * 1e-6)

print ('Time: %s' % timekridge)
expected = y_test
predicted = kridge.predict(X_test)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

LASSO LARS

```
from sklearn import linear_model
start = datetime.now()
#regr2=linear_model.LassoCV()
lassolars=linear_model.LassoLars(alpha=0.001)
lassolars.fit(X_train, y_train)
timelr=datetime.now() - start
timelr=timelr.seconds + (timelr.microseconds * 1e-6)

print ('Time: %s' % timelr)
expected = y_test
predicted = lassolars.predict(X_test)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
```

```
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

ELASTIC NET

```
from sklearn.linear_model import ElasticNet

start = datetime.now()
#regr5 = ElasticNet()
lnet=ElasticNet( alpha=0.001)
lnet.fit(X_train, y_train)
timelnet=datetime.now() - start
timelnet=timelnet.seconds + (timelnet.microseconds * 1e-6)

print ('Time: %s' % timelnet)
expected = y_test
predicted = lnet.predict(X_test)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

SUPPORT VECTOR REGRESSION

```
from sklearn.svm import SVR
start = datetime.now()
svr = SVR(kernel='rbf', C=1e1, gamma=0.01)
svr.fit(X_train, y_train)
timesvr=datetime.now() - start
timesvr=timesvr.seconds + (timesvr.microseconds * 1e-6)

print ('Time: %s' % timesvr)
expected = y_test
predicted = svr.predict(X_test)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

ANEXO 11. Código para el experimento de regresión de la librería Orange

CODIGO

```
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime
from scipy import stats
import pandas as pd
%matplotlib inline
```

REGRESSION RIDGE

```
import Orange
from Orange.evaluation import testing, scoring
train = Orange.data.Table("../datos/train_regresion.csv")
test = Orange.data.Table("../datos/test_regresion.csv")
start = datetime.now()
lr =
Orange.regression.linear.LinearRegressionLearner(train,ridge_lambda=0.1)
timelr=datetime.now() - start
timelr=timelr.seconds + (timelr.microseconds * 1e-6)
print ('Time: %s' % timelr)
results = Orange.evaluation.testing.test_on_data([lr], test)
expected = y_test
print('Coeficiente de determinación(R²) : %.4f' %
Orange.evaluation.scoring.R2(results)[0])
print("MSE: %.4f" % np.mean(Orange.evaluation.scoring.MSE(results)[0]))
# EMC/MSE Error cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],lr(test[i])))
```

LASSO LARS

```
import Orange
from Orange.evaluation import testing, scoring
train = Orange.data.Table("../datos/train_regresion.csv")
test = Orange.data.Table("../datos/test_regresion.csv")
start = datetime.now()
lr =
Orange.regression.lasso.LassoRegressionLearner(train,lasso_lambda=0.001)
timelr=datetime.now() - start
timelr=timelr.seconds + (timelr.microseconds * 1e-6)
print ('Time: %s' % timelr)
results = Orange.evaluation.testing.test_on_data([lr], test)
rmse = Orange.evaluation.scoring.RMSE(results)[0]
print('Coeficiente de determinación(R²) : %.4f' %
Orange.evaluation.scoring.R2(results)[0])
print("MSE: %.4f" % np.mean(Orange.evaluation.scoring.MSE(results)[0]))
# EMC/MSE Error cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],lr(test[i])))
```

SUPPORT VECTOR REGRESSION

```
import Orange
from Orange.evaluation import testing, scoring
train = Orange.data.Table("../datos/train_regresion.csv")
test = Orange.data.Table("../datos/test_regresion.csv")
from Orange.classification import svm

start = datetime.now()
lr = svm.SVMLearner(train,svm_type=svm.SVMLearner.Epsilon_SVR,
                            kernel_type=svm.kernels.RBF,C=1e1,
gamma=0.01
                           )

timelr=datetime.now() - start
timelr=timelr.seconds + (timelr.microseconds * 1e-6)
print ('Time: %s' % timelr)
results = Orange.evaluation.testing.test_on_data([lr], test)
rmse = Orange.evaluation.scoring.RMSE(results)[0]
print('Coeficiente de determinación(R²) : %.4f' %
Orange.evaluation.scoring.R2(results)[0])
print("MSE: %.4f" % np.mean(Orange.evaluation.scoring.MSE(results)[0]))
# EMC/MSE Error cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],lr(test[i])))
```

ANEXO 12. Código para el experimento de regresión de la librería Mlpy

CODIGO

```
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime
from scipy import stats
import pandas as pd
%matplotlib inline

# ASIGNACION DE DATOS PARA EL ENTRENAMIENTO
dftr = pd.read_csv("../datos/train_regresion.csv")
values=['cau_div','nac_hom','edad_hom','hijos_hom','niv_insth','nac_muj',
'edad_muj','hijos_muj','niv_instm','dur_mat']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['dur_mat']
X_train=X
y_train=y
# ASIGNACION DE DATOS PARA EL TEST
dftr = pd.read_csv("../datos/test_regresion.csv")
values=['cau_div','nac_hom','edad_hom','hijos_hom','niv_insth','nac_muj',
'edad_muj','hijos_muj','niv_instm','dur_mat']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['dur_mat']
X_test=X
y_test=y
```

REGRESSION RIDGE

```
import mlpy
start = datetime.now()
lnet=mlpy.Ridge(lmb=0.1)
lnet.learn(X_train, y_train)
timelnet=datetime.now() - start
timelnet=timelnet.seconds + (timelnet.microseconds * 1e-6)
print ('Time: %s' % timelnet)
expected = y_test
predicted = lnet.pred(X_test)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

KERNEL RIDGE

```
import mlpy
start = datetime.now()
K = mlpy.kernel_polynomial(X_train, X_train)
Kt = mlpy.kernel_polynomial(X_test, X_train)
#K = mlpy.kernel_linear(X_train, X_train)
#Kt = mlpy.kernel_linear(X_test, X_train)

kernel=mlpy.KernelPolynomial()
lnet=mlpy.KernelRidge(lmb=0.01)
lnet.learn(K, y_train)
timelnet=datetime.now() - start
timelnet=timelnet.seconds + (timelnet.microseconds * 1e-6)
print ('Time: %s' % timelnet)
expected = y_test
predicted = lnet.pred(Kt)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

ELASTIC NET

```
from mlpy import ElasticNetC
start = datetime.now()
ols = mlpy.ElasticNet(lmb=0.001, eps=0.001)
ols.learn(X_train, y_train)
timeols=datetime.now() - start
timeols=timeols.seconds + (timeols.microseconds * 1e-6)

print ('Time: %s' % timeols)
expected = y_test
predicted = ols.pred(X_test)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

SUPPORT VECTOR REGRESSION

```
import mlpy
start = datetime.now()
ols = mlpy.LibSvm(svm_type='epsilon_svr', kernel_type='rbf',C=1e1,
gamma=0.01)
ols.learn(X_train, y_train)
timeols=datetime.now() - start
timeols=timeols.seconds + (timeols.microseconds * 1e-6)

print ('Time: %s' % timeols)
expected = y_test
predicted = ols.pred(X_test)
```

```
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

ANEXO 13. Código para el experimento de regresión de la librería PyMVPA

CODIGO

```
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime
from scipy import stats
import pandas as pd
%matplotlib inline

# ASIGNACION DE DATOS PARA EL ENTRENAMIENTO
dftr = pd.read_csv("../datos/train_regresion.csv")
values=['cau_div','nac_hom','edad_hom','hijos_hom','niv_insth','nac_muj',
'edad_muj','hijos_muj','niv_instm','dur_mat']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['dur_mat']
X_train=X
y_train=y
# ASIGNACION DE DATOS PARA EL TEST
dftr = pd.read_csv("../datos/test_regresion.csv")
values=['cau_div','nac_hom','edad_hom','hijos_hom','niv_insth','nac_muj',
'edad_muj','hijos_muj','niv_instm','dur_mat']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['dur_mat']
X_test=X
y_test=y
```

REGRESSION RIDGE

```
from mvpa2.datasets import *
dtrain = dataset_wizard(X_train, y_train)
dtest=dataset_wizard(X_test, y_test)
from mvpa2.clfs.ridge import RidgeReg
start = datetime.now()
lnet=RidgeReg(lm=0.1)
lnet.train(dtrain)
timelnet=datetime.now() - start
timelnet=timelnet.seconds + (timelnet.microseconds * 1e-6)
print ('Time: %s' % timelnet)
expected = y_test
predicted = lnet.predict(dtest.samples)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

LASSO LARS

```
from mvpa2.clfs.lars import LARS
from mvpa2.datasets import *
dtrain = dataset_wizard(X_train, y_train)
dtest=dataset_wizard(X_test, y_test)
start = datetime.now()
lnet=LARS()
lnet.train(dtrain)
timelnet=datetime.now() - start
timelnet=timelnet.seconds + (timelnet.microseconds * 1e-6)
print ('Time: %s' % timelnet)
expected = y_test
predicted = lnet.predict(dtest.samples)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

ELASTIC NET

```
from mvpa2.datasets import *
dtrain = dataset_wizard(X_train, y_train)
dtest=dataset_wizard(X_test, y_test)
from mvpa2.clfs.enet import ENET
start = datetime.now()
lnet=ENET(lm=0.001)
lnet.train(dtrain)
timelnet=datetime.now() - start
timelnet=timelnet.seconds + (timelnet.microseconds * 1e-6)
print ('Time: %s' % timelnet)
expected = y_test
predicted = lnet.predict(dtest.samples)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

SUPPORT VECTOR REGRESSION

```
from mvpa2.datasets import *
dtrain = dataset_wizard(X_train, y_train)
dtest=dataset_wizard(X_test, y_test)
from mvpa2.clfs import svm
start = datetime.now()
kernel = svm.RbfSVMKernel(gamma=0.01)
lnet=svm.SVM(svm_impl="EPSILON_SVR",C=1e1, kernel=kernel)
lnet.train(dtrain)
timelnet=datetime.now() - start
timelnet=timelnet.seconds + (timelnet.microseconds * 1e-6)
print ('Time: %s' % timelnet)
expected = y_test
```

```
predicted = lnet.predict(dtest.samples)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

ANEXO 14. Código para el experimento de regresión de la librería Statsmodels

CODIGO

```
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime
from scipy import stats
import pandas as pd
%matplotlib inline

# ASIGNACION DE DATOS PARA EL ENTRENAMIENTO
dftr = pd.read_csv("../datos/train_regresion.csv")
values=['cau_div','nac_hom','edad_hom','hijos_hom','niv_insth','nac_muj',
'edad_muj','hijos_muj','niv_instm','dur_mat']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['dur_mat']
X_train=X
y_train=y
# ASIGNACION DE DATOS PARA EL TEST
dftr = pd.read_csv("../datos/test_regresion.csv")
values=['cau_div','nac_hom','edad_hom','hijos_hom','niv_insth','nac_muj',
'edad_muj','hijos_muj','niv_instm','dur_mat']
dftrain=dftr[values]
n_samples, n_features = dftrain.shape
lenf=len(values)-1
features = dftrain.columns[:lenf]
X=dftrain[features]
y= dftrain['dur_mat']
X_test=X
y_test=y
```

ORDINARY LEAST SQUARES

```
import statsmodels
import statsmodels.api as sm
from __future__ import print_function
from statsmodels.sandbox.regression.predstd import wls_prediction_std
start = datetime.now()
statsmod = statsmodels.regression.linear_model.OLS(y_train, X_train)
statsres = statsmod.fit()
time=datetime.now() - start
time=time.seconds + (time.microseconds * 1e-6)
print ('Time: %s' % time)
expected = y_test
predicted = statsres.predict(X_test)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

REGRESSION RIDGE

```
import statsmodels
import statsmodels.api as sm
start = datetime.now()
statsmod = statsmodels.regression.linear_model.OLS(y_train, X_train)
statsres = statsmod.fit_regularized(alpha=0.1, L1_wt=0)# If 0, the fit is
ridge regression
time=datetime.now() - start
time=time.seconds + (time.microseconds * 1e-6)
print ('Time: %s' % time)
expected = y_test
predicted = statsres.predict(X_test)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```

LASSO LARS

```
import statsmodels
import statsmodels.api as sm
start = datetime.now()
statsmod = statsmodels.regression.linear_model.OLS(y_train, X_train)
statsres = statsmod.fit_regularized(alpha=0.001, L1_wt=1)# If 1, the fit
is the lasso.
time=datetime.now() - start
time=time.seconds + (time.microseconds * 1e-6)
print ('Time: %s' % time)
expected = y_test
predicted = statsres.predict(X_test)
print('Coeficiente de determinación(R²) : %.4f' % r2_score(expected,
predicted))
print("MSE: %.4f" % np.mean((predicted - expected) ** 2)) # EMC/MSE Error
cuadrático medio
print ("\nPredicción de las primeras 6 instancias[VR]=Valor Real
[VP]=Valor Predicho\n")
for i in range(6):
    print("VR: %.1f VP:%.1f" % (expected[i],predicted[i]))
```