Aprendizaje automático

Grupo Sociofísica
Sebastián Pinto, Guillermo Pasqualetti, Gustavo Landfried

27 de septiembre de 2016

1. Introducción

El objetivo de este trabajo práctico es construir un clasificador automático de mensajes de correo electrónico en dos clases: "spam" y "ham". Para llevar a cabo esta tarea, fue necesario extraer atributos de los documentos. Con ellos exploramos diferentes modelos buscando mejorar la performance obtenidas en validación cruzada. A su vez, estudiamos algunas técnicas de reducción de dimensionalidad para acotar la cantidad de atributos elegidos inicialmente.

1.1. Metodología

Para la resolución de este trabajo utilizamos el lenguaje de progrmación *python*. Para la extracci ón de atributos 2 implementamos nuestros propios algoritmos y tomamos atributos de dos artículos científicos [1] y [6]. Los clasificadores (tree, naive bayes, k-nearest neighbors, random forest y support vector machine) y la validación cruzada que usamos para explorar modelos 3 fueron importados de la biblioteca *scikit-learn* [5]. Para seleccionar atributos 4 implementamos el ranking de ganancia de información de atributos y la búsqueda de optimos locales mediante hill climbing. La descomposición en valores singulares (SVD) fue importada de la librería *scipy.linalg* [3]. Fueron útiles los conceptos del libro "An Introduction to Statistical Learning" [4].

2. Extracción de atributos

Identificamos 127 atributos para la clasificación del mail, los cuales constituyeron dos grupos: características del texto y palabras involucradas.

Dentro de las características del texto elegimos: a) Largo del documento, b) Cantidad de espacios en blanco, c) Si el archivo contiene o no html, d) Si el mail es una respuesta, e) Cantidad de caracters no ASCII.

La hipótesis detrás de dichas elecciones es que un mail clasificado como *spam* es más probable que contenga un archivo *html* y una mayor cantidad de caracteres no ASCII, como sucede en el empleo de idiomas que no sea el inglés. Por otro lado, en un mail etiquetado como *ham* esperamos que tenga un largo menor que *spam*, que haya grandes diferencias en el empleo de uso de espacios en blanco, y si el mail es una respuesta, probablemente provenga de una conversación entre dos usuarios reales.

La elección de palabras involucradas se basó a su vez en dos criterios separados:

- La diferencia simétrica de los conjuntos de palabras más frecuentes en *spam* y *ham*
- Palabras reportadas en dos papers [1] y [6].

En primer lugar extrajimos las palabras más frecuentes en *spam* y *ham*. Para obtener variables representativas de cada clase nos quedamos únicamente con la diferencia simétrica. De ellas extrajimos los 25 términos más frecuentes del dataset tanto para *ham* como para *spam*. Para la

realización de esta tarea utilizamos la librería de python *nltk* ([2]). Los atributos elegido son la cantidad de veces que aparecen cada una de esas palabras en el texto. A su vez incluimos un listado de palabras propuestas por los trabajos [1] y [6] que fueron catalogadas como buenas discriminadoras entre mails *ham* y *spam*.

3. Modelos

En esta sección discutimos distintos modelos y evaluamos algunos de sus parámetros.

3.1. Decision Tree Classifier. DTC.

Con la finalidad de escoger el mejor árbol de desición exploramos el desempeño sobre los datos de entrenamiento en una grilla de híper-parámetros. De entre el total de híper-parametros del modelo decidimos profundizar sobre tres que nos parecieron especialmente relevantes:

- Profundidad máxima [Valor entero]: Máxima profundidad del árbol de desición.
- Criterio [Gini Impurity/Information Gain]: función que mide la cualidad del "split".
- Splitter [Best/Random]: estrategia usada para elegir el mejor "split" en cada nodo.

Los híper-parametros "Criterio" y "Splitter" no tuvieron un impacto significativo en la performance; para ellos escogimos los valores Gini Impurity y Best, respectivamente. El híper parametro "Profundidad máxima" tuvo, en cambio, un impacto importante sobre el desempeño. La figura siguiente muestra la relación funcional entre ambos.

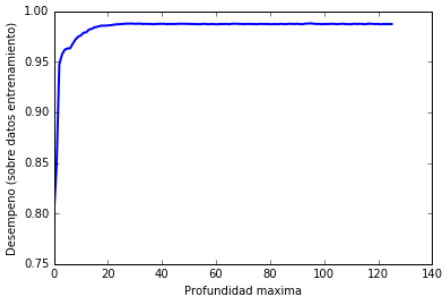

Figura 1

Vemos que el desempeño presenta una fuerte mejora a medida que se aumenta la profundidad máxima permitida del árbol hasta profundidades de 15 a 20 niveles. Luego el desempeño prácticamente no crece. Para evitarnos posibles problemas de 'overfitting', decidimos limitar en 15 la profundidad de los árboles usados con un desempeño correspondiente de 0.982.

3.2. Naive Bayes. NB.

El método de Naive Bayes no resultó adecuado para este set de datos. El desempeño sobre los datos de entrenamiento fue de tan solo $0{,}52 \pm 0{,}01$. La hipótesis sobre este mal desempeño es que los atributos elegidos no son lo suficientemente independientes como para estar dentro de la hipótesis del método. En la sección 4.3.1 realizamos un nuevo estudio sobre este método, buscando mejorar su eficacia.

3.3. k-nearest neighbors. KNN

Para determinar la cantidad de vecinos óptimo evaluamos el desempeño en cross validation con 10-folds con $k \in \{1, 10, 20, 40, 80, 160, 320, 640\}$ vecinos. Los resultados son los siguientes,

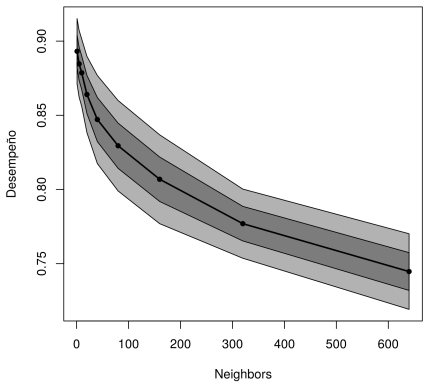

Figura 2

El desempeño máximo fue menor a $0,9$ y resultó ser bajo en comparación con otro métodos. Nos sorprendió particularmente que la cantidad de vecinos óptimo para este problema haya sido con un único vecino.

3.4. Random Forest. RF.

Para determinar la cantidad de árboles que genera cada random forest, probamos el rendimiento de la validación cruzada 10-fold con t árboles $t \in \{1, 5, 10, 20, 40, 80, 160, 320, 640\}$

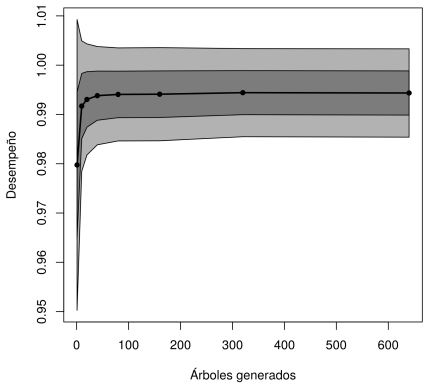

Figura 3

40 árboles parecen ser suficiente para optimizar el método random forest.

La cantidad máxima de atributos a que se utiliza en cada iteración la dejamos con su valor por defecto $a = \sqrt{(|A|)}$. En el libro "An Introduction to Statistical Learning" [4] se discute ese tema y se concluye que es una valor bueno en general.

3.5. Suport Vector Machine. SVM.

Decidimos realizar el estudio para SVM con un subconjunto del dataset original. Del tutorial de *scikit-learn* ([5]) observamos que la complejidad computacional es cuadrática en el tamaño de la muestra. Este método es muy lento para entrenar y validar con la totalidad de los datos. Por lo tanto utilizamos un dataset acotado a 10000 mails (50 % ham - 50 % spam) con el cual obtuvimos las eficacias de la figura 4 para distintos kernels y valores del parámetro C, que constituye un valor de tolerancia ante datos mal clasificados.

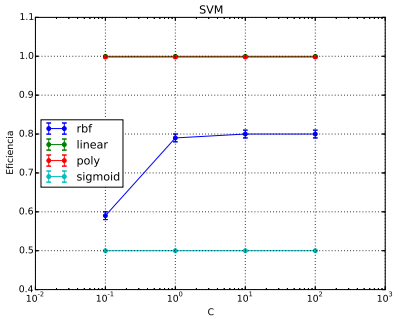

Figura 4: Eficacia de SVM para distintos kernels y valores de C, para un subconjunto de datos.

Los tiempos de ejecución para cada kernel, promediados para los valores de C empleados son los siguientes:

Kernel	Tiempo (s)
Linear	1850
Poly (degree = 3)	202
Rbf	192
Sigmoid	154

Cuadro 1: Tiempo de validación de SVM, para distintos kernels.

De la tabla 1 y la figura 4 concluímos que SVM para un los kernels *linear* y *poly* dan una muy buena eficacia, que el tiempo para *linear* es mucho mayor que para *poly*, pero que ambos casos, el tiempo de validación es claramente más alto que los otros métodos estudiados, aún al trabajar con un dataset acotado.

4. Reducción de dimensionalidad

En esta sección implementamos varios métodos de reducción de dimensionalidad. Optamos por la descomposición en valores singulares (svd) para seleccionar atributos.

4.1. Features Ranking

Evaluamos la ganancia de cada variable individualmente y obtuvimos el siguiente ranking.

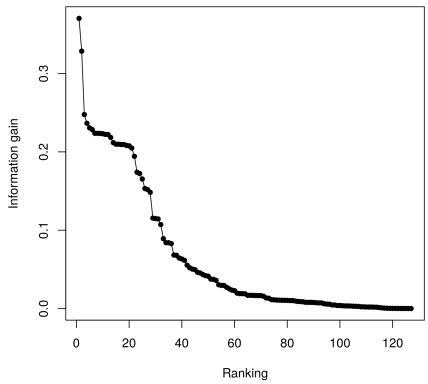

Figura 5

Este ranking no considera la dependecia entre variables. Dos variables correlacionadas pueden tener individualmente la misma ganancia de información, pero al ser tomadas juntas una puede resultar superflua. Para tener en cuenta la interección implementamos otros métodos.

4.2. Hill climbing

Implementamos completo un algoritmo de hill climibing. Este método de selección de variables busca maximizar la performance resultante de aplicar a un conjunto de datos, D, un modelo L, evualuado sobre un subconjunto de atributos S variable. La vecindad de un subconjunto S son todos los subconjuntos que se pueden contruir sacando de S un único elemento, o agregando a S un único elemento. Corrimos 50 procesos de hill climbing evaluando la perfomance en cada punto ejecutando un árbol de decisión con cross validation 5-fold. Encontramos 50 óptimos locales, sin repetición. Las performance de los óptimos locales estuvieron entre 0.88 y 0.96. Este proceso no fue suficiente para decidir por un único subconjunto de atributos para la selección de atributos.

La performance que se obtiene de la validación cruzada depende de la partición utilizada. Esa variabilidad puede hacer que ciertos subconjuntos S sean a veces consideramos ótpimos locales y otras veces no. Una forma de solucionar esto sería realizar validación cruzada *leave one out*, pero este método es muy costoso. Otra opción es fijar 5-folds iguales para todos las pruebas de validación cruzada y considerar únicamentre la performance obtenida sobrea esa partición particular.

Sin embargo, las diferencias entre los ótpimos locales obtenidos no fueron significativas para determinar con certeza un único subconjunto de variables. Para eso implementamos otro método.

4.3. Principal Component Analysis. PCA

De los atributos seleccionados, realizamos un reducción de la dimensionalidad mediante la técnica de PCA. Dada la matriz de *mails* x *atributos*, calculamos la matriz de covarianza de los atributos, y realizamos una descomposición en valores singulares (SVD) mediante la función *scipy.linalg.svd* [3]. La matriz de covarianza es una matriz simétrica, por lo tanto sus valores singulares coinciden con sus autovalores, que además son reales y no negativos. En la figura 6, observamos el valor de los mismos, ordenados de mayor a menor.

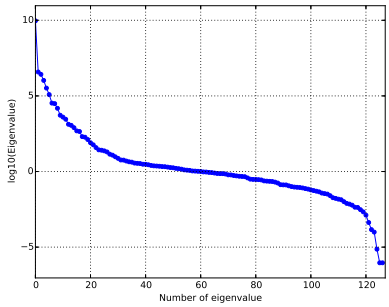

Figura 6: Valor de los autovalores surgidos de la descomposición SVD, ordenados de derecha a izquierda.

Los autovectores obtenidos de la factorización son una combinación lineal de los atributos elegidos originalmente. Dichos autovectores pueden ser tomados como nuevos atributos, los cuales, a partir de sus autovalores asociados, sabemos en cuáles hay una mayor variabilidad de los datos.

Para dar una interpretación a los nuevos atributos, observamos la representación de los autovectores en el espacio de atributos originales. Como criterio, estudiamos qué componentes tienen un valor absoluto mayor a 0,1 en el espacio de atributos originales. En la tabla 2 mostramos el resultado para las 5 direcciones principales. De la tabla vemos que las dos direcciones principales prácticamente coindicen con dos de los atributos elegidos originalmente, asociados al formato del mail en cuestión. Los autovectores siguientes se constituyen de un grupo de términos, de los cuales el 4º y 5º autovector muestran una correlación en la terminología (o procedencia) de los miembros del grupo más visible que en el 3º autovector.

Autovector	Componentes principales
1	Largo del documento.
2	Cantidad de espacios en blanco.
3	Términos: germ, hi, how, think, valuable, enron, republic, content-class, thread-index.
4	Términos: x-origin, x-filename, x-cc, binary
5	Términos: receive, email, upgrade, fast, spam

Cuadro 2: Principales componentes de los autovectores surgidos de PCA

4.3.1. Naive Bayes + PCA

El método de Naive Bayes no resultó satisfactorio para el dataset original, como reportamos en la sección 3.2. La principal hipótesis de tan mal desempeño, fue el hecho de que Naive Bayes suponga que los atributos son independientes entre sí. Por lo tanto, esperábamos que con los nuevos atributos obtenidos mediante la técnica PCA, el método funcione mejor, debido a que dichos atributos se construyen buscando ortogonalidad entre los mismos, resultando en atributos más descorrelacionados que los atributos originales entre sí. En la tabla 3 reportamos los valores de eficacia, donde prácticamente no se ven diferencias en tomar un subconjunto de atributos surgidos de PCA y el desempeño original. [1]

[1] Cuando estudiamos el caso de Naive-Bayes con el dataset original, encontramos grandes diferencias en los resultados obtenidos, según en qué ordenador corríamos el script. El problema estaba en una diferencia de versiones de la librería *scikit-learn*. Los resultados reportados se corresponden con la versión 0.17.1.

Cuadro 3

Cantidad de atributos	Eficacia	error
5	0.51	0.01
10	0.52	0.02
20	0.52	0.01
120	0.51	0.01
original	0.52	0.01

4.3.2. knn + PCA

Al evaluar el clasificador *knn* sobre las variable transformadas, la cantidad de vecinos óptimo fue nuevamente $k = 1$. En la siguiente figura se muestra el resultado con $k = 1$ y $k = 10$ al aumentando la cantidad de atributos que se utilizan para clasificar.

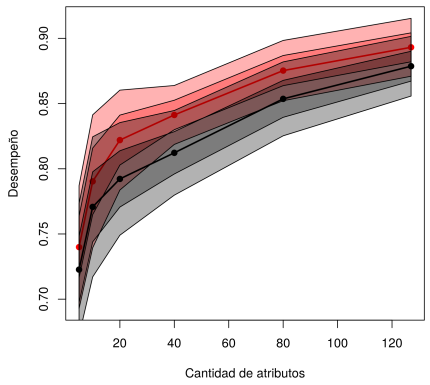

Figura 7: Rojo: $k = 1$. Negro: $k = 10$

Tampoco se observa una mejora con respecto a los atributos no transformados

4.3.3. árboles + PCA

Fijamos los parámetros a partir del análisis ya realizado, Máxima profundidad: 15 Criterio: Gini Splitter: Best

La figura siguiente **??** muestra el desempeño en función del número de componentes.

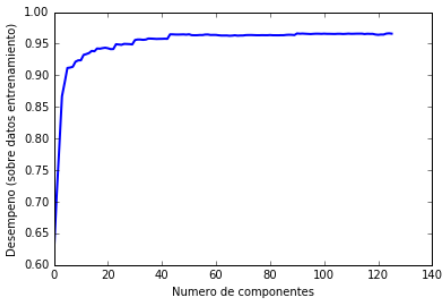

Figura 8

Como esperabamos, el desempeño crece abruptamente al comienzo para luego hacerlo en forma

moderada hasta la componente 43 cuando se alcanza prácticamente el valor máximo. Parece entonces ser esa una cantidad adecuada de componentes a considerar.

4.3.4. random forest + PCA

Probamos random forest con 40 árboles con v variables en el nuevo sistema de coordenadas, $v \in \{5, 10, 20, 40, 127\}$

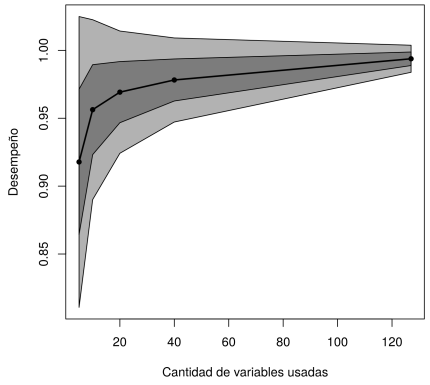

Figura 9

No se observa mejora con respecto a los atributos no transformados.

4.3.5. SVM + PCA

Estudiamos el caso de SVM con kernel polinomial y variables reducidas, sin embargo el tiempo de ejecución se incrementó significativamente respecto al caso de tomar todas los atributos originales, por lo que descartamos que sea una mejora para este modelo.

5. Resultados y conclusiones

Luego de analizar diferentes modelos, concluimos que los métodos que mostraron una mejor performance para el conjunto de atributos son SVM y Random Forest, teniendo el último un tiempo de validación y entrenamiento considerablemente menor que el primero.

La generación de nuevos atributos mediante la reducción de dimensionalidad con técnicas tales como PCA, no supuso una mejora significante en el balance entre tiempo de ejecución y eficacia de los modelos.

Por otro lado, vale destacar que los resultados reportados corresponden de realizar un promedio de los rendimientos obtenidos al hacer *cross-fold validation*, por lo que somos concientes que podemos estar realizando un *over-fitting* sobre los datos analizados, y la perfomance de los modelos ante una base de datos nueva puede diferir de la predicha en este trabajo.

6. Discusión

Algunos hechos que fueron surgiendo a medida que se desarrollaba el trabajo práctico, que al principio nos parecían eventos aislados, pensamos que pueden ser considerados en realidad como un indicio de un problema originado en la extracción de atributos. Los hechos aparentemente aislados serían

- La baja performance de *naive bayes*.
- El cantidad de vecinos óptimo para determinar la clase mediante *KNN* igual a 1.
- La variedad de óptimos locales al realizar *hill climbing*.

Hoy suponemos que la baja performance de naive bayes se debe a una fuerte dependencia entre los atributos. A su vez pensamos que la cantidad de vecinos óptimo para knn sea un único vecino podría estar indicando que los datos no están bien separados en el espacio, es decir, que los atributos no distinguen con suficentemente claridad una clase de otra. Finalmente sospechamos que la variedad de óptimos locales se debe a la alta correlación entre los datos. Algunas cosas que deberíamos implementar para mejorar para la extracción de atributos

- Sacar el encabezado del mail para la selección de palabras.
- Sacar el html para la selección de palabras.
- Estandarizar los string de tipo archivo adjunto (más de tres aaa) como una única variable

Referencias

[1] Günal, S., Ergin, S., Gülmezoğlu, M.B., Gerek, Ö.N.: On feature extraction for spam e-mail detection. In: International Workshop on Multimedia Content Representation, Classification and Security. pp. 635–642. Springer (2006)

[2] http://www.nltk.org:

[3] http://www.scipy.org/:

[4] James, G., Witten, D., Hastie, T., Tibshirani, R.: An Introduction to Statistical Learning: With Applications in R. Springer Publishing Company, Incorporated (2014)

[5] http://www.scikit learn.org:

[6] Vaughan-Nichols, S.J.: Saving private e-mail. IEEE Spectrum 40(8), 40–44 (2003)