Facultad de Ingeniería - UBA

Organización de Datos 75.06

Trabajo Práctico: Diseño

Fine Food Reviews

Integrantes:
Agustina Barbetta *96528* agustina.barbetta@gmail.com
Santiago Lazzari *96735* santilazzari1994@gmail.com
Luciano Mintrone *95463* lucianomintrone@gmail.com
Manuel Porto *96587* manu.porto94@gmail.com

Rayo Privatizador

30 de noviembre de 2016

Resumen

En el presente trabajo se analiza un set de datos con reseñas de productos y se estudian distintas estrategias para desarrollar un modelo capaz de predecir el puntaje de las mismas.

Índice

1. Introducción

El objetivo de este trabajo es predecir la cantidad de estrellas de un review en base a los datos del mismo.
En el presente informe, se parte de un análisis de los sets de datos dados, es decir, todo lo relacionado con las distintas métricas, para luego investigar los potenciales algoritmos a utilizar en la búsqueda de una solución definitiva. En la sección **Investigación** se presenta cada algoritmo individualmente, con una explicación del funcionamiento del mismo, sin compararlos entre sí. Más adelante se relatan los modelos utilizados hasta el momento junto con sus resultados. Finalmente, en la sección de **Modelo a implementar**, se presentará la solución elegida justificando el por qué de nuestra decisión y su preferencia sobre los otros algoritmos.

2. Pre-procesamiento

Antes de comenzar el análisis, escribimos un *script* para; eliminar HTML markup, caracteres especiales, números y pasar a minúscula todo el texto correspondiente a los campos `Summary` y `Text` de cada *review*.

Además, utilizamos la biblioteca `nltk`[1] para obtener versiones sin *stopwords* y con *stemming* de los sets de datos.

Como no utilizaremos todos los pasos de este proceso para los diferentes tipos de análisis, los enumeramos en la siguiente lista a la cual haremos referencia en las próximas secciones.

1. Eliminación de HTML
2. Eliminación de caracteres especiales
3. Eliminación de caracteres numéricos
4. Pasaje a lower case
5. Eliminación de stopwords
6. Stemming

Para este trabajo, buscamos extraer información a partir de diferentes textos. El motivo de este pre-procesamiento es filtrar de los documentos a aquellas palabras que no provean información relevante, así como también, llevar todas las palabras con un significado equivalente a un mismo formato.

3. Análisis

3.1. Utilidad de los campos del set

Uno de los primeros análisis realizados consistió en evaluar la utilidad de los campos presentes en el set. A continuación se enumeran todos los campos existentes:

1. **Id:** El Id que identifica a cada review
2. **ProductId:** El Id del producto
3. **UserId:** El Id del usuario

[1] Natural Language Toolkit: http://www.nltk.org/

4. **ProfileName:** El nombre del usuario
5. **HelpfulnessNumerator:** El numerador indicando la cantidad de usuarios que juzgaron al review como util
6. **HelpfulnessDenominator:** El denominador indicando la cantidad de usuarios que evaluaron si el review fue útil o no
7. **Prediction:** La cantidad de estrellas del review. Valor entero entre 1 y 5.
8. **Time:** Un timestamp para el review
9. **Summary:** Un resumen del review
10. **Text:** Texto del review

En principio, los campos "Id", "ProfileName" y "Time" no fueron considerados de utilidad. El primero porque su única función es la de identificar cada review y no aporta ningún tipo de información sobre la misma. El segundo no es necesario ya que se cuenta con el Id del usuario, además de que el nombre de perfil de un usuario no necesariamente es único, no siendo el caso de su Id. Por último, el campo "Time" también fue descartado ya que no se encontró ninguna razón por la cual el timestamp de una reseña pueda ser de utilidad para predecir el puntaje de la misma.

El Id del usuario podría aportar información útil ya que, junto con sus puntuaciones de productos es posible predecir como suele calificar los productos. Sin embargo, si se realiza un análisis sencillo sobre el set, se advierte que le promedio de reviews por usuario es demasiado bajo (`2 reviews/usuario`) y que solo el 17,7 % de los usuarios se encuentra por encima de este valor. Esto no quiere decir que el campo "UserId" deba ser descartado, pero en caso de utilizarse se deberá tener en cuenta lo recién expuesto para filtrar a aquellos usuarios con una cantidad de reviews insuficientes para poder aportar información útil al modelo.

Una situación similar ocurre con los campos "HelpfulnessNumerator" y "HelpfulnessDenominator". En principio, esta información podría ser de utilidad para asignar pesos a las distintas reseñas, es decir, se le daría mas importancia a aquellas reviews que hayan sido útiles para los usuarios y se le quitaría importancia a aquellas que no lo sean. Además de que este razonamiento no necesariamente implique un beneficio al modelo ya que, el hecho de que una reseña sea útil para un grupo de usuarios no implica que lo sea para el algoritmo utilizado, estos campos presentan la misma problemática que el "UserId". En el campo "HelpfulnessDenominator" que indica a cuántos usuarios les pareció útil o no una reseña se encuentra un valor promedio de 2,24. Además, solo el 21,8 % de los reviews se encuentran por encima de esta marca. Esto es importante ya que es muy probable encontrar reseñas en donde la utilidad sea muy alta o baja pero que la cantidad de personas que lo hayan determinado sea muy baja por lo que el dato deja de tener sentido.

La conclusión de este breve análisis es que en principio se trabajará únicamente con los campos "ProductId", "Prediciton", "Summary" y "Text". Los dos primeros por obvias razones y los dos últimos porque presentan la fuente principal de información sobre cada review. Esto no quiere decir que los campos restantes no puedan ser incluidos eventualmente, pero llegado el caso se los utilizara considerando lo anteriormente expuesto.

3.2. Cantidad de reviews por puntuación

Observando el siguiente gráfico, puede advertirse un claro desbalance en la distribución de las clases de los reviews del set de entrenamiento. Aproximadamente 3/4 partes de las reseñas

tienen una puntuación de 5 estrellas mientras que el cuarto restante se reparte entre el resto de las clases.

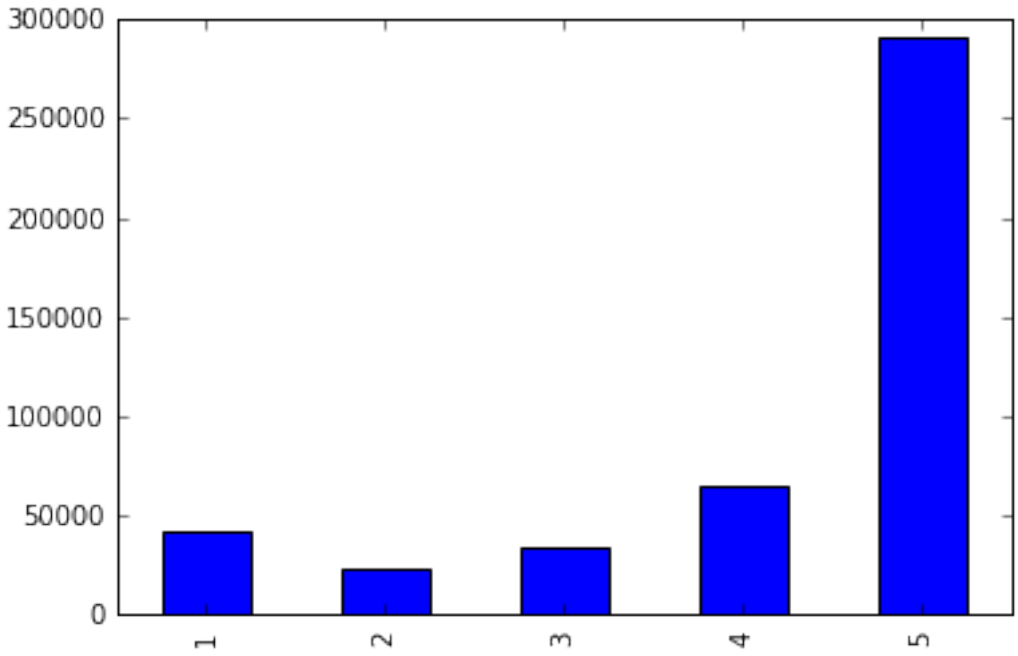

Figura 1: Distribución de reviews según puntuación. Cantidad de reviews en el eje y.

El conocimiento de la existencia de esta problemática es significativo ya que, existen algoritmos de aprendizaje que al utilizarse sobre sets de entrenamiento desbalanceados se comportaran de igual manera que un simple algoritmo que clasifique los datos según la clase mayoritaria. Esto ocurre porque la mayoría de estos algoritmos operan sobre dos premisas fundamentales:

1. El objetivo es maximizar la precisión
2. El clasificador operara sobre un set de datos de igual distribución que el de entrenamiento

En síntesis, utilizar algoritmos de clasificación estándares sobre sets de datos desbalanceados sin ajustarlos para considerar esta característica, puede ocasionar graves errores en los resultados de la predicción. [1]

3.3. Palabras significativas por puntuación

En este análisis, contamos la cantidad de apariciones de cada palabra para cada puntuación. Creemos que de esta forma podremos obtener la probabilidad de que un nuevo review del *test set* pertenezca a una determinada clase (puntuación).

Comenzamos pre-procesando los campos `Summary` y `Text` con los pasos 1 a 4. Luego agrupamos estos campos según su clase y graficamos las siguientes nubes de palabras por medio de la biblioteca `wordcloud`[2]. Cabe aclarar que este generador de nubes provee una lista de *stop words* que excluye del gráfico.

[2] https://github.com/amueller/word_cloud

Figura 2: Nube de palabras para clase 1

Figura 3: Nube de palabras para clase 2

Figura 4: Nube de palabras para clase 3

Figura 5: Nube de palabras para clase 4

Figura 6: Nube de palabras para clase 5

Observamos que varias de las palabras predominantes son las mismas para todas las clases (Por ejemplo, *taste*, *flavor*, *good*, son grandes en todas las nubes). Procedimos a calcular algunas probabilidades de la forma:

$$p_{palabra} = \frac{apariciones}{total} \tag{1}$$

Siendo **apariciones** la cantidad de apariciones de la palabra en las *reviews* de la clase x y **total** la cantidad total de palabras utilizadas para las *reviews* de la clase x.

A continuación graficamos los resultados:

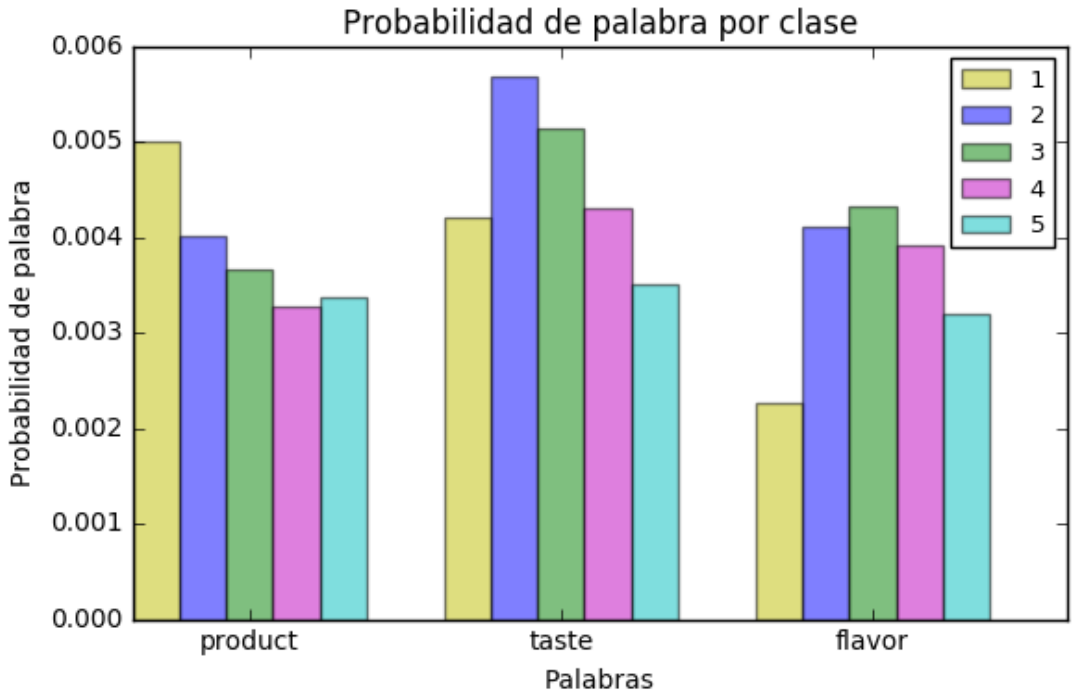

Figura 7: Probabilidad de palabra por clase para palabras probables en todas las clases

Finalmente, comparamos la visualización anterior con una de probabilidades para palabras predominantes por clase, como por ejemplo; *love* que sólo es grande en la nube clase 5.

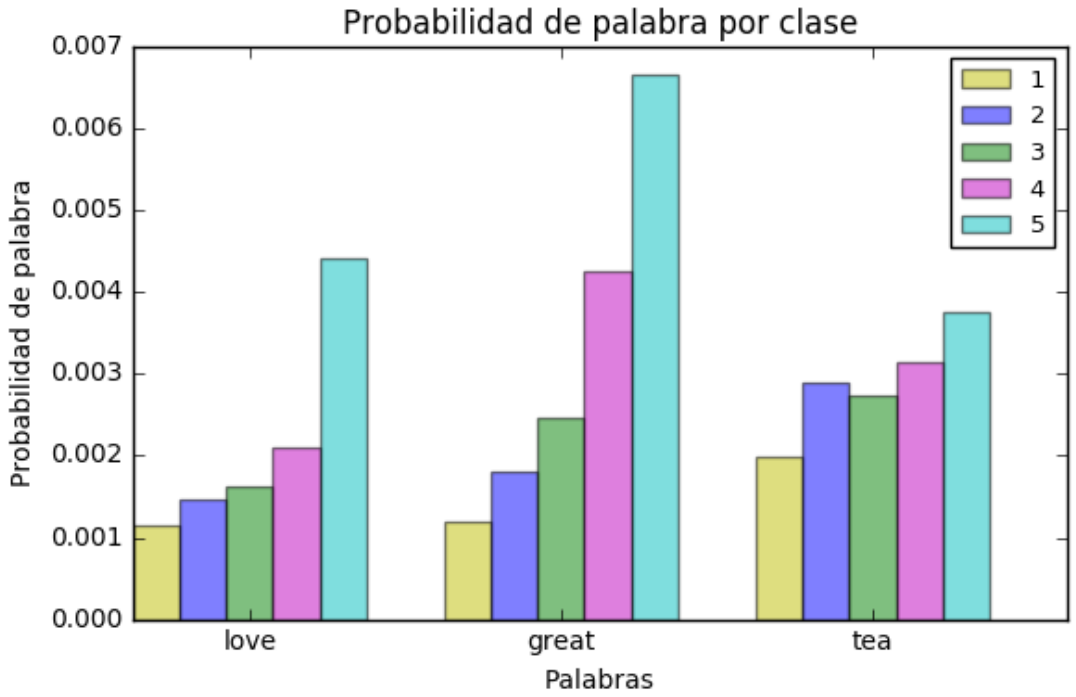

Figura 8: Probabilidad de palabra por clase para palabras probables en *reviews* de clase 5

Se observa como, en el primer gráfico, ninguna barra se despega mucho de las demás, mientras que en el segundo, la barra de clase 5 supera ampliamente al resto de las clases excepto en la palabra *tea*, que aunque parece predominante sólo en la nube 5, es probable que se presente en *reviews* de cualquier clase.

3.4. Entradas repetidas

A medida que fuimos explorando el *training set*, nos dimos cuenta de la existencia de *reviews* repetidas. Para contabilizarlas de forma eficiente escribimos un *script* en `PySpark`. A partir del mismo y con el set de entrenamiento original (sin pre-procesamiento), encontramos 48618 *reviews* que poseen repeticiones (Campos `Summary`, `Text` y `Prediction` idénticos) y un total de 129432 copias de ellas. Todas las entradas respetan la unicidad del campo `Id` y, por lo que pudimos explorar, fueron escritas por el mismo usuario en el mismo momento (Mismo `UserId`, `UserName` y `Time`). Difieren en `HelpfulnessNumerator`, `HelpfulnessDenominator` y, sorprendentemente, en `ProductId`.
En conclusión, tendremos en cuenta a la hora de proponer una solución la posibilidad de recortar estas entradas repetidas, ya que en caso de no utilizar los *features* `ProductId` y `Helpfulness` podemos reducir nuestro set de entrenamiento un $129432/454760 * 100 = 28{,}4\,\%$

Además de encontrar entradas repetidas en el set de train pueden haber reviews en el set de test idénticos a uno o mas reviews del otro set. Para optimizar la predicción en este caso, podemos agrupar los reviews por su `minhash` así al compararlos pasamos de un problema de $O(n^2)$ a uno de $O(n)$. El `minhash` se calcula a partir de aplicarle una función de hashing a cada shingle del texto y quedándonos con el mínimo.

4. Investigación

4.1. Perceptrón

Perceptrón es un algoritmo de aprendizaje supervisado para clasificadores binarios y de tipo lineal.

Para este algoritmo se utiliza un vector de **n** dimensiones, siendo **n** igual al total de palabras sin repetir en el diccionario del set de datos. Así cuando un review es leído va a crearse un vector con un 1 en las dimensiones que aparecen. Por ejemplo si el universo de palabras es: U = ('organización', 'de', 'datos') y el review contiene las palabras: j = ('de', 'datos') el vector **x** va a quedar definido como x = (0,1,1). También se debe definir un vector de pesos **w** de **n** dimensiones, el cual generalmente comienza con valores iguales a 0 (cero): w = (0,0,0). Así para cada fila (**j**) leída del set de datos se debe calcular la predicción (y_j) con una función a definir. Por lo tanto, la ecuación resultante seria:

$$y_j = f[w_0 x_{j,0} + w_1 x_{j,1} + w_2 x_{j,2}]$$

Una vez calculado dicho valor, se debe actualizar el vector de pesos con la formula:

$$w_i = w_i + (d_j - y_j) x_{j,i}$$

para las dimensiones $0 \leq i \leq n$. Con d_j el valor a predecir.

Luego se deben repetir estos dos últimos pasos hasta llegar a un error establecido o una cierta cantidad de iteraciones.

Si bien es un algoritmo diseñado para trabajar con dos clases, puede generalizarse para clasificar mas clases. Incluso en problemas de clasificación de textos. Sin embargo, según los resultados vistos en problemas similares al expuesto en el presente trabajo, este método no resulta ser el mas preciso en comparación con otros algoritmos utilizados.[2]

4.2. Red multicapa

Una red multicapa utiliza el concepto de los perceptrones donde cada perceptrón es una neurona de una red que conecta varias de estas neuronas. Una red multicapa se puede visualizar de la siguiente manera.

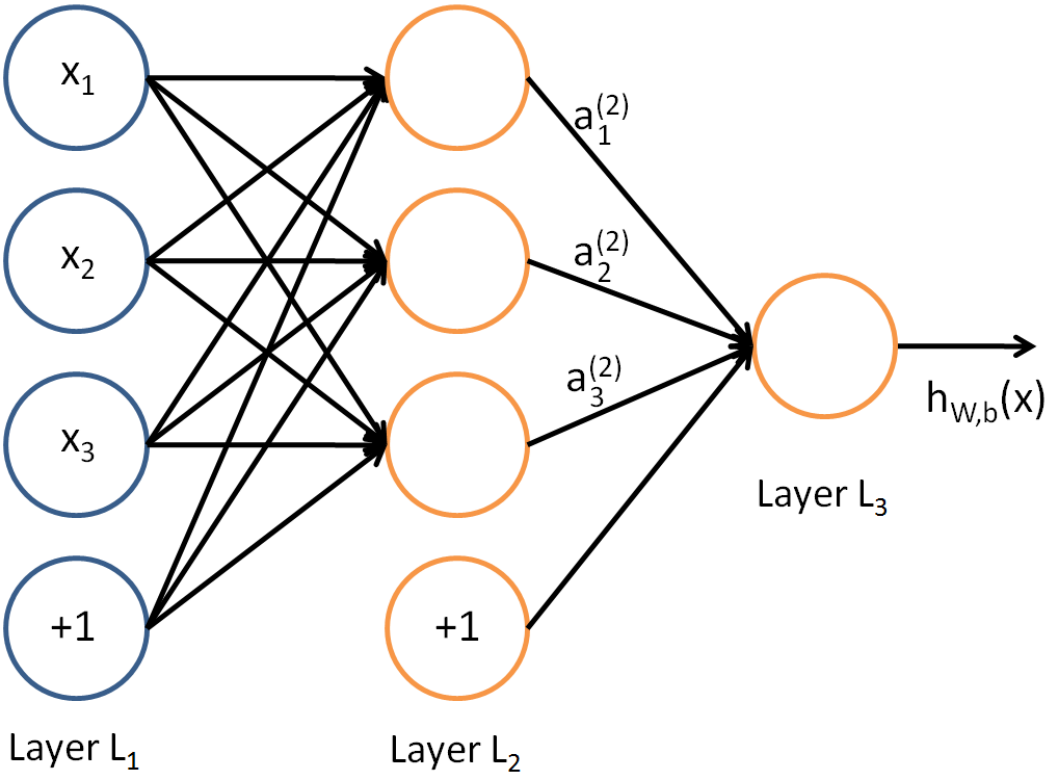

Figura 9: Red Neuronal

Donde la primer capa es la **Input layer** que deriva la información que llega de *input* y la reparte en la siguiente capa. Las subsiguientes capas son las **Hidden layers** donde se van distribuyendo los datos capa a capa y finalmente se llega al **Output layer** que es el resultado. En la imagen presentada el *output layer* es una sola neurona, sin embargo no es necesario, podría haber n neuronas en el *output layer*.

En la siguiente sección, describimos algunos de los métodos de aprendizaje para las redes neuronales:

4.2.1. Modelos de aprendizaje

Los modelos de dividen en **Algoritmos de aprendizaje**

- **Corrección de error**
- **Aprendizaje Hebbiano**
- **Aprendizaje Competitivo**

y, por otro lado, **Paradigmas de aprendizaje**:

- **Aprendizaje Supervisado**
- **Aprendizaje reforzado**
- **Aprendizaje no supervisado**

El **Aprendizaje Supervisado** es el tipo de entrenamiento en el cual se provee al sistema con información de las entradas al igual que se proveen las salidas esperadas o destinos correspondientes a dichas entradas a modo que el sistema tenga los destinos como punto de referencia para evaluar su desempeño en base a la diferencia de estos valores y modificar los parámetros libres en base a esta diferencia.[3]

En un **Aprendizaje no supervisado** los parámetros libres del sistema son modificados únicamente en base a las entradas del sistema de manera que aprenden a categorizar las entradas y clasificarlas sin necesidad de una referencia.

La **Regla delta** o **Algoritmo de corrección de error** se refiere al modelo de aprendizaje que se basa en minimizar una función de error en relación a los parámetros libres del sistema. Una vez definida la función de costo a minimizar la corrección de error es un problema estrictamente de optimización.

El **Aprendizaje Hebbiano** indica que las conexiones entre las neuronas de entrada activas y las neuronas de salida activas se refuerzan durante el entrenamiento: coincidencias entre actividad de entrada y actividad de salida se intensifican. Mientras que las conexiones entre neuronas de entrada inactivas y neuronas de salida (Activas o Inactivas) no se refuerzan.

Este método de aprendizaje puede ser tanto supervisado como no supervisado. Cuando es supervisado, la respuesta correcta para el dato de entrada es introducida para cada neurona de salida, y los pesos sinápticos entre las neuronas activas se incrementan, mientras que los pesos entre neuronas que no estén activas simultáneamente permanecen igual que estaban.

En el **Aprendizaje competitivo** suele decirse que las neuronas compiten (Y Cooperan) unas con otras con el fin de llevar a cabo una tarea dada. Con este tipo de aprendizaje se pretende que cuando se presente a la red cierta información de entrada, sólo una de las neuronas de salida de la red, o una por cierto grupo de neuronas, se active (Alcance su Valor de Respuesta Máximo). Por tanto las neuronas compiten para activarse quedando finalmente una, o una por grupo, como neurona vencedora y el resto quedan anuladas y siendo forzadas a sus valores de respuesta mínimos.

4.3. The Hashing Trick

También conocido como feature hashing, al igual que otros algoritmos como preceptrón, toma ventaja de la separación en *tokens* del review y crea un vector de features. A cada feature, en nuestro caso a cada palabra, del vector que define una fila en el set de datos se le aplica una función de hashing h_1, que como sabemos devuelve un número. A dicho número se le aplica la función **mod d** con $d \leq n$. Siendo **n** el numero de dimensiones del vector original y **d** el número de dimensiones del vector final. Así la idea es reducir el numero de dimensiones de nuestro set de datos. Esto tiene una desventaja que puede generar colisiones, es decir, textos que son distintos pero al ser hasheados parecen similares. Dicho problema se puede solucionar con una segunda función de hashing h_2 que para dimensión devuelva dos valores +1 o -1. Entonces según h_2 sea positiva o negativa se va a sumar o restar 1 a la posición indicada por h_1. Esto reduce las probabilidades de colisiones y nos permite encontrar textos similares con mayor facilidad.

Otra ventaja de este algoritmo es que el vector resultante suele ser muy disperso, por lo que guardarlo con un algoritmo adecuado puede ayudarnos a reducir mucha memoria.

Una desventaja de este algoritmo es que se pierde la interpretación de los datos.

Según las ultimas aplicaciones del hashing trick clasificando textos con funciones de hashing aleatorias y decenas de miles de columnas en el vector de salida no presenta problemas de performance, hasta incluso sin el uso de h_2.[4]

4.4. Naïve bayes

Naïve Bayes es una extensión de Bayes para n características, las cuales se asumen independientes.

$$P(y|x_1,..,x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1,..,x_n)}$$

En donde 'y' seria la puntuación y 'x' cada feature que en nuestro caso corresponde a cada palabra. Entonces vamos a calcular a

1. P(y) como la $\frac{review-de-clase-x}{total-de-reviews}$
2. $P(x|y)$ como la frecuencia de la palabra x en la clase y $p_{palabra} = \frac{apariciones}{total}$
3. $P(x_1,..,x_n)$ como la probabilidad de que en un review aparezcan esas palabras.

La frecuencia de las palabras se almacena en tablas de hash durante la fase de entrenamiento. De esta forma vamos a predecir a un review con el $P(y|x_1,..,x_n)$ más grande.[5]

Si el clasificador se encuentra con una palabra que no ha visto antes en el set de entrenamiento, nos encontramos con un problema de probabilidad cero. La solución más simple al problema es sumar 1 a la frecuencia de cualquier palabra en cualquier contexto, a esto se lo llama **ajuste de Laplace**.

Una desventaja de Bayes es el overfitting, al encontrar palabras poco frecuentes que aparecen sólo en una clase, el algoritmo clasifica al documento dentro de esta clase y cree que cualquier entrada con esa palabra pertenece exclusivamente a la clase en la que la observó. Esto puede evitarse ignorando las palabras con baja frecuencia o que solo pertenecen a una clase.[6]

Su ventaja principal es la capacidad de escalar grandes sets de datos sin problemas. Se pueden eliminar las palabras duplicadas de los documentos, ya que no agregan información adicional; a este tipo de algoritmo se lo llama **Bernoulli Naïve Bayes**. Incluir la presencia de una palabra en lugar de su recuento mejora la performance significativamente cuando hay muchas entradas en el set de entrenamiento.[7]

4.5. SVM

SVM o Support Vector Machines son modelos de aprendizaje supervisado asociados con algoritmos para resolver problemas de clasificación y regresión. SVM es particularmente útil para categorización de textos e incluso obtiene mejores resultados que otros algoritmos convencionales.[8]

El modelo teórico de SVM es muy sencillo y puede expresarse a partir de los conceptos del modelo "Perceptrón". En este ultimo se buscaba un hiperplano o recta que separara a las clases que se quieren clasificar, en SVM no solo se busca dicho hiperplano si no que se pretende encontrar el mejor de ellos para realizar la clasificación. Para encontrar el hiperplano que mejor separa las clases se utiliza el concepto del "margen", en SVM el mejor hiperplano es aquel que maximiza el margen entre las clases.[6] El margen es la región comprendida entre dos hiperplanos que separan los elementos. Esta región es de la mayor distancia posible mientras que los elementos continúen separados.

Si bien SVM es introducido como un modelo para resolver problemas de clasificación lineal, es posible adaptarlo para resolver problemas en donde existen mas de dos clases, como es el caso

del presente trabajo. Una de las opciones para lograr solucionar este problema es utilizar varios clasificadores lineales que separen a una de las clases con el resto, es decir que se transforma el problema de clasificación entre mas de dos clases en varios problemas de clasificación binarios independientes. Esta estrategia imposibilita la relación entre distintas clases ya que pertenecen a problemas independientes entre si.[9] Existen casos en donde SVM fue utilizado para clasificación de textos en problemas con mas de dos clases y se obtuvieron resultados satisfactorios e incluso mejores que los obtenidos con otros algoritmos. [8]

5. Modelos utilizados

5.1. Vowpal Wabbit

Como parte de un finger exercise, realizamos predicciones sobre el test set utilizando la herramienta **Vowpal Wabbit**.
El proyecto Vowpal Wabbit es un sistema de aprendizaje rápido diseñado para procesar sets de datos completos que son demasiado grandes para caber en la memoria principal de una computadora.

Hay dos maneras de tener un algoritmo de aprendizaje rápido:

- Comenzar con un algoritmo lento y acelerarlo
- Construir un algoritmo de aprendizaje intrínsecamente rápido

Este proyecto se basa en el segundo enfoque, y se llegó a un estado en el que puede ser útil a los demás como una plataforma para investigación y experimentación.

Existen varios algoritmos de optimización disponibles, siendo la base un *gradient descent (GD)* disperso en una *loss function* (De las múltiples disponibles).[10]

Antes de comenzar a predecir, pre-procesamos ambos sets de datos con los pasos 1 a 6 y tomamos exclusivamente los features `Summary`, `Text`, `Prediction` y `Id` (Este último como tag única para cada review) pasándolos al formato de entrada pedido por vw.

Comenzamos entrenando con los siguientes flags[11]:

- `--passes 300`: Realiza 300 pasadas sobre el training set
- `--ngram 7`: Crea 7-gramas
- `-b 24`: Utiliza hashes de 24 bits (Por defecto son 18)
- `--ect 5`: Utiliza *error correcting tournament*, notifica la existencia de 5 labels posibles y pide que se elija una por entrada

logrando predecir con un puntaje de `0.51373` en Kaggle.

En un segundo intento, entrenamos con los flags:

- `--passes 300`: Explicado anteriormente
- `--ngram 2`: Crea bigramas

- `-b 24`: Explicado anteriormente
- `--skips 2`: Genera saltos de a bigramas
- `--decay_learning_rate=.95`: Establece un factor de decaimiento, entre pasadas, para la tasa de aprendizaje
- `--ftrl`: Utiliza la optimización *Follow the (Proximal) Regularized Leader*

Logrando un puntaje final de `0.41703` en Kaggle.

5.2. Dynamic vector prediction

Inspirados en la charla de las jaiio y de la manera en que facebook hacia sentiment analisis, propusimos un algoritmo similar que lo bautizamos Dynamic vector prediction.

El algorimo consiste en armar una tabla de la forma palabra:cardinalidad de la palabra por cada predicción. Por ejemplo, en el caso de la predicción 1.

```
{
"tast":22508,
"br":22305,
"like":21614,
"product":21051,
"one":15308,
"food":14724
.
.
.
}
```

Esto se genera recorriendo todos los reviews y de una misma predicción y sumando 1 por cada palabra. Ésta tabla de frecuencias seria el equivalente al aprendizaje.

Para poder predecir un review por ejemplo “The food was awful”

Se hace un split lo cual queda [“The”, “food” , “was”, “awful”]. Teniendo esto que llamaremos, la referencia, se genera un vector, llamado vector dinámico con la cardinalidad de cada palabra. Es decir, si alguna palabra se repitiera, no estaría duplicada en las referencias pero si aumentaría la cardinalidad en el vector dinámico. En este caso el vector dinámico quedaría de la siguiente forma. [1, 1, 1, 1].

Teniendo las tablas de frecuencias de las 5 predicciones, se hace lo mismo con las palabras de referencia. Por ejemplo, si “The” tiene cardinalidad 10000, “food” 1500,“was” 10000 y “awful”500, en la predicción 1 entonces el vector dinámico es [10000, 1500, 1000, 500]. Esto se repite con todas las predicciones.

A esta altura se generaron 6 vectores dinámicos, el de la review + las 5 predicciones. Para poder comprar todos los reviews de la misma forma, sabiendo que el set de datos está desbalanceado, se procede a normalizar los 6 vectores.

Finalmente se calcula la distancia coseno entre la review contra todas las predicciones y se queda con la menor. Esto define a cuál se parece más y esa es la predicción.

Los resultados de éste algoritmo se probaron con tres distancias distintas y se hizo un post en Kaggle con la mejor de ellas. Basándonos en cuánto se ajustaba al train set.

Con la distancia del coseno los resultados fueron 377 predicciones correctas con respecto a mil. Con la Euclideana fueron 345. Mientras que con Jaccard 77.

Se obtuvo una puntuación de `5.06451` en Kaggle, por lo que concluimos en que el algoritmo no es bueno para este tipo de problema.

6. Modelo a implementar

6.1. Pre-procesamiento

Para la solución a implementar se decidió utilizar los pasos 1 a 4 sobre los sets training y test, obviando la eliminación de stopwords, ya que desempeñan una función importante a la hora de entender los sentimientos de los usuarios.

En segundo lugar, se propone realizar un manejo de las negaciones. Al utilizar cada palabra como un feature, la palabra "good" en la frase "not good" contribuirá a un sentimiento positivo en lugar de a uno negativo, debido que la presencia de "not" no se tiene en cuenta. Para resolver este problema se escribirá un *script* que transforme una palabra seguida de un "not" o "n't" en "not_" + palabra. Las palabras a negar se detectan con una variable de estado, cuando esta es verdadera, a todas las palabras leídas se les antepondrá la negación. La variable vuelve a su valor inicial cuando se encuentra un signo de puntuación u otra negación.

6.2. Procesamiento

Previo al entrenamiento del modelo elegido, se deberá preparar el set de acuerdo a los requisitos del algoritmo. Además se determinará qué campos serán incluidos para realizar el entrenamiento, en base a lo explicado en apartados anteriores.

El procesamiento propuesto consiste en construir una matriz compuesta por los Id's de los productos, su puntaje y todas las palabras halladas en los campos "Summary" y "Text" de todas las reviews. Cada fila de esta matriz corresponde a un review individual. Cada una de las palabras corresponderá a un atributo y tendrán un valor entre 0 y N dependiendo de la cantidad de veces que dicha palabra aparezca en la review. A continuación se presenta un ejemplo de la matriz:

ProductId	Prediction	good	not_good	taste
1	4	3	0	1
2	1	0	2	2
3	2	1	3	0
4	2	0	1	0

Cuadro 1

Dado que la gran cantidad de palabras existentes tiene como consecuencia que las matrices tengan una dimensión demasiado grande, se filtrarán las palabras en base a su ocurrencia. Sabiendo que la cardinalidad de las palabras por predicción se distribuye como una ley de potencias, se podrá encontrar algún porcentaje empírico de palabras, las más relevantes, que se utilizarán para entrenar. Y siendo ésta distribución una ley de potencias, dicha relación es invariante a la escala, es decir, que para todas las predicciones (independientemente de la cantidad de reviews que tengan) se podrá usar la mismo proporcionalidad.

Otra forma de reducir éste número es aplicarle the hashing trick a las palabras y así poder reducir las dimensiones de la matriz. Se probará si hace falta aplicarle the hashing trick, y si hiciera falta, cuán costoso és.

6.3. Predicción

El modelo que se utilizará para predecir el puntaje de los reviews será SVM, como ya se explicó, SVM es un algoritmo especialmente útil para clasificación de textos. Este modelo puede ser utilizado para problemas no lineales, como es el caso del presente trabajo [9] y se cuenta con

evidencia empírica de que obtiene mejores resultados que otros algoritmos para problemas de clasificación no lineales. [8]

Según lo investigado, SVM no funciona bien para sets de datos desbalanceados[12], lo cual es nuestro caso (como fue detallado en el análisis del set de datos). Para solucionar este problema se pueden usar dos aproaches; Una solución posible es balancear el set de datos, recortándolo aleatoriamente o aprovechando a sacar las entradas repetidas. Por otro, si esto último no funciona, optaremos por cambiar el predictor por otro, como un perceptrón multicapa.

7. Bibliografía

Referencias

[1] Foster Provost. Machine learning from imbalanced data sets 101. http://pages.stern.nyu.edu/ %7Efprovost/Papers/skew.PDF, 2000.

[2] L. Tolosi Y. Ivanov G. Georgiev V. Zhikov, I. Nikolova. Multi-class, label and language document classification: System and features. https://ontotext.com/documents/publications/2012/classifextraction.pdf, 2012.

[3] UDLAP. Perceptrón multicapa. http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/mejia_s_ja/capitulo3.pdf.

[4] Argerich Luis. Can you explain feature hashing in an easily understandable way? https://www.quora.com/Can-you-explain-feature-hashing-in-an-easily-understandable-way, 2016.

[5] Vik Paruchuri. Naive bayes: Predicting movie review sentiment. https://www.dataquest.io/blog/naive-bayes-tutorial/, 2015.

[6] Argerich Luis. Organización de datos, apunte del curso. 2016.

[7] Arjun Bhatia Vivek Narayanan, Ishan Arora. Fast and accurate sentiment classification using an enhanced naive bayes model. https://arxiv.org/pdf/1305.6143.pdf.

[8] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. https://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf, 1997.

[9] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. http://jmlr.csail.mit.edu/papers/volume2/crammer01a/crammer01a.pdf, 2001.

[10] Microsoft Research. Vowpal wabbit. https://github.com/JohnLangford/vowpal_wabbit/wiki.

[11] Microsoft Research. Vowpal wabbit: Command line arguments. https://github.com/JohnLangford/vowpal_wabbit/wiki/Command-line-arguments.

[12] Luis Argerich. For what kind of classification problems is svm a bad approach. https://www.quora.com/For-what-kind-of-classification-problems-is-SVM-a-bad-approach/answer/Luis-Argerich?srid=kXaw.