

**UNIVERSIDAD ANDRÉS BELLO**

**FACULTAD DE INGENIERÍA**

**INGENIERÍA CIVIL INFORMÁTICA**

**Trabajo 2 Etiquetado – Aprendizaje Supervisado (SVM)**

**ANDRÉS EDUARDO VALENZUELA GONZÁLEZ**

**SANTIAGO - CHILE**

**SEPTIEMBRE, 2017**

Contenido

[**1.** **Introducción** 3](#_Toc493106243)

[**2.** **Descripción de la Vectorizacion** 3](#_Toc493106244)

[**1.** **Cross-Validation** 6](#_Toc493106245)

[**3.** **Desarrollo** 7](#_Toc493106246)

[3.1. Cuando se tengan los diez archivos de resultados. 7](#_Toc493106247)

[**4.** **Conclusión** 8](#_Toc493106248)

# **Introducción**

Para comprender el desarrollo de este informe, se utilizaron las dos mil etiquetas junto con sus perfiles de la tarea anterior para ser vectorizadas y luego utilizar los algoritmos de *SVM* (y otros creados por el alumnado) para lograr un aprendizaje supervisado.

Junto con las etiquetas proporcionadas (en formato *csv*) por el servidor de la *Universidad Nacional Andrés Bello*, se proporcionó además el conjunto de perfiles que fueron etiquetados por el alumnado (en formato *dat*), los cuales fueron unidos para luego generar una bolsa de palabras, de las cuales, además, se descontaron las *stopwords*.

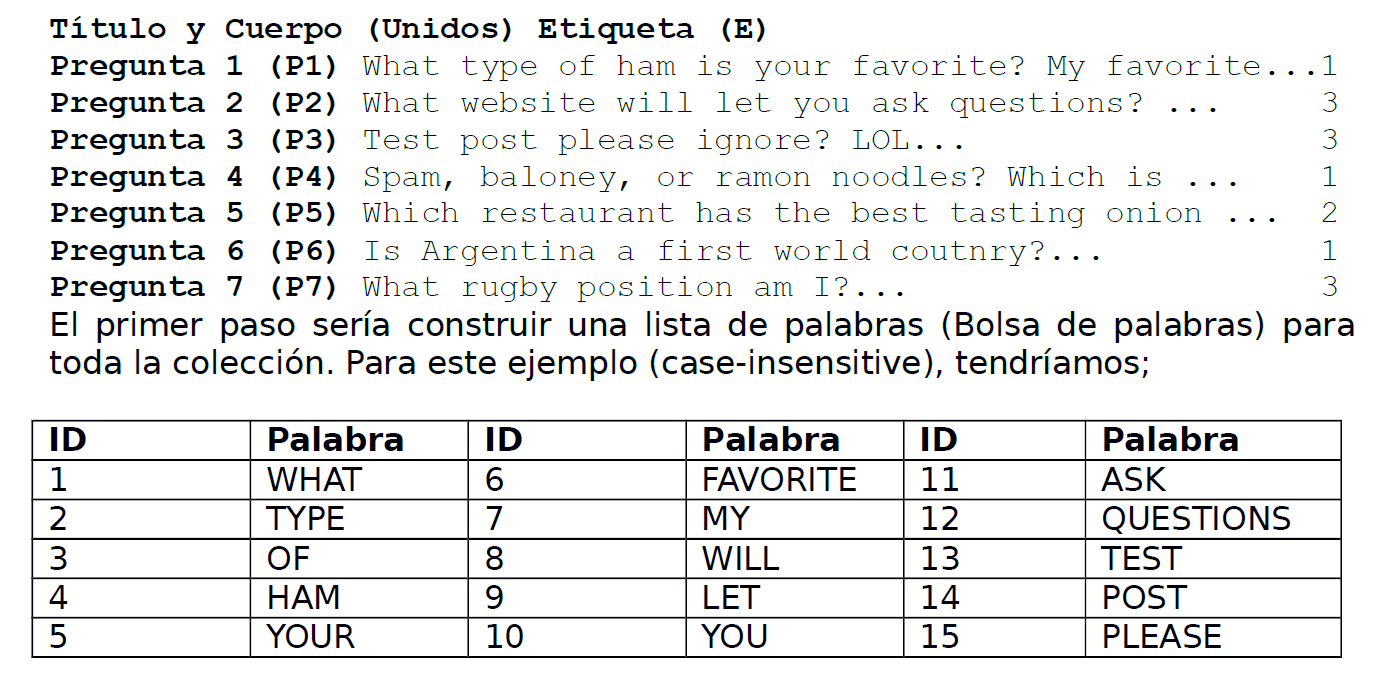
Para realizar el aprendizaje supervisado se debió modelar el problema mediante vectores.   
Si asumimos que cada una de las palabras es un atributo, entonces podríamos crear una representación vectorial que indique la presencia o ausencia de esta palabra en cada una de las descripciones del usuario etiquetadas en la primera tarea. Para la etiqueta se utilizará un valor desde 1 hasta 4:

1 = *Undetermined* 2 = *Non-USA* 3 = *World* 4 = *USA only*.

Cada etiqueta debe estar asociada con un único valor numérico. *Extracto del enunciado Tarea 2, Espacio Vectorial, 28 de agosto 2017 [consulta: 13 septiembre 2017, 20:12 hrs]. Disponible en:* <https://gitlab.com/Choapinus/SistemasInteligentes/blob/master/Tarea%202/enunciado_informe/Tarea%202.pdf>

# **Descripción de la Vectorizacion**

Para explicar el proceso de vectorizacion, se considerara el siguiente ejemplo que trabaja con preguntas *cQA* (cada perfil es una pregunta):



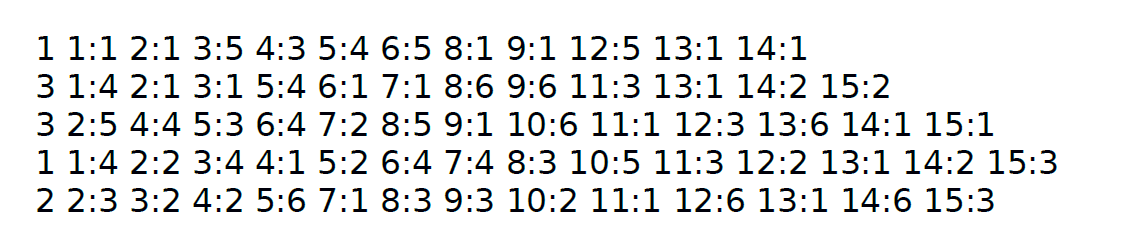
Se representaron los perfiles como vectores, indicando la frecuencia de cada palabra en la pregunta con su índice correspondiente respecto a la bolsa de palabras (Por bolsa de palabras entenderemos a las palabras ordenadas respecto al orden de llegada desde los perfiles, con las *stopwords* descontadas). Luego, se generaron los vectores con formato *SVM multiclass*, descrito a continuación:

<target> <feature>:<value> <feature>:<value> ... <feature>:<value>

En donde:

* **Target:** Valor de 1 a 4, representa la clase a la que pertenece.
* **Feature:** id de la palabra ubicada dentro de la bolsa de palabras.
* **Value:** Veces que se repite la palabra dentro del perfil vectorizado.

Aquellas palabras que no se encuentren presentes en el perfil se pueden omitir (si se quiere incluir, se iguala su atributo *value* = 0). Para el ejemplo descrito anteriormente, se tiene:

Cabe destacar que cada ID debe referirse a la misma palabra a través de todas las instancias de entrenamiento y que el formato de *SVM multiclass* exige que se escriban en orden ascendente (en términos de ID). *Extracto del enunciado Tarea 2, SVM Multiclass, 28 de agosto 2017 [consulta: 13 septiembre 2017, 20:12 hrs]. Disponible en:* <https://gitlab.com/Choapinus/SistemasInteligentes/blob/master/Tarea%202/enunciado_informe/Tarea%202.pdf>

Con base en el ejemplo anterior, el alumnado utilizo un programa de su autoría hecha en *Python* para contar las palabras, descontar las *stopwords* y generar los vectores con sus respectivas clases y sus pertenecientes palabras.

A continuación, un ejemplo de los vectores creados:

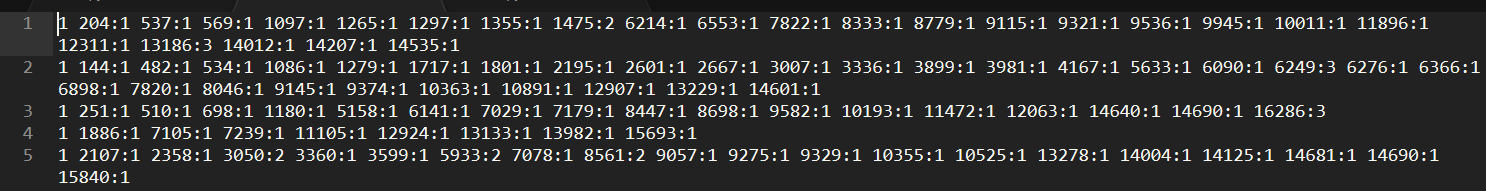


Ilustración 1.- Vectores

Se dispone acceso al repositorio en gitlab para disponer del ejemplo:

<https://gitlab.com/Choapinus/SistemasInteligentes/blob/master/Tarea%202/codigos/make_vectors/data/vectores.txt>

Y del codigo que genera los vectores con el csv del alumnado:

<https://gitlab.com/Choapinus/SistemasInteligentes/tree/master/Tarea%202/codigos/make_vectors>

Si se desea hacer uso del código, se ruega leer las instrucciones adjuntas.

# **Cross-Validation**

Una vez obtenidos los vectores, hay que dividir el set de entrenamiento en diez partes iguales (por ejemplo, si en total se tienen 2000 perfiles, cada *split* (sub-set) consta de 200 perfiles etiquetados). Luego, se debe repetir diez veces el proceso de asignar un *split* para evaluar y los restantes nueve como entrenamiento. *Extracto del enunciado Tarea 2, Cross-Validation, 28 de agosto 2017 [consulta: 13 septiembre 2017, 20:12 hrs]. Disponible en:* <https://gitlab.com/Choapinus/SistemasInteligentes/blob/master/Tarea%202/enunciado_informe/Tarea%202.pdf>

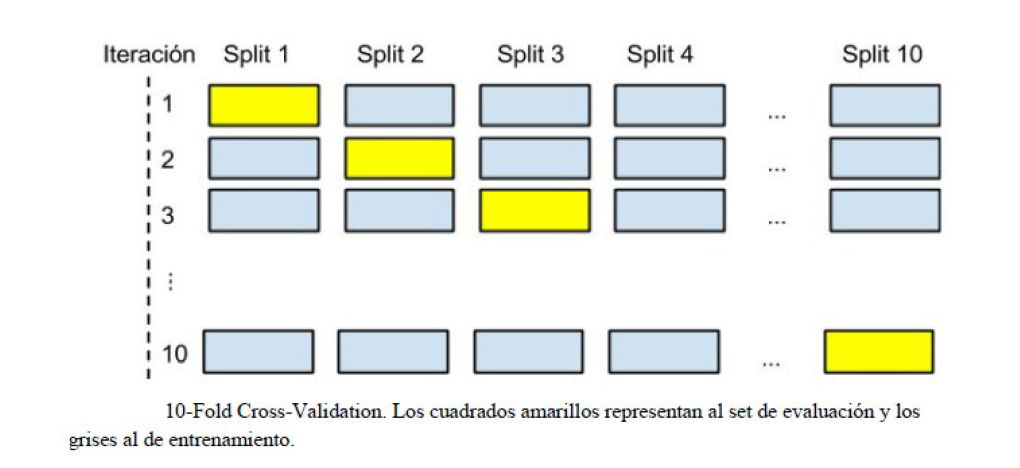


Ilustración 2.- 10-Fold Cross-Validation

Luego, se crearon los sets de entrenamientos de la siguiente manera:

E 1 = S2 + S3 + S4 + S5 + S6 + S7 + S8 + S9 + S10  
E 2 = S1 + S3 + S4 + S5 + S6 + S7 + S8 + S9 + S10  
E 3 = S1 + S2 + S4 + S5 + S6 + S7 + S8 + S9 + S10  
E 4 = S1 + S2 + S3 + S5 + S6 + S7 + S8 + S9 + S10  
E 5 = S1 + S2 + S3 + S4 + S6 + S7 + S8 + S9 + S10  
E 6 = S1 + S2 + S3 + S4 + S5 + S7 + S8 + S9 + S10  
E 7 = S1 + S2 + S3 + S4 + S5 + S6 + S8 + S9 + S10  
E 8 = S1 + S2 + S3 + S4 + S5 + S6 + S7 + S9 + S10  
E 9 = S1 + S2 + S3 + S4 + S5 + S6 + S7 + S8 + S10  
E 10 = S1 + S2 + S3 + S4 + S5 + S6 + S7 + S8 + S9

Es importante destacar que los datos de prueba **no deben ser usados para entrenar**.

Por recomendación del docente que imparte el ramo, se opto por realizar un “*random sampling Cross-Validation*” para cada split de datos con 200 muestras del archivo de vectores original. Esto debido a que permite que el entrenamiento se efectue con una mayor diversidad de perfiles y con una mayor probabilidad de que existan todas las clases dentro del split (además de entretenido , se obtuvo un entrenamiento más preciso).

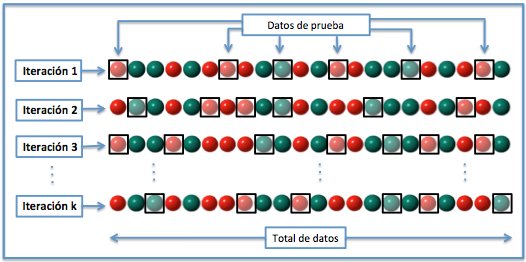


Ilustración 3.- Random Sampling Cross-Validation

Los datos de entrenamiento se construyeron de igual forma como se especifica en la formula descrita anterior al *RS Cross-Validation*.

E 1 = S 2 +S 3 +S 4 +S 5 +S 6 +S 7 +S 8 +S 9 +S 10  
E 2 = S 1 +S 3 +S 4 +S 5 +S 6 +S 7 +S 8 +S 9 +S 10  
…

Se dispone del código para su uso:

<https://gitlab.com/Choapinus/SistemasInteligentes/tree/master/Tarea%202/codigos/check_predictions>

Desde ya se advierte que por cada ejecución del código de creación de splits, se generara un split totalmente distinto a su antecesor.

# **Entrenamiento**

Una vez construidos los conjuntos de entrenamiento, se deben generar los modelos.   
Para esto, se debe utilizar el comando “*svm\_multiclass\_learn -c X Ei Mi*”, con lo que se genera un modelo (*Mi*) por cada uno de los *Ei*. *X* corresponde a un factor de penalización que utiliza el algoritmo. Para evaluar el modelo, debe ejecutarse sobre el split que no fue considerado en el respectivo *Ei*, es decir *Si*, mediante el comando “svm\_*multiclass*\_classify *Si* *Mi* *Ri*” donde *Ri* es el archivo en donde se almacenará el resultado. Por ejemplo:

**$ ./svm\_multiclass\_learn –c 5000 train.dat model.dat  
$ ./svm\_multiclass\_classify test.dat model.dat predictions.dat**

Luego, el resultado de ejecutar el comando de prueba queda en el archivo predictions.dat. *Extracto del enunciado Tarea 2, Cross-Validation, 28 de agosto 2017 [consulta: 13 septiembre 2017, 20:12 hrs]. Disponible en:*

<https://gitlab.com/Choapinus/SistemasInteligentes/blob/master/Tarea%202/enunciado_informe/Tarea%202.pdf>

Para fines demostrativos, se ejecutaron los siguientes comandos:

**$ svm\_multiclass\_learn.exe -c 5000 E\_train/E\_01.dat models/modelo\_01.dat  
$ svm\_multiclass\_learn.exe -c 5000 E\_train/E\_02.dat models/modelo\_02.dat  
$ svm\_multiclass\_learn.exe -c 5000 E\_train/E\_03.dat models/modelo\_03.dat**

**…**

Luego de obtener los modelos, se obtuvieron las predicciones con los siguientes comandos:

**$ svm\_multiclass\_classify.exe S\_01.dat modelo\_01.dat prediction\_01.dat  
$ svm\_multiclass\_classify.exe S\_02.dat modelo\_02.dat prediction\_02.dat  
$ svm\_multiclass\_classify.exe S\_03.dat modelo\_03.dat prediction\_03.dat**

**…**

Resultando en lo siguiente:

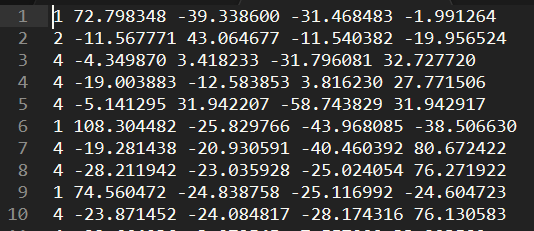


Ilustración 4.- prediction\_01.dat

# **Desarrollo**

## Cuando se tengan los diez archivos de resultados, se debe calcular el desempeño del clasificador SVM para predecir la ubicación geográfica.

Luego de entrenar y clasificar, se obtuvieron los siguientes resultados:

|  |  |  |
| --- | --- | --- |
| ***Clasificador XX*** | ***Accuracy*** | ***Average Loss*** |
| Clasificador 01 | 86.5 % | 13.5 % |
| Clasificador 02 | 82 % | 18 % |
| Clasificador 03 | 81.5 % | 18.5 % |
| Clasificador 04 | 83.5 % | 16.5 % |
| Clasificador 05 | 81 % | 19 % |
| Clasificador 06 | 83.5 % | 16.5 % |
| Clasificador 07 | 84 % | 16 % |
| Clasificador 08 | 83 % | 17 % |
| Clasificador 09 | 82 % | 18 % |
| Clasificador 10 | 82 % | 18 % |

## Calcule el MRR y la matriz de confusión. Entregue también los promedios de los diez splits.

En la siguiente tabla se exponen los MRR de cada clasificador:

|  |  |
| --- | --- |
| *Clasificador XX* | *MRR* |
| Clasificador 01 | **0.9225** |
| Clasificador 02 | **0.890833333333** |
| Clasificador 03 | **0.895** |
| Clasificador 04 | **0.90125** |
| Clasificador 05 | **0.892083333333** |
| Clasificador 06 | **0.904166666667** |
| Clasificador 07 | **0.90875** |
| Clasificador 08 | **0.90625** |
| Clasificador 09 | **0.89375** |
| Clasificador 10 | **0.89875** |

*MRR* promedio: 0.901333333333

## Comente acerca de los errores ¿Hay algún patrón?, ¿Cómo podría mejorar?, ¿Qué puede decir de los errores?, ¿Con respecto a las ubicaciones geográficas (strings) nota algún patrón en los errores?

Los errores que comete la *SVM* dependen bastante de la magnitud del entrenamiento que reciba y de la calidad de entrenamiento, es decir, si recibe un set de entrenamiento que contiene solo dos clases, sus predicciones pueden ser inadecuadas para un set de prueba que contenga las cuatro clases. Tambien se sospecha de cercanias en cuanto a vectores con distintas clases asignadas (vectores similares pero catalogados con distintas clases).

La clase *USA only* fue la que “peor” (por peor se entiende que proviene del clasificador que peor clasifico -> Clasificador 05) predicciones obtuvo dentro de las 10 pruebas realizadas. Esto podría deberse a la gran cantidad de perfiles que solo especificaron su ubicación geográfica, por lo cual se entiende que su perfil atiende a ser un vector más pequeño y por lo tanto tendera a ser poco significativo comparado con un vector más sustancioso en su descripción de perfil, como también cabe la posibilidad de que se utilizaran palabras tan comunes que dentro del hiperplano se acerque demasiado a las demás clases, por lo tanto, es mal catalogado por la *SVM*.

La *SVM* podría mejorar su índice de acierto si:

1. Se crea un ranking de las palabras mas usadas y descartar las menos usadas.
2. Se descartan los perfiles con una cantidad de palabras inferior a 5 (factor propuesto al azar) y/o perfiles ambiguos.
3. Se entrena la *SVM* con un mayor espectro de datos de pruebas (1800 vectores de entrenamiento y 200 de prueba son poquísimos, algoritmos como *HyperGAN*, *CBLSTM* y *PixelCNN* son entrenados con miles de datos y sus resultados son prometedores, es decir, a mayor iteración de entrenamiento, mayor será el performance del modelo predictorio (aunque no se debería comparar redes neuronales con un clasificador *SVM*)). De todas maneras, un porcentaje de acierto del 86.5% es bastante aceptable.
4. Minimizar el error humano. Puede que la *SVM* este en lo cierto con sus predicciones …
5. Disminuir el espectro de clases (incompatible con nuestra problemática)
6. Apoyar a la *SVM* con algún otro clasificador (clasificador Bayesiano de textos)
7. Apoyar con generadores y ejecutores, los cuales vayan generando distintos sets de entrenamiento y los segundos aplicándolos a la *SVM* para luego ser comparados y verificar que set de datos obtuvo un mejor rendimiento.

## Calcule la curva ROC, Curva Lift, F(1)-score y el estimador AUC asumiendo cada una de las variables como positivas.