



**DEPARTAMENTO
DE COMPUTACION**

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Número 1

28 de septiembre de 2016

Aprendizaje Automático

Integrante	LU	Correo electrónico
Bordón, Pablo	794/07	bordonpablo@gmail.com
Gasco, Emilio	171/12	gascoe@gmail.com
Gatti, Mathias	477/14	mathigatti@gmail.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

0. Extracción de atributos	3
1. Modelos	5
1.1. Árboles de decisión	5
1.2. Random Forest	6
1.3. K Vecinos Mas Cercanos	7
1.4. Naive Bayes	8
1.5. Support Vector Machine	8
2. Reducción de dimensionalidad	10
2.1. Descripción	10
2.2. Resultados	10
3. Resultados en Testing Set	11
4. Discusión	12

0. Extracción de atributos

1. A continuación enumeramos los atributos que escogimos como potencialmente útiles y que luego implementamos para extraer automáticamente de los mails, estos fueron ideados por nosotros excepto por el atributo *spell_error_count* el esta inspirado en el artículo *Personalized Spam Filtering with Natural Language Attributes*¹, en este paper se mencionan algunas buenas ideas mas las cuales no llegamos a implementar como contar la cantidad de palabras poco comunes o la cantidad de verbos en el texto.

- Los correos de spam suelen ser enviados a un único destinatario, para capturar esta característica se extraen 3 atributos del encabezado del correo: *recipient_count*, *has_cc* y *has_bcc* para extraer cantidad de destinatarios del correo, si hay destinatarios en copia y si hay destinatarios en copia oculta respectivamente.
- *headers_count* cantidad de encabezados.
- *mailer* Software utilizado para envío de correo.
- *has_body* Nos dice si el correo tiene cuerpo o si solo consta de encabezados
- *content_type* Tipo del contenido del cuerpo de correo. Por ejemplo: text/plain, text/html, multipart/related, multipart/alternative, etc.
- *content_transfer_encoding* la codificación utilizada para la transferencia del correo
- *is_multipart* Nos dice si el cuerpo consta de varias partes
- *subject_length* Largo del título del correo.
- *raw_mail_len* Largo del cuerpo del mensaje.
- *raw_body_count_spaces* Cantidad de espacios en cuerpo de correo.
- *has_dollar* Nos dice si aparece el símbolo \$ en el cuerpo del correo.
- *has_link* Indica presencia de link http dentro del cuerpo del correo.
- *has_html* Indica presencia de html dentro del cuerpo del correo.
- *has_attachment* Indica la presencia de archivos adjuntos analizando content-type de las partes de correos con múltiples partes. Se consideran archivos adjuntos a las partes que no sean del tipo text/*.
- *uppercase_count* Frecuencia de caracteres de letras mayúsculas en cuerpo de correo.
- *has_non_english_chars* Indica presencia de caracteres de idiomas diferentes al inglés dentro del cuerpo.
- *spaces_over_len* frecuencia de espacio en cuerpo de correo
- En correos de tipo ham se puede observar alta frecuencia de conjunciones y artículos. Por lo que tenemos atributos para calcular la frecuencia de los mismos. Por ejemplo: a, and, for, of, to, in, the . La frecuencia se mide por separado a cada uno de los listados. La lista surgió de analizar palabras mas frecuentes en correos de ham en comparación con correos de spam.

¹<https://pdfs.semanticscholar.org/8c64/21ad08277291813690c41c14a55916e46034.pdf>

- *parts_count* Cantidad de partes en correo de múltiples partes.
- *spell_error_count* Cantidad de errores ortográficos en cuerpo de correo.
- Por último analizamos el set de entrenamiento de correos de spam y ham en busca de palabras de interes, principalmente las que aparecieran mucho en un tipo de mail pero no en otro. Por cada palabra normalizabamos su cantidad de apariciones por sobre el total de palabras y calculamos la diferencia entre el resultado obtenido en ham y spam. A partir de estos valores nos quedamos con las 2000 palabras con mayor score.

El atributo *content_type* es el atributo con mas ganancia de información, posicionando en la raíz de los clasificadores de arboles cuando no se limitaba la selección de atributos a un subconjunto aleatorio de atributos. El atributo *headers_count* no resulto ser muy efectivo, la cantidad de encabezados suele ser uniforme entre correos spam y ham, variando por la inclusión de encabezados cc y bcc que ya son capturados por otros atributos.

2. El conjunto original de mails fue dividido para tener por un lado un set de entrenamiento con el cual trabajar y un set de testing para probar al final si realmente nuestros clasificadores generalizaban bien y podían clasificar correctamente instancias nuevas. Al entrenar los árboles de decisión surgió la necesidad de medir su performance de alguna manera, para el dominio del problema en particular no pareció valido utilizar F0.5 como unidad de medida, ya que la precisión es lo que tiene mayor peso en un filtro de spam debido a que se busca evitar que el clasificador catalogue como spam un mail importante.
3. Experimentamos con distintos hiper-parámetros utilizando al técnica de grid search. Por limitaciones de computo no pudimos hacer una búsqueda demasiado exhaustiva, de todas maneras logramos obtener resultados considerablemente mejores que los obtenidos al probar con los valores seteados por defecto en los clasificadores. Se discutirá esto en mayor profundidad en la sección de resultados.

1. Modelos

1.1. Árboles de decisión

Este modelo tiene la ventaja de ser muy rápido pero presenta el problema de que si se deja crecer mucho el árbol el clasificador puede sobre ajustar los datos de entrenamiento. Las primeras pruebas estuvieron orientadas a estimar un rango de alturas en donde el clasificador tenga buena eficacia sin sobre ajustar. Para lograrlo calculamos la métrica $f0.5$ de árboles de alturas de 1 a 30 para datos de entrenamiento y para cross validation con $K=10$. En la figura 1 puede ver un gráfico de los resultados.

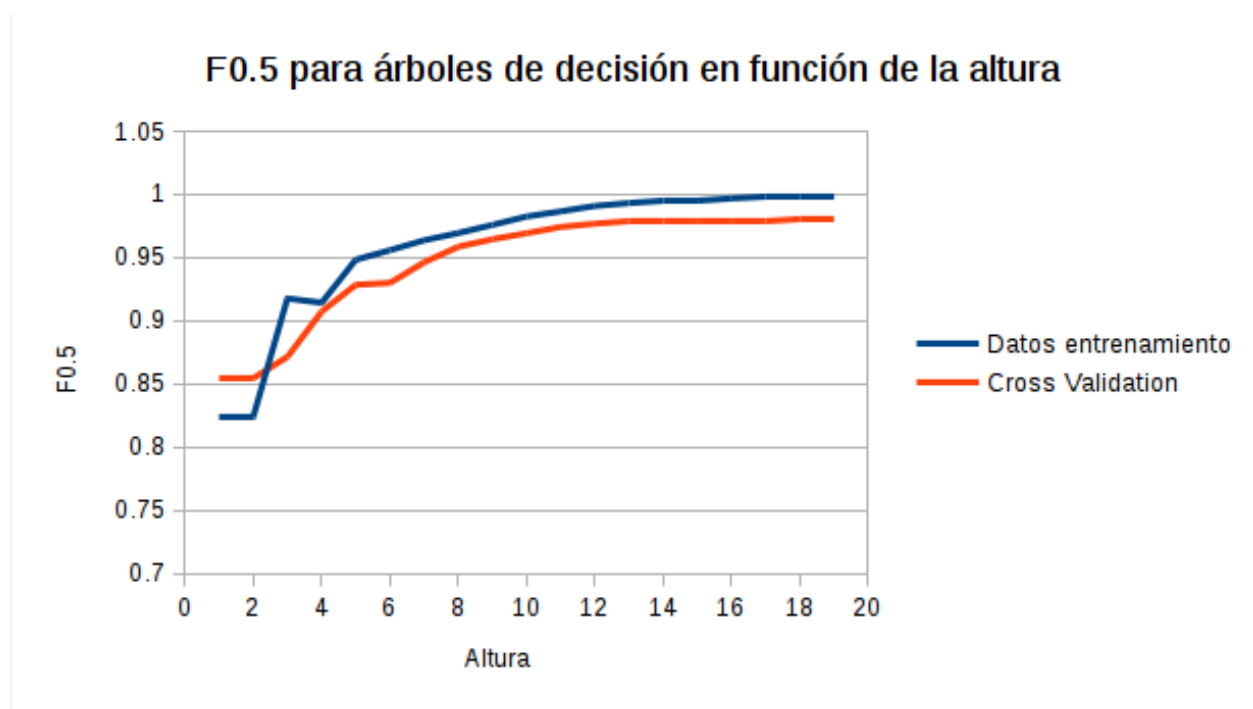


Figura 1: $f0.5$ en función de la altura del árbol

El resultado de cross validation es un estimador de la eficacia que vamos a obtener clasificando correos nuevos y a pesar de que por lo general no aproxime correctamente la eficacia nos permite predecir que estamos haciendo sobre ajuste de datos de entrenamiento cuando al aumentar la altura baja o se mantiene la eficacia. Como se puede ver en la figura 1 a partir de árboles de altura 9 el aumento en la eficacia al variar la altura empieza a bajar hasta que se estanca llegando a árboles de altura 15. Luego para este rango de alturas ejecutamos un grid search a fin de obtener los mejores hiper-parámetros para el clasificador.

1. *criterio de selección:* Se probaron criterios, gini y ganancia de información. No se observaron grandes diferencias en la eficacia del clasificador al variar el criterio. El mejor resultado se obtuvo con ganancia de información.
2. *Cantidad máxima de atributos considerados:* Este hiper-parámetro permite limitar la cantidad de atributos considerados al hacer un split. Tiende a favorecer al aparición

de arboles que no sean dominados por los atributos que provean mayor ganancia de información, dado que es posible que varios atributos más débiles trabajando en conjunto logren mejor eficacia. En las pruebas realizadas se obtuvieron mejores resultados sin limitar la cantidad de atributos

3. *Cantidad mínima de elementos en división:* Limita la división de nodos internos a lo que superen en cantidad de elementos la cantidad especificada. Los mejores resultados se obtuvieron con el valor mínimo de 2. Al aumentar el valor y al mismo tiempo limitar la cantidad de atributos limitados se puede observar una caída de eficacia en cross validation.
4. *Cantidad mínima de elementos en hojas:* Las hojas del árbol no pueden tener menos de la cantidad especificada. Los mejores resultados se obtuvieron con el valor mínimo de 2. Al igual que con el hiper-parámetro anterior, aumentar el valor y al mismo tiempo limitar la cantidad de atributos limitados se puede observar una caída de eficacia en cross validation.

1.2. Random Forest

La idea de random forest es utilizar múltiples árboles de decisión para mejorar la eficacia del clasificador, reduciendo la alta varianza que se observa en la utilización de un único árbol. Esto es posible gracias al bajo costos que lleva entrenar y consultar a cada árbol. La reducción en la varianza suele ser mas significativa cuando el bosque se arma con árboles que utilicen diferentes atributos. Para lograr esto los atributos utilizados en las divisiones de los nodo al construir el árbol se limitan a un subconjunto aleatorio de los atributos, reduciendo la probabilidad que lo atributos con mayor ganancia de información dominen todos los arboles. Utilizando los mismos parámetros que para el clasificador de un único árbol volvimos a hacer pruebas variando la cantidad de árboles del bosque y el tamaño del subconjunto de atributos a considerar. En la figura ?? se muestran los resultados para bosques de diferentes tamaños considerando n , \sqrt{n} y $\log(n)$ atributos, donde n es la cantidad total de atributos. Los bosques en los cuales se consideran n atributos muestran mejores resultados, pero cuando consideran \sqrt{n} atributos, la varianza es inferior por lo que esperamos que la eficacia con datos nuevos sea más parecida a lo que observamos con cross validation sobre datos de entrenamiento.

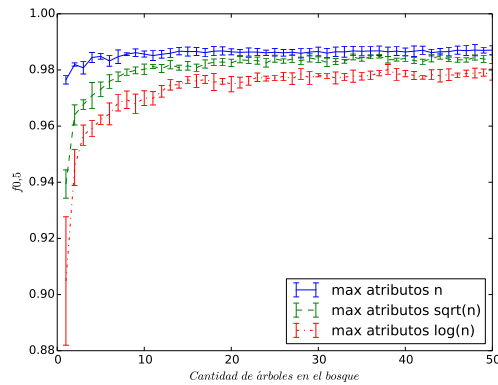


Figura 2: $f_{0.5}$ en función de la cantidad de árboles

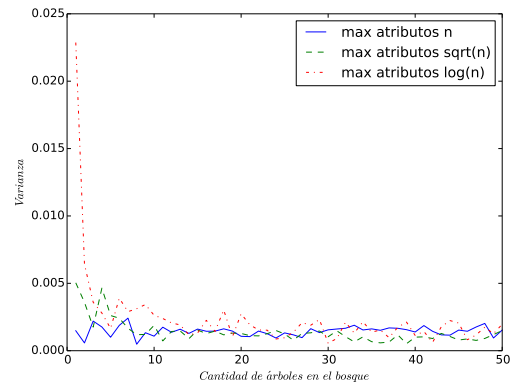


Figura 3: Varianza en función de la cantidad de árboles

1.3. K Vecinos Mas Cercanos

Para este clasificador consideramos un enfoque parecido al de árbol de decisión, a fin de limitar las pruebas necesarias para conseguir los mejores hiperparámetros, primero realizamos tests variando la cantidad de vecinos, k , considerados y calculando $f_{0.5}$ para cross validation de 10 folds. Obteniendo los resultados de la figura 4.

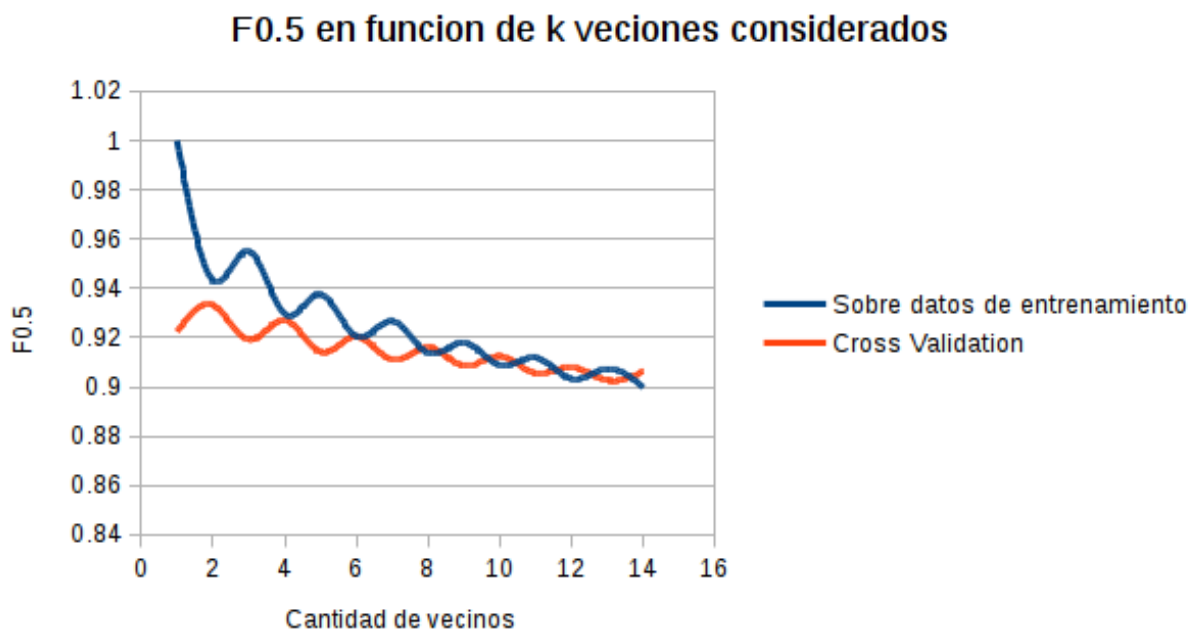


Figura 4: $f_{0.5}$ en función de k vecinos considerados

A demás de la cantidad de vecinos este clasificador cuenta con los siguientes atributos:

- a) *Peso*: Este parámetro ponderar el peso de cada uno de los vecinos según su distancia o puede tratarlos a todos uniformemente ignorando la distancia. En nuestro caso se obtuvieron mejores resultados considerando los vecinos uniformemente.
- b) *Métrica de distancia*: Se probaron diferentes métricas de distancias, obteniendo los mejores resultados con la distancia manhattan en la cual la distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas.

En el gráfico 4 se observan los resultados del clasificador corriendo con los mejores atributos de *Peso* y *Métrica de Distancia* que encontramos para este problema.

1.4. Naive Bayes

Para este clasificador tuvimos que elegir primero la distribución que asumiríamos que iba a tener $P(x_i|y)$, las opciones que consideramos fueron una distribución Gaussiana, Binomial y Multinomial. La distribución que mejores resultados obtuvo fue la Binomial lo cual tiene sentido debido a que modela bien la gran cantidad de atributos binarios, quizás si estos tuvieran mas de dos resultados posibles la distribución Multinomial o la Gaussiana habrían sido mejores opciones. Los hiperparametros que consideramos fueron *alpha* y *fit_prior*, el primero suaviza la función de distribución de cada atributo y el segundo intenta ir mejorando la estimación de los atributos y las clases a priori, a medida que va analizando nuevas instancias.

	Gaussiana	Multinomial				Bernoulli			
Fit Prior		True		False		True		False	
Alpha		0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0
Desvío		0.00092	0.00092	0.00092	0.00092	0.0056	0.0057	0.0056	0.0057
Media	0.612	0.54	0.54	0.54	0.54	0.91	0.91	0.91	0.91

Figura 5: Media y Desvío Estandar de Clasificadores Naive Bayes luego de aplicar Cross Validation

1.5. Support Vector Machine

Debido a limitaciones de tiempo y computo solo pudimos probar SVM con los parametros dados por defecto en Sklearn ², obtuvimos una performance del 83.5 % al realizar cross validation de 10 folds con el set de entrenamiento utilizando F0.5 como medida de score.

El clasificador con los parametros por defecto contiene como características principales un Kernel de Base Radial ³ con parametro gamma = 1/numero_de_features y la

²<http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC>

³https://en.wikipedia.org/wiki/Radial_basis_function_kernel

heurística llamada *Shrinking* la cual permite realizar los computos del Kernel de forma mas rápida.

2. Reducción de dimensionalidad

2.1. Descripción

Utilizamos distintas técnicas de reducción de la dimensionalidad, principalmente porque teníamos una gran cantidad de features los cuales teníamos dudas de que fueran realmente útiles. Las principales dudas estaban en las 1000 palabras que habíamos escogido a partir del score que definimos previamente. Para descartar atributos utilizamos tres selectores y luego calculamos la unión entre esos tres conjuntos resultando en 300 atributos al final en lugar de los 1490 con los que empezamos.

- *Support Vector Classifier* con penalidad norma 1.
- A partir de un árbol de decisión, entrenamos a este con los valores de entrenamiento y vemos los atributos que fueron utilizados primero por este.
- A partir de seleccionar los 100 valores con puntaje mas alto, utilizando para puntuar a la función χ^2 la cual mide la independencia entre un atributo y una clase. Mientras mas dependa el valor de la clase del atributo (Algo deseable) mayor será el puntaje.

2.2. Resultados

Durante esta etapa descartamos algunos atributos que a priori no parecían ser tan malos, por ejemplo la longitud de palabra promedio, contar espacios o la cantidad de palabras. También fue interesante ver como palabras que quizás uno relacionaría fuertemente con el spam como "cheaper" fueron descartadas y en su lugar tuvieron mejores resultados algunos caracteres especiales como &, demostrando quizás las limitaciones que tenemos para reconocer atributos importantes y la utilidad herramientas que realicen estos cálculos por nosotros.

Las técnicas utilizadas fueron basadas principalmente en los recursos encontrados en http://scikit-learn.org/stable/modules/feature_selection.html

3. Resultados en Testing Set

Una vez seleccionados los mejores hiper-parámetros para cada modelo se procedió a probar los mismo clasificando los datos de testing separados al comienzo del trabajo. Los resultados se muestran en la siguiente tabla.

Modelo	Hiper-parámetros	f0.5
Decision Tree	Criterio de selección=entropy, altura máxima=14	0.68
Random Forest	Criterio de selección=entropy, altura máxima=14,cantidad máxima de atributos=n, cantidad de árboles=35	0.73
Random Forest	Criterio de selección=entropy, altura máxima=14,cantidad máxima de atributos=sqrt, cantidad de árboles=35	0.93
Gaussian Naive Bayes	N/A	0.61
Multinomial Naive Bayes	fit_prior=True, alpha=1.0	0.54
Bernoulli Naive Bayes	fit_prior=True, alpha=1.0	0.91
K Nearest Neighbors	Vecinos considerados = 2, Métrica distancia = Manhattan	0.61

4. Discusión

Al finalizar el trabajo terminamos optando por Random Trees Classifier como nuestro algoritmo predilecto, este tuvo la mejor performance, se desempeñó bien utilizando los atributos que consideramos y superó en gran medida a los demás clasificadores. La excepción está quizás con Bernoulli Naive Bayes, este método también obtuvo resultados muy buenos, al final terminamos descartándolo porque su performance era levemente inferior pero quedaron en el tintero varias ideas que quedarán como trabajo a futuro, consideramos crear un clasificador que corra ambos métodos (Random Trees y Naive Bayes) y decida como un árbitro a cuál hacerle caso o que conteste con cierto nivel de seguridad según si ambos responden igual o no. Otro tema a explotar podría ser el del preprocesamiento, la falencia del mismo nos trajo problemas al buscar palabras clave en los textos ya que el código html generaba ruido en los resultados, de todas maneras pudimos superar dichos problemas parcialmente y conseguimos un corpus de palabras que creemos se desempeña de forma bastante exitosa.