DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Aprendizaje automático

Trabajo práctico 1

SpamFilter

Resumen

Aprendizaje automático - SpamFilter

Integrante	LU	Correo electrónico
Negri, Franco	893/13	franconegri2004@hotmail.com
Rey, Martín Gastón Podavini	483/12	marto.rey2006@gmail.com
Valdés Castro, Tobías	800/12	tobias.vc@hotmail.com

Palabras claves:

TP

Índice

1. Introducción

En este trabajo práctico nos propondremos clasificar mails en *spam* como en no *spam* (conocido también como *ham*).

Para esto analizaremos los datos en busca de atributos útiles y luego experimentaremos con diversos clasificadores con la intención de encontrar los que mejor clasifiquen.

Además utilizaremos técnicas de reducción de dimensionalidad con la intención de mejorar aún más los modelos de los puntos anteriores.

Para tener una buena métrica de los resultados obtenidos, apartamos una porción del set de datos que nos fue entregado para ser utilzado al final de la experimentación y así tener una visión realista de qué tan buenos son los clasificadores elegidos. Con el resto de los datos, realizamos *kfolds* con $k = 10$ con la intención de minimizar en cierta medida el *overfitting* de datos.

2. Desarrollo

Como ya adelantamos en la seccion anterior, el objetivo de este trabajo práctico será clasificar mails en *spam* y *ham*. Para ello, comenzaremos seleccionando atributos adecuados que consideramos que dividen bien el problema.

2.1. Selección de atributos

Para la selección de atributos observamos parte del *JSON* entregado en busca de palabras claves que pudieran distinguir entre *spam* y *ham*.

De este análisis se encontró que palabras tales como *viagra* o *nigeria* son muy frecuentes en los mensajes de *spam*, como también aquellas que hagan referencia a negocios o a dinero, por lo que incluímos atributos que cuenten las veces que son mencionadas estas palabras en el cuerpo y el encabezado del mensaje. Así tambien observamos que los mails de *spam* tienden a tener enlaces hacia sitios web o contenido HTML en el cuerpo del mensaje por lo que también contamos la cantidad de ocurrencias de palabras tales como *html* o *http* y símbolos especiales como la barra invertida, o el *hashtag*.

De esta menera reunimos alrededor de *100* atributos que utilizaremos a continuación para la experimentación.

2.2. Experimentación

Los clasificadores que utilizaremos para experimentar en este trabajo serán:

- Árboles de Decisión
- Multinomial Naive Bayes
- Vecinos más cercanos
- SVM (SVC)
- Random Forest

Para cada uno de ellos utilizaremos *GridSearch* para intentar encontrar los hiperparámetros que logren la mejor clasificación.

En cada una de las experimentaciones utilizamos *GridSearch* de *sklearn* con 10 *kfolds*.

2.2.1. Árboles de decisión

Como vimos en clase, este clasificador intentará aplicar una serie de reglas sucesivas para determinar la clasificación. Por ejemplo, un posible árbol de decisión para clasificar *spam* podría ser: si el mensaje contiene la palabra *nigeria* mas de 4 veces, es *spam*, si no, podríamos ver cuántas veces se menciona *html* en el mensaje, si este resultado está entre 4 y 7 es *spam*, si no no... y así sucesivamente.

En particular, los hiperparámetros que queremos encontrar para este algoritmo en particular son: la máxima profunididad del árbol, y la cantidad mínima de muestras necesarias para dividir un nodo.

Experimentamos variando los hiperparámetros entre los siguientes valores:

- $max\_depth : 1, 3, 5, 10, 15, 50, 100$
- $min\_samples\_split : 1, 3, 5, 10, 15$

El mejor resultado obtenido fue de 0,965975308642 con un max_depth de 50 y un *min sample split* de 1.

2.2.2. Multinomial Naive Bayes

El objetivo del Naive Bayes será encontrar la clase más probable de cada instancia en base al cálculo de la probabilidad bayesiana.

Para este clasificador queremos encontrar el mejor suavizado laplaciano ($alfa$), para ello experimentamos con los siguientes valores:

- $alfa$: 0,1, 0,2, 0,3, 0,4, 0,5, 0,6, 0,7, 0,8, 0,8, 0,9, 1

El resultado del *GridSearch* arroja que el mejor score resulto ser de 0,587740740741 con un alpha de 0,1

2.2.3. Vecinos más cercanos

La idea tras este clasificador es dada una nueva instancia a clasificar, comparar sus atributos contra todos los de la base de entrenamiento y quedarnos con k vecinos más *cercanos*. La *cercanía* de una instancia con otra puede calcularse como la norma 2 de los atributos al cuadrado o alguna otra función que nos permita definir una distancia relativa entre dos hiperplanos. Para este clasificador queremos determinar la cantidad de vecinos óptima y la función de peso a utilizar en la predicción. Para esto último tendremos dos posibilidades, con pesos uniformes o por distancia. Para dar un ejemplo sobre cuál es la diferencia, supongamos que seteo como hiperparámetro que la cantidad de vecinos es igual a 3 y para un mensaje dado obtengo que el primer vecino es *spam* y los dos restantes son *ham*. Con una función uniforme, se determinará que el mail es *ham*, ya que *ham* es mayoría. Pero podría darse el caso en que el primer vecino se encuentre a una distancia ínfima del mail a clasificar y que los otros dos se encuentren a una distancia extremadamente grande. En ese caso sería más razonable asignarle mayor peso al primer vecino de manera tal que el mail sea clasificado como *spam*. Esta segunda solución será la de utlizar pesos variables que dependan de la distancia.

Para este clasificador probamos con $1, 3, 5, 7, 10$ y 15 vecinos y el mejor resultado obtenido fue de 0,908358024691 con una cantidad de vecinos igual a 1 y pesos uniformes.

2.2.4. SVM (SVC)

2.2.5. Random Forest

Este clasificador consiste en crear una gran cantidad de árboles de desición, y al momento de clasificar se elige el resultado que sea la moda de las clasificaciones, determinadas por los árboles individuales.

Para este estimador queremos definir la cantidad de árboles de decisión a utilizar, la profundidad máxima que se le permitirá tener a cada árbol, y la máxima cantidad de atributos a considerar cuando se esté realizando la división de un nodo.

Experimentamos con $2, 5, 10, 15, 40, 100$ árboles, $2, 5, 10, 20$ atributos a considerar al momento de la división de un nodo y una profundidad máxima de $3, 5, 20$ y sin restricciones.

Para este estimador, el mejor resultado fue de 0,979098765432 con una cantidad de árboles igual a 100, una profundidad irrestricta y una cantidad de *features* a examinar igual a 10.

2.3. PCA

Para intentar mejorar la precisión de los resultados utilizamos PCA para obtener las componentes principales de nuestro set de atributos.

Además, utlizamos *GridSearch* para obtener cuál era la cantidad óptima de componentes principales para cada uno de los algoritmos. Utilizamos $2, 5, 10, 40, 70$ componentes para la experimentación.

Lo que observamos fue que tanto en árboles de decisión como en Random Forests las respuestas resultaron ser levemente peores, pasando de un mejor resultado de 0,965975308642 a 0,931592592593 y de 0,967962962963 a 0,979098765432, respectivamente.

Para el algoritmo de Naive Bayes esta vez utilizamos un modelo gaussiano, pero los resultados siguieron sin ser muy buenos, obteniendo un score del 0,596308641975

Para vecinos más cercanos, el cambio de performance no fue notable, obteniendo una respuesta de 0.908.

Si bien entendemos que los resultados del *GridSearch* podrían haber sido mejores variando nuevamente los parámetros del apartado anterior junto con la cantidad de componentes (sobre todo en el caso de vecinos más cercanos), esto también resultaba muy caro computacionalmente y por cuestiones de tiempo *no* lo incluímos en el trabajo.

3. Concluciones