DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1

Aprendizaje Automático

Integrante	LU	Correo electrónico
Gustavo Cairo	89/13	gj.cairo@gmail.com
Germán Pinzón	475/13	pinzon.german.94@gmail.com
Ángel More	931/12	angel_21_fer@hotmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción

El presente trabajo tiene como objetivo elaborar un clasificador de mails en Spam y Ham. Para lograr esto trabajaremos con distintos modelos de predicción.

Dado que los modelos necesitan de ciertos atributos para poder analizar, catalogar y así predecir , extraeremos un conjunto de atributos significativos de nuestro conjunto de mail.

Trabajaremos sobre un total de 90K de mails, de los cuales utilizaremos 70K como base de entrenamiento para los modelos y el resto como test set. Evaluaremos distintas combinaciones y variaciones dentro de los modelos comparando la exactitud de cada uno así como el costo temporal que estos implican.

Finalmente seleccionaremos los mejores y los utilizaremos para clasificar a los mails perteneciente al test set y de esta manera tener una aproximación de como se comportaría el clasificador frente a nuevos datos.

El código será elaborado en lenguaje python. Como bibliotecas destacadas utilizaremos $Pandas$, esta nos permitirá el almacenamiento de los mails y sus atributos dentro de la estructura $DataFrame$.

Las funciones que representen a los distintos modelos serán provistas por la biblioteca $sklearn$.

1.1. Extracción de atributos

Se realizaron un total de 505 extracciones de atributos sobre, aproximadamente, 70K de mails. Dicho conjunto de mails sera nuestro training set mientras que los 20K restantes se corresponderán con nuestro test set. Estos 20.000 mails de test fueron extraídos de manera aleatoria utilizando la función $random.sample$, y no fueron utilizados para el entrenamiento de los clasificadores.

Los atributos seleccionados fueron los siguientes:

- *len*: longitud de cada mail.
- *count_spaces*: cantidad de espacios que contienen los mails.
- *links*: cantidad de enlaces que contienen los mails. Lo consideramos importante ya que suponemos que un tipo de mail spam generalmente es aquel con publicidades y estos contienen distintos enlaces a sus diversos servicios comerciales; o así haciendo referencia a noticias.
- *tags*: cantidad de tags HTML encontrados. A veces podemos encontrar en los mails no sólo enlaces, sino imágenes, publicidades, noticias, etc. que se pueden visualizar dentro del propio mail, así como también formateo excesivo de los mismos (muchos colores, etc.) por lo que la presencia y/o cantidad de etiquetas HTML deberían constituir otro atributo distintivo.
- *rare*: este atributo representa la cantidad de caractere
- s que podrían ser considerados poco comunes de encontrar en un mail del tipo Ham.
- *500 palabras que más aparecieron en los mails*: se extrajeron las 500 palabras más utilizadas del total de mails utilizados como base de entrenamiento.
  Para la extracción se exceptuó tanto el encabezado de cada mail como la porción de texto que corresponde a mensajes reenviados. Tomamos estas decisiones porque de esta manera estamos excluyendo palabras que usualmente pueden estar presentes sin importar el tipo de mail, y por lo que considerarlas no nos brindaría ningún tipo información representativa a la hora de querer clasificarlos.

1.2. Modelos

Decidimos trabajar con los siguientes algoritmos de aprendizaje:

- Decision Tree Classifier
- Multinomial Naive Bayes Classifier
- Gaussian Naive Bayes Classifier
- KNN Classifier
- SVM Classifier
- Random Forest Classifier

Para todos los algoritmos que tuvieran hiperparámetros, elegimos un subset de los mismos (en base a nuestro conocimiento sobre ellos) y ejecutamos grid searches con varias alternativas de valores para cada uno. Ésto nos aseguró conseguir los mejores hiperparámetros (de entre los propuestos por nosotros) para cada clasificador.

A continuación presentamos cuáles fueron los hiperparámetros seleccionados para cada algoritmo, y cuál fue su score correspondiente luego de ejecutar cross-validation de 10 folds con dichos parámetros.

1.2.1. Decision Tree Classifier

Para el Decision Tree Classifier variamos los siguientes hiperparámetros:

- *criterion:* el criterio para elegir qué atributo usar para hacer split.
- *max_features:* el número de atributos a tener en cuenta para buscar el mejor split.
- *max_depth:* la profundidad máxima del árbol (o sin límite).
- *min_samples_split:* el mínimo número de instancias necesarias para subdividir un nodo.

Grid Search arrojó que, entre todas las combinaciones propuestas, el mejor *Decision Tree Classifier* era el que usaba *entropy* como *criterion*, tenía *max_features* igual a 10, *max_depth* sin límite, y *min_samples_split* igual a 5.

El *accuracy* arrojado luego de hacer cross-validation fue aproximadamente 0.96 ($+/-0.012$).

1.2.2. Multinomial Naive Bayes Classifier

Para el Multinomial Naive Bayes Classifier variamos los siguientes hiperparámetros:

- *alpha:* este parámetro modifica (*suaviza*) la probabilidad de que un parámetro pertenezca a una clase.[1]
- *fit_prior:* este boolean determina si el algoritmo debería aprender la probabilidad de cada clase, o suponer una distribución uniforme.

Grid Search arrojó que, entre todas las combinaciones propuestas, el mejor *Multinomial Naive Bayes Classifier* era el que usaba *fit_prior* en `true`, y *alpha* en 0.25.

El *accuracy* arrojado luego de hacer cross-validation fue aproximadamente 0.65 ($+/-0.06$).

1.2.3. Gaussian Naive Bayes Classifier

El Gaussian Naive Bayes Classifier no toma parámetros, por lo que simplemente ejecutamos cross-validation. El *accuracy* arrojado fue de 0.6 ($+/-0.1$).

[1] Está relacionado a la frecuencia relativa. Más información: http://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes

1.2.4. KNN Classifier

Para el KNN Classifier variamos los siguientes hiperparámetros:

- *n_neighbors:* este parámetro define la cantidad de vecinos a usar.
- *weights:* este parámetro define si los pesos dados a los vecinos son uniformes, o inversos a su distancia.

Hay un tercer parámetro con el cual intentamos experimentar, *algorithm*, pero como utilizamos estructuras de datos esparsas para guardar el dataset y los atributos, el único algoritmo que funciona con estas estructuras (sin convertirlas en densas) es fuerza bruta. Al intentar utilizar algoritmos distintos, las máquinas en las cuales intentamos correr la Grid Search se quedaban sin memoria, así que optamos por utilizar simplemente el algoritmo de fuerza bruta.

Grid Search arrojó que, entre todas las combinaciones propuestas, el mejor *KNN Classifier* era el que usaba *n_neighbors* en 3, y *weights* igual a *distance*.
El *accuracy* arrojado luego de hacer cross-validation fue aproximadamente 0.89 ($+/-0.03$).

1.2.5. Gaussian Naive Bayes Classifier

El Gaussian Naive Bayes Classifier no toma parámetros, por lo que simplemente ejecutamos cross-validation. El *accuracy* arrojado fue de 0.6 ($+/-0.1$).

1.2.6. SVM Classifier

En SVM buscamos variar los siguientes hiperparámetros:

- *kernel:* Este parámetro sirve para especificar el tipo de algoritmos que se utilizará
- *degree:* Parámetro utilizado como grado para las funciones de kernel del tipo polinómica.

Si bien intentamos variar el tipo de kernel y los grados. Sólo obtuvimos la perfomance para kernel en modo rbf. La causa fue el tiempo que demoro este clasificador. Sin reducciones el tiempo total fue de aproximadamente 15 hs. Aunque intentamos seguir experimentando aplicándole reducciones y/o transformaciones, el tiempo seguía siendo excesivamente alto comparándolo con los demás. Por lo que nos concentramos en tratar de probar las mayores y mejores variantes en el resto de los clasificadores.

1.2.7. Random Forest Classifier

Para el Random Forest Classifier variamos los siguientes hiperparámetros:

- *n_estimators:* este parámetro define la cantidad de árboles en el bosque.
- *criterion:* el criterio para elegir qué atributo usar para hacer split.
- *max_features:* el número de atributos a tener en cuenta para buscar el mejor split.
- *max_depth:* la profundidad máxima del árbol (o sin límite).
- *min_samples_split:* el mínimo número de instancias necesarias para subdividir un nodo.

Grid Search arrojó que, entre todas las combinaciones propuestas, el mejor *Random Forest Classifier* era el que usaba 20 árboles, *criterion* igual a *gini*, *max_features* igual a 1, *max_depth* igual a 2, y *min_samples_split* también igual a 2.
El *accuracy* arrojado luego de hacer cross-validation fue aproximadamente 0.98 ($+/-0.006$).

1.3. Reducción de dimensionalidad

Las técnicas que decidimos emplear para reducir la dimensionalidad fueron PCA, selección de K mejores atributos y selección de percentil. Estas tres técnicas fueron combinadas siempre que fue posible y probadas con distintos parámetros en el caso de las últimas dos.

Para el caso de selección de K mejores atributos variamos el K con los siguientes valores: 20, 50, 100, 150, 200, 250, 300, 320, 370, 400, 420 y 450 ya que pensamos que constituían una gama relativamente amplia de valores posibles, considerando que estamos trabajando con un total de 505 atributos. En el caso de la selección de percentil variamos el parámetro percentil con los valores 5, 10, 15 y 25. Además, dado que en algunos casos decidimos hacer una combinación de técnicas aplicando primero PCA y luego alguna de las otras dos, también utilizamos dos funciones distintas en el parámetro $score\_func$ correspondiente a $SelectKBest$ y $SelectPercentile$.

Las funciones que utilizamos en el parámetro $score\_func$ fueron $chi2$ y $f\_classif$. Cuando decidimos aplicar primero PCA y luego algún tipo de selección entonces tuvimos que utilizar la función $f\_classif$ debido a que si no se presentaban inconvenientes por valores negativos que aparecían en la matriz luego de la transformación que aplicaba PCA ($chi2$ sólo acepta valores positivos, y PCA dejaba algunos valores negativos en el nuevo espacio). Sin embargo cuando aplicamos las funciones de selección por separado, sin combinarlas con PCA, pudimos experimentar tanto con $chi2$ como con $f\_classif$.

Un aspecto importante a rescatar es que tuvimos que hacer utilización de la clase $Pipeline$ de sklearn. Esto se debe a que como vamos a estar realizando ciertas transformaciones a los datos con los que nuestro clasificador vaya a entrenar, es necesario que dichas transformaciones se realicen en el momento indicado que es dentro de la función $cross\_val\_score$ (que usamos para medir la performance de nuestro clasificador en esta instancia). Así, si por ejemplo decidimos aplicar primero PCA y luego $SelectKBest$ con $chi2$ y $K = 200$, entonces estas dos funciones (con parámetros en el caso de la segunda) van a ser steps en el pipeline que utilizará $cross\_val\_score$.

Finalmente para poder tener cierta variedad de transformaciones posibles en los datos a la hora de aplicar los clasificadores y decidir con cual quedarnos, vamos a estar aplicando lo siguiente:

- PCA
- $SelectKBest$ con los valores de K mencionados anteriormente y el parámetro $score\_func$ tanto con $f\_classif$ como con $chi2$
- $SelectPercentile$ con los valores de $percentile$ mencionados anteriormente
- PCA y luego $SelectKBest$ (mismos valores de K pero con $f\_classif$ en $score\_func$)
- PCA y luego $SelectPercentile$ (mismos valores de $percentile$)

Luego de hacer las transformaciones anteriores a los datos se aplicará cada uno de los clasificadores elegidos a excepción de Naive Bayes Multinomial en los casos donde se utilice PCA ya que este clasificador no puede tratar ciertos valores negativos que aparecen en la matriz luego de la transformación.

1.4. Resultados y selección del mejor clasificador

Podemos describir nuestra estrategia para la elección del mejor clasificador de la siguiente forma:

- Luego de realizar Grid Search y buscar los mejores parámetros dentro de cierto rango para cada clasificador (a excepción de SVM por su complejidad temporal y $GaussianNB$ porque no recibe parámetros), comparamos los valores de *accuracy* arrojados por cross-validation (detallados en la sección de Modelos). En esta etapa ya es notorio que los clasificadores más performantes son Random Forest y Decision Tree, seguidos de KNN, y por último y con bastante peor desempeño, Naive Bayes Multinomial (se mostrarán los gráficos con los valores específicos más abajo).

- Por otro lado, como explicamos en la sección de reducción de dimensionalidad, también calculamos el score de cross-validation para cada clasificador sobre los datos transformados por cada una de las técnicas (o combinación de técnicas).

Una vez hecho lo anterior, procedimos a elegir la mejor configuración de datos que recibe cada clasificador. Es decir, en algunos casos una reducción de dimensionalidad pudo resultar más beneficiosa y en otros no, con lo cual para cada clasificador decidimos si vamos a transformar los datos antes de utilizarlo (y qué tipo de transformación vamos a estar realizando) o no.

Si bien más abajo se mostrarán los resultados de una manera gráfica y precisa, por lo general pudimos observar que aplicar alguna transformación a los datos siempre va a resultar mejor que no hacerlo. Una vez hecho esto, y seleccionados los clasificadores con sus correspondientes transformaciones, los aplicamos sobre los datos de test (que no fueron utilizados antes). Esto nos brinda una visión mucho más realista de la performance de nuestros clasificadores y nos da cierta seguridad de no haber estado haciendo *overfitting* ya que, como mencionamos, estos datos fueron separados desde el principio y los clasificadores no fueron entrenados con ellos.

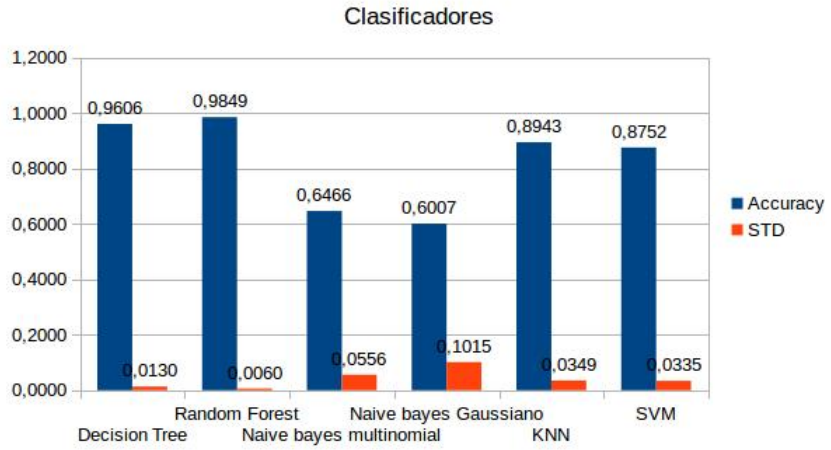

Figura 1: Se muestran los clasificadores con un accuracy mayor a 0.8, sin realizar ningun tipo de transformación o reducción

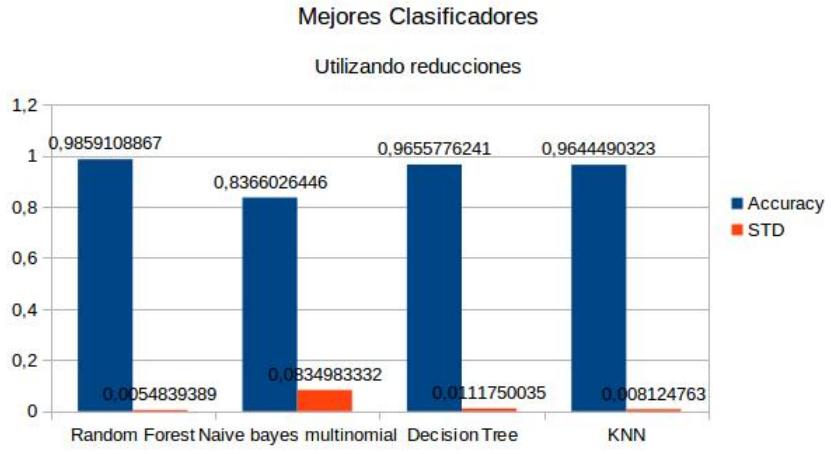

Figura 2: Se muestran los clasificadores con un accuracy mayor a 0.8 y las siguientes reducciones:
Random Forest: 400 mejores atributos, usando chi2.
Naive Bayes Multinomial: 200 mejores atributos, usando f_classif.
Decision tree: 200 mejores atributos, usando chi2.
KNN: 50 mejores atributos, usando PCA y f_classif.

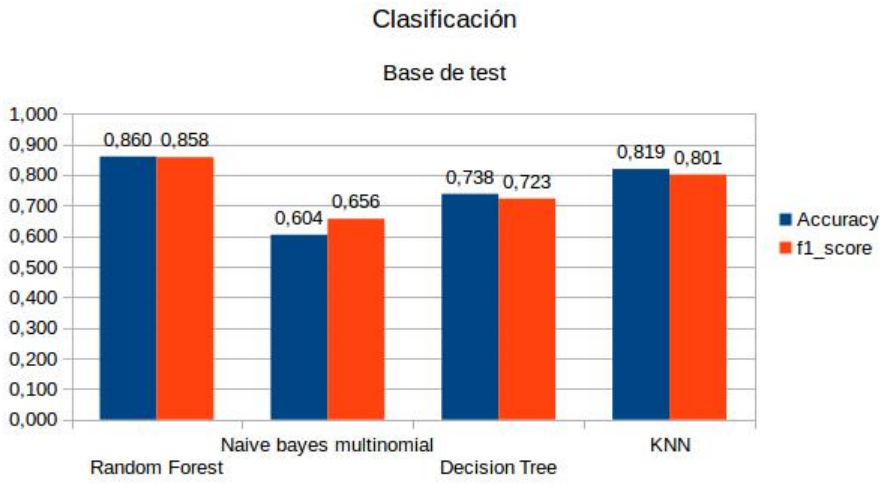

Figura 3: Se muestran los clasificadores aplicados a la base de test, con las reducciones de la figura anterior.

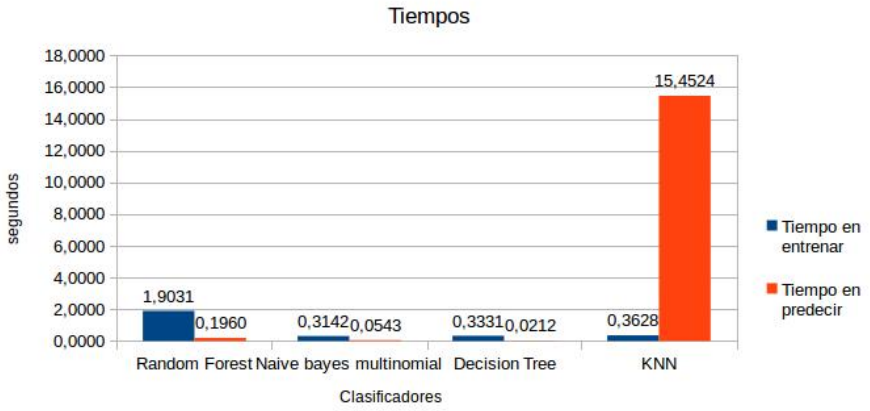

Figura 4: Tiempos registrados durante la calsificación de los mails de la base de test

1.5. Discusión

A partir de las técnicas utilizadas y los distintos clasificadores hemos podido ver que los atributos seleccionados como los mejores generalmente variaban de un modelo a otro. Muchas veces esto ocurrió aún dentro de un mismo clasificador pero variando parámetros por ejemplo $chi2$ o $f\_classif$.

Aunque en nuestra selección de atributos predominan los del tipo ”500 palabras mas usadas”hemos visto que en algunos casos se ha tomado como los de mayor poder predictivo a los otros atributos. Con la reducción de dimensionalidad pudimos observar por un lado que los tiempos de ejecución de los distintos clasificadores disminuyeron respecto de sus versiones sin reducción. Además, observamos que en la mayoría de los casos al incrementar la cantidad de atributos el acurracy aumentaba pero cada vez de manera menos significativa. Incluso en algunos casos al aumentar el número de atributos el accuracy bajaba levemente de una iteración a otra.

Esto nos llevo a buscar el mejor equilibrio entre las cantidad por un lado con pocos atributos tengo información escasa para poder distinguir entre un Spam y Ham pero, con demasiados atributos no solo aumenta el tiempo de computo sino que tener determinada combinación de atributos pueden empeorar la perfomance o caer en overfitting.

Desde un primer momento observamos que Decision Tree y Random Forest fueron los que predominaron en todo momento la mayor exactitud se encontraba entre estos clasificadores. Aun así luego de observar todos los datos decidimos quedarnos con aquellos clasificadores que habían presentado una

exactitud mayor al 80 %, ya que por lo general estos clasificadores mantuvieron dicha calidad a lo largo de todas la experimentaciones. Una vez probados con la base de test este patrón vario un poco. En primer lugar la exactitud disminuyo mas de lo que habíamos pensado, lo cual puede ser causa de que, aun con las reducciones elegidas, estábamos overfiteando. Lo cual también pudo ser causa de que los clasificadores que lideraron la exactitud, no fueron los mismos que durante la experimentación. Si bien Random Forest predomino, el segundo mejor fue KNN.

Mientras que en lo que respecta a los clasificadores del tipo Naive Bayes en casi todos los casos experimentados la perfomance fue bastante baja. La causa principal de esto podría ser que los atributos están demasiados relacionados como para ser tratados de manera independiente. Siendo desfavorable para el algoritmo.