Caso de estudio

Detección de SPAM

Introducción a los métodos de aprendizaje automático

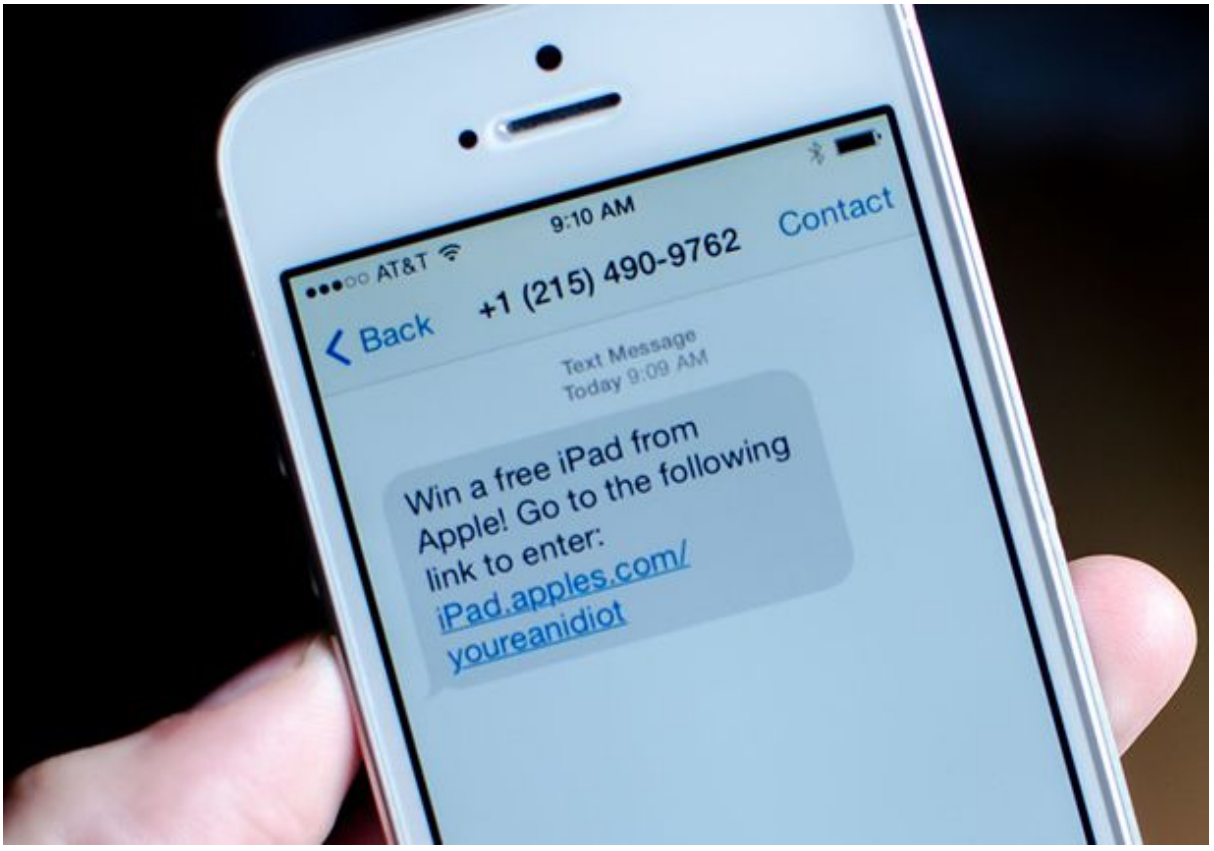
Noviembre 2017

Autores:

- Santiago Casás
- Agustín Betancor
- Martín Rose
- Mauricio Coniglio
- Gianni Iaquinta

Índice

Introducción



Hoy en dia es muy común que las empresas comuniquen sus promociones a través de mensajes de texto, esto genera malestar dado que muchas veces las ofertas/promociones que llegan no son del agrado o no interesan al usuario y para empeorar la situacion aun mas, llegan con mucha frecuencia.
A raíz de este problema es que nos resultó interesante investigar acerca de esto y poder encontrar una manera de determinar de forma automática cuales mensajes corresponden a spam y cuáles no. Esto podría ser de gran ayuda si luego se quisiera automatizar la eliminación de los mismos o crear un aplicación que bloquee este tipo de mensajes.
Es por esta razón que se busco un dataset el cual nos permitiera realizar dicha investigación.
Lo que se busca con este caso de estudio es poder crear un modelo que permita predecir si un mensaje es SPAM o no.

Análisis del problema

El problema que se plantea es poder a través de un atributo de entrada que corresponde a un SMS en inglés, poder predecir si el mismo corresponde a un mensaje de SPAM o no. Claramente el caso de estudio plantea un problema de clasificación dado que se busca clasificar el SMS en alguna de las clases de salida. En este caso al solo poder tomar dos valores, SPAM o NO SPAM, y al ser un problema de clasificación podemos hilar más profundo y ver que nos encontramos frente a un claro problema de clasificación binaria.

Análisis de los datos

El dataset utilizado fue obtenido se llama “SMS Spam Collection”, el mismo se puede encontrar entre los dataset ofrecidos por Kaggle a través de la siguiente URL: https://www.kaggle.com/uciml/sms-spam-collection-dataset
El dataset contiene 5.574 observaciones y no presenta valores faltantes. El mismo está compuesto por dos atributos:

- **result**: Corresponde al resultado de la observación, el mismo puede tomar los valores “ham” para el caso cuando el SMS es legítimo, mientras que toma el valor “spam” cuando el SMS es realmente SPAM. El atributo es binomial.
- **message**: Este atributo contiene el contenido del SMS, el cual es un mensaje en idioma inglés. Este atributo es polynomial.

De interés

Total de datos: 5.574
Total de atributos: 2
Spam: 747
Ham: 4826

Atributos

Para la solución planteada se utilizaron ambos atributos, dado que son los dos de relevancia para el desarrollar del modelo y también cabe destacar que no hay datos faltantes. Por lo que no se tuvo que realizar ningún tipo de reemplazo para este tipo de datos.

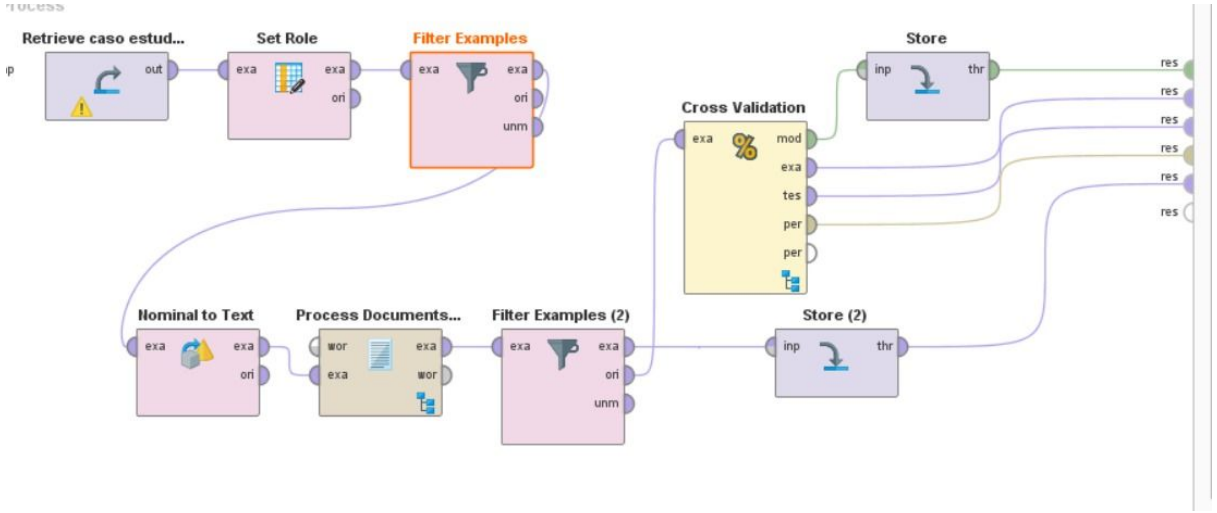
Selección del Operador

Como operador se optó por utilizar un ensamble vote con tres operadores, SVM,, Naive Bayes y W-Logistics(algoritmo que se encuentra en la extensión de Weka).
También se debió agregar distintos tipos de operadores para poder separar las palabras y generamos un archivo con palabras que representen algo importante para poder generar el modelo.

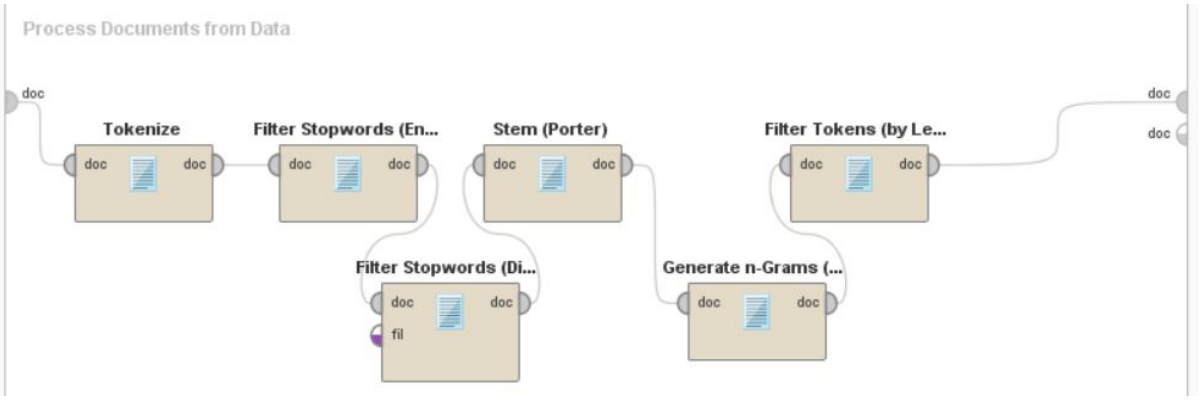
Diagrama del Proceso en RapidMiner

Dado que se realizó un sistema el cual será consumido por una aplicación externa es necesario que la predicción sea los mas rapido posible. Por lo tanto fue necesario separar el entrenamiento del modelo de la predicción de un nuevo ejemplo.
Esto se consiguió utilizando RapidMiner Server, y dividiendo el trabajo en dos procesos distintos. En un proceso se generó y entrenó el modelo, y en el otro se realizó la predicción con el modelo entrenado anteriormente.
A continuación se detallaran ambos procesos.

Proceso local para generar el modelo

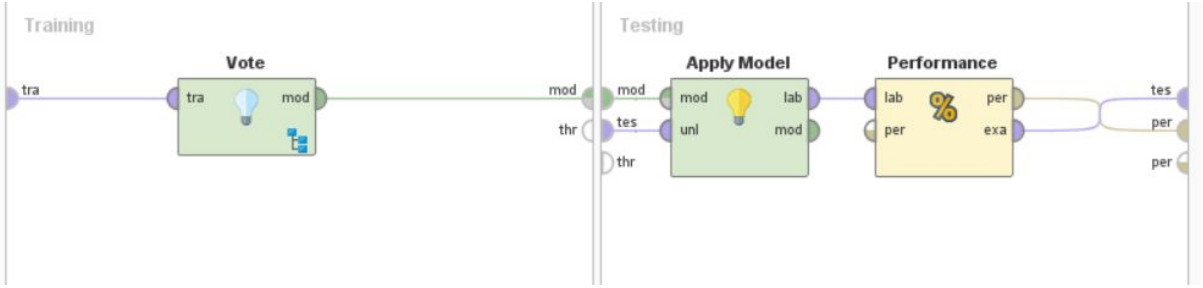


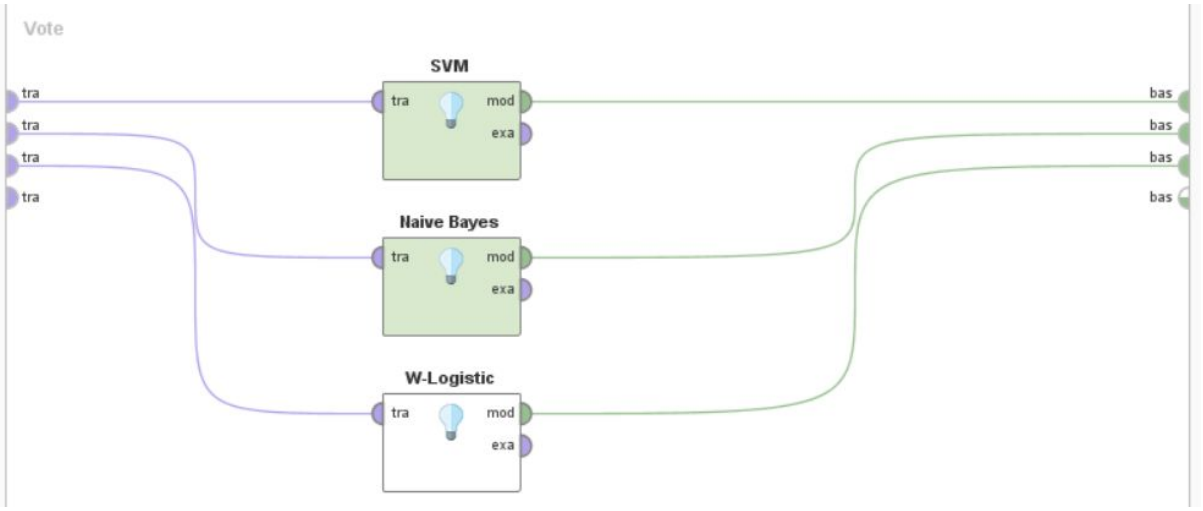
El proceso comienza con la preparación de los datos, lo primero que se hace es establecer el atributo “result” como label utilizando el bloque Set Role. Luego se filtran todos las tuplas con valores faltantes. Y por último se convierte el atributo “message” de polinomial a texto para poder ser procesado en la siguiente etapa.



A continuación se realiza el proceso de extracción de palabras claves del texto(o Tokens), para que luego sean estas las que sean utilizadas para entrenar al modelo. Para esto se utilizan varios bloques entre los que se encuentran: Tokenize, Filter Stopwords, Stem, Generate n-Grams y Filter Tokens(by Length). Tokenize se utiliza para la generación de los tokens individuales. Los bloques Filter Stopwords(English) y Filter Stopwords(Dictionary) se utilizan para eliminar palabras comunes o que no representan nada(conectores, escapes, etc). Generate n-Grams se utiliza para generar Tokens compuesto de más de una palabra, en nuestro caso lo configuramos para que cree Tokens de máximo 2 palabras. Por último se encuentra el bloque Filter Tokens(by Length) que se utiliza para quitar todos los Tokens que no se encuentren dentro del largo especificado en el bloque, nosotros utilizamos palabras con un mínimo de 4 letras y un máximo de 25.
Luego de generados todos los Tokens nos interesa guardar todos esos atributos que tenemos para que luego sean utilizados en el proceso de predicción, por lo tanto filtramos todas las tuplas del dataset(dejandolo vacio) y lo guardamos con un operador de Store.

Por otra parte el Dataset que es generado por el Process Documents es enviado a un Cross Validation en el cual se genera el modelo y luego se evalúa su performance.





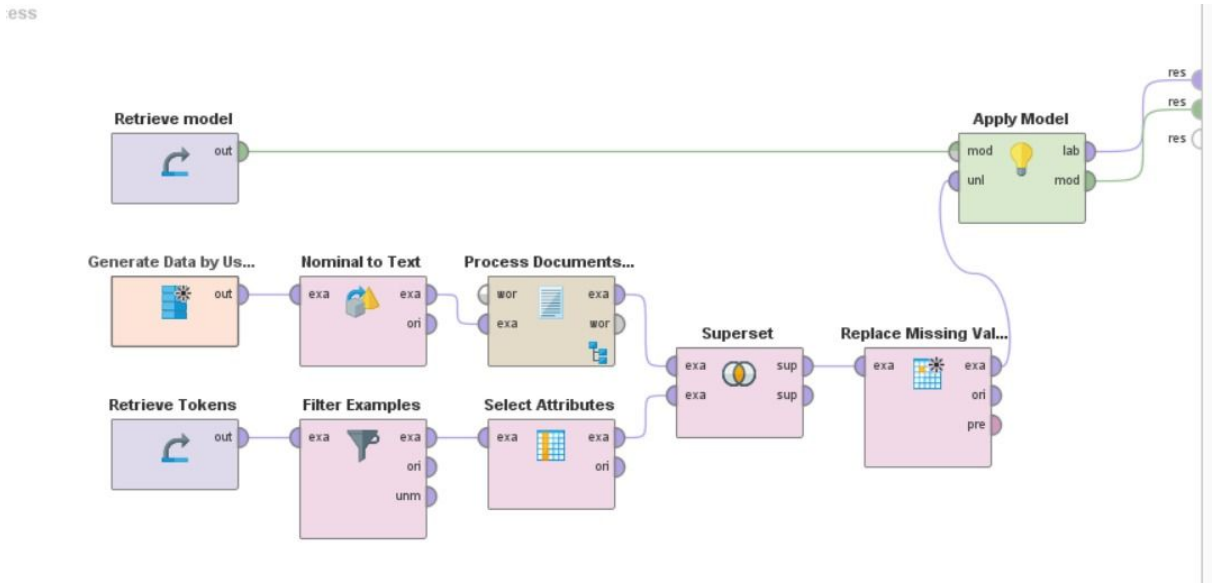
Dentro del Cross validation se utilizó un Vote con tres algoritmos distintos: SVM, Naive Bayes y W-Logistics(algoritmo que se encuentra en la extensión de Weka). Se opto por esta ya que fue uno de los modelos que mejor resultados obtuvo. En especial en lo que se trata de precision(90.60%) y Recall(85.14%), se tuvo un acierto del 96.82%.

accuracy: 96.82% +/- 0.80% (mikro: 96.82%)

	true ham	true spam	class precision
pred. ham	4760	111	97.72%
pred. spam	66	636	90.60%
class recall	98.63%	85.14%	

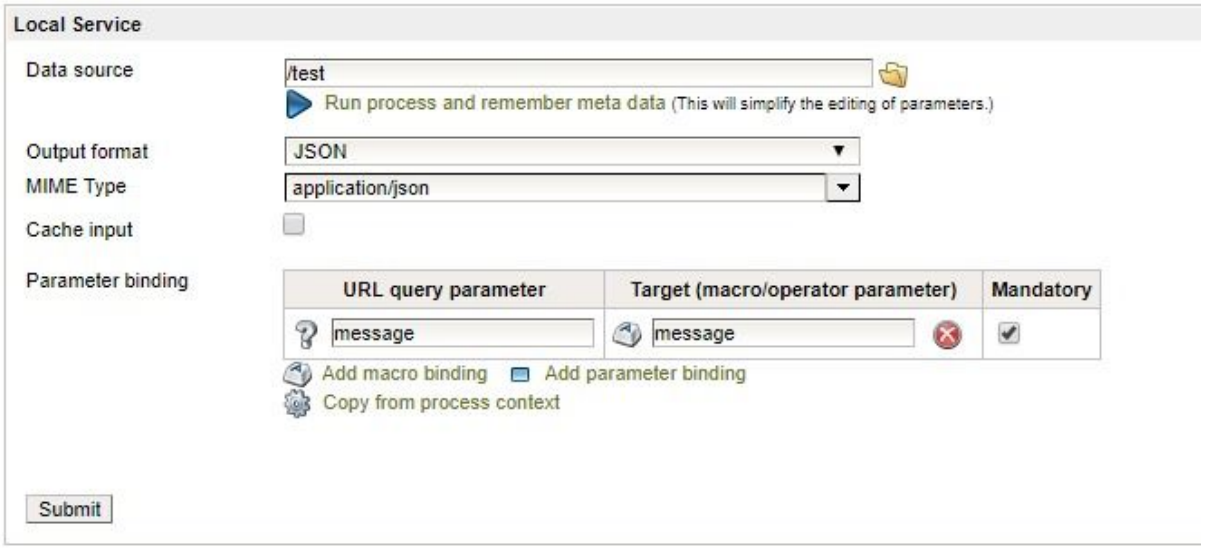
Otros modelos no obtuvieron tan buenos resultados o requieren cantidades muy grandes de tiempo y procesamiento para solo aumentar el rendimiento en 0.5%. También se dieron casos en los cuales no se pudieron utilizar los modelos ya que no podían ser guardados dentro del servidor utilizando el operador Store, como fue el caso de un operador K-NN dentro del Vote. Más adelante se mostraran algunas de la soluciones que se probaron.
Por último, y antes de continuar al siguiente proceso, se guarda el modelo generado dentro del cross validation para su posterior utilización.

Proceso en el server para la aplicación



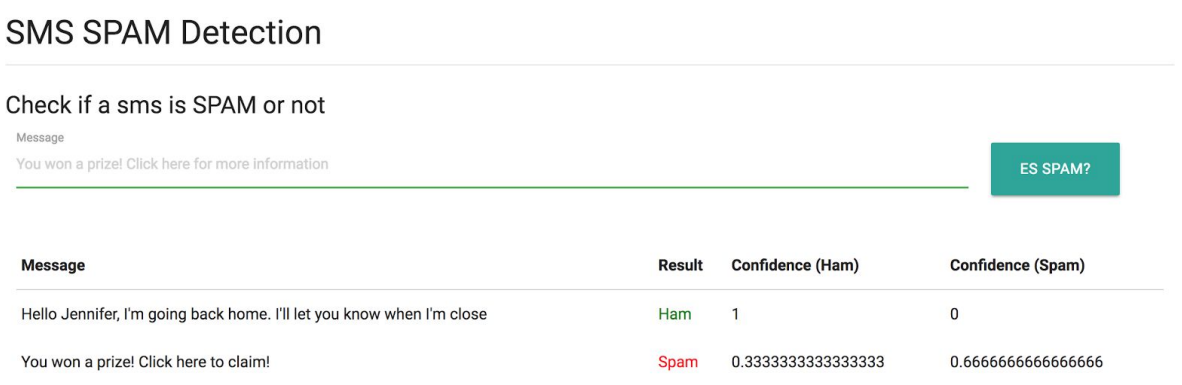
El segundo proceso es el de la predicción. En este proceso se producirá el nuevo valor ingresado. Se comienza generando la tupla con los datos ingresados por la persona. Para esto se tuvo que configurar el server y asociar un query parameter con un macro del proceso(más sobre esto a continuación). Al igual que en el otro proceso, luego se pasa a cambiar el atributo al tipo text y a procesarlo para generar los Tokens.
Por otro lado se carga el dataset con todos los Tokens que fue guardado en el otro proceso, se filtra para sacarle las tuplas que tenga(solo como una precaución), y se quita el atributo result.
Luego de cargados todos los Tokens del proceso anterior y luego de generar los nuevos Tokens se continua a juntar esas tablas. Esto generará una tabla que contiene una sola tupla con las columnas que estaban en una o la otra tabla. Esta nueva tupla tendrá valores numéricos en la columnas que estaban presentes en la tabla de Tokens generada para el nuevo mensaje, y valores missing en todo el resto. Es por eso que el siguiente paso es transformar todos los valores missings en 0.
En este momento ya se cuenta con el nuevo mensaje ya pronto para ser procesado por el modelo, es por eso que lo siguiente que se hace es cargar el modelo generado en el otro proceso, y utilizar un Apply Model para realizar la predicción. Esta predicción es exportada junto con la confianza para cada una de las dos opciones.

Servicio Web



Para consumir el proceso encargado de realizar la predicción del nuevo mensaje se utilizo un servicio web, para esto fue necesario crearlo y configurarlo en Rapidminer Server. El servicio se creó utilizando el proceso que se realizo anteriormente en el Rapidminer Studio para la predicción del nuevo mensaje. Se estableció que devolviera la información como JSON. Y se le agrego un query parameter con el mismo nombre del macro que se agregó anteriormente en el proceso, y se puso como obligatorio.

Luego de que el servicio web quedó funcionando se desarrolló una aplicación web que consumiera ese servicio para realizar predicciones a partir del modelo generado. La aplicación lleva a cabo la autenticación con las credenciales del servidor y luego hace uso del servicio enviando un mensaje y recibiendo el resultado de la predicción.

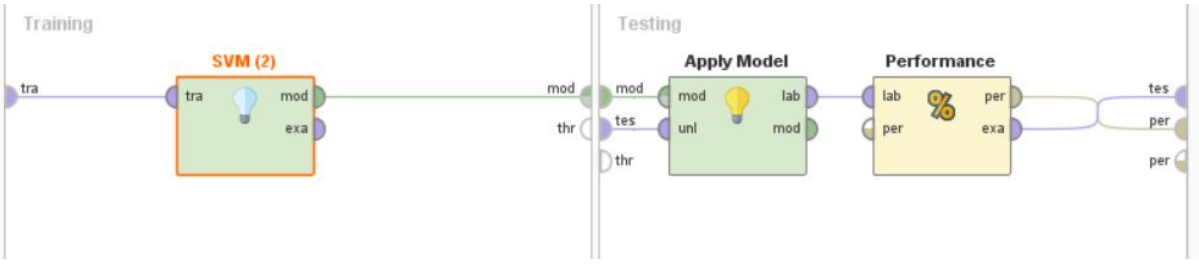


Conjunto de Entrenamiento y Testing

Utilizamos el operador Cross Validation para realizar el conjunto de entrenamiento y test, lo que utilizamos fue una validación cruzada de 10 iteraciones, lo que quiere decir que vamos a tener 10 subconjuntos con 550 datos cada uno aprox.

Soluciones Alternativas

Se probaron muchas alternativas además de la presentada en el proceso final. Algunas obtenían resultados muy malos y otros muy buenos, o demoraban mucho y no mejoraron sustancialmente, y otros no eran compatibles con el servidor de rapidminer.
A continuación algunos de las soluciones que se probaron:



accuracy: 96.70% +/- 0.83% (mikro: 96.70%)

	true ham	true spam	class precision
pred. ham	4805	163	96.72%
pred. spam	21	584	96.53%
class recall	99.56%	78.18%	

Uno de los primeros modelos que se utilizó fue un SVM, que luego de un par de iteraciones para ajustar los valores se obtuvo un acierto del 96.70% y una precisión del 96.53% pero un recall de solo 78.18%.
Otro caso que se probó fue la utiliza fue la utilización de K-NN, este modelo mejoraba en alg

Análisis de los Resultados

En el documento ya mostramos los resultados obtenidos por el proceso utilizado pero vamos a analizar los mismos.

accuracy: 96.82% +/- 0.80% (mikro: 96.82%)

	true ham	true spam	class precision
pred. ham	4760	111	97.72%
pred. spam	66	636	90.60%
class recall	98.63%	85.14%	

Como se puede ver en la imagen la exactitud obtenida es del casi 97%, esto quiere decir que casi no hay falsos positivos y/o falsos negativos. La manera de verificar si el resultado obtenido es bueno es verificando el porcentaje recall del proceso, dado que el recall nos

dice que cantidad de spam el modelo predijo sobre toda la cantidad de spam que tiene el dataset.

$$recall = Fue\ interesante\ aplicar\ algoritmos\ de\ machine\ learning\ en\ conjunto\ con\ técnicas\ de\ procesamiento\ de\ lengu$$

tp: es la cantidad de spam que el modelo predijo que era spam y es realmente spam en el dataset.

fn: es la cantidad de spam que hay en el dataset y que el modelo no predice como spam.

Como la clase a predecir es si el mensaje es spam o no obtuvimos casi un 85% de acierto, por lo que la cantidad de fn es baja.

Aparte la precision tambien es buena, porque obtuvimos casi 90% de acierto; lo que nos quiere decir que de todo lo que el modelo predijo como spam, solo una pequeña cantidad no lo fue.

$$precision = \frac{tp}{tp+fp}$$

fp: es la cantidad de predicciones que hizo mal el modelo, es decir que es la cantidad de lo que el modelo predijo como spam y era legitimo.

Conclusiones

Fue interesante aplicar algoritmos de machine learning en conjunto con técnicas de procesamiento de lenguaje natural. Gracias a la gran cantidad de operadores que tiene Rapid Miner pudimos encontrar algunos de gran utilidad para el caso de estudio planteado.

A pesar de obtener un resultado positivo somos conscientes que la cantidad de instancias en el dataset son pocas y pudiendo aplicar este modelo a un dataset mas grande puede ajustarse mejor y tener como resultado un modelo utilizable en un entorno real.