

Universidad Nacional del Sur

Tesis de Licenciatura en Ciencias de la  
Computación

---

Minería de Opinión en Twitter

Clasificación de Tweets y visualización de  
tendencias de opinión

---

Autor  
Leonardo Nieto

Directora  
Dra. Ana Gabriela Maguitman

Co-Director  
Dr. Carlos M. Lorenzetti

Departamento de Ciencias e Ingeniería de la Computación



<b>1. Introducción</b>	<b>3</b>
1.1. Motivación	3
1.2. Objetivos	3
1.3. Metodología de Trabajo	4
<b>2. Sistema de Clasificación y Visualización en Tiempo Real</b>	<b>6</b>
2.1. Tecnologías	6
2.2. Arquitectura	7
2.2.1. Base de Datos	7
2.2.2. Backend	8
2.2.2.1. Motor de Clasificación	8
2.2.2.2. API REST	8
2.2.4. Frontend	9
2.3. Interfaz con el usuario	10
2.3.1. Funcionalidades	10
2.3.1.1. Búsqueda Histórico	11
2.3.1.2. Búsqueda en Tiempo Real	11
2.3.1.3. Referencias	13
2.3.1.4. Grilla de Tweets	13
2.3.2. Limitaciones de la Interfaz de Usuario	14
<b>3. Clasificación de Tweets - SVM, Aprendizaje Supervisado</b>	<b>15</b>
3.1. TASS Corpus	15
3.2. Feature Vectors	16
3.2.1. Pre-Procesamiento de Tweets	16
3.2.2. Generación de Feature Vectors	17
3.3. Generación del Modelo	17
3.3.1. SVM Kernels	17
3.3.2. F1-Measure	18
3.3.2. Modelo Seleccionado	19
<b>4. Evaluación de Desempeño</b>	<b>20</b>
4.1. Métricas utilizadas	20
4.2. Evaluación	21
4.2.1. Support Vector Machine	22
4.2.1.1. Resultados - Linear Kernel	22
4.2.1.2. Resultados - RBF Kernel	23
4.2.2. Simple Classifier	24

4.2.2.1. Resultados	24
4.2.3. Comparación entre los modelos	25
<b>5. Conclusión</b>	<b>26</b>
<b>Anexo</b>	<b>27</b>
Material de este trabajo	27
SVM, Kernels y sus Parámetros	27
Linear Kernel	27
RBF Kernel	28
Grid Search	29
Cross Validation	29
K-Fold Cross Validation	29
<b>Bibliografía</b>	<b>31</b>

# 1. Introducción

## 1.1. Motivación

El advenimiento de redes sociales y las herramientas que estas proveen nos permiten utilizar su flujo de información para análisis y procesamiento. En particular, este trabajo se centra en el uso Twitter<sup>1</sup>, que gracias a su Streaming API<sup>2</sup> podemos acceder, en tiempo real, al flujo de mensajes que en esta red social se generan.

Combinando la información obtenida de la red social junto con las técnicas de minería de datos y clasificación podemos generar modelos que nos permitan identificar aspectos de nuestro interés en los mensajes. Además contamos con tecnologías como Google Maps Platform<sup>3</sup> que nos permite geolocalizar la información de forma sencilla y práctica así como con bases de datos orientadas a documentos que nos facilitan trabajar con grandes volúmenes de información no estructurada.

Este trabajo tiene un fin pragmático, combinar estas tecnologías para generar una aplicación que nos permita visualizar en tiempo real las tendencias de opinión que se generan en el flujo de información de Twitter.

## 1.2. Objetivos

Twitter es un servicio de microblogging muy utilizado. Los usuarios crean pequeños mensajes (de no más de 280 caracteres) llamados tweets. Muchos de estos tweets expresan opinión sobre diferentes temas.

Este trabajo tiene como objetivo extraer de estos tweets, escritos en idioma español, tendencias de opinión, determinar la polaridad de cada uno y volcar la información en un mapa

---

<sup>1</sup> <https://twitter.com/>

<sup>2</sup> <https://developer.twitter.com/en/docs/basics/getting-started>

<sup>3</sup> <https://cloud.google.com/maps-platform/>

que permita visualizar el "humor" social de cada zona en tiempo real.

Para evaluar la tendencia de opinión (polaridad) de cada tweet se cuenta con un clasificador que permite establecer la clase a la que pertenece cada mensaje. Estas clases son: positivo, negativo, neutral y ninguno (no expresa opinión). Para poder visualizar la tendencia de opinión por zona y en tiempo real se usó Stream API de Twitter, se toman los tweets que se generan en el momento y se procede a realizar la geolocalización de cada uno en un mapa. El modelo de clasificación generado es el responsable de clasificar cada tweet según su polaridad y de esta manera, junto con la geolocalización de los tweets, podemos visualizar las tendencias de opinión por zona y en tiempo real.

### 1.3. Metodología de Trabajo

Este trabajo tiene como objetivo la clasificación de tweets en idioma español por lo tanto se utilizó como sets de entrenamiento y testeo del clasificador el corpus TASS<sup>4</sup>. El corpus general contiene cerca de 68000 tweets escritos en español entre Noviembre de 2011 y Marzo de 2012. Si bien el contexto de extracción de este corpus está enfocado en España, la diversas nacionalidades de los autores, incluyendo personas de España, México, Colombia, Puerto Rico, Usa y otros países, hace que este corpus cubra una audiencia hispano parlante representativa.

Con este corpus se construyó un clasificador supervisado que utiliza SVM. Para esto se realizó un preprocesamiento de los tweets (stemming, tratamiento de emoticones y signos de puntuación, remover stop words, llevar el texto a lowercase, etc.). Luego de este preprocesamiento se generan los "feature vectors", inputs del clasificador.

El modelo generado se encuentra integrado en una aplicación que, junto con Streaming API de Twitter, permite la clasificación en tiempo real de los tweets.

---

<sup>4</sup> "TASS 2015 : SEPLN 2015 workshop on: Sentiment Analysis at SEPLN." 2015. 10 Jul. 2016  
<<http://www.sepln.org/workshops/tass/2015/tass2015.php>>

Para visualizar la geolocalización de los tweets así como su polaridad se desarrolló una interfaz web que permite graficarlos en un mapa. Además la interfaz permite ingresar términos de búsqueda para filtrar los mensajes acorde a los términos de interés y así poder analizar la polaridad de opinión basada en un tópico particular.

Para poder analizar la performance del modelo de clasificación construido se realizó una comparación contra un esquema de clasificación donde sólo tenemos en cuenta el número de ocurrencias de cada término positivo y negativo en cada tweet, es decir, un tweet con mayor cantidad de palabras positivas será positivo, en sentido inverso, de tener mayor cantidad de términos negativos será considerado negativo, de tener similar número de términos negativos y positivos, será considerado neutro y de no contar con términos positivos o negativos será considerado como un tweet que no expresa opinión.

De ambos clasificadores se tomaron las medidas de precisión, recall y f-measure y bajo estas métricas se realizó la comparación de un método con respecto al otro.

## 2. Sistema de Clasificación y Visualización en Tiempo Real

### 2.1. Tecnologías

Se utilizó el lenguaje Python en el desarrollo del modelo de clasificación y el backend de la aplicación. La interfaz de usuario se generó utilizando HTML, CSS y Javascript.

Las principales librerías utilizadas en el desarrollo del modelo de clasificación son:

```
nltk==3.2.1
pandas==0.19.2
scikit-learn==0.19.1
scipy==1.0.0
```

En particular, scikit-learn provee el conjunto de herramientas que se utilizó para la clasificación, evaluación y optimización del modelo.

Las principales librerías utilizadas en el backend de la aplicación son:

```
falcon==1.4.1: Python web API framework
gunicorn==19.7.1: Python WSGI HTTP Server for UNIX
pymongo==3.7.1: conexión a base de datos MongoDB
tweepy==3.6.0: abstracción de Streaming API de Twitter.
```

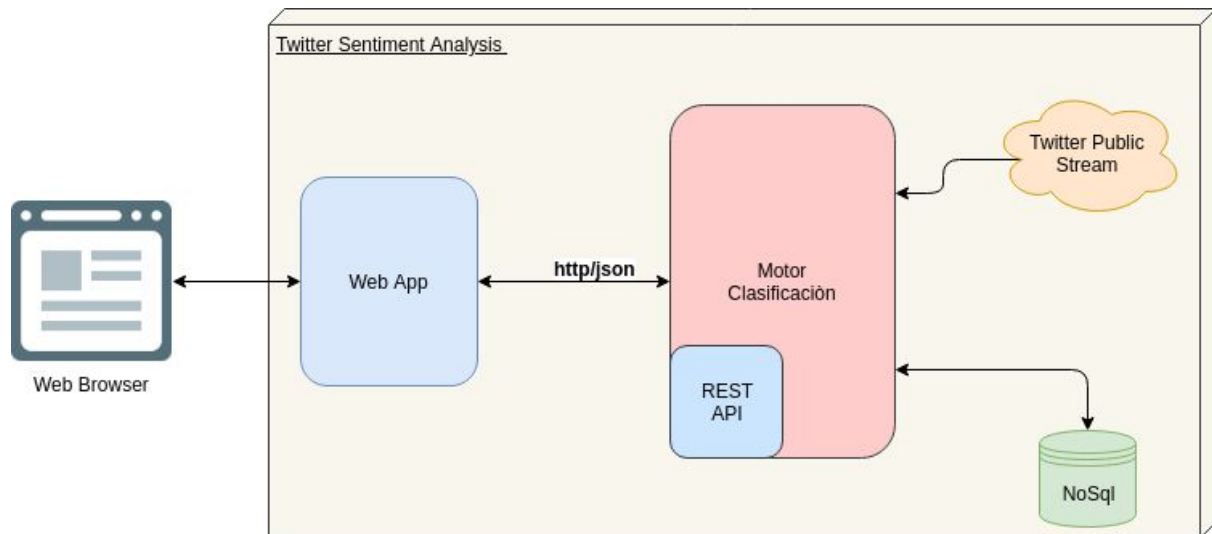
Las principales librerías utilizadas en el frontend (interfaz de usuario) de la aplicación son:

```
jquery==3.3.1
jsgrid==1.5.3
Maps JavaScript API: parte de Google Maps Platform
```

En particular la interfaz hace uso del elemento **input datetime-local** que aún no cuenta con un soporte amplio entre

los browsers. Se aconseja utilizar la última versión de Chrome para operar la interfaz de usuario.

## 2.2. Arquitectura



### 2.2.1. Base de Datos

Como base de datos se eligió MongoDB<sup>5</sup>. Esta base de datos pertenece a la familia de bases de datos llamadas NoSQL. En particular es una base de datos con un modelo orientada a documentos que utiliza una representación basada en JSON para almacenar los datos.

Este tipo de base de datos nos brinda flexibilidad al trabajar con grandes volúmenes de información no estructurada. Por tales motivos fue elegida para trabajar en este proyecto, el stream de tweets que leemos de Twitter utiliza JSON como formato de los mensajes y nos permite almacenarlos de forma rápida, eficiente y sencilla. El posterior trabajo que se hace con los tweets también se ve facilitado ya que no es necesario realizar un mapeo a otra representación.

---

<sup>5</sup> <https://www.mongodb.com/>



### 2.2.2. Backend

El backend de la aplicación está compuesto por el motor de clasificación, la interfaz hacia Streaming API de Twitter, la conexión hacia la base de datos y expone una API REST que brinda servicios a la interfaz de usuario.

#### 2.2.2.1. Motor de Clasificación

El motor de clasificación, su componente principal es el modelo obtenido del entrenamiento del clasificador basado en SVM. El modelo generado se encuentra almacenado en un archivo que se carga en memoria y se utiliza para clasificar los tweets provenientes del stream de Twitter.

**El flujo de operación es el siguiente:**

1. Un thread lee continuamente el flujo proveniente del stream de Twitter y se encarga de almacenarlo en base de datos.
2. Un segundo thread lee los mensajes nuevos (en base de datos) no procesados, los clasifica y genera una nueva representación de cada tweet que incluye su polaridad y los tokens en los que se transformó el mensaje.

#### 2.2.2.2. API REST

A continuación se listan los endpoints que expone el API.

**GET /tweets/{fechaDesde}/{fechaHasta}**

- Permite obtener los tweets almacenados en base de datos entre las fechas especificadas.
- Formato de fecha: %Y-%m-%dT%H:%M-UTC%Z. Donde:
  - ◆ %Y - year including the century
  - ◆ %m - month (01 to 12)
  - ◆ %d - day of the month (01 to 31)
  - ◆ %H - hour, using a 24-hour clock (00 to 23)

- ◆ %M - minute
- ◆ %z - time zone or name or abbreviation

#### GET /realtimetweets/{timestamp}

- Permite obtener los tweets almacenados en base de datos cuyo timespan sea mayor o igual al ingresado en el parámetro.
- Formato timestamp: int. Timestamp, en segundos, transcurridos desde Epoch.

#### POST /classifications

- Inicia clasificación de tweets en tiempo real.
- POST Data (JSON): Array(String) => Lista de términos.
  - ◆ Filtra los tweets según la lista de términos.  
Ejemplo: ['termino 1', 'termino 2']
- POST Data: Empty.
  - ◆ No aplica filtro de términos sobre los tweets. Busca geolocalizar los tweets sobre la región Argentina:  
[-73.616669,-55.185078,-53.637451,-21.781168]

#### PUT /classifications

- Si está ejecutando, detiene la clasificación en tiempo real de tweets.

### 2.2.4. Frontend

El frontend se encuentra desarrollado en HTML, Javascript y CSS. Es una interfaz que permite manejar las operaciones que expone el motor de clasificación, visualizar los tweets procesados en una grilla y si cuentan con geolocalización, se vuelcan en el mapa (Google Maps).

La comunicación con el backend se realiza por llamadas AJAX implementadas con jquery y el formato de datos en los mensajes es JSON.

## Estructura de la solución

### ./classification\_interface

#### assets

scripts.js: scripts que utiliza la página.

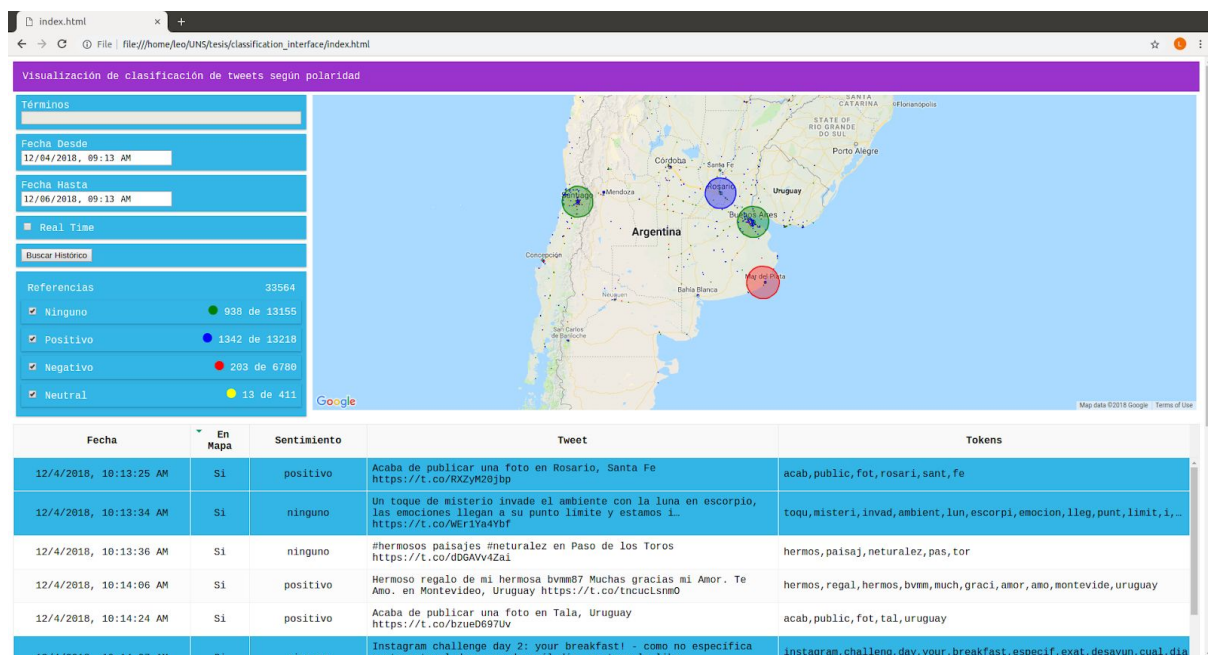
spin.css: estilos del indicador de espera.

styles.css: estilos generales de la página.

index.html: página que define la interfaz.

## 2.3. Interfaz con el usuario

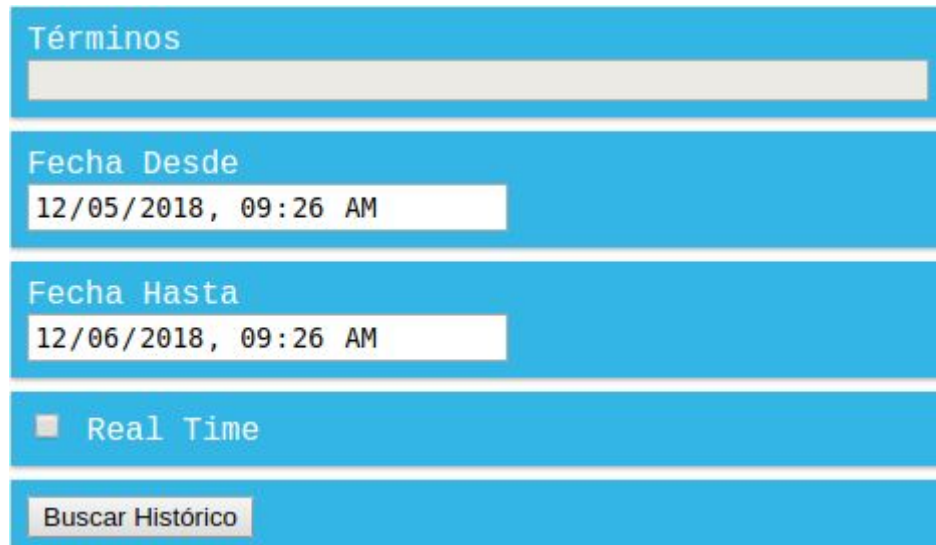
La interfaz nos permite visualizar los tweets obtenidos y procesados del stream de Twitter. Podemos ver, para aquellos tweets que cuenten con coordenadas de geolocalización, su disposición en el mapa. Por otro lado, también podemos ver, dentro de la grilla, todos los tweets obtenidos y procesados.



### 2.3.1. Funcionalidades

La interfaz cuenta con dos modos de operación, Búsqueda de Histórico y Búsqueda en Tiempo Real. Estos dos modos son excluyentes, es decir, o se trabaja en uno o en otro y dependiendo de cual esté activo se configura la interfaz para poder operar en el modo seleccionado.

#### 2.3.1.1. Búsqueda Histórico



Términos

Fecha Desde  
12/05/2018, 09:26 AM

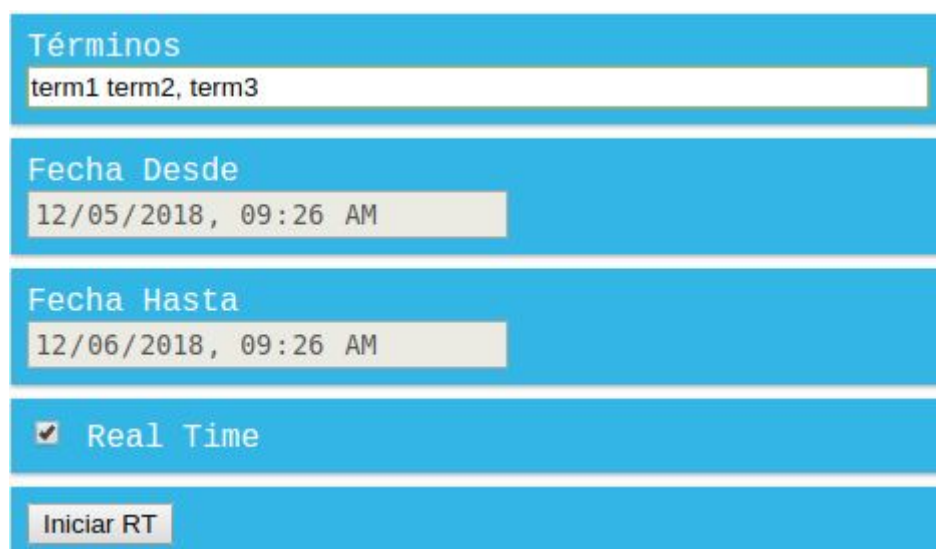
Fecha Hasta  
12/06/2018, 09:26 AM

☐ Real Time

Buscar Histórico

Este modo se encuentra activo cuando el checkbox "Real Time" no se encuentra seleccionado. Permite ingresar "Fecha Desde" y "Fecha Hasta", al presionar en "Buscar Histórico" consulta los tweets procesados en el margen de tiempo ingresado. Notar que en este modo el ingreso de Términos no está permitido, por lo tanto, el cuadro de input se deshabilita impidiendo ingresarlos.

#### 2.3.1.2. Búsqueda en Tiempo Real



Términos  
term1 term2, term3

Fecha Desde  
12/05/2018, 09:26 AM

Fecha Hasta  
12/06/2018, 09:26 AM

☒ Real Time

Iniciar RT

Este modo se encuentra activo cuando el checkbox "Real Time" se encuentra seleccionado, al presionar "Iniciar RT" se inicia la lectura y procesamiento en tiempo real de los tweets que genera el streaming de Twitter.

El modo de operación permite ingresar una lista de términos para filtrar tweets que los incluyan. Si la lista de términos está vacía y se inicia el procesamiento, busca todos los tweets que se encuentren geolocalizados en el radio de la Argentina así como aquellos tweets que no tengan geolocalización alguna. Notar que al seleccionar "Real Time" se deshabilitan los inputs de Fecha ya que el modo no requiere estos parámetros.

### **Sobre la búsqueda por términos**

Cuando se ingresan los términos de búsqueda, los espacios en blanco separando los términos funcionan como un operador "AND" y las comas ',' funcionan como un operador "OR". Es decir, cuando los términos se encuentran separados por un blanco filtra los mensajes que incluyen ambos términos pero sin importar el orden en el que aparecen en el tweet. Por otro lado, si están separados por ',' filtra los mensajes conteniendo alguno de los términos.

En el ejemplo que figura en el screenshot, (búsqueda de términos: term1 term2, term3), filtra los mensajes conteniendo los términos term1 y term2, sin importar el orden en que aparecen, o los mensajes que contienen term3.

### **Sobre la interacción de la interfaz con el backend**

Cuando iniciamos una búsqueda en tiempo real la interfaz hace polling sobre un servicio que expone el backend para ir consultando periódicamente los nuevos mensajes obtenidos y procesados. Cuando se detiene, cuando presiona "Stop RT", la interfaz hace una llamada por AJAX al backend indicando que detenga el procesamiento de tweets en tiempo real. Si en lugar de detener la búsqueda en tiempo real cerramos el navegador, esta queda ejecutándose en el backend. Al volver al sitio y presionar "Iniciar RT", la interfaz llama al backend para iniciar la búsqueda pero como ya se encuentra trabajando retorna el timestamp que marca el inicio del procesamiento

previo. Esto nos permite continuar con la visualización del procesamiento en tiempo real que habíamos iniciado y después “abandonado” cerrando el sitio.

#### 2.3.1.3. Referencias

Referencias		17840
<input checked="" type="checkbox"/> Ninguno		647 de 6656
<input checked="" type="checkbox"/> Positivo		712 de 7341
<input checked="" type="checkbox"/> Negativo		154 de 3625
<input checked="" type="checkbox"/> Neutral		9 de 218

El cuadro de referencias nos permite ver el total de los tweets procesados así como la cantidad de los que se encuentran en el mapa y el color que los representa. En el ejemplo que vemos en el screenshot vemos que procesamos 17840 tweets de los cuales 7341 son positivos y 712 tienen coordenadas de geolocalización que nos permite visualizarlo en el mapa.

Otro aspecto importante de las referencias son los checkbox, al estar seleccionados o no nos permite filtrar los mensajes diagramados en el mapa, es decir, si se desmarca la selección de “Positivo”, en el mapa vamos a ver todos los mensajes menos aquellos positivos.

#### 2.3.1.4. Grilla de Tweets

La grilla contiene los tweets procesados, su texto así como la polaridad estimada y el vector de tokens que se utilizó para clasificarlo. La misma se puede ordenar de forma ascendente o descendente según la selección en las columnas. En particular, si el tweet declara “Si” en la columna “En Mapa”, nos indica que existe un punto que lo representa en el mapa, si lo seleccionamos, vemos ese punto resaltado.

### 2.3.2. Limitaciones de la Interfaz de Usuario

La interfaz trabaja con todos los datos en memoria, la paginación que vemos en la grilla es local al conjunto de tweets que estamos manejando en memoria. Por tal motivo, de trabajar con un volumen de datos excesivos podemos generar problemas de retardo e incluso agotar la memoria y generar un "cuelgue" del browser.

Hay optimizaciones que se pueden hacer, como por ejemplo, resolver la paginación en el backend y que la UI trabaje solo con los tweets correspondientes a la página. O bien podemos hacer un desdoble con respecto a los tweets que cuentan con coordenadas para su geolocalización y los que no ya que estos últimos son los que más suman al volumen de información y así trabajar con todos los tweets geolocalizables en memoria y los que no, traerlos paginados por demanda.

Estas optimizaciones se encuentran fuera del scope de este trabajo y se dejan a futuros desarrollos.

### 3. Clasificación de Tweets - SVM, Aprendizaje Supervisado

El modelo para la clasificación de tweets se construyó utilizando Support Vector Machine. Los feature vector se construyeron mediante un pre-procesamiento de los mensajes, cada componente del vector cuenta con uno de dos valores [0 (False), 1 (True)] indicando si el término está presente o no en el tweet. Estos valores se eligieron bajo la idea de que al ser mensajes cortos no importa la cantidad de ocurrencia del término, sino la ocurrencia del término en el tweet.

La validación del modelo y la obtención de sus Hiper-parámetros se realizó utilizando técnicas de cross-validation y grid search.

El pre-procesamiento de los mensajes y la generación de los feature vector se realizaron mediante algoritmos utilizando el lenguaje Python y el soporte de las librería NLTK<sup>6</sup> y scikit-learn<sup>7</sup>.

#### 3.1. TASS Corpus

Como se mencionó anteriormente este trabajo utiliza el corpus TASS para generar un modelo de clasificación que nos permita predecir la polaridad de un tweet. La entidad Tweet en este corpus está modelada utilizando XML de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<tweet>
  <tweetid>142378325086715906</tweetid>
  <user>jesusmarana</user>
  <date>2011-12-02T00:03:32</date>
  <lang>es</lang>
  <sentiments>
    <polarity>
      <value>N</value>
    </polarity>
  </sentiments>
```

---

<sup>6</sup> Natural Language Toolkit: <http://www.nltk.org/>

<sup>7</sup> Scikit-Learn: <https://scikit-learn.org/>



```

<topics>
  <topic>política</topic>
</topics>
<content>Portada 'Público', viernes. Fabra al banquillo por
'orden' del Supremo; Wikileaks 'retrata' a 160 empresas espías.
http://t.co/YtpRU0fd</content>
</tweet>

```

Donde la polaridad asociada a cada uno puede ser uno de los siguientes valores: **N+** (muy negativo), **N** (negativo), **NEU** (neutral), **P** (positivo), **P+** (muy positivo), **NONE** (no expresa sentimiento). En este trabajo, al utilizar cuatro niveles de polaridad (ninguno, positivo, negativo y neutral), se realizó el siguiente mapeo a cada uno de los valores:

```

NONE → ninguno
P, P+ → positivo
N, N+ → negativo
NEU → neutral

```

NOTA: dentro del corpus también existen tweets que modelan la polaridad asociada a las entidades que se mencionan dentro del mensaje. Estas polaridades a nivel de entidad no se consideraron, solo se tuvo en cuenta la polaridad global del tweet. Para mayores detalles ver sección "General Corpus" en <http://www.sepln.org/workshops/tass/2015/tass2015.php>

## 3.2. Feature Vectors

### 3.2.1. Pre-Procesamiento de Tweets

Para el pre-procesamiento de los tweets se llevaron adelante las siguientes acciones:

1. Stemming (lemmatization). Se utilizó SnowballStemmer incluido en NLTK.
2. Tratamiento de emoticones. Se consideran un token.
3. Tratamiento signos de puntuación. Se remueven los signos de puntuación.
4. Stop words. Se remueven stop words.
5. Lower case words. Se lleva a minúsculas todos los tokens.
6. Se eliminan las URLs y direcciones de E-Mails.

7. Referencias a usuarios, @user. Se eliminan las referencias a usuarios.
8. Hashtags. Se elimina el símbolo de hash, "#hashtag" se transforma en "hashtag".
9. Reducción de longitud de las palabras. Los caracteres repetidos más de tres veces se contraen a tres, por ejemplo "hooooooooola" se transforma en "hoooola".

De estas acciones se desprenden los Tokens que forman los componentes que proyectan los feature vector. Por ejemplo,

Tweet: Una de las mejores series que he visto!! #Merlí 🙌🙌🙌  
<https://t.co/37...>

Tokens: mejor, seri, vist, merl, 🙌, 🙌, 🙌

### 3.2.2. Generación de Feature Vectors

Para la generación de los vectores se utilizó el módulo CountVectorizer presente en la librería scikit-learn. Esta clase convierte los tokens generados en un vector compuesto por valores booleanos, [0, 1], dependiendo si el token está presente en el tweet o no. Un vector tiene tanto componentes como tokens extraídos del corpus y el valor de cada uno para un mensaje determinado está dado por la presencia (1) o ausencia (0) de este token en el tweet que el vector representa.

## 3.3. Generación del Modelo

En la generación del modelo de clasificación se utilizó SVM. La optimización de los parámetros se buscó empleando Grid Search junto con Cross Validation. La métrica empleada en la evaluación de los distintos modelos fue F1-Measure.

### 3.3.1. SVM Kernels

Los kernels utilizados para la búsqueda del mejor modelo fueron Linear y RBF. Sobre los parámetros C y Gamma, input a estos kernels, primero se realizó una búsqueda de parámetros general, una vez encontrado el mejor valor del conjunto, se

hizo una búsqueda en torno al valor encontrado inicialmente con el objetivo de obtener un mejor modelo.

### **Linear Kernel**

- Primer conjunto de valores

$$C = 2^{-5}..2^{15}$$

- Segundo conjunto de valores

$$C = 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0$$

### **Radial Basis Function Kernel (RBF)**

- Primer conjunto de valores

$$C = 2^{-5}..2^{15}$$

$$\text{Gamma} = 2^{-15}..2^3$$

- Segundo conjunto de valores

$$C = 2^{0.5}..2^{1.5}$$

$$\text{Gamma} = 2^{-3.5}..2^{-2.5}$$

### 3.3.2. F1-Measure

Esta métrica se puede interpretar como el promedio ponderado entre las métricas de precision y recall, donde alcanza su mejor valor en 1 y el peor en 0. La contribución relativa de estas métricas al valor obtenido por F1-Score son idénticas, es decir, ambas pesan lo mismo.

#### **Formulación:**

$$F1 = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

En particular, para la evaluación de los distintos modelos resultados de la variación en sus parámetros, y dado que es una clasificación entre cuatro clases (ninguno, positivo, negativo, neutral) se utilizó F1-Score ponderada por la

cantidad de instancias que efectivamente pertenecen a la clase. Esto nos permite llegar a un valor final de F1-Score resultado del promedio ponderado entre los valores de F1 obtenidos para cada clase, donde las clases con más instancias son las más relevantes.

En términos de `sklearn.metrics.f1_score`<sup>8</sup> el valor utilizado para el parámetro `average` fue `'weighted'`.

### 3.3.2. Modelo Seleccionado

Finalmente, los valores obtenidos para el modelo utilizado en la clasificación en tiempo real de tweets fue:

**Kernel:** RBF

**C** =  $2^{1.5}$

**Gamma** =  $2^{-3.5}$

---

<sup>8</sup> [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1\\_score.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)

## 4. Evaluación de Desempeño

### 4.1. Métricas utilizadas

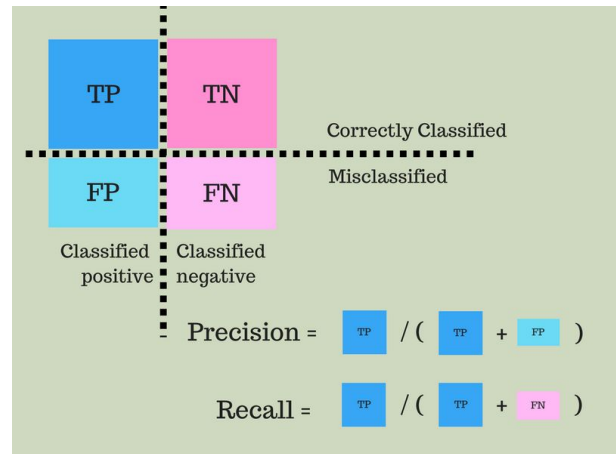
El desempeño se evalúa en términos de las métricas de Precision, Recall y F-Measure.

$$Precision = \frac{TP}{TP + FP}$$

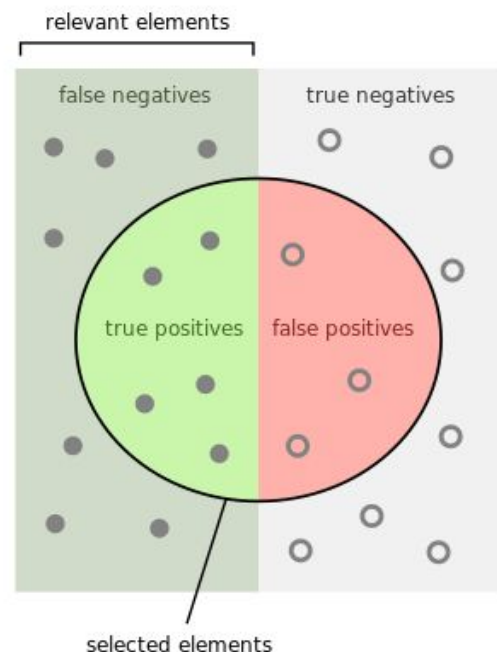
$$Recall = \frac{TP}{TP + FN}$$

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

Donde TP (True Positive) son tweets correctamente clasificados, en cambio, FP (False Positive) y FN (False Negative) son tweets clasificados de forma errónea.



En el contexto de clasificación y pensando en términos de la clasificación a una clase determinada, Precision es un indicador de cuantas instancias son correctamente clasificadas en la clase. En cambio, Recall es un indicador de cuantas instancias relevantes fueron clasificadas en dicha clase.



How many selected items are relevant?

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

How many relevant items are selected?

$$Recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

## 4.2. Evaluación

Para entrenar el modelo de clasificación se utilizó el 80 % del Corpus dejando el 20 % restante para su posterior validación.

Tener en cuenta lo siguiente en las tablas con los valores de las métricas Precision, Recall y F1-Measure:

- **Instancias** es la cantidad de instancias que pertenecen a la clase.
- **AVG/Total** es el promedio ponderado por la cantidad de instancias que existen de la clase. En términos de scikit learn: The reported averages are a prevalence-weighted macro-average across classes (equivalent to `func: `precision_recall_fscore_support` with `average='weighted'``).

#### 4.2.1. Support Vector Machine<sup>9</sup>

##### 4.2.1.1. Resultados - Linear Kernel<sup>10</sup>

**Parámetros:** C = 0.25

#### **Precision, Recall y F1-Measure**

	Precision	Recall	F1-Measure	Instancias
Ninguno	0.68	0.76	0.72	4288
Positivo	0.82	0.79	0.80	4430
Negativo	0.72	0.70	0.71	3180
Neutral	0.16	0.02	0.03	262
AVG/Total	0.73	0.74	0.73	12160

#### **Confusion Matrix**

	Real Ninguno	Real Positivo	Real Negativo	Real Neutral
Predicho Ninguno	3265	743	736	46
Predicho Positivo	466	3482	197	84
Predicho Negativo	553	194	2235	127
Predicho Neutral	4	11	12	5

---

<sup>9</sup> Generación del modelo:

<https://github.com/nietol/tesis/blob/master/twitter-learn/clasification.py>

<sup>10</sup> Evaluación del modelo:

<https://github.com/nietol/tesis/blob/master/twitter-learn/Validator.py>

#### 4.2.1.2. Resultados - RBF Kernel

**Parámetros:**  $C = 2^{1.5}$  -  $\text{Gamma} = 2^{-3.5}$

#### **Precision, Recall y F1-Measure**

	Precision	Recall	F1-Measure	Instancias
Ninguno	0.69	0.76	0.72	4288
Positivo	0.83	0.79	0.81	4430
Negativo	0.71	0.72	0.71	3180
Neutral	0.37	0.03	0.05	262
AVG/Total	0.74	0.74	0.74	12160

#### **Confusion Matrix**

	Real Ninguno	Real Positivo	Real Negativo	Real Neutral
Predicho Ninguno	3260	700	711	39
Predicho Positivo	447	3512	186	79
Predicho Negativo	579	214	2277	137
Predicho Neutral	2	4	6	7



#### 4.2.2. Simple Classifier<sup>11</sup>

Para comparar el modelo generado en términos de SVM se construyó un clasificador que determina la clase a la que pertenece un mensaje en término de las ocurrencias de palabras positivas y negativas presentes en él.

Un mensaje con mayor cantidad de palabras positivas será positivo, en sentido inverso, de tener mayor cantidad de términos negativos será considerado negativo, de tener similar número de términos negativos y positivos, será considerado neutro y de no contar con términos positivos o negativos será considerado como un mensaje que no expresa opinión.

**NOTA:** en el caso de este clasificador, no necesita entrenamiento, por lo tanto todo el Corpus se utilizó como conjunto de prueba.

##### 4.2.2.1. Resultados<sup>12</sup>

#### **Precision, Recall y F1-Measure**

	Precision	Recall	F1-Measure	Instancias
Ninguno	0.55	0.66	0.60	21416
Positivo	0.64	0.65	0.64	22233
Negativo	0.68	0.37	0.48	15844
Neutral	0.06	0.19	0.10	1305
AVG/Total	0.61	0.57	0.57	60798

---

<sup>11</sup> Implementación:

<https://github.com/nietol/tesis/blob/master/twitter-learn/SimpleClassifier.py>

<sup>12</sup> Generación de Métricas:

[https://github.com/nietol/tesis/blob/master/twitter-learn/simple\\_classifier\\_metrics.py](https://github.com/nietol/tesis/blob/master/twitter-learn/simple_classifier_metrics.py)

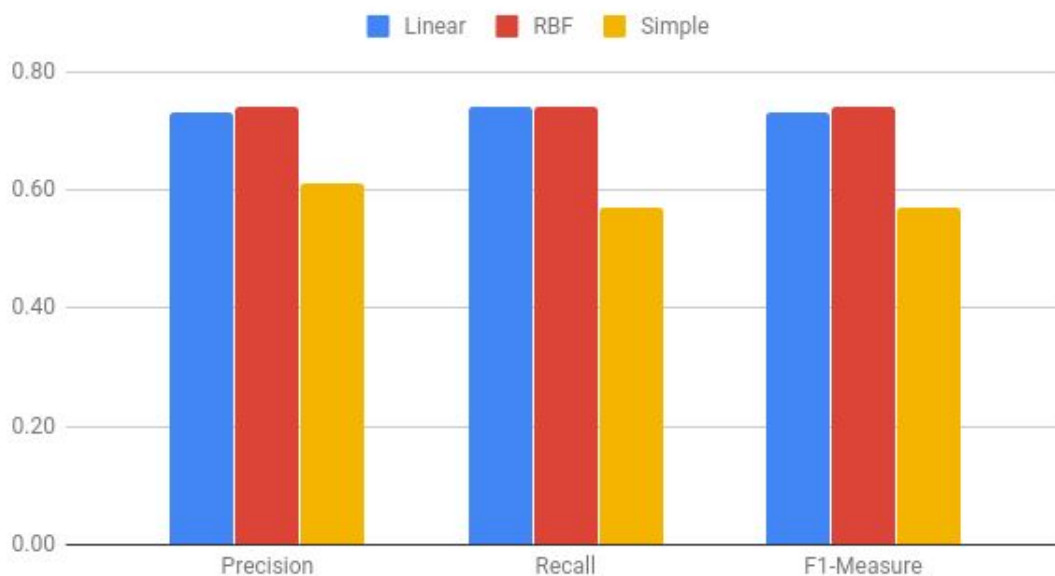
## Confusion Matrix

	Real Ninguno	Real Positivo	Real Negativo	Real Neutral
Predicho Ninguno	14161	5981	5199	241
Predicho Positivo	4609	14473	3078	525
Predicho Negativo	1802	682	5837	287
Predicho Neutral	844	1097	1730	252

### 4.2.3. Comparación entre los modelos

Las métricas de los modelos construidos con SVM no exhiben diferencias significativas. En cambio, comparando estos modelos (linear and rbf kernels) contra **Simple Classifier**, notamos mejores resultados de SVM, tanto en término de la cantidad de items relevantes vs la cantidad de items clasificados en la clase (precision) así como en la cantidad de items relevantes vs la cantidad total de items relevantes pertenecientes a la clase (recall).

#### AVG/Total



## 5. Conclusión

El gran volumen de datos que se genera en las redes sociales nos brinda un sinfín de recursos para poder trabajar con algoritmos de machine learning y transformar esos datos en información útil.

Tal volumen de datos accesible para todos junto con herramientas de machine learning (como scikit-learn, utilizada en este trabajo) ampliamente disponibles en la Web y la gran biblioteca que es Internet nos permite aprender y generar herramientas como la construida en este trabajo.

El propósito de este trabajo fue construir una herramienta que nos permita visualizar la polaridad de los tweets que se generan en tiempo real. Dicho propósito pudo ser alcanzado gracias a la disponibilidad de recursos mencionada.

Si bien el modelo de clasificación es algo que siempre puede mejorarse, se lograron buenos resultados en término de las métricas calculadas y la interfaz construida nos permite ver cómo se distribuyen los tweets según su polaridad así como visualizar el texto de los tweets procesados y el vector de características generado para su clasificación.

## Anexo

En este anexo se hace una breve explicación de conceptos y herramientas utilizadas en este trabajo. Para información detallada remitirse a la bibliografía correspondiente.

### Material de este trabajo

Todo el material relativo a esta tesis (corpus, código fuente, informes, etc) se encuentra en GitHub.

<https://github.com/nietol/tesis>

En el repositorio, archivo README.md, se describe el proceso de setup del ambiente para ejecutar los proyectos.

### SVM, Kernels y sus Parámetros

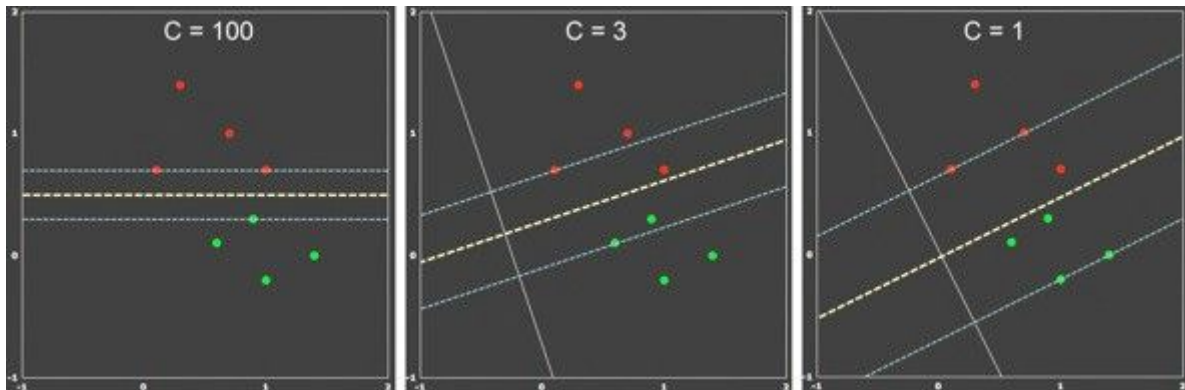
Support Vector Machine busca generar un hiperplano que divida los vectores según a la clase a la que pertenecen. Durante el entrenamiento de un clasificador utilizando este modelo buscamos encontrar el hiperplano cuya distancia al vector más cercano de cada una de las clases sea máxima, es decir, SVM busca maximizar la distancia entre el hiperplano y los vectores más cercano de cada uno de las clases. Estos son los support vectors del algoritmo.

Según el Kernel que utilicemos y las variaciones de sus parámetros obtenemos distintos hiperplanos y distintos support vectors.

#### Lineal Kernel

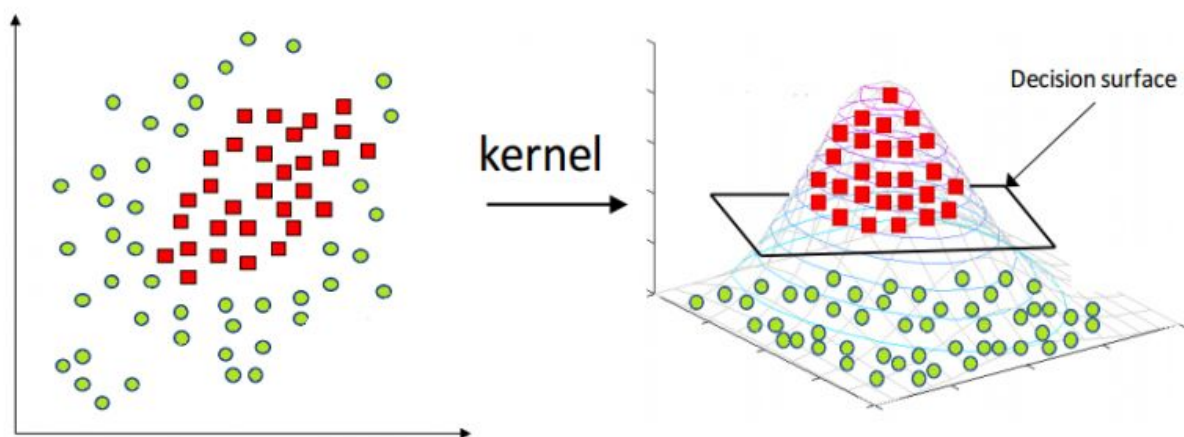
Este kernel busca separar el hiperplano de los vectores de forma lineal. Como input cuenta con el parámetro C. Intuitivamente este parámetro influencia al algoritmo en términos de si estamos dispuestos a dejar fuera outliers como support vectors en post de obtener un margen de separación

mayor al hiperplano. A valores chicos de  $C$  buscamos mayor separación al hiperplano y en valores grandes buscamos bajar la tasa de clasificación errónea a costa de que el margen de separación sea menor.



## RBF Kernel

Cuando los datos no son separables de forma lineal en el espacio en que se encuentran, es decir, no se puede establecer un hiperplano que separe los datos en sus clases, se utiliza un kernel no lineal como es el caso de Radial Basis Function. Estos kernels proyectan el espacio a dimensiones superiores donde se pueda establecer el hiperplano que separe las instancias.



Este kernel utiliza la distribución normal (Gaussian function) como función de similitud entre vectores.

## Grid Search

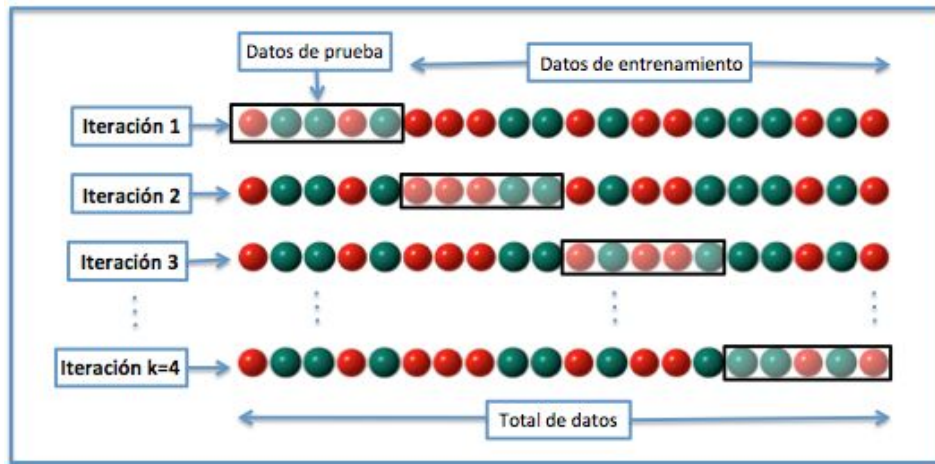
Técnica utilizada para la optimización de los parámetros que definen el modelo de clasificación. Se realiza una búsqueda exhaustiva en un rango de valores definido para los parámetros involucrados. Se evalúa los distintos modelos obtenidos empleando técnicas de cross validation o dejando un subconjunto de datos de prueba fuera del conjunto de entrenamiento, para evaluar el desempeño. De esta manera nos permite encontrar los parámetros que mejor definen nuestro modelo en término de los datos de entrenamiento y prueba.

## Cross Validation

Esta técnica busca dividir los datos de entrenamiento en conjuntos complementarios, un conjunto lo utilizamos para entrenar al clasificador y el otro para validar nuestro modelo. Se itera varias veces sobre los datos generando distintas divisiones de conjuntos y la evaluación final del modelo es el resultado del promedio de las obtenidas en cada una de las iteraciones.

## K-Fold Cross Validation

Con esta técnica dividimos los datos en  $k$  subconjuntos. Tomamos  $k-1$  como datos de entrenamiento y dejamos uno para validar el modelo. Este proceso se repite  $k$  veces con cada uno de los subconjuntos como dato de prueba y el promedio de las evaluaciones obtenidas entre las  $k$  iteraciones es el resultado de la evaluación de nuestro modelo.



# Bibliografía

scikit-learn. *Cuenta con excelente documentación no solo sobre el uso de las herramientas del framework sino también sobre métodos y algoritmos asociados a machine learning.*

<https://scikit-learn.org/>

Wikipedia. *Support vector machine.*

[https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)

Wikipedia. *Cross-validation.*

[https://en.wikipedia.org/wiki/Cross-validation\\_\(statistics\)](https://en.wikipedia.org/wiki/Cross-validation_(statistics))

Julio Villena-Román, Sara Lana-Serrano, Eugenio Martínez-Cámara, José Carlos González-Cristóbal. *TASS - Workshop on Sentiment Analysis at SEPLN*. Procesamiento del Lenguaje Natural, Revista no 50 marzo de 2013, pp 37-44.

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. *A Practical Guide to Support Vector Classification.*

Hao Wang, Jorge A. Castanon. *Sentiment Expression via Emoticons on Social Media.*

Bo Pang and Lillian Lee. *Opinion Mining and Sentiment Analysis*. Foundations and Trends in Information Retrieval Vol. 2, Nos. 1-2 (2008) 1-135.

David Vilares, Miguel A. Alonso y Carlos Gómez-Rodríguez. *Una aproximación supervisada para la minería de opiniones sobre tuits en español en base a conocimiento lingüístico*. Departamento de Computación, Universidade da Coruña Campus de Elviña, 15011 A Coruña.