Escuela Politécnica Superior de Jaén
Grado en Ingeniería Informática

Universidad de Jaén
Escuela Politécnica Superior
de Jaén

TRABAJO FIN DE GRADO

Sistema basado en machine learning para clasificación sentimientos en opiniones

Alumno

Pablo Luna Zaragoza

Tutora

Mª DOLORES MOLINA GONZÁLEZ,
(Departamento de Telecomunicaciones)

Junio, 2020

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Doña Mª DOLORES MOLINA GONZÁLEZ, tutora del Trabajo Fin de Grado titulado: '**Sistema basado en machine learning para clasificación sentimientos en opiniones**', que presenta Don Pablo Luna Zaragoza, otorga el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Junio de 2020

El alumno: La tutora:

Pablo Luna Zaragoza Mª DOLORES MOLINA GONZÁLEZ,

(Página intencionalmente en blanco)

Agradecimientos

Gracias a todos los profesores por haberme enseñado, formado en estos cuatro años que por fin acabo, y por lo que me siento realmente satisfecho y agradecido y también quiero agradecer a mi familia todo su apoyo desde el primer momento que elegí hacer este grado.

Índice de contenido

Índice de Ilustraciones

Índice de tablas

1 ESPECIFICACIÓN DEL TRABAJO

1.1 Introducción

Desde siempre una parte fundamental de nuestro comportamiento era saber que piensan otras personas. Actualmente las redes sociales como Twitter, Facebook, tienen un papel muy importante, por ejemplo, Twitter basado en micro Blogs nos proporciona una cantidad bastante considerable de información la cual podemos utilizar en aplicaciones de Análisis de sentimientos (Sentiment Analysis SA), estas aplicaciones tratan de extraer información útil de esta gran cantidad de datos y clasificarlo en lo que serían las distintas clases (sentimientos).

Procesar esta gran cantidad de información de forma manual es un trabajo prácticamente imposible, por lo que surgen técnicas como el Machine Learning (ML), con las cuales buscamos que sea el propio ordenador el que aprenda y nos proporcione los resultados que de otra forma no podríamos conseguir.

El objetivo de este TFG es crear una de estas aplicaciones, que basándose en distintitos algoritmos de ML, sea capaz de encontrar los parámetros que proporcionen los mejores resultados para un corpus proporcionado por el usuario.

La aplicación se divide principalmente en tres apartados, el primero es la entrada del corpus y su preprocesamiento mediante las distintas técnicas de procesamiento del lenguaje natural (PLN). Una vez hecho esto pasaremos a la parte de entrenar los distintos algoritmos para así ver cuál de ellos da mejores resultados y con qué parámetros, y finalmente todo esto estará representado en pantalla mediante una interfaz simple e intuitiva.

Para ello haré uso de Python, un lenguaje muy potente de alto nivel que cuenta con muchas bibliotecas que nos ayudan tanto con PLN como con algoritmos de ML, como también para crear la propia interfaz de la aplicación.

Para llevar a cabo esta herramienta, he realizado una etapa de investigación y estudio, ya que durante la carrera varios de los campos relacionados con este trabajo se ven muy por encima, como pueden ser el tema del ML, del cual solo se nos proporciona un ligero conocimiento teórico, o en el caso del PLN que es una

asignatura opcional de cuarto, la cual no he cursado y que tras la realización de este trabajo me parece muy interesante.

1.2 Objetivos del trabajo

Los principales objetivos que me propuse para la realización de este trabajo fueron los siguientes:

1. Aprender sobre las distintas técnicas de ML.
2. Aprender sobre el estado del arte del AS.
3. Continuar aprendiendo a programar en Python.
4. Estudiar las distintas tecnologías para implementar la herramienta.
5. Aprender a usar bibliotecas de PLN y de ML.
6. Diseñar una interfaz fácil de manejar e intuitiva.
7. Aprender a implementar interfaces en Python.
8. Implementar la interfaz internacionalizada.
9. Decidir un formato para la entrada y salida de los datos.

1.3 Antecedentes y estado del arte

El SA es el proceso de recolectar y analizar datos basándonos en las opiniones y pensamiento de las personas. También es conocido como “opinion mining” ya que extrae características importantes de lo que opina la gente. El SA es una combinación de varias técnicas de ML, modelos estadísticos y Procesamiento del Lenguaje Natural, para la extracción de las características.

El SA se puede hacer tanto a nivel de archivo, como a nivel de frase o incluso a nivel de palabra. A nivel de archivo se toma en cuenta un resumen con la idea principal del mismo, y esta idea es la que valoramos como positiva o negativa. La oración se tiene en cuenta para comprobar la polaridad. En el nivel de oración, cada oración es clasificada en una clase particular para proporcionar el sentimiento (Prabowo, 2009).

El SA tiene una gran cantidad de aplicaciones en la actualidad, se utiliza para generar opiniones, valoraciones de la gente en las redes sociales mediante el

análisis de los textos que publican. Otra forma, por ejemplo, en la que se está usando, es para detectar casos de ciber-acoso, bulling y este tipo de comportamiento en redes sociales mediante el cual si el sistema detecta este tipo de comportamientos, directamente se elimina el mensaje e incluso se llega a penalizar al usuario que lo publicó. También, es común usarlo para obtener comentarios sobre cualquier producto o película, para obtener el informe financiero de cualquier empresa, para predicciones o marketing (Liu, 2012).

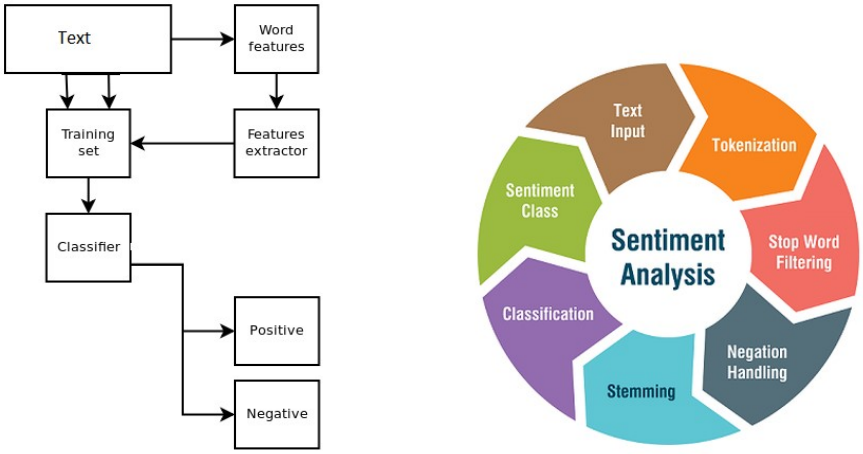


Ilustración 1 Resumen SA

Aunque no de forma generalizada, la mayoría de sistemas de SA utiliza algoritmos de ML.

Toda esta gran cantidad de datos no tendría sentido si la procesáramos manualmente, para crear un sistema que sea capaz de funcionar automáticamente requerimos de la utilización de técnicas de ML:

El ML es una rama de la informática cuya idea principal es crear sistemas que pueden aprender por sí solos.

Es una tecnología que permite hacer automáticas una serie de operaciones con el fin de reducir la necesidad de que intervengan los seres humanos. Esto puede suponer una gran ventaja a la hora de controlar una ingente cantidad de información de un modo mucho más efectivo (Alpaydin, 2020).

Los inicios del ML los encontramos en los años 50s, cuando Arthur Samuel, pionero en el campo de los juegos informáticos e Inteligencia Artificial (IA), escribió el primer programa de aprendizaje informático.

Los principales tipos de algoritmos de ML son:

Aprendizaje no supervisado:

En este tipo de aprendizaje no se usan valores verdaderos o etiquetas. Estos sistemas tienen como finalidad la comprensión y abstracción de patrones de información de manera directa. Este es un modelo de problema que se conoce como clustering. Es un método de entrenamiento más parecido al modo en que los humanos procesan la información.

Aprendizaje por refuerzo:

En la técnica de aprendizaje mediante refuerzo, los sistemas aprenden a partir de la experiencia. Como ejemplo, se puede observar el comportamiento de un coche autónomo. Cuando el vehículo toma una decisión errónea, es penalizado, dentro de un sistema de registro de valores. Mediante dicho sistema de premios y castigos, el vehículo desarrolla una forma más efectiva de realizar sus tareas.

Aprendizaje supervisado

El aprendizaje supervisado es una técnica cuya tarea es deducir una función de etiquetado dada una serie de muestras de entrenamiento. Las muestras que utilizamos para el aprendizaje supervisado consisten en un gran conjunto de ejemplos para un tema en particular. En el aprendizaje supervisado, cada muestra de entrenamiento viene en un par de entrada (vector quantity) y valor de salida (el valor deseado) (Marsland, 2015). Estos algoritmos analizan datos y generan una función de salida, que se utiliza para mapear nuevos conjuntos de datos a las clases respectivas. Este trabajo se va a centrar en este tipo de algoritmos.

Los principales algoritmos no supervisados que vamos a tratar:

- Naive Bayes
- Decision Tree
- Random Forest
- SVC (Support Vector Classifier)

Naive Bayes

Los métodos de Bayes son un conjunto de algoritmos de aprendizaje supervisado fundamentados en el teorema de Bayes, y algunas hipótesis simplificadoras adicionales. Es a causa de estas simplificaciones, que se suelen resumir en la hipótesis de independencia entre las variables predictoras, por las que recibe el apelativo de naive, es decir, ingenuo.

Funciona en base a la probabilidad condicional. La probabilidad condicional es la probabilidad de que algo suceda, dado que ya ha ocurrido algo anteriormente. Usando la probabilidad condicional, podemos calcular la probabilidad de un evento utilizando su conocimiento previo.

El teorema de Bayes establece la siguiente relación, dada la variable de clase y el vector de característica dependiente x1 sobre xn

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Lo anterior podría reescribirse en lenguaje común como:

$$\text{Posterior} = \frac{\text{Anterior} \times \text{Probabilidad}}{\text{Evidencia}}$$

Los diferentes clasificadores NB difieren principalmente por los supuestos que hacen con respecto a la distribución.

Los que se estudiarán en este trabajo son:

1. MultinomialNB es una de las dos variantes clásicas utilizadas en la clasificación de texto, donde los datos se representan típicamente como recuentos de vectores de palabras. La distribución está parametrizada por vectores de tipo $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ para cada clase y, donde n es el número de características (en la clasificación de texto, el tamaño del vocabulario), y θ_{yi} es la probabilidad $P(x_i \mid y)$ de la característica i de aparecer en una muestra que pertenece a la clase y.

2. Gaussian NB que supone que la probabilidad de las características es gaussiana.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Algunas Ventajas que nos proporciona este clasificador son:

1. Los clasificadores de NB han funcionado bien en muchas situaciones del mundo real, por ejemplo, en clasificación de documentos y filtrado de spam.
2. Requieren una pequeña cantidad de datos de entrenamiento para estimar los parámetros necesarios para la clasificación.
3. Los algoritmos NB pueden ser extremadamente rápidos.
4. Permite clasificación multiclase.

Por otro lado, aunque NB se conoce como un clasificador decente, se sabe que es un mal estimador, por lo que las salidas de probabilidad no se deben tomar demasiado en serio.

Los Clasificadores de Bayes pueden aprender la importancia de las características individuales, pero no pueden determinar la relación entre las características.

Decision Trees

Los árboles de decisión (DT) son un método de aprendizaje supervisado no paramétrico utilizado para la clasificación y la regresión. El objetivo es crear un modelo que prediga el valor de una variable mediante el aprendizaje de reglas de decisión simples inferidas de las características de los datos.

El árbol de decisión crea modelos de clasificación o regresión en forma de árbol, desglosa un conjunto de datos en subconjuntos cada vez más pequeños. El resultado final es un árbol con nodos de decisión y nodos hoja. Un nodo de decisión tiene dos o más ramas, el nodo hoja representa una clasificación o decisión.

Un árbol de decisión se construye de arriba hacia abajo desde un nodo raíz e implica la partición de los datos en subconjuntos que contienen instancias con valores similares (homogéneos) (Sehra, 2018).

El funcionamiento de los árboles de decisión sería tal que:

1. Se calcula la entropía del nodo. La entropía se usa para calcular la homogeneidad de una muestra. Si la muestra es completamente homogénea, la entropía es cero y si la muestra está igualmente dividida, tiene una entropía de uno.

2. El conjunto de datos se divide en los diferentes atributos. Se calcula la entropía para cada rama. Luego, se suma proporcionalmente para obtener la entropía total para la división. La entropía resultante se resta de la entropía antes de la división. El resultado es la ganancia de información. Se elige el atributo con la mayor ganancia de información como nodo de decisión y se divide el conjunto de datos entre sus ramas, por último, repetimos el mismo proceso en cada rama.

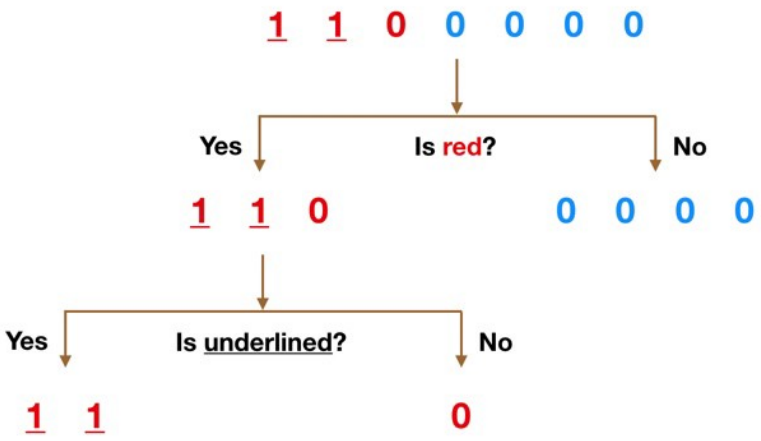


Ilustración 2 Ejemplo DT

Algunas ventajas de los árboles de decisión son:

1. Simple de entender e interpretar. Los árboles pueden ser visualizados.

2. El costo de usar el árbol para predecir datos es logarítmico en función a la cantidad de datos utilizados para entrenar el árbol.

3. Capaz de manejar datos numéricos y categóricos. Otras técnicas suelen estar especializadas en el análisis de conjuntos de datos que tienen un solo tipo de variable.
4. Capaz de manejar problemas de salida múltiple.
5. Utiliza un modelo de caja blanca. Si una situación dada es observable en un modelo, la explicación de la condición se explica fácilmente por la lógica booleana. Por el contrario, en un modelo de caja negra (por ejemplo, en una red neuronal artificial), los resultados pueden ser más difíciles de interpretar.
6. Es posible validar un modelo utilizando pruebas estadísticas. Eso permite tener en cuenta la fiabilidad del modelo.

Las desventajas de los árboles de decisión incluyen:

1. Los árboles de decisión pueden crear árboles demasiado complejos que no generalizan bien los datos. Esto se llama sobreajuste. Para evitar este problema, son necesarios mecanismos como la poda, establecer el número mínimo de muestras requeridas en un nodo de hoja o establecer la profundidad máxima del árbol.
2. Los árboles de decisión pueden ser inestables porque pequeñas variaciones en los datos pueden generar un árbol completamente diferente.
3. Los algoritmos prácticos de aprendizaje del árbol de decisiones se basan en algoritmos heurísticos como el algoritmo “greedys,” donde se toman decisiones localmente óptimas en cada nodo. Tales algoritmos no pueden garantizar la devolución del árbol de decisión globalmente óptimo. Esto puede mitigarse entrenando múltiples árboles.
4. Se pueden crean árboles sesgados si algunas clases dominan. Por lo tanto, se recomienda equilibrar el conjunto de datos antes de ajustarlo con el árbol de decisión.

Random Forest

Un bosque aleatorio (RF) es un algoritmo que se ajusta a varios clasificadores de árbol de decisión en varias submuestras del conjunto de datos y utiliza el promedio para mejorar la precisión predictiva y controlar el sobreajuste.

Es un algoritmo de promedio basado en árboles de decisión aleatorios. Se basa en métodos de perturbar y combinar específicamente diseñados para árboles. Esto significa que se crea un conjunto diverso de clasificadores al introducir la aleatoriedad en la construcción del clasificador. La predicción del conjunto se da como la predicción promedio de los clasificadores individuales.

El propósito de estas dos fuentes de aleatoriedad es disminuir la varianza del RF. De hecho, los árboles de decisión individuales suelen presentar una gran variación y tienden a sobreajustarse. La aleatoriedad inyectada en los bosques produce árboles de decisión con errores de predicción algo desacoplados. Al tomar un promedio de esas predicciones, algunos errores pueden cancelarse. Los bosques aleatorios logran una variación reducida al combinar diversos árboles, a veces, a costa de un ligero aumento en el sesgo. En la práctica, la reducción de la varianza a menudo es significativa, por lo que se obtiene un mejor modelo general.

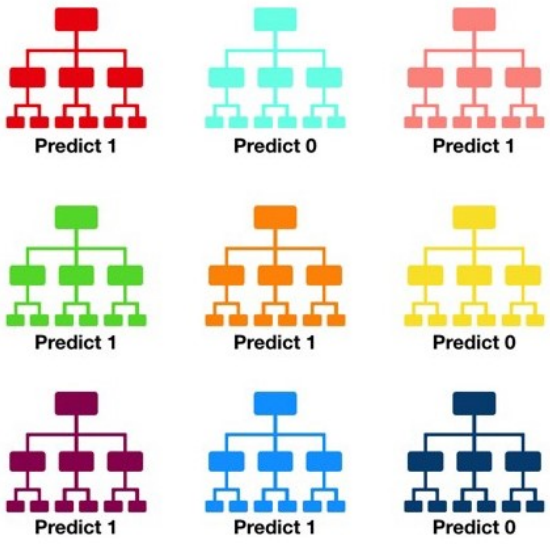


Ilustración 3 Ejemplo RF

Support Vector Machines

Las máquinas de vectores de soporte (SVM) son un conjunto de métodos de aprendizaje supervisado utilizados para la clasificación, regresión y detección de valores atípicos.

Lo que hacen los SVM es encontrar una línea de separación (o hiperplano) entre los datos de dos clases. SVM es un algoritmo que toma los datos como entrada y genera una línea que separa esas clases si es posible.

El problema viene de que pueden existir múltiples líneas que dividan el espacio, entonces ¿cuál es la mejor de estas líneas?

SVM lo hace de la siguiente forma: primero encontramos los puntos más cercanos a la línea de ambas clases. Estos puntos se denominan vectores de soporte. Ahora, calculamos la distancia entre la línea y los vectores de soporte. Esta distancia se llama margen. Nuestro objetivo es maximizar el margen o lo que es lo mismo encontrar la línea que optimice el hiperplano. (Pupale, 2018).

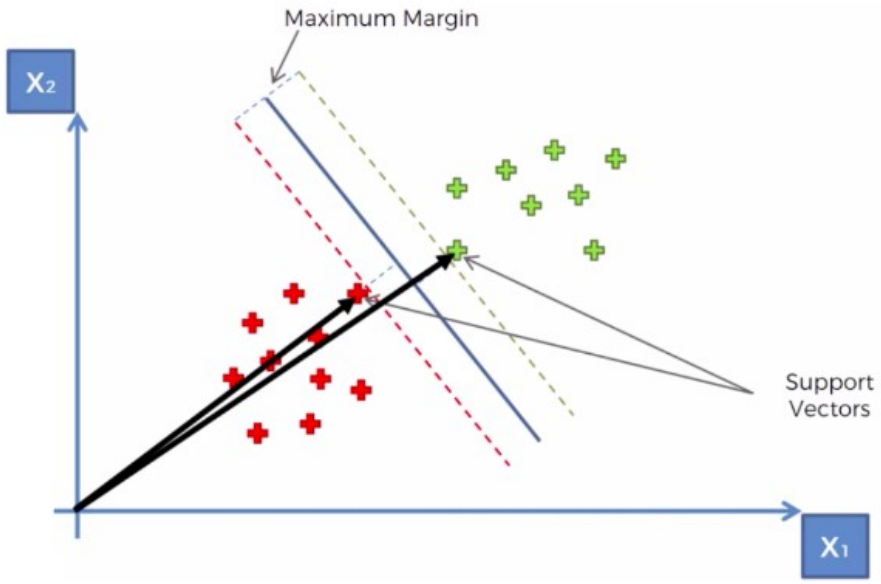


Ilustración 4 Ejemplo SVM

Supongamos que tenemos un vector w que siempre es normal al hiperplano (perpendicular a la línea en 2 dimensiones). Podemos determinar qué tan lejos está una muestra de nuestro límite de decisión proyectando el vector de posición de la muestra en el vector w. Se calcula con el producto escalar.

Si se trata de una muestra positiva, vamos a insistir en que la función de decisión en curso (el producto de punto de w y el vector de posición de una muestra dada más alguna constante) devuelve un valor mayor o igual a 1, y si es negativo devolverá un valor menor o igual que -1.

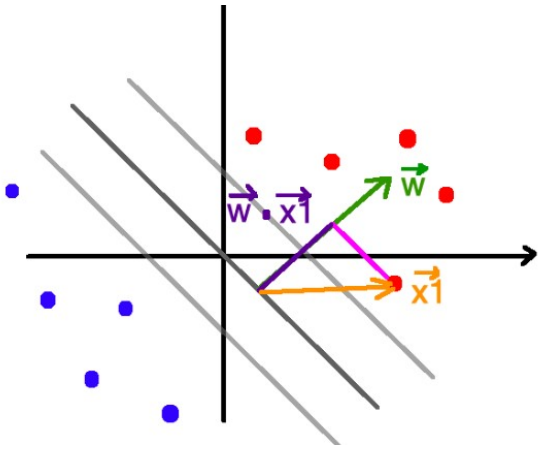


Ilustración 5 Ejemplo SVM

Las ventajas de las máquinas de vectores de soporte son:

1. Efectivo en espacios de altas dimensiones.
2. Sigue siendo efectivo en casos donde el número de dimensiones es mayor que el número de muestras.
3. Utiliza un subconjunto de puntos de entrenamiento en la función de decisión (llamados vectores de soporte), por lo que son eficientes en memoria.
4. Versátil: se pueden especificar diferentes funciones de Kernel para la función de decisión. Se proporcionan núcleos comunes, pero también es posible especificar núcleos personalizados.

Las desventajas de las máquinas de vectores de soporte incluyen:

1. Si el número de características es mucho mayor que el número de muestras, hay que evitar el ajuste excesivo al elegir las funciones del núcleo y el término de regularización es crucial.

2. Las SVM no proporcionan directamente estimaciones de probabilidad, estas se calculan utilizando una costosa validación cruzada.

1.4 Descripción de la situación de partida

Partimos de la necesidad de conocer cuál es el mejor algoritmo, el que mejor se adapta para un conjunto de datos, para lo que se requiere de un nuevo software gráfico que permita rápidamente tener una comparativa de cuáles son los mejores parámetros para un algoritmo en concreto, y cuál de estos algoritmos es mejor.

El objetivo es crear una aplicación de escritorio que sea capaz de hacer lo mencionado anteriormente.

Lo fundamental para la realización del trabajo, era tener unos conocimientos previos en algún lenguaje de programación que tuviera a su disposición bibliotecas o algún tipo de ayuda con respecto a PLN y ML, por lo que barajé distintas opciones como Matlab y Python.

En cuanto a los conocimientos que yo dispongo, cursé una asignatura en la que las prácticas se desarrollaban en Python, un lenguaje que me sorprendió para bien, por lo que tenía cierta experiencia.

Sin embargo, con respecto a PLN, no tenía ningún conocimiento previo por lo que tuve que estudiar los conceptos básicos para la realización del trabajo.

Con respecto al ML se nos proporciona los conceptos básicos en la asignatura de IA y Metaheurística, por lo que no partía desde cero.

1.5 Requisitos iniciales 

En un principio la aplicación debía contar con los siguientes requisitos funcionales:

-Debe cargar un corpus con un formato dado.

-Debe tener una interfaz gráfica.

-Debe tener una ventana gráfica en la que se muestren los resultados.

-Debe poder guardar los mejores parámetros.

-Debe poder entrenar algoritmos.

-Debería poder seleccionar el tipo de procesamiento de texto.

-Debe usarse el método Cross-Validation para la creación de lo sets.

1.6 Estudio de alternativas y viabilidad

Alternativa lenguaje de programación:

Uno de los requisitos para la realización del TFG era disponer de conocimientos en un lenguaje de programación que tuviera facilidades con el PLN y con el ML. Teniendo en cuenta estos factores, los dos lenguajes con más ayudas con respecto a estos campos eran Matlab y Python.

El problema de Matlab era que las librerías con las que trabajaba no me daban tanta facilidad ni libertad como me daban las de Python, que disponían de multitud de ejemplos y de configuraciones. Además, las librerías de Python son tan fáciles de instalar como poner un simple comando.

Y por último, yo ya sabía programar en Python (tenía ciertos conocimientos básicos), mientras que con Matlab sería comenzar desde cero.

Alternativa IDE de desarrollo:

En lo que se refiere al IDE estaba dudoso entre Pydev y Pycharm.

Pydev es gratuito y trae consigo multitud de funciones muy interesantes para la programación eficiente de Python. Es un plugin open source que se ejecuta en Eclipse.

Las principales ventajas de Pydev son: el autocompletado de código, soporte multilingüe, depuración integrada de Python, análisis de código, plantillas de código, marcado de errores o la integración de control de código.

PyCharm es un IDE muy completo para Python desarrollado por JetBrains. PyCharm es un IDE profesional y tiene dos versiones: la open source, más básica, y la profesional.

La mayoría de las características están disponibles en la versión gratuita, incluyendo el autocompletado de código, la navegación intuitiva por el proyecto, calidad de código verificado y refactorizado con PEP8, (guía de estilo para la programación en Python) y depurador gráfico.

Me decanté por esta última, por la gran ayuda que proporciona el depurador gráfico y porque su tamaño es inferior al de Eclipse.

1.7 Descripción de la solución propuesta

La solución por la que he optado ha sido por la de crear tres módulos independientes, el de ML, el de SA, y el de idioma que se unen en la interfaz.

En cuanto al PLN:

1. El tipo de procesamiento que se le dará al texto será eliminar stopwords, emoticonos y puntuaciones innecesarias.
2. Este procesamiento será opcional pudiéndose elegir una opción u otra o las dos o ninguna.

En cuanto al ML:

1. Los algoritmos que utilizaremos serán SVM, DT, RF y NB.
2. Los algoritmos tendrán una configuración de parámetros limitados ya que si el número de parámetros aumentara mucho, el tiempo de cómputo para entrenar los algoritmos aumentaría considerablemente.
3. El número de ejecuciones de un algoritmo será siempre predefinido, teniendo la opción de ejecutar menos veces en los algoritmos que consuman más tiempo.
4. Los sets de datos se dividirán usando el método de Cross-Validation.
5. La salida de los parámetros se hará en el siguiente formato determinado que será proporcionado en las iteraciones finales.

En cuanto a la interfaz:

1. Interfaz simple, intuitiva y manejable con atajos de teclado y multidioma.
2. Pestaña principal que muestra los resultados y tiene los distintos menús.
3. Menú de archivo para cargar corpus y guardar los parámetros.
4. Menú con los parámetros de los distintos algoritmos.
5. Menú con los distintos idiomas.

1.8 Material y métodos

El principal material del que dispongo para este trabajo son una serie de datasets / corpus que vamos a utilizar tanto para entrenar como para validar los algoritmos de ML.

Se me proporcionaron dos, uno con reviews de cine, y otro con tweets. Adicionalmente, para la realización de las pruebas iniciales, usé los datasets de ejemplo que proporcionan las propias librerías las cuales no requieren de ningún tipo de preprocesamiento.

1.9 Tecnologías utilizadas

Python es un lenguaje de programación dinámico de alto nivel. Se utilizó la versión Python 3.7.3. Es un lenguaje interpretado que realiza las pruebas y la depuración extremadamente rápido ya que al ser un leguaje de scripting no es necesario compilar. Hay una amplia fuente abierta de bibliotecas disponibles para esta versión de Python y una gran comunidad de usuarios.

Python es un lenguaje de programación simple pero potente, interpretado y dinámico, que es bien conocido por su funcionalidad de procesamiento de datos en lenguaje natural usando NLTK.

Otra alternativa al uso de Python podría haber sido Matlab, ya que parecía fácil de usar, pero no ofrecía la misma flexibilidad y libertad que Python puede ofrecer, además de que para mí, habría sido comenzar desde cero con ese lenguaje mientras que con Python ya tenía conocimientos previos.

Natural Language Toolkit (NLTK) es una biblioteca de Python que proporciona una fuerte base para aplicar distintas técnicas de PLN como el tagging, stemming,

tokenization, etc... Y además, también nos proporciona ciertos algoritmos de clasificación, aunque en este trabajo voy a usar otra biblioteca para los algoritmos y utilizaré ésta solamente para el procesamiento del texto.

Esta herramienta, junto con algunas expresiones regulares, son las que me ayudarán a preprocesar el texto para extraer así las características que me interesen, y así, poder utilizar cualquier tipo de texto ya sean tweets o reviews en nuestro proceso de entrenamiento de algoritmos.

NLTK dispone además, de una gran cantidad de datasets de prueba y de tutoriales que son de gran ayuda para comenzar a entrar en este mundo.

Adicionalmente NLTK, es compatible con la biblioteca de ML Sckit-Learn que dispone de los principales algoritmos que voy a usar para este trabajo.

Scikit-learn es un módulo de Python que integra una amplia gama de algoritmos de ML de última generación para problemas supervisados y no supervisados a mediana escala. Sus principales ventajas son su gran facilidad de uso, rendimiento, documentación y la gran cantidad de ejemplos que podemos encontrar.

Para la creación de la interfaz, he utilizado la biblioteca Tkinter, un binding de la biblioteca gráfica Tcl/Tk para el lenguaje de programación Python. Se considera un estándar para la interfaz gráfica de usuario (GUI), para Python y es el que viene por defecto con la instalación para Microsoft Windows.

En cuanto al entorno de desarrollo, he estado utilizando dos:

1. Principalmente, Sublime Text, un procesador de textos de software libre, totalmente acomodado para la programación al que podemos instalar algunos pluggins para que nos ayude aún más con Python.
2. Y Pycharm, un IDE de JetBrains, gratuito, específico para Python que me ayudará en algunos temas de depuración más complejos.

Ilustración 6 Herramientas utilizadas

1.10 Metodología de desarrollo de software

No he escogido ningún método de desarrollo en concreto, ya que se me proporcionó total libertad a la hora de realizar este trabajo aunque, si pudiese adjuntarlo a alguna metodología, sería desarrollo incremental, en la que me autoadjudiqué mis propias iteraciones y en la que al final de cada iteración en la que tuviera algo que se pudiera enseñar, quedaba con mi tutora para mostrárselo y acordar qué cosas le parecían bien y cuáles quería que cambiara. Esto lo hemos hecho mediante la realización de videos explicativos y el correo electrónico.

Creo que este método es de los más adecuados debido a la situación en la que se ha desarrollado este TFG, ya que con el motivo de la pandemia Covid-19, tanto alumnos como profesores, hemos estado más ocupados y disponíamos de menos tiempo.

1.11 Planificación temporal

La planificación temporal que llevé a cabo durante el curso se basó en el siguiente diagrama de forma más o menos exacta:

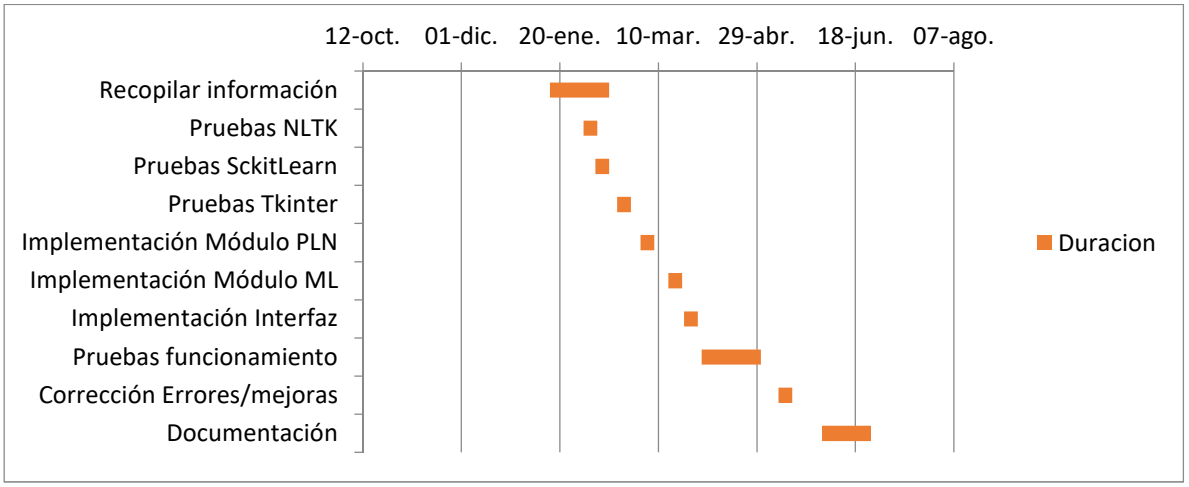


Ilustración 7 Diagrama de Gantt

2 DISEÑO INICIAL

El fundamento básico de todo este tipo de aplicaciones se basa en tres pasos claramente diferenciados.

El primer paso se basa en la depuración de los datos de entrada. Los conjuntos de datos dados pueden estar compuestos por muchos tweets, no estructurados, que deben ser preprocesados para hacer un modelo de PNL. En este proyecto probamos las siguientes técnicas de preprocesamiento:

-Eliminación de signos de puntuación.

-Eliminación de palabras de uso común (Stopwords).

-Normalización de palabras.

Los signos de puntuación siempre serán una perturbación en PNL, especialmente los hashtags y "@" juegan un papel importante en los tweets. Las puntuaciones excluidas y otras notaciones inusuales se eliminarán con expresiones regulares y la ayuda de NLTK.

El siguiente paso será la vectorización y selección del modelo. Antes de entrenar con los datos, tenemos que representar numéricamente los datos preprocesados. En este caso voy a usar la técnica CountVectorization.

CountVectorization genera una matriz dispersa que representa todas las palabras del documento, como aparecen en esta imagen.

document	w1	w2	w3	w4	...	wn	sentiment
d1	2	1	3	1		1	positive
d2	1	5	5	5		1	negative
d3	3	8	6	8		2	positive
d4	2	5	1	5		3	positive
d5	3	0	3	0		0	negative
d6	0	0	0	0		0	negative
d7	2	0	0	0		0	positive
d8	9	2	9	2		2	negative
...	...	...	...	...	...	...	...

Ilustración 8 Ejemplo vectorización

Bag of Words (BoW) o CountVectorizer describe la presencia de palabras dentro del texto. Da un resultado de 1 si está presente en la oración y 0, si no está presente. Por lo tanto, crea una bolsa de palabras con un recuento de matriz de documentos en cada documento de texto.

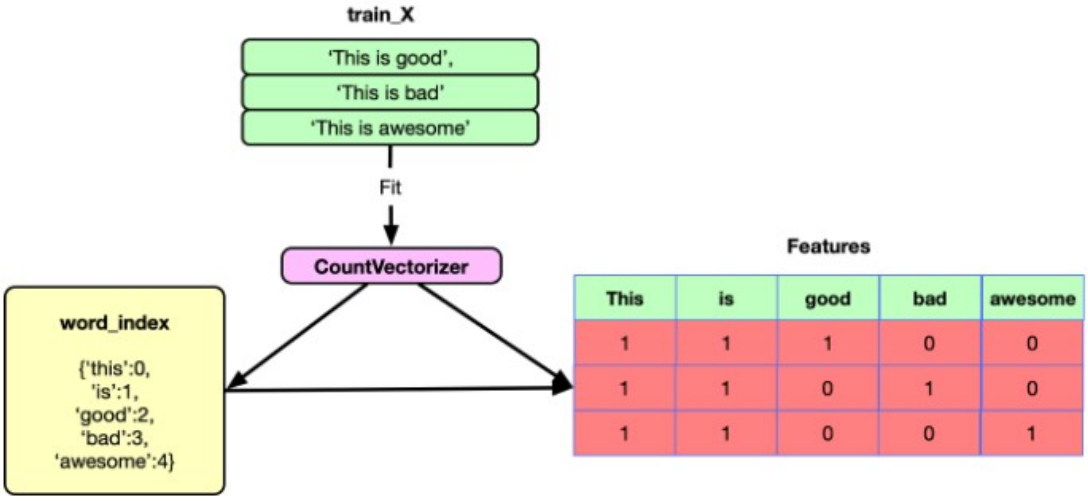


Ilustración 9 Ejemplo countVectorization

El siguiente paso, una vez ya tenemos el conjunto de datos vectorizado, va a ser crear los sets de entrenamiento y testeo. En este caso, uno de los requisitos del proyecto era realizar un Cross-Validation que usaremos para dividir el conjunto de datos.

Cross-Validation se utiliza principalmente para estimar modelos de ML. Hace uso de una muestra limitada para estimar cómo se espera que el modelo funcione en general cuando se use para hacer predicciones sobre datos que no se usaron durante el entrenamiento del modelo.

Es un método popular porque es fácil de entender y porque generalmente da como resultado una estimación menos sesgada o menos optimista de la habilidad del modelo que otros métodos. En mi caso, haré una división simple de entrenamiento / prueba con la utilidad que me proporciona la biblioteca Sckitlearn.

En esta aplicación el porcentaje de datos que usaremos para entrenar podrá ser elegido por el usuario.

Una vez hecho todo lo anterior, el paso final será entrenar y validar los algoritmos.

Teniendo en cuenta todo lo anterior, y con la visión de querer una aplicación gráfica interactiva, se creó un diseño inicial tal que:

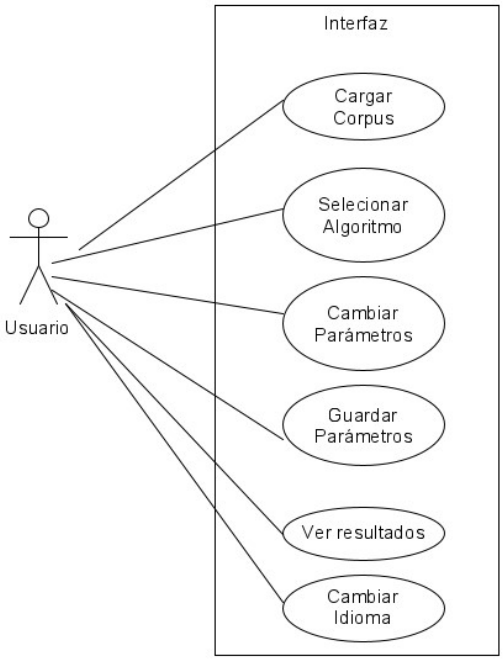


Ilustración 10 Diagrama de caso de uso inicial

Cargar corpus: El usuario podía cargar un corpus especificando en un campo de texto la ruta del mismo. En caso de que esa ruta fuera errónea, aparecerá un mensaje de error al tratar de abrir un fichero inexistente. El formato de este fichero se decidirá en iteraciones posteriores. Este paso incluye los 2 primeros pasos mencionados anteriormente, (preprocesamiento y creación del BoW).

Seleccionar algoritmo: El usuario podrá elegir entre los distintos algoritmos para entrenarlos.

Cambiar parámetros: El usuario podrá elegir los parámetros que desee para ejecutar los algoritmos.

Guardar parámetros: El usuario podrá guardar los parámetros con los que ejecutó el algoritmo, el formato de estos datos se especificará más adelante.

Ver resultados: El usuario podrá validar el entrenamiento de los algoritmos contra el test de prueba generado por el Cross-Validation.

2.1 Especificaciones y diseño del sistema

El diseño se ha basado en la implementación de tres módulos independientes que finalmente hacen llamadas unos a los otros.

El sistema fundamental será la interfaz, que será el que incorpore finalmente a los demás módulos.

El módulo de ML se encargará principalmente de crear los sets, entrenar los algoritmos y devolver los parámetros usados y los resultados finales.

El módulo de PLN se encargará del preprocesamiento del texto, eliminar puntaciones, tokenizar etc...

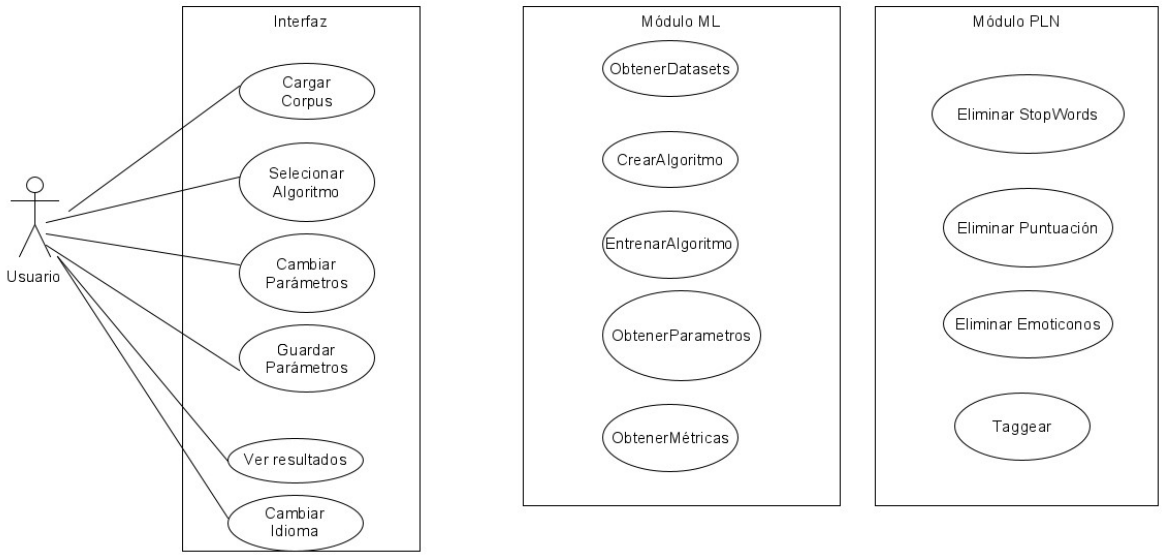


Ilustración 11 Diagrama de casos de uso

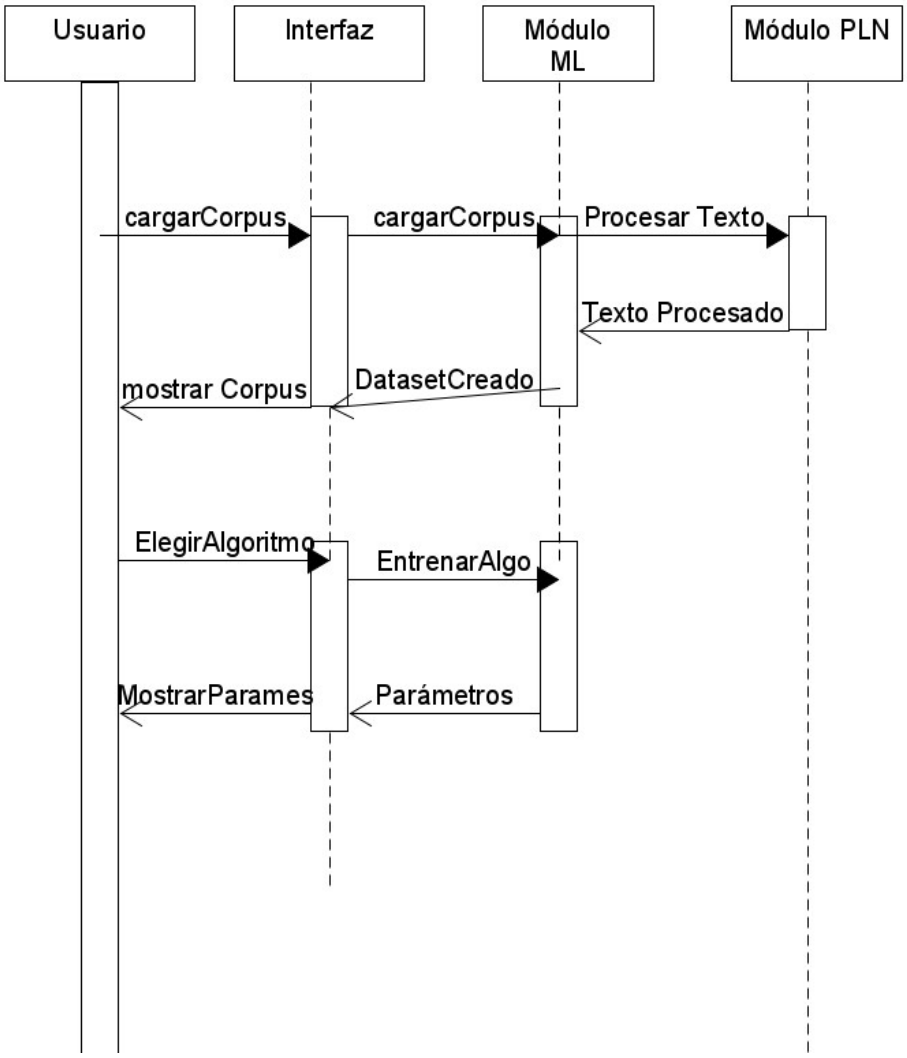


Ilustración 12 Diagrama de secuencia

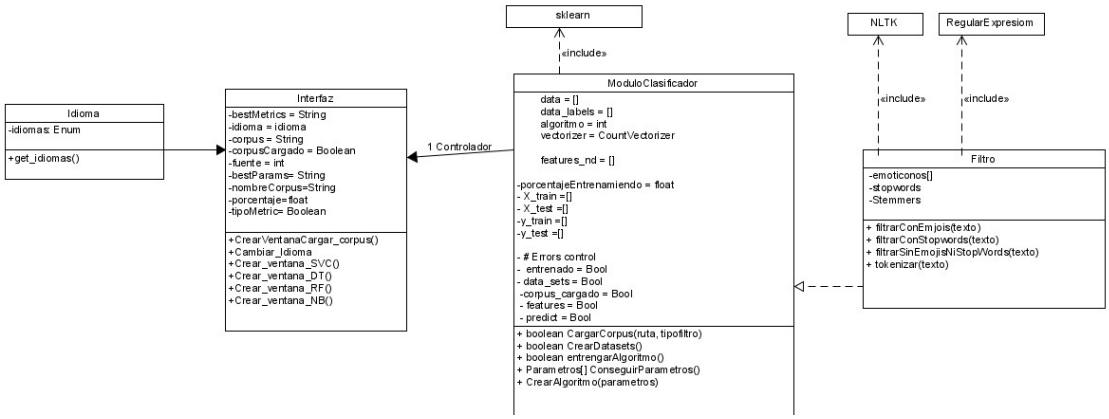


Ilustración 13 Diagrama de clases

3 DESARROLLO

3.1 Primera Iteración

Durante los primeros meses, el trabajo que hice fue puramente de investigación. Se basó principalmente en recopilar información sobre el PLN, el SA, y qué bibliotecas usaría para realizar el trabajo, etc…

Una vez tuve claro con qué iba a realizar cada cosa, comencé a probar con programas muy básicos de ejemplo de ML, en los que principalmente probaba a ejecutar algoritmos con dataset de prueba de la biblioteca para ver cómo funcionaba.

El mecanismo era simple, solo había que elegir un dataset, escoger el modelo y hacer el fit, con esto ya teníamos el modelo del algoritmo que quería entrenado y listo para obtener los resultados y sus parámetros.

El proceso de entrenamiento de un algoritmo sería:

1. Seleccionar el algoritmo que queremos.
2. Ajustar los parámetros del algoritmo.
3. Hacer el fit (entrenar y validar).
4. Obtener los resultados.

Todo este proceso se realiza gracias a la utilidad que nos proporciona la biblioteca Sckit-learn la cual traduce este proceso en:

1. Construir un algoritmo con los parámetros que queramos (todos son objetos con constuctores).
2. Llamar al método Fit pasando como parámetro el conjunto de entrenamiento.
3. Calcular los resultados llamando a los métodos que nos calculan las estadísticas que necesitemos pasando como parámetros el conjunto de datos de prueba.

El siguiente desafío fue cómo hacer para que los datasets fueran los míos y no los de prueba de la biblioteca, esto se resolverá en la siguiente iteración.

De forma paralela a las pruebas de ML durante esta iteración, tuve el primer contacto con la creación de interfaces en Python con Tkinter.

Esta biblioteca proporciona idea muy básica para la creación de aplicaciones gráficas. Su núcleo se basa en la creación de una pestaña principal y un bucle de actualización de la misma.

Lo primero que haremos será crear la pestaña principal mediante su constructor Tk() . Con esto ya tendremos una pestaña vacía.

El siguiente paso será añadir a lo que esta biblioteca llama widgets, que son los distintos botones, campos de texto, etc… Esto lo haremos simplemente llamando al constructor del widget en concreto y pasándole los parámetros con los que queramos que se cree, además de pasarle el objeto de la ventana a la cual pertenece. Y por último, llamando al método place() del widget ,lo colocaremos donde queramos en la ventana.

Tkinter tiene varias formas de colocar los widgets en sus ventanas, la más útil, sería la de la división en forma de malla (grid), la cual nos ayuda a colocar widgets simplemente especificando en que fila y columna lo queremos ubicar.

Como quería una aplicación simple, decidí crear un menú con una serie de botones que crearan nuevas ventanas, que supuso el nuevo reto de cómo conseguir que los botones funcionaran, y el cómo ubicarlo en una barra de menú.

El funcionamiento de los botones es simple, funciona como un widget normal, al que le podemos asignar una función que se llamará cuando éste sea pulsado.

Mientras que la barra de menús era un conjunto de objetos de tipo Menús, que a su vez son una agrupación de objetos de tipo Button.

El último paso de esta versión fue crear nuevas pestañas que permitieran configurar los parámetros mediante cajas de texto.

Para ello hago uso de la función topWindow() que crea una nueva ventana. Este será el mecanismo principal de los menús.

Y el primer diseño de la interfaz sería el siguiente:

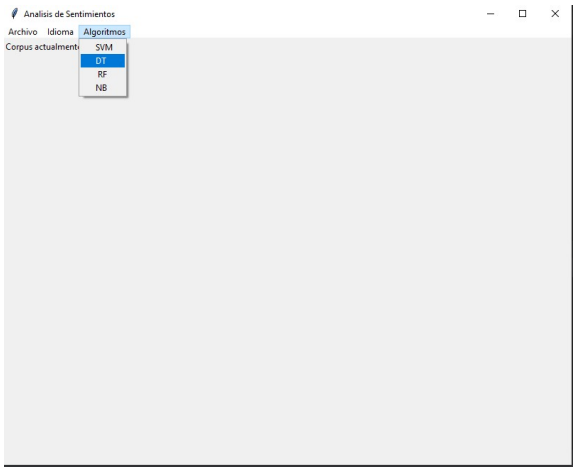


Ilustración 14 Interfaz original 1

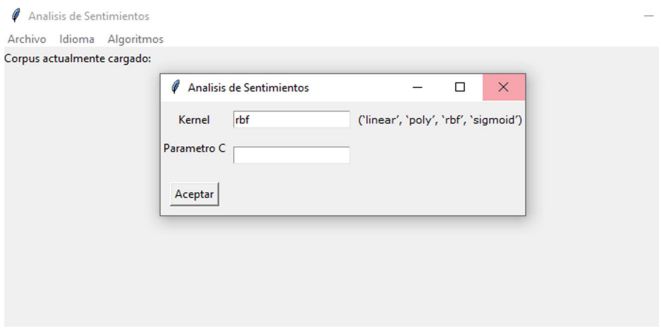


Ilustración 15 Interfaz original 2

Como vemos, la interfaz en su primera versión, permitía ejecutar un algoritmo del tipo SVM pudiendo cambiar solamente sus parámetros Kernel y C.

Además, se mostraba en la pestaña principal el corpus que estaba actualmente cargado.

Esta interfaz primigenia no contaba con validación de datos, ni con múltiples ejecuciones por lo que había que acceder a este menú para realizar varias pruebas con cada algoritmo y había que introducir todos los parámetros de forma manual.

Además, tenía diversos fallos, como que podías acceder al menú de algoritmos sin haber cargado un corpus, o que el foco de las nuevas ventanas no se mantenía, es decir, podías volver a la pestaña principal mientras tenías abierta la de configuración de parámetros.

En resumen, durante esta iteración hice pruebas para comprobar funcionamiento de los algoritmos de ML, y fue una toma de contacto a cómo crear interfaces con Tkinter.

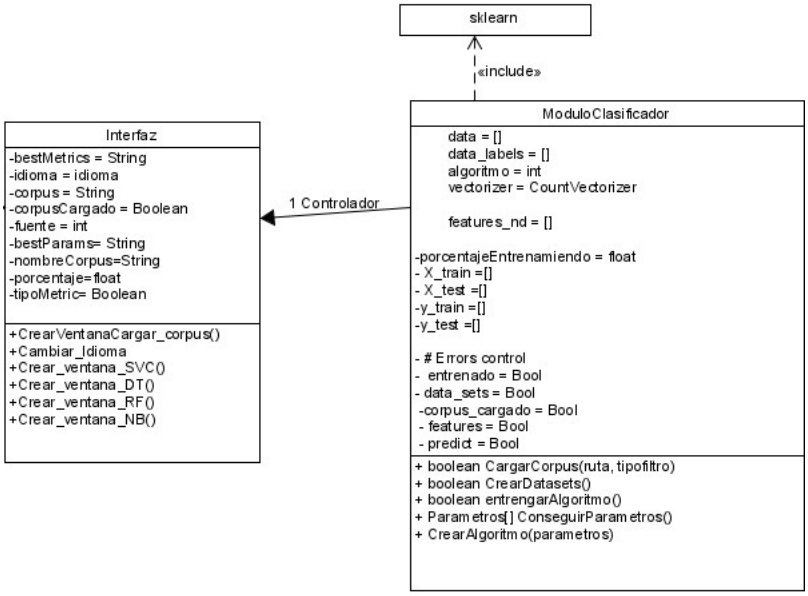


Ilustración 16 Diagrama de clases inicial

3.2 Segunda Iteración

En esta segunda iteración añado la parte de procesamiento de texto y la de idiomas, además de corregir diversos fallos que había en la interfaz anterior, y habilito la posibilidad de utilizar nuestros propios corpus de datos.

Lo primero en lo que me centré fue en tratar de cargar mis propios corpus, para ello comencé a crear el módulo de PLN al que llamé filtro (su utilidad principal es filtrar palabras). Este módulo lo que hace simplemente es, dado un texto, eliminar las palabras que no queramos que influyan en las técnicas de ML.

Esto lo podemos lograr mediante el uso de la biblioteca NLTK y RE, la primera aporta utilidades para dividir el texto en palabras simples, hacer que todo el texto sea minúscula e incluso eliminar palabras que no aportan información

(stopwords). Mientras que la segunda la usamos para crear expresiones regulares, que nos son de utilidad para eliminar emoticonos, enlaces etc…

Además, intenté hacer tagging a las palabras, es decir, clasificarlas en sustantivo, adjetivo, adverbio…, pero por problemas de memoria en mi ordenador no conseguí que terminara de procesar el dataset completo, por lo que decidí eliminarlo.

Una vez logré que el módulo de preprocesamiento de texto fuera funcional, me puse con el problema de hacer que mis corpus fueran compatibles con la biblioteca de ML. Fue entonces cuando investigué la biblioteca de ML, vi que tenía ciertos apartados para trabajar con SA y encontré la utilidad CountVectorization (explicado en la parte de Diseño).

Una vez hecho esto teníamos la información compatible con los algoritmos de ML, y ya era cuestión de crear los distintos sets de entrenamiento y prueba con el Cross-Validation.

En esta versión de la aplicación lo que hice fue sumar a la interfaz un botón con el que una vez cargado el corpus, procesaba el texto quitándole stopwords, emoticonos, puntuaciones e información poco útil como enlaces y # de twitter. No daba opción a otro tipo de procesamiento.

Con esto resuelto, ya tendríamos una aplicación funcional a la que le pasamos un dataset, le aplicamos el procesamiento de texto, entrenamos el modelo con los parámetros que queramos y obtenemos los resultados.

En esta versión se añadieron el resto de algoritmos (DT, RF y NB). Y se corrigieron los errores de foco de las pestañas gracias a la utilidad de Tkinter grabfocus().

3.3 Tercera Iteración

En esta versión se me propusieron los siguientes cambios:

1. El tagging no era necesario hacerlo.
2. Para entrenar los algoritmos había que hacer un número determinado de ejecuciones con unos parámetros distintos por cada ejecución, y había que hacerlo de forma metódica.

3. El usuario puede guardar el resultado del entrenamiento de los algoritmos.

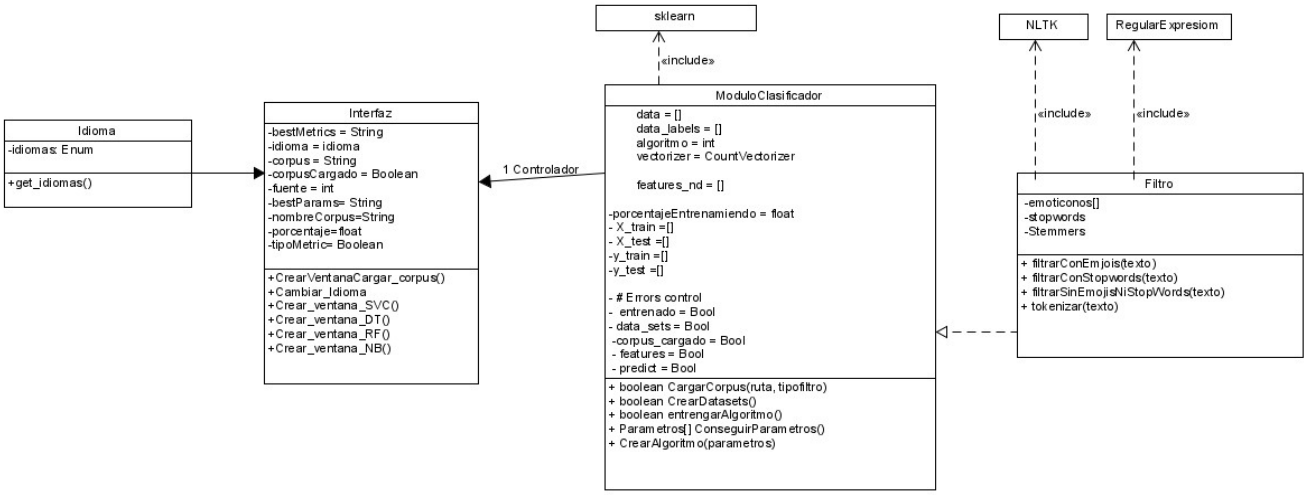


Ilustración 17 Diagrama de clases final

El tercer punto fue implementado añadiendo en el menú de archivo un botón de guardado, similar al de cargar el corpus, pero en modo de escritura con los parámetros de los algoritmos y sus resultados.

En cuanto a las múltiples ejecuciones, opté por poner un rango en los distintos parámetros (min mid max), que se probaran todos con todos de forma secuencial, quedándome al final con la combinación que mejores resultados proporcionara.

Simplemente hice un bucle que pasaba por todas las combinaciones de todos los parámetros, y por cada iteración del bucle se crea un algoritmo con una combinación de parámetros específica y se entrena. En esta versión los resultados solo se calculaban a la parte positiva del dataset, y el mejor resultado se consguía como la mayor de las sumas de los resultados.

3.4 Iteración final

Para esta iteración se me propusieron los siguientes cambios:

1. El menú debe tener los siguientes apartados:

   -Archivo

   -Idioma

   -Algoritmos

   -Ejecuciones

   -Ayuda

2. Cambiar la ventana de cargar el corpus.
3. Una vez cargado el corpus, poner su nombre en la parte superior imitando a un programa como Word.
4. El preprocesamiento del texto podría ser opcional.
5. El botón de ayuda, podría tener dos partes. En una tendría un pequeño manual de trabajo, en la otra parte, descripción del autor, correo, etc…
6. El formato de las estadísticas ha de ser el siguiente:

```
******DECISION TREE******
Nombre del corpus: corpus.csv
Valor de los parámetros:
    -Criterio: entropy
    -min_samples_split: 20
    -min_samples_leaf: 5
Accuracy Score : 81.19%
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       324
           1       0.59      0.59      0.59        96

    accuracy                           0.81       420
   macro avg       0.73      0.74      0.73       420
weighted avg       0.81      0.81      0.81       420
```

Ilustración 18 Salida de datos

7. A La clase 0 (Negativo) y clase 1(Positivo), calcular recall, precision, f1 y las macros-R, macro-P, macro-F1 y accuracy.
8. Tomar como mejores resultados macro-F1 o accuracy.

9. Al guardar los parámetros se usará por defecto el nombre del corpus actualmente cargado, y si este fichero de resultados ya existe, no se sobrescribirá, si no que se añadirá al final del fichero.

Estos cambios no afectaron en nada a lo que sería el diseño de clases de la iteración anterior, simplemente supuso la modificación de algunos métodos del módulo de ML y la realización de unos ajustes en la interfaz.

Enlace al video de esta versión:

https://drive.google.com/open?id=1A9CILC6_Zsve_ZIf4Ng9oz9MJ0Yfgn9z

3.4.1 Pruebas de verificación del sistema

El sistema dispone de gran robustez, mostrando diversos errores e informando en dónde se ha equivocado el usuario al introducir datos mediante la validación de campos.

Esta aplicación ha sida testeada exclusivamente por mí en mi ordenador, por lo que los resultados de las experimentaciones son algo limitados.

La aplicación soporta bien corpus de entre 500-1000 opiniones. A partir de este límite, dependiendo de la longitud de las opiniones y del número máximo de features encontrado por corpus, la aplicación comenzará a congelarse, aunque seguirá trabajando, aun así, en segundo plano.

Ajustando el parámetro de Max_Features podremos conseguir una mayor fluidez de la aplicación.

La aplicación muestra los resultados por pantalla en el formato acordado y también podemos guardarlos en fichero.

3.4.2 Validación de la usabilidad del sistema

El sistema cuenta con atajos de teclado para abrir las pestañas más usadas como son cargar corpus y guardar parámetros. Además, todas las pestañas de los parámetros de los algoritmos pueden ser ejecutadas pulsando la tecla [Enter] y, podemos pasar de un campo de input a otro mediante la tecla [TAB].

Dispone también de un botón para cambiar el idioma. Es muy fácil añadir nuevos idiomas, ya que la aplicación se diseñó desde un principio pensando en ello.

4 EXPERIMENTACIÓN, RESULTADOS Y DISCUSIÓN

Con respecto a las experimentaciones, han de tenerse en cuenta dos factores, los parámetros de cada algoritmo, y las métricas que vamos a evaluar de cada uno de ellos.

En cuanto a los parámetros que vamos a modificar de los algoritmos nos vamos a centrar en:

-SVM:

1. Parámetro C: Controla la compensación entre el límite de decisión suave y la clasificación correcta de los puntos de entrenamiento. Un valor grande de c significa que obtendrás más puntos de entrenamiento correctamente.
2. Kernel: Tipo de kernel que va a usar el algoritmo (lineal, polinómico, sigmoide, función de base radial (RBF)).
3. Gamma: Coeficiente del Kernel, define hasta dónde llega la influencia de un solo ejemplo de entrenamiento. Si tiene un valor bajo, significa que cada punto tiene un alcance lejano y, por el contrario, un valor alto de gamma, significa que cada punto tiene un alcance cercano. Si gamma tiene un valor muy alto, entonces el límite de decisión dependerá de los puntos que están muy cerca de la línea, lo que efectivamente hace que se ignoren algunos de los puntos que están muy lejos del límite de decisión. Esto se debe a que los puntos más cercanos obtienen más peso. Por otro lado, si el valor gamma es bajo, incluso los puntos lejanos obtienen un peso considerable y obtenemos una curva más lineal.

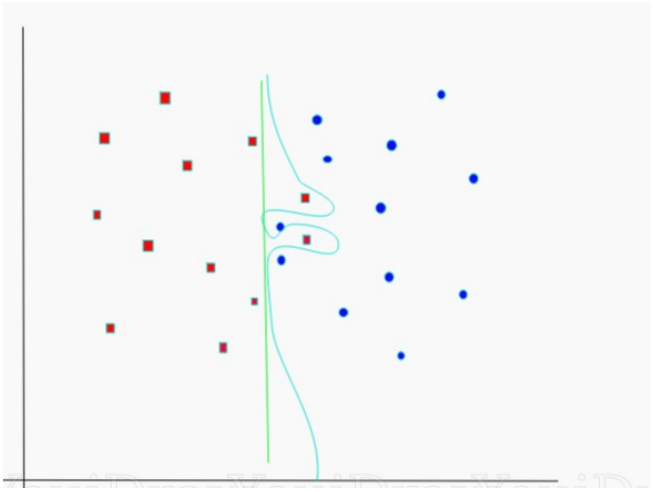


Ilustración 19 Límite de decisión suave vs clasificar correctamente

-DT:

1. Criterion: Función para medir la calidad de una división. Los criterios admitidos son "gini" para la impureza de Gini, y "entropía", para la ganancia de información.
2. Splitter: La estrategia utilizada para elegir la división en cada nodo. Las estrategias admitidas son "mejor" para elegir la mejor división, y "aleatorio," para elegir la mejor división aleatoria.
3. Max_depth: La profundidad máxima del árbol. Si es ninguno, los nodos se expanden hasta que todas las hojas sean puras o hasta que todas las hojas contengan menos de min_samples_split.
4. Min_samples_split: El número mínimo de muestras necesarias para dividir un nodo interno.

-RF:

Hace uso de los mismos parámetros que el DT añadiendo:

1. N_stimators: Número de árboles.

-MultinomialNB:

1. Alpha: Parámetro de suavizado adicional. Si el valor es mayor que 0, tiene en cuenta las muestras que no están presentes en el conjunto de aprendizaje y evita probabilidades de cero en los cálculos posteriores. La

configuración se denomina suavizado de Laplace si es igual a uno, mientras que se le llama suavizado de Lidstone, si es menor que uno.

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

-GaussianNB:

1. Parte de la mayor variación de todas las características que se agrega para la estabilidad del cálculo.

Por otro lado, los factores (métricas) que vamos a medir son:

1. Accuracy Score: Precisión total de la clasificación.
2. Precision: La precisión es intuitivamente la capacidad del clasificador de no etiquetar como positiva una muestra que es negativa.

   Sigue la relación tp / (tp + fp)

   Donde tp es el número de positivos verdaderos y fp, el número de falsos positivos.
3. Recall: Capacidad del clasificador para encontrar todas las muestras positivas. Es la relación tp / (tp + fn) donde tp es el número de verdaderos positivos y fn, el número de falsos negativos.
4. F1_Score: Se puede interpretar como un promedio ponderado de Precision y recall. Sigue la fórmula:

   F1 = 2 * (precision * recall) / (precision + recall)
5. Support: Es el número de ocurrencias de cada clase en los valores correctos.

4.1 Experimentaciones y pruebas

Conforme la aplicación se iba desarrollando, se iban realizando distintas pruebas para comprobar cómo afectaban los distintos parámetros a los resultados de los algoritmos.

Durante la primera iteración obtuve los siguientes resultados:

Algoritmo	Accuracy	F1	Precision	Recall
SVC(Kernel=rbf gamma = scale)	0.819047619047619	0.8204158790170132	0.8410852713178295	0.8007380073800738
SVC(Kernel=linear)	0.820952380952381	0.8206106870229007	0.849802371541502	0.7933579335793358
SVC(Kernel=poli 3 gamma = scale)	0.5447619047619048	0.2691131498470948	0.7857142857142857	0.16236162361623616
SVC(Kernel=poli 5 gamma = scale)	0.5009523809523809	0.09027777777777779	0.7647058823529411	0.04797047970479705
SVC(Kernel = Sigmoid gamma = scale	0.7828571428571428	0.7764705882352942	0.8284518828451883	0.7306273062730627
RF(Trees= 100)	0.7923809523809524	0.7985212569316081	0.8	0.7970479704797048
RF(Trees= 300)	0.7866666666666666	0.7963636363636363	0.7849462365591398	0.8081180811808119
RF(Trees= 100 , criterion=entropy)	0.7714285714285715	0.7727272727272728	0.7937743190661478	0.7527675276752768
NB Multinomial	0.8095238095238095	0.8175182481751825	0.8086642599277978	0.8265682656826568
NB Multinomial alpha =0	0.7295238095238096	0.760942760942761	0.6996904024767802	0.8339483394833949
NB Multinomial FitPrior = False	0.8076190476190476	0.8153564899451554	0.8079710144927537	0.8228782287822878
DT default	0.6361904761904762	0.6389413988657845	0.6550387596899225	0.6236162361623616
DT entropy	0.6323809523809524	0.6266924564796906	0.6585365853658537	0.5977859778597786
DT spliter = random	0.64	0.6493506493506495	0.6529850746268657	0.6457564575645757

Tabla 1 Primeros resultados

Aquí no tenía aún demasiada idea de cómo funcionaban los algoritmos, ni qué significaban exactamente estos parámetros ya que además solo se mostraban los resultados de la clase positiva.

Estas pruebas se realizaron con el dataset completo de opiniones de cine sin tener ningún tipo de preprocesamiento en el texto. Aun así, ya se puede notar por valores como el accuracy, qué tipo de algoritmo va a resultar mejor para los datasets y qué parámetros dan mejores resultados.

Podemos observar como la gran mayoría llega a acercarse al 80% de acierto, siendo la única excepción el DT, que como comenté en su sección es necesario que para que no generalice y llegue a un sobreajuste, implementar un mecanismo de poda que no usé en estas pruebas.

También es curioso ver la gran diferencia que existe entre los distintos kernels del SVC ya que hay una gran diferencia entre el linear, (el que mejores resultados me ha proporcionado casi siempre), y el de tipo polinomio, del que podemos configurar el grado, cosa que no hice en esta experimentación.

Pruebas de tiempo:

El orden de rapidez a la hora de entrenar algoritmos sería el siguiente:

NB<DT<RF<SVM

Pruebas realizadas con datasets de 250 opiniones

Número Ejecuciones/tiempo(s)	1	10	50
NB	0.7s	9.3s	47.3s
DT	1.2s	13.23s	57.3s
RF	4.10s	20.33s	76.12s
SVM	11.2s	130.4s	603.2s

Tabla 2 Mediciones de tiempos

Como vemos, existe una gran diferencia entre los distintos tiempos de los diferentes algoritmos, llegando algunos incluso a durar varios minutos con un número bastante reducido de opiniones. Por esto, se implementó un apartado para ejecutar algoritmos con menos iteraciones.

Resultado de búsqueda de mejores parámetros:

****SVM****

Valor de los parámetros:

-C parám: 0.01

-Kernel: linear

-Gamma: 0.1

Accuracy Score: 80.0%

	precision	recall	F1_socre	support
0	0.87	0.82	0.84	33
1	0.68	0.76	0.72	17
Accuracy			0.80	50
Macro avg	0.78	0.79	0.78	50
Weighted-avg	0.81	0.80	0.80	50

Tabla 3 Mejores parámetros SVM

****DT****

Valor de los parámetros:

-Criterion: gini

-Splitter: best

-Max_Depth: None

-Min_Samples_split: 2

Accuracy Score: 78.0%

	precision	recall	F1_socre	support
0	0.84	0.82	0.83	33
1	0.67	0.71	0.69	17
Accuracy			0.78	50
Macro avg	0.76	0.76	0.76	50
Weighted avg	0.78	0.78	0.78	50

Tabla 4 Mejores parámetros DT

****** RANDOM FOREST ******

Valor de los parámetros:

-N_Árboles: 400

-Criterion: entropy

-Max_Depth: None

-Min_Samples_split: 3

Accuracy Score: 82.0%

	precision	recall	F1_socre	support
0	0.80	0.97	0.88	33
1	0.90	0.53	0.67	17
Accuracy			0.82	50
Macro avg	0.85	0.75	0.77	50
Weighted avg	0.83	0.82	0.81	50

Tabla 5 Mejores parámetros RF

****** MULTINOMIAL NAIVEBAYES ******

Valor de los parámetros:

-Alpha: 0.1

Accuracy Score: 84.0%

	precision	recall	F1_socre	support
0	0.88	0.88	0.88	33
1	0.76	0.76	0.76	17
Accuracy			0.84	50
Macro avg	0.82	0.82	0.82	50
Weighted avg	0.84	0.84	0.84	50

Tabla 6 Mejores parámetros MNB

****** GAUSSIAN NAIVEBAYES ******

Valor de los parámetros:

-Smoothing: 0.01

Accuracy Score: 76.0%

	precision	recall	F1_socre	support
0	0.73	1.00	0.85	33
1	1.00	0.29	0.45	17
Accuracy			0.76	50
Macro avg	0.87	0.65	0.65	50
Weighted avg	0.82	0.76	0.71	50

Tabla 7 Mejores parámetros GNB

	Accuracy texto crudo	Accuracy texto procesado
SVM	0.79	0.75
DT	0.81	0.79
RF	0.83	0.82
GNB	0.65	0.60
MNB	0.80	0.80

Tabla 8 Comparación tweets

	Accuracy texto crudo	Accuracy texto procesado
SVM	0.81	0.80
DT	0.76	0.79
RF	0.80	0.82
GNB	0.73	0.76
MNB	0.84	0.84

Tabla 9 Comparación reviews de cine

4.2 Resultados y discusión

Como podemos observar, todos los algoritmos consiguen una precisión cercana al 80% haciendo uso de datasets pequeños (300-500 opiniones), realizando entre 30-50 por algoritmo, variando todos los parámetros de forma iterativa. Ajustando los parámetros cercanos al mejor, conseguiríamos los mejores resultados.

En proporción accuracy/tiempo entrenamiento, el mejor algoritmo sería el multinomialNB.

Las últimas pruebas que hice fueron comprobando los distintos resultados de mismos datasets pero con distintos preprocesamientos del texto.

Resultó bastante curioso. Con respecto a los datasets con tweets, daban mejores resultados el texto crudo que el texto procesado, mientras que era al contrario con las reviews de cine.

Estas pruebas se realizaron eliminando tanto stopwords como emoticonos, y el resultado tiene cierta lógica ya que en los tweets, podemos encontrar información útil en los emoticonos a la hora de polarizar una oración, mientras que en las reviews de cine, lo óptimo es quitar las stopwords que lo único que hacen es interferir a la hora de encontrar las características clave de un texto.

5 CONCLUSIONES Y TRABAJOS FUTUROS

La rama de la minería de datos en general me ha sorprendido para bien. Creo que la aplicación es útil y cumple con los requerimientos que se me propusieron.

Este trabajo era un nuevo reto ya que no tenía demasiados conocimientos sobre los temas que trataba, pero todo lo referido a ML me gustaba ya que son ramas que en mi opinión van a ser muy importantes en un futuro muy cercano.

La curva de aprendizaje del ML me parece alta, ya que al principio no sabía cómo comenzar y tuve que empezar desde lo más básico mirando videos y tutoriales para comprender el funcionamiento de los algoritmos.

En general, el tema del procesamiento de datos automáticos me ha parecido de gran utilidad y creo que podría ser una de las ramas en las que me centre en el futuro.

Algunas posibles mejores que se podrían implementar para esta aplicación son:

-Mejorar la estética de la interfaz.

-Añadir más tipos de algoritmos.

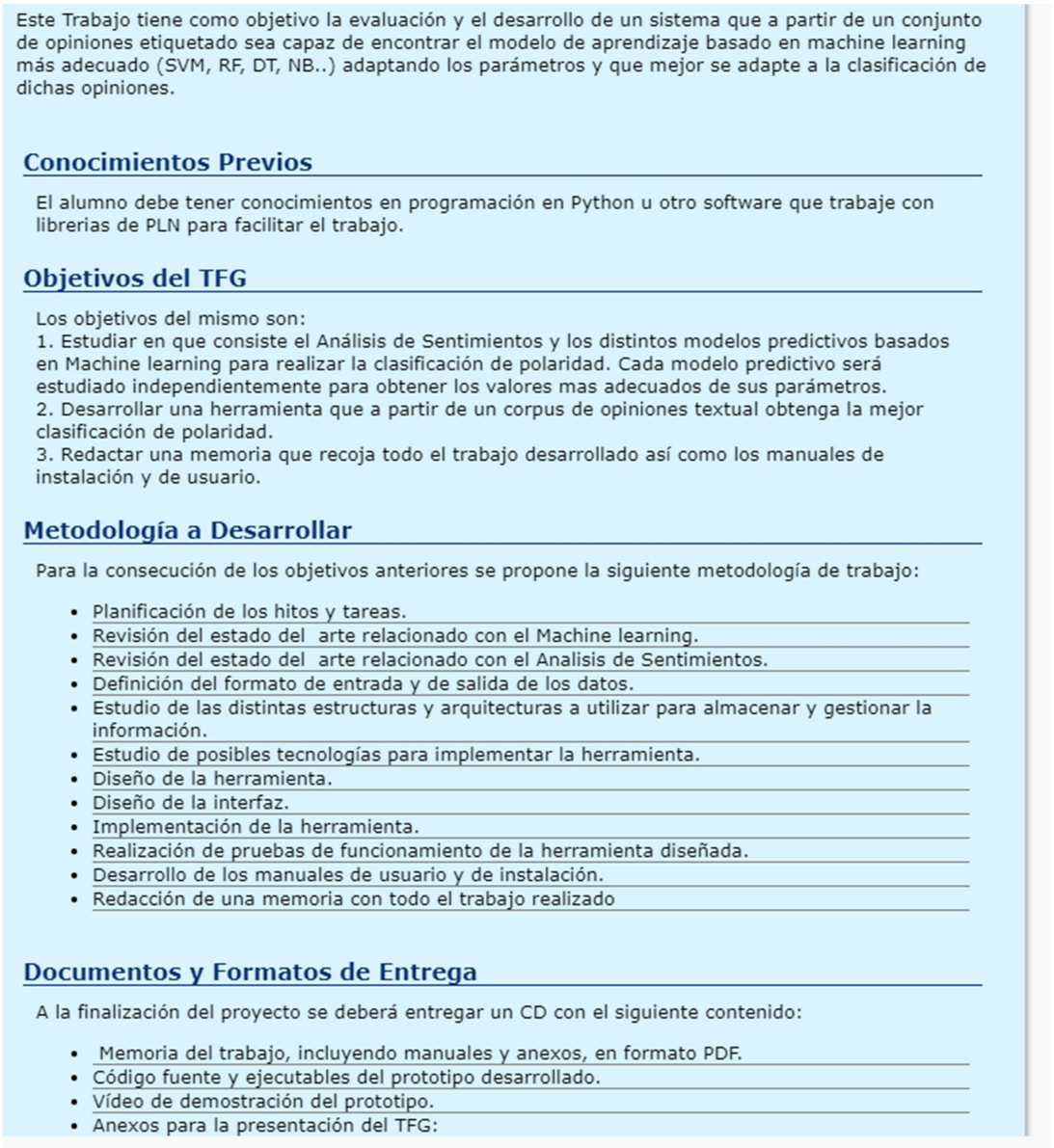
-Añadir más parámetros de configuración.

-Hacer la aplicación compatible con más formatos de entrada.

-Paralelizar la ejecución de los algoritmos en varios hilos.

6 APÉNDICES

6.1 Guía original del Trabajo Fin de Título



Este Trabajo tiene como objetivo la evaluación y el desarrollo de un sistema que a partir de un conjunto de opiniones etiquetado sea capaz de encontrar el modelo de aprendizaje basado en machine learning más adecuado (SVM, RF, DT, NB..) adaptando los parámetros y que mejor se adapte a la clasificación de dichas opiniones.

Conocimientos Previos

El alumno debe tener conocimientos en programación en Python u otro software que trabaje con librerias de PLN para facilitar el trabajo.

Objetivos del TFG

Los objetivos del mismo son:
1. Estudiar en que consiste el Análisis de Sentimientos y los distintos modelos predictivos basados en Machine learning para realizar la clasificación de polaridad. Cada modelo predictivo será estudiado independientemente para obtener los valores mas adecuados de sus parámetros.
2. Desarrollar una herramienta que a partir de un corpus de opiniones textual obtenga la mejor clasificación de polaridad.
3. Redactar una memoria que recoja todo el trabajo desarrollado así como los manuales de instalación y de usuario.

Metodología a Desarrollar

Para la consecución de los objetivos anteriores se propone la siguiente metodología de trabajo:

- Planificación de los hitos y tareas.
- Revisión del estado del arte relacionado con el Machine learning.
- Revisión del estado del arte relacionado con el Analisis de Sentimientos.
- Definición del formato de entrada y de salida de los datos.
- Estudio de las distintas estructuras y arquitecturas a utilizar para almacenar y gestionar la información.
- Estudio de posibles tecnologías para implementar la herramienta.
- Diseño de la herramienta.
- Diseño de la interfaz.
- Implementación de la herramienta.
- Realización de pruebas de funcionamiento de la herramienta diseñada.
- Desarrollo de los manuales de usuario y de instalación.
- Redacción de una memoria con todo el trabajo realizado

Documentos y Formatos de Entrega

A la finalización del proyecto se deberá entregar un CD con el siguiente contenido:

- Memoria del trabajo, incluyendo manuales y anexos, en formato PDF.
- Código fuente y ejecutables del prototipo desarrollado.
- Vídeo de demostración del prototipo.
- Anexos para la presentación del TFG:

6.2 Manual de instalación y configuración

La principal forma de crear un ejecutable en Python es mediante la utilidad pyinstaller, que incluye todas las dependencias y librerías que hayas utilizado en el proyecto.

En cuanto a la instalación de la aplicación existe un problema bien conocido de la biblioteca NLTK a la hora de intentar crear el ejecutable:

-https://stackoverflow.com/questions/54659466/NLTK-hook-unable-to-find-NLTK-data/54760437

Este problema se puede solucionar, pero al arreglarlo surgen otros los cuales generan el ejecutable pero hacen que la aplicación no se abra. No he podido encontrar la solución a este problema.

Por tanto, la alternativa que me queda para instalar la aplicación, se basa en instalar y descargar las bibliotecas que he utilizado de forma manual.

Para ello necesitaremos:

-Tener instalado Python https://www.Python.org/downloads/

-Abrir una consola de comandos y poner lo siguiente:

pip install NLTK

pip install scikit-learn

pip install Tkinter

pip install re

Por último, ejecutar la aplicación moviéndonos al directorio de instalación de la app y ejecutando:

Python interfaz.py

Adicionalmente, dejo unos archivos .bat que ejecutan automáticamente estos comandos en un sistema Windows.

6.3 Manual de usuario

Esta es una aplicación basada en ML que trata de encontrar los mejores parámetros dados un dataset/corpus para los siguientes algoritmos:

- SVM (Support Vector Machine).
- DT (Decision Tree).
- RF (Random Forest).
- NB (Naive Bayes).

Para un correcto uso de la aplicación el dataset de entrada deberá tener el siguiente formato:

ID|text|score (0-1).

El dataset ha de ser binario (las pruebas se han realizado con sets del tipo 0, 1 / 1,-1).

Guía de uso:

-1 Menú de Archivo:

1.1 En el menú de archivo tendremos disponible la opción de cargar un corpus.

Con el botón de cargar corpus se nos abrirá una pestaña en la que tendremos la opción de seleccionar qué corpus cargar. Una vez seleccionado el dataset que queramos, se nos proporcionará la opción del tipo de preprocesamiento que le queramos dar al texto ya sea:

-Texto plano tal y como está.

-Texto sin emojis.

-Texto sin stopWords.

-Texto sin emojis, ni stopwords.

Una vez cargado el dataset en el título de la aplicación, aparecerá el nombre del dataset con el que estemos trabajando.

1.2 La siguiente opción que tendremos en este menú será la de guardar los mejores parámetros. Esta opción estará deshabilitada hasta que al menos hayamos entrenado un algoritmo. Una vez hecho esto, este botón nos permite seleccionar una ruta en la que queramos guardar los parámetros, por defecto la ruta de guardado es ./resultados/nombredelcorpusactual.txt. Si seleccionamos un archivo que no exista se creará, mientras que si elegimos un archivo que ya existe, añadirá al final del archivo los nuevos parámetros

1.3 La última opción de este menú es la de configuración. Esta opción nos da acceso a la pestaña de configuración general de la aplicación. Aquí tendremos la opción de elegir cuál es el criterio de mejor nota por algoritmo que puede ser o bien accuracy o bien macro_f1. También podremos seleccionar el porcentaje del cross- validation que usamos para entrenar los algoritmos y el número máximo de feautres por archivo.

-2 Menú de Algoritmos: Este menú estará deshabilitado hasta que se haya cargado al menos un corpus correctamente. Es la funcionalidad principal de la aplicación, la cual permite entrenar los distintos algoritmos.

2.1 SVM Abre la pestaña de parámetros del SVM que consta de:

-Parámetro C (estrictamente positivo).

-Parámetro Gamma (estrictamente positivo).

-Kernel [linear, poly, rbf, sigmoid].

2.2 DT Abre la pestaña de parámetros del Decision Tree:

-Profundidad (estrictamente positivo) o None (solo se puede poner None en el primer recuadro de profundidad).

-Min Samples (estrictamente positivo).

-Splitters [best, random].

-Criterion [gini, entrophy].

2.3 RF Abre la pestaña de parámetros del Random Forest:

-Profundidad (estrictamente positivo) o None (solo se puede poner None en el primer recuadro de profundidad).

-Min Samples (estrictamente positivo).

-Número de Árboles (estrictamente positivo).

-Criterion [gini, entrophy].

2.4 MultNB Abre la pestaña de parámetros del multinomial Naive Bayes:

-Suavizado (estrictamente positivo).

2.5 GaussianNB Abre la pestaña de parámetros de la Gaussian Naive Bayes:

-Suavizado (estrictamente positivo).

Todos estos algoritmos tienen puestos unos parámetros por defecto que son los recomendados por la biblioteca de ML.

Además, los primeros tres algoritmos tienen más parámetros que no son configurables por esta aplicación debido a la falta de potencia del ordenador.

-3 Menú de Ejecución: En este menú podremos ejecutar los algoritmos SVM, DT y RF con un menor número de ejecuciones aligerando así la carga de trabajo (Solo están estos porque los demás ya son ligeros de base).

-4 Menú de Ayuda:

Manual de uso: Abre este mismo manual de usuario.

Sobre mí: Abre una pestaña con información del autor.

7 DEFINICIONES Y ABREVIATURAS

Lista de abreviaciones:

-PLN: Procesamiento del lenguaje natural

-IA: Inteligencia Artificial

-NLTK: Natural Languaje processing

-NB: Naive-Bayes

-SVM: Support Vector Machine

-DT: Decision Tree

-RF: Random Forest

-ML: MachineLearning

-SA: Sentiment Analysis/Análisis de sentimientos

-BoW: Bag of Words

Definiciones:

-Tokenizar: Dividir el texto en palabras simples llamadas tokens, eliminando así puntuaciones.

-Features: Características, palabras clave de un texto en las que nos centramos en los algoritmos de ML.

-Stemmer: Conseguir el estema de una palabra. Ejemplo: Panadería-> Pan

-Tagging: Clasificar las palabras por el tipo que son: sustantivo, adjetivo, adverbio, etc...

8 BIBLIOGRAFÍA

(s.f.). Obtenido de sckitlearn: https://scikit-learn.org/stable/index.html

Alpaydin, E. (2020). *Introduction to machine learning. MIT press.*

Boiy, E. &. (2009). . A machine learning approach to sentiment analysis in multilingual Web texts. Information retrieval, 12(5), 526-558.

Liu, B. (2012). *Sentiment analysis and opinion mining.*

Marsland, S. (2015). *Machine learning: an algorithmic perspective.* CRC press.

Munir, S. (2019). Basic Sentiment Analysis using NLTK. *towardsdatascience*.

nltk. (s.f.). Obtenido de https://www.nltk.org/

Prabowo. (2009). Sentiment analysis: A combined approach. *Journal of Informetrics*, 143-157.

Pressman, R. (2010). *Ingeniería del Software.* McGraw-Hill.

Pupale, R. (2018). Support Vector Machines(SVM) — An Overview. *towardsdatascience*.

Sehra, C. (2018). Decision Trees Explained Easily. *Medium*.

towardsdatascience. (s.f.). Obtenido de towardsdatascience: https://towardsdatascience.com/natural-language-processing-nlp-for-machine-learning-d44498845d5b