5.2 Naturaleza del tuit

En esta fase del proyecto el objetivo es discernir si el contenido del texto de un tuit que hemos obtenido de Twitter a través de la búsqueda descrita en la sección 4.2, es efectivamente relevante para seleccionar a su autor como un posible candidato a la oferta de trabajo que nos ocupa. El criterio para que un tuit tenga un contenido relevante será que describa o haga referencia a alguna de las habilidades o conocimientos requeridos a los candidatos. Es decir, no basta con que el tuit hable de Big Data o de ciencia de datos en general, o de noticias relacionadas con ellos, sino tiene que mostrar algún signo de dichas habilidades y conocimientos.

Para ello, hemos desarrollado un modelo ad-hoc que pasamos a describir a continuación.

5.2.1 Construcción del modelo

En este apartado hemos elegido la opción de un modelo supervisado para clasificación de textos con objetivo de tener más eficiencia en la selección de usuarios relevantes.

La clasificación de los tuits mediante un modelo no supervisado (tipo “bag of words", por ejemplo), es una técnica que ya exploramos en la clasificación del idioma del tuit que hemos tratado en la sección anterior, y en la clasificación del perfil de usuario que trataremos a continuación.

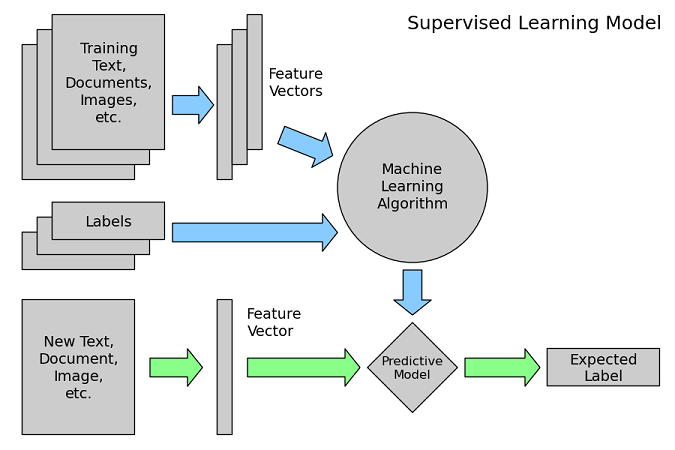


Figura X.X.X: Proceso de construcción y funcionamiento para clasificación de textos. .

Como en la figura x.x.x, tenemos los siguientes procesos para nuestro modelo:

**Training Text:**

Para ello, obtuvimos una nueva base de datos de Twitter con 50 mil tuits, con una búsqueda distinta a la descrita en la sección 4.2 pero también por palabras clave, a través del API Search de Twitter, y procedimos como en el caso de la base de datos para la extracción de candidatos, a almacenarla en local en una base de MongoDB.

**Labels:**

Trabajamos sobre esa base de datos para construir un dataframe que contuviera los textos de los tuits, así como otra información relevante de los tuits (aunque luego no se usa en el modelo), y procedimos a etiquetar manualmente 2000 de esos tuits. Los tuits etiquetados están almacenados en un fichero Excel Python/relevance model/ds sample2.xlsx de nuestro repositorio de GitHub. Como curiosidad, nos gustaría comentar que al bajar los tuits por palabras claves como Python o SAS, obtuvimos tuits que hablaban de serpientes o de los problemas del Sistema Andaluz de Salud.

**Feature Vector:**

Antes de entrenar el modelo, intentamos eliminar el “ruido" de los textos sobre los que se va a realizar, sometiéndolos a una limpieza en la que retiramos de los mismos urls, números, signos de puntuación, “stop words", palabras de menos de dos letras y distinción entre mayúsculas y minúsculas.



Figura 5.2.1: Función para limpiar los tuits y entrenar el modelo de clasificación.

Además, los estimadores implementados en scikit-learn necesitan transformar las palabras en vectores de características y para eso elegimos el método TfidfVectorizer porque los textos de tuits son imprevisibles y el TF-IDF ajusta la situación de que hay palabras que son más comunes y menos relevantes, además de ser altamente recomendado en análisis de textos por su facilidad de uso y su parametrización.

Para ello, crea un “vocabulario" con las palabras que aparecen en los textos y sus características en forma de diccionario. En nuestro contexto, los niveles o características TF-IDF, son números que reflejan la importancia de cada palabra, de forma proporcional a la frecuencia en la que aparece en el documento, pero penalizados por la frecuencia total de ocurrencia de la palabra en todo el corpus. Más concretamente, [35], para cada palabra t y cada documento d:

tf (t; d) =

(

1 + log2(frec(t; d)) si frec(t; d) > 0

0 en otro caso.

idf(t) = log jDj

jDtj ;

2Los métodos y algoritmos usados en el entrenamiento y predicción con este modelo son los que nos ofrece el

paquete scikit-learn de Python.

54 OCTOPUS DATA INSIGHTS

donde

frec(t; d) es el número de veces que aparece t en d

D es la colección de todos los documentos

Dt es la colección de los documentos que contienen la palabra t

jAj es el cardinal del conjunto A:

Finalmente, el TF-IDF de un par palabra-documento es

tfidf(t; d) = tf (t; d) \_ idf(t):

El resultado del método TfidfVectorizer aplicado a un corpus, es una matriz en la que cada documento del corpus es codificado como un vector cuyas entradas son los índices TF-IDF de cada una de las palabras de dicho documento, con respecto a ese mismo documento.

**Machine Learning Algoritmo:**

Para entrenar el modelo dividimos la muestra de 2000 tuits en un 30% de datos de test y los restantes datos de entrenamiento a través de la función y con esta división procedimos a entrenar el modelo2.

Nos hemos decidido por un algoritmo de tipo Naive Bayes, que invocaremos a través del modelo MultinomialNB() de scikit-learn, porque después de buscar e investigar concluimos que ese algoritmo aunque tenga algoritmos más simples siendo más fácil de usarlo y comprenderlo, tiene muy buena performance, velocidad de procesamiento y funciona muy bien en clasificación de textos p.ej: clasificar spam.

Un modelo de clasificación de tipo Naíve Bayes, [36], se basa en asignar a cada observación aquella clase o resultado de la clasificación, a la que es más probable que pertenezca, dados los datos que componen la observación. En concreto, nuestras observaciones están detenidas por una serie de variables X1;X2;X3; : : : , y queremos adivinar para cada observación el valor de otra variable Y (discreta y finita). En un modelo Naive Bayes, el valor de Y = yi que asignaremos a una observación (x1; x2; x3; : : : ) es aquel que hace máxima

P(Y = yijX1 = x1;X2 = x2;X3 = x3; : : : ):

Por la fórmula de Bayes,

P(Y = yijX1 = x1;X2 = x2;X3 = x3; : : : )

=

P(X1 = x1;X2 = x2;X3 = x3; : : : jY = yi)P(Y = yi)

P(X1 = x1;X2 = x2;X3 = x3; : : : )

=

P(X1 = x1;X2 = x2;X3 P = x3; : : : jY = yi)P(Y = yi)

j P(X1 = x1;X2 = x2;X3 = x3; : : : jY = yj)P(Y = yj)

El cálculo de probabilidades condicionadas en la muestra puede ser muy costoso. El nombre de

Naíve en el método viene de la simplificación que el método Naive Bayes adopta para llevar a cabo estos cálculos, y que consiste en suponer que las variables X1 = x1;X2 = x2;X3 = x3; : : : son independientes entre s\_\_. En ese caso,

P(X1 = x1;X2 = x2;X3 = x3; : : : jY = yi) =

Y

i

P(Xi = xijY = yi);

y el cálculo se aligera notablemente. Sin embargo, esta hipótesis es generalmente la que más problemas nos puede dar con estos modelos, [8], debido a que es muy poco habitual en los problemas reales que las características sobre las que construimos el modelo sean independientes.

OCTOPUS DATA INSIGHTS 55

Según la documentación de scikit-learn sobre el modelo MultinomialNB()3, este modelo en principio deberrá usarse para la clasificación con características discretas (por ejemplo, conteo de palabras en el caso de clasificación de textos), ya que la distribución multinomial normalmente requiere valores discretos. Sin embargo, en la práctica, conteos no enteros, como el del TF-IDF, también funcionan.

Las siguientes líneas de código son las que hacen el trabajo de entrenamiento del modelo

:

Figura 5.2.2: Código para el entrenamiento del modelo de clasificación de relevancia.

Una vez entrenado el modelo, lo queremos usar en dos fases: en esta de selección de usuarios y en una posterior para la ordenación de los candidatos (ver sección 6.1). Por ello, hemos guardado el modelo para exportarlo al \_chero Python/relevance model/modelo clf.sav y tener la ocasión posteriormente de importarlo de nuevo. El uso que vamos a hacer del modelo para clasificación espor ello a través de una importación:



Figura 5.2.3: Código para usar el modelo Naive Bayes de clasificación de contenidos ya entrenado en Python/relevance model/Clasificacion1.py.

Previo a predecir con el modelo, los textos han de pasar por el mismo proceso de limpieza que aquellos textos con los que se entrenó el modelo. Por ello, antes de llamar a esta función, hay que procesar los textos sobre los que vamos a aplicarlo:



Figura 5.2.4: Código para limpiar los textos y posteriormente usar el modelo Naive Bayes de clasificación de contenidos ya entrenado en Python/user selection.py.

3http://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.MultinomialNB.html

56 OCTOPUS DATA INSIGHTS

**Resultados del modelo de clasificación:**

Para entender mejor como el modelo está funcionando, tenemos las métricas de precision, recall, f1-score , confusion matrix. Pero antes tenemos que entender algunas terminologías para clasificación.

|  |  |
| --- | --- |
| TP (true positive): clasificación correcta de la clase positiva.  TN (true negative): clasificación correcta de la clase negativa.  FP (false positive): clasificación incorrecta de la clase positiva.  FN (false negative): clasificación incorrecta de la clase negativa. | Image result for confusion matrix |

Precision: TP / (TP + FP) => cuanto preciso es el modelo para detectar positivos

Recall: TP / (TP + FN) => cuanto sensitivity es el modelo para detectar positivos

F1-Score: 2\* ((Precision \* Recall) / (Precision + Recall) => métrica más completa para medir la eficiencia del modelo

Confusion Metrics:

1 0

1 [[ 11 61]

0 [ 12 516]]

Métricas de clasificación:

precision recall f1-score support

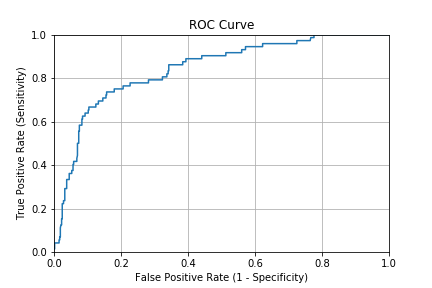
0 0.89 0.98 0.93 528

1 0.48 0.15 0.23 72

avg / total 0.84 0.88 0.85 600

Acurácia ROC: 0.842500526094

ROC and AUC of MultinomialNB



Los resultados están buenos y hemos decidido empezar el proyecto con el modelo así para hacer las pruebas hasta el final del proyecto, pero estamos seguros que es posible mejorarlo, principalmente en la precisión de los aciertos en la clase positivas (TP) y la curva ROC. Hemos empezado a hacer pruebas con otros algoritmos y métodos de optimización.

5.2.1 Fase de mejoras

Para ello, hemos hechos pruebas con Multinomial NB y SVM, buscando los mejores parámetros para optimizar la eficiencia del modelo (Hyperparameters Optimization) a través de GridSearchCV() y cambiando el método de muestra de simples de separación (70/30) entre entreno y teste para el uso del método K-fold cross validation que separa el dataset en K “folds” de igual tamaño y cada “fold” actúa una vez como base de teste y K-1 vez como entreno.

**Mejorando el modelo con GridSearchCV – Multinomial NB**

*parameters = {'tfidf\_\_ngram\_range': [(1,1),(1,2)],*

*'tfidf\_\_use\_idf': (True, False),*

*'tfidf\_\_sublinear\_tf': (True, False),*

*'tfidf\_\_min\_df':(0, 5, 10),*

*'tfidf\_\_max\_df':(0.95, 0.90, 0.85),*

*'clf\_\_alpha': (0.001, 0.01, 0.1, 1, 10),*

*}*

*text\_cl = Pipeline([('tfidf', TfidfVectorizer()),('clf', MultinomialNB())])*

*gs\_clf = GridSearchCV(text\_cl, parameters, cv = 5, scoring='roc\_auc', n\_jobs=-1)*

*gs\_clf = gs\_clf.fit(x\_train, y\_train)*

*# Aplicando el modelo en la base de teste*

*gs\_pred = gs\_clf.predict(x\_test)*

Confusion Metrics:

[[524 4]

[ 30 42]]

precision recall f1-score support

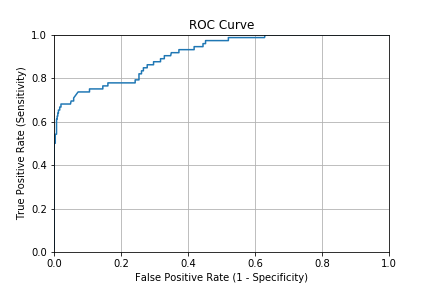
0 0.95 0.99 0.97 528

1 0.91 0.58 0.71 72

avg / total 0.94 0.94 0.94 600

Acuracia ROC: 0.910208859428

ROC and AUC of MultinomialNB



**Mejorando el modelo con GridSearchCV – SVM**

*parameters = {'tfidf\_\_ngram\_range': [(1,1),(1,2)],*

*'tfidf\_\_use\_idf': (True, False),*

*'tfidf\_\_sublinear\_tf': (True, False),*

*'tfidf\_\_min\_df':(0, 5, 10),*

*'tfidf\_\_max\_df':(0.95, 0.90, 0.85),*

*'clf\_\_C': (1, 10, 100, 1000),*

*'clf\_\_tol':(0.001, 0.01, 0.1),*

*}*

*text\_cl = Pipeline([('tfidf', TfidfVectorizer()),*

*('clf', LinearSVC())])*

*gs\_clf = GridSearchCV(text\_clf, parameters, cv = 5, scoring='roc\_auc', n\_jobs=-1)*

*gs\_clf = gs\_clf.fit(x\_train, y\_train)*

*# Aplicando el modelo en la base de teste*

*gs\_pred = gs\_clf.predict(x\_test)*

Confusion Metrics:

[[521 7]

[ 26 46]]

precision recall f1-score support

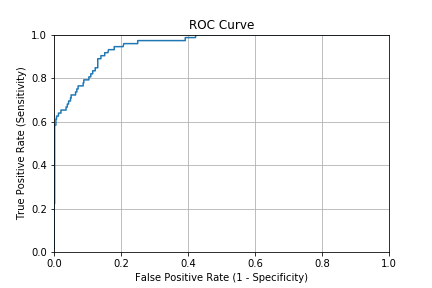
0 0.95 0.99 0.97 528

1 0.87 0.64 0.74 72

avg / total 0.94 0.94 0.94 600

Acuracia ROC: 0.952033354377

ROC and AUC of SVM



Conclusión:

Los Modelos han resultado en acurácia y métricas muy parecidas. El modelo SVM está un poco mejor que el NB, sin embargo, el NB tiene un procesamiento más rápido y simples. Por eso creo que será necesario procesar los dos modelos en el proyecto para llegar a una conclusión, pero si en el futuro es necesario tener velocidad, usaríamos el NB.