informe

December 7, 2016

```
Reducción de dimensionalidad: Describir brevemente las técnicas empleadas.
Resultados: Describir los resultados conseguidos por los distintos modelos y conjur
Discusión: Analizar los resultados, buscando responder cuestiones como, por ejemplo
```

La longitud sugerida del informe es de entre 3 y 5 páginas de texto (sin contar tablas o figuras). Además pueden incluirse tablas y figuras, pero siempre deben ser referenciadas y explicadas en el texto.

Si se tomaron ideas de la literatura (papers, libros, blogs, wikipedia o lo que sea), citar claramente las fuentes (autor, título, tipo de publicación, año de publicación, URL si corresponde, etc.).

No incluir código. Si es necesario describir un algoritmo, hacerlo en pseudocódigo

```
Populating the interactive namespace from numpy and matplotlib
```

1 Resumen del trabajo

El objetivo de este trabajo fue explorar la performance de distintos clasificadores en la tarea de detección de spam. Se procedió, sobre un corpus de mails etiquetados positiva o negativamente como spam, primeramente a elegir un conjunto de atributos.

Para cada uno de los tipos de clasificadores explorados (`Decision Tree`, `Random Forest`, `KNN`, `Näive Bayes`, y `Support Vector Machine`) optimizamos sus parámetros para mejorar la performance sobre el dataset, a la vez que investigamos cómo afectaba a su rendimiento la utilización de `Principal Component Analysis`.

2 Métodos y Materiales

2.1 Dataset

El dataset constó de 90000 mails, equilibrados entre `spam` y `ham`. De este total de mails, separamos un 10% para test, y usamos el 90% restante como desarrollo.

2.2 Extracción de Atributos

Los atributos principales que se extrajeron de los mails fueron:

- Atributos generados por TF-IDF
- Atributos escogidos manualmente

Nos quedamos con 200 atributos del primero, mientras que del segundo extrajimos 73 adicionales, sumando un total de 273 atributos por mail.

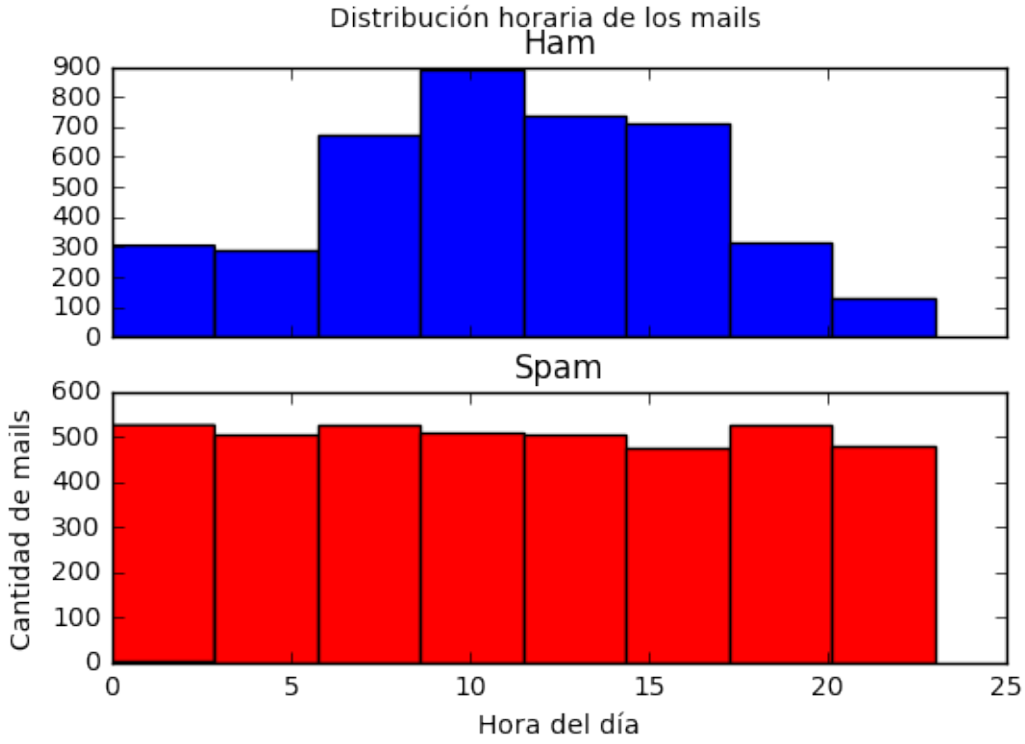
Los atributos extraídos mediante `TF-IDF` fueron realizados sobre el payload del mail y no sobre el texto completo. Para el parsing de los mails en texto plano utilizamos la librería estándar `email`[cita de Python mientras que para la generación de los atributos con `TF-IDF` utilizamos el vectorizador correspondiente de `sklearn`.

Los atributos manuales que extrajimos fueron tanto del header como otros del payload, luego de un análisis observacional del dataset. Del encabezado del mensaje extrajimos:

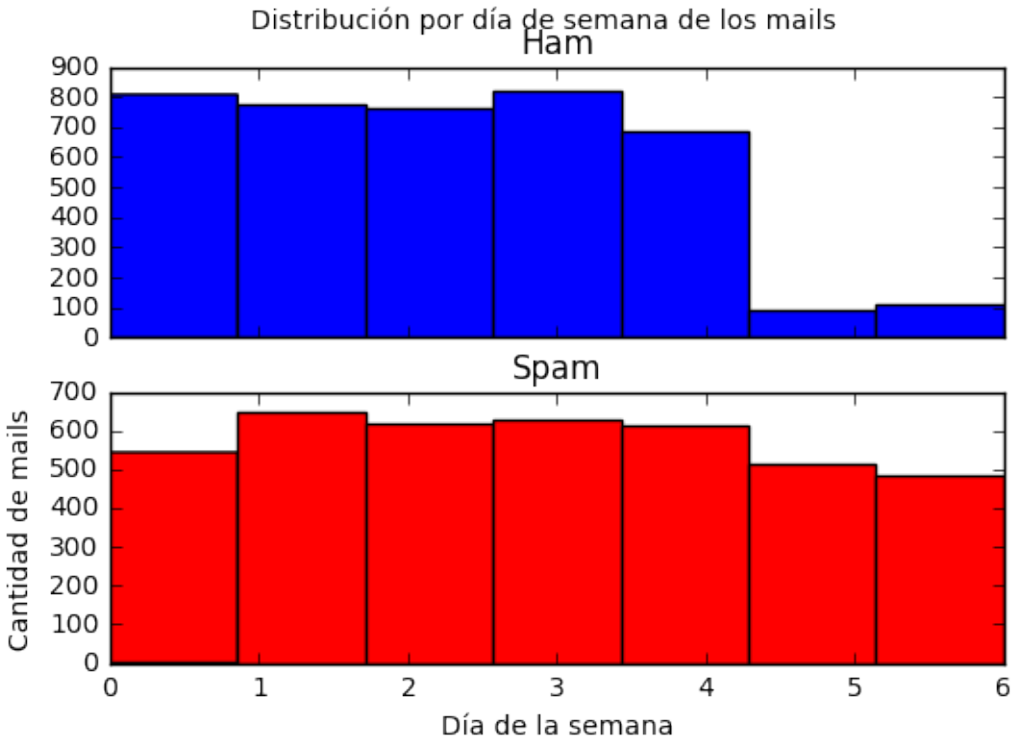
- El `content-type` del mail: si son texto, imágenes, html, etc o bien una mezcla de estos
- La hora en la que fue enviado
- ¿El mail fue enviado entre las 7 y las 20hs?
- El día de la semana (Lunes a Domingo)
- ¿Fue enviado el fin de semana?
- El día del mes (1-31)
- El año en el que fue enviado
- La cantidad de receptores del mail
- ¿El nombre del sender tiene algún caracter no ASCII?

Los atributos temporales del mail fueron extraídos luego de analizar el dataset. Por ejemplo, podemos ver en las figuras siguientes que la distribución horaria del ham tiene casi el 77% concentrado en el rango de 7 a 20 hs, mientras que el spam se reparte uniformemente. También puede observarse como el 95% del ham se envía durante los días de semana, mientras que el spam también se reparte de igual manera en cada día.

```
Frecuencia de Ham entre 5 y 20 hs = 0.77037037037
```



```
Frecuencia de Ham en día de semana 0.94962962963
```



2.3 Clasificadores Utilizados

Los tipos de clasificadores explorados en este trabajo fueron los siguientes:

- Decision Tree
- Random Forest
- K-Nearest Neighbors
- Naive Bayes
- Support Vector Machines

Veamos brevemente los hiperparámetros de cada uno de estos modelos.

2.3.1 Decision Trees

La implementación de árboles de decisión en `sklearn` implementa CART. Los hiperparámetros que elegimos optimizar fueron:

- min_samples_split: cuántos elementos tengo que tener como mínimo para partir un nodo

- criterion: gini o entropy, dos medidas distintas de dispersión
- max_depth: profundidad máxima del árbol
- max_features: qué porcentaje de variables (aleatorio) tomo a la hora de partir un nodo

2.3.2 Random Forest

Leyendo la documentación de sklearn respecto a Random Forest, decidimos optimizar los siguientes parámetros:

- min_samples_split y max_features: Idem a Decision Trees
- n_estimators: Cantidad de árboles

Si bien es verdad que cuantos más árboles mejor performance tiene el modelo, también es cierto que a partir de cierto número de estos no mejora significativamente, con lo cual decidimos probarlo también.

2.3.3 K-Nearest Neighbors (KNN)

En el caso de KNN experimentamos con los hiperparametros `n_neighbors`, `weights` y `metric`. `n_neighbors` parametriza la cantidad de consideramos el rango entre 100 y 1000. Dada una nueva instancia el algoritmo devolvera la clase mas ocurrente entre las n_neighbors mas cercanas weights: hace referencia al peso de cada punto, al elegir distance como unica opcion de weight estamos diciendo que los vecinos mas cercanos tienen mas influencia sobre la nueva instancia que los mas lejanos. metric: se refiere a como medir la distancia que mencionamos en el hiperparametro anterior. Dentro de las metricas disponibles en sklearn.neighbors.DistanceMetric decidimos experimentar con "canberra" y "braycurtis".

2.3.4 SVM

Para SVM decidimos exploramos los siguientes conjuntos de hiperparametros: - kernel : lineal o rbf - C: np.arange(0.2, 5.0, 0.4)

En el caso del kernel rbf, consideramos distintas opciones para gamma (entre 0.01, 1.00 con saltos de 0.01) Intentamos estudiar el caso donde el kernel era un polinomio de grado 3 pero luego de 24 horas no habia finalizado asi que decidimos descartar ese caso.

2.3.5 Naive Bayes (Okus)

Aplicamos los clasificadores BernoulliNB y GaussianNB.

2.3.6 Optimización de hiperparámetros

Una vez elegidos los conjuntos de hiperparametros donde nos interesaba estudiar cada uno de los algoritmos, aplicamos RandomizedSearchCV. En todos los casos, excepto en SVM, elegimos usar cross validation de 10 folds. En SVM consideramos solo 3 folds. Esta funcion analizaba distintas combinaciones de hiperparametros y nos devolvia la mejor combinacion en cada caso. La eleccion de la mejor combinacion se realizo comparando los valores roc_auc. Es decir,nos quedabamos con la combinacion de hiperparametros que nos daba un valor mayor de area bajo la Curva Roc.

- ROC AUC

- Cross validation (10 vs SVM 3)
- Usamos Randomized (vs Grid)
- PCA (JM)

3 Resultados y Discusión

Clasificador	precision	accuracy	f1_score	recall_score	roc_auc_score
Decision	0.956531	0.957444	0.957487	0.958444	0.957444
_____	_____	_____	_____	_____	_____
_____	_____	_____	_____	_____	_____
_____	_____	_____	_____	_____	_____
_____	_____	_____	_____	_____	_____
_____	_____	_____	_____	_____	_____

- TABLAS
- Cómo se comportaron con PCA
-

4 Conclusiones

5 Decision Trees

6 Los hiperparámetros que nos parecieron interesantes, provistos por la librería fueron.

** Criterio de partición ('criterion')**

Establece la función que mide la calidad de una partición de los nodos del árbol. Existen dos provistos por la librería, "Gini" y "Entropía". El último responde al Information Gain visto en la materia.

Profundidad máxima ('max_depth')

Permite hacer una poda sobre el árbol.

Splitter

Tiene dos opciones, best y random. Establece si se utiliza el mejor criterio de partición o usa un mecanismo aleatorio para buscarlo. Como ya dijimos antes, la calidad de la partición estará dada por el hiperparámetro 'criterion'

Subconjunto de valores de un atributo('max_features')

Establece el máximo de valores de un atributo que se tiene en cuenta. Visto más gráficamente, establece el ancho del árbol. Se puede establecer un número fijo, o un porcentaje, entre otras posibilidades.

Criterio de corte para un nodo('min_samples_split')

La mínima cantidad de muestras necesarias para que se justifique la partición en un nodo.

** La mejor combinación de hiperparámetros fue **

'min_samples_split' = 82

'splitter' = 'best'
'criterion' = 'entropy'
'max_depth'= 18
'max_features' = 0.90000000000000013
El valor fue 0.987803482259 con el scoring roc_auc.

7 VER QUE ONDA CON EL ROC AUC SCORE QUE VARIA CON EL ANTERIOR MOSTRADO

precision_score	accuracy_score	f1_score	recall_score	roc_auc_score
0.956531	0.957444	0.957487	0.958444	0.957444

8 Naive Bayes

Dada la naturaleza del clasificador, hay que tener también en cuenta la distribución de las muestras. En ese sentido, la librería provee tres clasificadores de Naive Bayes, usando una distribución Gaussiana, una de Bernoulli y una Multinomial.

Las primeras pruebas arrojaron resultados nulos con la multinomial y la de bernoulli. Cuando hablamos de resultados nulos nos referimos a que, con pruebas varias que hacíamos, la clasificación era completamente errada.

Dado que la Gaussiana no arrojaba ese tipo de resultado, decidimos trabajar con esa propuesta.

Ahora bien, cuando graficábamos la distribución de algunos atributos en los mails veíamos que se parecía más a una Exponencial que a una Gaussiana. Además, los gráficos perdían proporcionalidad y no se podían entender. Al intentar aplicarle logaritmo a los datos para darle otra escala vimos que tomaban una forma de campa de gauss.

Como además aplicar el logaritmo preserva las desigualdades, nos parecía razonable usar esta conversión previa para trabajar con el clasificador.

Este clasificador no tiene parámetros así que no tuvimos que hacer pruebas variadas en relación a modificación de hiperparámetros.

Los resultados obtenidos sin previa normalización usando logaritmo fueron:

| precision_score | accuracy_score | f1_score | recall_score | roc_auc_score | |———————-|———————|———————|———————|———————|
| 0.504766 |0.509222 | 0.665556 | 0.976667 | 0.509222 |

Con normalización fueron:

| precision_score | accuracy_score | f1_score | recall_score | roc_auc_score | |———————|———————-|———————-|———————|———————| | 1.0 | 0.500778 | 0.003106 | 0.001556 | 0.500778 |

9 AGREGAR LO DE BERNOULLI QUE APARECIÓ DESPUÉS

10 KNN

11 SVM

12 RANDOM FOREST