# Aprendizaje Automático
# Trabajo Práctico 1
# Detección de Spam

16 de diciembre de 2016


| Integrante | LU | Correo electrónico | Carrera |
|---|---|---|---|
| Martín Baigorria | 575/14 | martinbaigorria@gmail.com | Computación (Licenciatura) |
| Damián Furman | 936/11 | damian.a.furman@gmail.com | Computación (Licenciatura) |
| Germán Abrevaya | - | germanabrevaya@gmail.com | Física (Doctorado) |


**Reservado para la cátedra**

| Instancia | Docente | Nota |
|---|---|---|
| Primera entrega | | |
| Segunda entrega | | |

# Índice

# 1. Introducción

El objetivo de este trabajo practico es experimentar con diferentes técnicas de aprendizaje automático para entrenar un clasificador de mensajes de correo electrónico en dos clases, 'spam' y 'ham'. Para ello, se utilizaran diferentes modelos y métodos para buscar los diferentes atributos que utilizaremos. Se experimentaran con diferentes técnicas de reducción de dimensionalidad. Además, se evaluaran los diferentes hiperparametros de cada modelo para intentar buscar una mejor performance de los algoritmos de aprendizaje. Dejamos el 10 % de los datos como testing set, utilizando el resto para experimentar con los diferentes modelos. Al elegir los hiperparametros sobre el training set, caemos en la posibilidad de estar haciendo overfitting. Siguiendo el Principio de Occam, a performance similar, optaremos siempre por el modelo mas simple.

# 2. Extracción de atributos

## 2.1. Bag of words

En una primera etapa, antes de intentar seleccionar atributos de forma manual, la idea fue analizar la frecuencia de cada palabra en cada mail, ignorando el orden de las palabras. Notar que los mails están en formato MIME, por lo que no se podía simplemente hacer un análisis de frecuencia sobre el MIME en plaintext, si no que había que utilizar un parser para luego poder analizar las diferentes partes de cada correo. Además, muchos de los correos no estaban en UTF-8, lo que dificultaba el análisis. Por esa razón decidimos ignorarlos para el análisis de frecuencia.

Dado que el alto tiempo de computo para calcular la frecuencia de cada palabra en cada mail, decidimos simplemente tomar una variable binaria por mail que indica si la misma esta o no en el mismo. La idea fue calcular por clase cuantas veces aparecía cada palabra en el cuerpo de un correo, para luego ordenar todas las palabras por la diferencia absoluta en frecuencia relativa y de esa forma seleccionar los mejores atributos. Seguramente existe un orden mejor, pero lo que buscábamos era encontrar las palabras que mas distinguían entre las dos clases. Una idea que teníamos a priori además es que todas las conjunciones iban a aparecer con la misma frecuencia en ambas clases, por lo que al calcular su respectivo score se compensarían y no aparecerían en nuestros resultados finales. Además ignoramos todos los números, dado que creemos que no es un buen predictor.

En una primera etapa, para decidir que atributos se iban a tener en cuenta tomamos todas las palabras que mostraban una diferencia de frecuencia relativa entre clases de mails mayor o igual a 500. De esta forma terminamos preseleccionando 1.019 atributos. Entre ellos se encuentran:

['subject', 'please', 'original message', 'thanks', 'will', 'have', 'this', 'know', 'that', 'for', 'let', 'enron', 'would', 'any', 'attached', 'questions', 'call', 'are', 'here', 'with', 'gas', 'date', 'corp', 'has', 'monday', 'file', 'friday', 'and', 'houston', 'october', 'wednesday', 'not', 'energy', 'you', 'meeting', 'tuesday', 'need', 'thursday', 'group', 'california', 'power', 'but', 'following', 'there', 'november', 'should', 'final', 'was', 'more', 'they', 'schedule', 'review', 'think', 'john', 'deal', 'ect', 'start', 'scheduling', 'portland', 'what', 'money', 'been', 'contract', 'fax', 'professional', 'may', 'mark', 'hou', 'last', 'work', 'contact', 'iso', 'issues', 'schedules', 'log', 'vince', 'viagra', 'still', 'however', 'hourahead', 'can', 'solicitation', 'back', 'between', 'comments', 'mailto', 'also', 'discuss', 'thank', 'software', 'january', 'messages', 'save', 'were', 'mike', 'your', 'received', 'site', 'had', 'changes', 'advertisement', 'size', 'could', 'smith', 'prices', 'txt', 'westdesk', 'parsing', 'morning', 'louise', 'david', 'sure', 'well', 'going', 'click', 'working', 'visit', 'font', 'only', 'jeff', 'trading', 'stop', 'next', 'like', 'plan', 'did', 'tomorrow', 'now', 'north', 'regarding', 'product', 'best', 'agreement', 'online', 'forward', 'conference', 'texas', 'fyi', 'prescription', 'products', 'point', 'family', 'process', 'these', 'team', 'world', 'adobe', 'results', 'kaminski', 'help', 'cialis', 'offers', 'note', 'market', 'management', 'many', 'employees', 'kevin', 'michael', 'scheduled', 'his', 'capacity', 'august', 'which', 'daily', 'bill', 'photoshop', 'end', 'data', 'request', 'dollars', 'probably', 'chris', 'james', 'below', 'two', 'act', 'him', 'microsoft', 'during', 'pipeline', 'future', 'how', 'ena', 'done', 'copies', 'prohibited', 'requested', 'delete', 'services', 'put', 'another', 'either', 'robert', 'macromedia', 'into', 'desk', 'able', 'weight', 'global', 'flash', 'related', 'sender', 'asked', 'jim', 'yourself', 'hope', 'markets', 'each', 'quality', 'distribution', 'kitchen', 'email', 'update', 'might', 'good', 'discussed', 'stocks', 'phone', 'our', 'set', 'differ', 'security', 'anything', 'offer', 'attachments', 'talk', 'wish', 'afternoon', 'privileged', 'yet', 'whether', 'create', 'look', 'low', 'easy', 'available', 'advice', 'uncertainties', 'summary', 'already', 'confidential', 'xls', 'meaning', '100 %', 'shipping', 'report', 'joe', 'copy', 'until', 'richard', 'cause', 'premiere', 'provide', 'west', 'risk']

Por un lado unificamos las palabras 'original' y 'message', que en general muestran mensajes donde se hizo 'reply' o 'reply all'. Por otro lado, la palabra 'enron' aparecía bastante, lo que nos hizo pensar que el dataset que teníamos era el de Enron. Nos cuestionamos un poco que tan representativo es este dataset, dado que corresponde en general a mails corporativos. Notar que este mismo procedimiento se puede hacer para el título de los correos.

### 2.2. Selección manual

Luego de seleccionar los primeros atributos utilizando nuestra variacion de Bag of Words, decidimos elegir algunos otros atributos de forma manual. Estos fueron:

- has_closing_tags
- has_links
- cant_capital
- capital_in_a_row
- is_multipart

Con estos atributos buscamos captar distintas propiedades frecuentes en los mails de spam: la presencia de links hacia páginas web de contenido malicioso o de spam, el formato HTML que otorga formato a cierta publicidad, la presencia de muchas palabras en mayúscula o títulos o subtítulos con varias letras con mayúsculas seguidas.

A su vez, a las palabras determinadas por el Bag of Words le agregamos varias palabras que consideramos de posible aparición frecuente en mails de spam, a saber: offer, sex, viagra, Nigeria, discount entre varias otras.

## 3. Model Selection

Luego de definir nuestros atributos, probamos distintos modelos con el objetivo de evaluar su performance. Decidimos utilizar para nuestra comparación Naive Bayes, Decision Tree Classifier, Random Forest Classifier, Support Vector Machine y K Nearest Neighbours y AdaBoost.

Antes de entrenar los modelos, se separó el 10 % de los datos totales para ser empleados como test set, y el resto como training set. Para seleccionar un modelo final, la idea es probar todos los modelos bajo diferentes configuraciones haciendo cross validation sobre el training set, y luego seguir trabajando con el que nos de los mejores resultados. El test set lo guardamos para cuando finalmente tengamos seleccionado un modelo, podamos tener una idea de que tan bien generaliza bajo la elección de hiperparametros que hagamos.

### 3.1. Support Vector Machines

SVM resultó poco practico debido al descomunal tiempo de computo necesario para entrenar el modelo. Quitándole 90 atributos calculados mediante Bag of Words el tiempo de cómputo para entrenar al modelo (sin cross-validation) fue de 2:17 horas. El test mean accuracy resultó en este caso de 0.7933, la precision fue de 0.8829 y el recall, 0.6739. SVM utiliza quadratic programming lo cual conlleva un gran tiempo de cálculo cuando se trabaja con muchas dimensiones. No hay un criterio marcado de localidad para buscar hiperplanos separadores. Para sacarle más provecho a este método haría falta un kernel especifico.

## 3.2. Naive Bayes

A continuación se pueden ver los resultados de diferentes configuraciones de Naive Bayes con probabilidades gausianas. Dado el supuesto de independencia de los features, decidimos ir agregando atributos de a poco para observar como se comportaban las diferentes métricas de performance.

| | **200 Atributos** | **500 Atributos** | **Todos** |
|---|---|---|---|
| Train mean accuracy | 0.6798 | 0.6851 | 0.6493 |
| False positives | 2593 | 2680 | 3060 |
| False negatives | 288 | 154 | 96 |
| True positives | 4220 | 4354 | 4412 |
| True negatives | 1899 | 1812 | 1432 |
| Precision | 0.6194 | 0.6189 | 0.5904 |
| Recall | 0.9361 | 0.9658 | 0.9787 |

Cuadro 1: Resultados de Naive Bayes al ir agregando atributos al modelo. Los resultados son calculados tomando el promedio de cada métrica al hacer 10 fold cross validation.

Lo primero que nos llamo la atención fue el bajo tiempo de computo (menor a 5 minutos) en comparación con SVM. Otro factor interesante que observamos es que agregar atributos no necesariamente es mejor. La ganancia en accuracy en pasar de 200 a 500 atributos no parece significativa, y de hecho baja cuando agregamos todos los atributos que teníamos disponibles. En este caso parece que los mejores resultados los logramos con 500 atributos. En caso de que los otros modelos no logren una diferencia significativa en performance, quizás puede ser interesante seguir probando con Naive Bayes y técnicas de feature selection.

## 3.3. Random Forests

A continuación experimentamos con la técnica de Random Forests, variando el hiperparametro que define la cantidad de arboles utilizados en en el ensemble y utilizando todos los atributos disponibles.

| | **30 árboles** | **50 árboles** | **100 árboles** | **150 árboles** | **200 árboles** |
|---|---|---|---|---|---|
| Train mean accuracy | 0.9795 | 0.9786 | 0.9812 | 0.9801 | 0.9808 |
| False positives | 82 | 94 | 83 | 93 | 88 |
| False negatives | 102 | 98 | 96 | 86 | 84 |
| True positives | 4406 | 4410 | 4422 | 4422 | 4424 |
| True negatives | 4410 | 4398 | 4409 | 4399 | 4404 |
| Precision | 0.9817 | 0.9791 | 0.9810 | 0.9796 | 0.9813 |
| Recall | 0.9773 | 0.9767 | 0.9780 | 0.9802 | 0.9804 |
| Time taken | 15.84s | 23.69s | 46.27s | 66.67s | 89.44s |

Cuadro 2: Resultados de Random Forest cambiando la cantidad de arboles. Los resultados son calculados tomando el promedio de cada métrica al hacer 10 fold cross validation.

En general el tiempo de computo fue menor a 2 minutos, hasta en el caso en el que utilizamos 200 arboles. Parece que el tiempo es lineal en la cantidad de arboles. En cuanto a performance, Random Forests con 100 arboles tiene la mejor accuracy y precision, por lo que hasta el momento es el modelo que utilizaremos.

Se puede observar que aumentar la cantidad de árboles en general tiende a mejorar la *accuracy* aunque su variación es relativamente pequeña (alrededor del 0,2 %). Sin embargo, a partir de los 100 arboles la *accuracy* comienza a disminuir, aunque también con una tasa de decrecimiento baja. En general, a partir de los 100 arboles los resultados tampoco parecen cambiar mucho, por lo que optar por el modelo mas simple parece la mejor opción.

### 3.4. kNN:

Para K Nearest Neigbours consideramos intentar mejorar los resultados a partir de variar los k vecinos considerados. Mientras que en una primera instancia utilizamos un k = 5, testeamos luego los resultados de aplicar un k = 10 y k = 20.

| | **K = 5** | **K = 10** | **K = 20** |
|---|---|---|---|
| Train mean accuracy | 0.8244 | 0.8141 | 0.7994 |
| False positives | 821 | 708 | 792 |
| False negatives | 759 | 965 | 1013 |
| True positives | 3749 | 3543 | 3495 |
| True negatives | 3671 | 3784 | 3700 |
| Precision | 0.8203 | 0.8127 | 0.8152 |
| Recall | 0.8316 | 0.8334 | 0.7752 |

Cuadro 3: Resultados de k Nearest Neighbours cambiando la cantidad de vecinos utilizados para clasificar una nueva instancia. Los resultados son calculados tomando el promedio de cada métrica al hacer 10 fold cross validation.

En cuanto a tiempo de computo, los tres modelos tomaron menos de 10 segundos. Podemos observar que en la medida que aumenta la cantidad de vecinos que consideramos al clasificar una nueva instancia k, los resultados en general empeoran. Esto se debe a que al clasificar una nueva instancia, estamos considerando puntos que probablemente no son locales. Dados estos resultados, Random Forests con 100 arboles sigue siendo el que utilizaremos.

## 4. Reducción de dimensionalidad

Se probó el uso de PCA como técnica de reducción de dimensionaidad y algunos selectores de features, basados en test estadísticos univariados. Para ambos casos se vio que la eficiencia, en cuanto a accuracy, precision y recall, disminuía sin importar con cuantos features nos quedásemos. Por lo tanto decidimos simplemente utilizar todos los features que teníamos disponibles.

## 5. Modelo final y performance

Finalmente, dada nuestra experimentación decidimos quedarnos con el modelo que utiliza Random Forests con 100 arboles, sin reducción de dimensionalidad. A continuación podemos ver los resultados que tuvimos entrenando sobre el training set entero y calculando las métricas sobre el test set, comparados con los resultados que obtuvimos sobre el training set.

| **Validacion** | **Cross validation** | **Test set** |
|---|---|---|
| Train mean accuracy | 0.9812 | 0.9795 |
| False positives | 83 | 86 |
| False negatives | 96 | 98 |
| True positives | 4422 | 4410 |
| True negatives | 4409 | 4406 |
| Precision | 0.9810 | 0.9808 |
| Recall | 0.9780 | 0.9782 |
| Time taken | 46.27s | 0.9782 |

Cuadro 4: Performance de Random Forests. En la primera columna, la performance se mide tomando el promedio de cada métrica al hacer 10 fold cross validation. Por otro lado, la segunda columna muestra el modelo entrenado sobre el traning set y validado sobre el test set.

Por lo que podemos ver, en general el modelo parece haber generalizo bien y mantenido su performance.

# 6. Discusión

Luego de experimentar con diferentes técnicas de aprendizaje, observamos que los mejores resultados se obtuvieron utilizando métodos basados en arboles, en particular uno con 100 arboles. En el caso del spam, estamos especialmente interesados en disminuir la cantidad de falsos positivos. Random Forests tuvo el menor numero de falsos positivos comparado con otros métodos. El tiempo de ejecución fue otro factor que consideramos relevante. A diferencia de otros métodos como SVM, en general Random Forests fue relativamente rápido de computar.

Son interesantes algunas de las estrategias que emplean los spammers para evitar este tipo de filtros, como por ejemplo *bayesian poisoning*, en la cual procuran emplear la información con la que se entrenan a los filtros de spam para disminuir su efectividad. Incluyen en sus mails spam palabras y otros atributos estadísticamente característicos de mails ham con el fin de aumentar las probabilidades de ser clasificados como ham. Por otro lado, si los filtros son entrenados con estos mails spam 'envenenados', tenderá a aumentar la cantidad de falsos positivos.