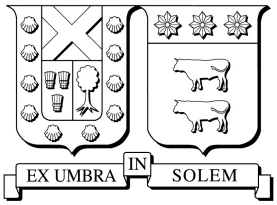



# INF-393. Machine Learning. Tarea 2.
## Métodos para Clasificación

Prof. Ricardo Ñanculef
*jnancu@inf.utfsm.cl*

## Temas

- Implementación y evaluación de clasificadores en *sklearn*.
- Tipos de fronteras de clasificación.
- LDA versus QDA.
- PCA versus LDA. Reducción de dimensionalidad para clasificación.
- Selección de híper-parámetros estructurales en Regresión Logística, SVM, Arboles de Decisióny y $k$-NN.
- Métodos de *Kernel*
- Clasificadores bayesianos ingenuos (Bernoulli, Multinomial).
- Preprocesamiento de datos brutos de entrada.
- Distintas representaciones para datos de entrada.

## Formalidades

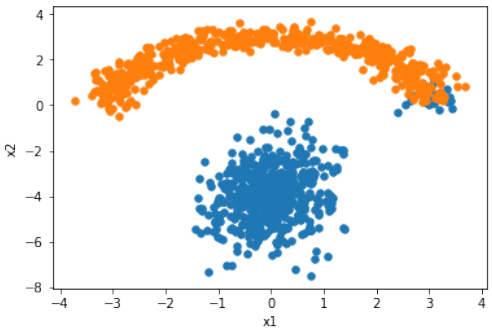
- Equipos de trabajo de: 2 personas.[*]
- Se debe preparar un (breve) Jupyter/IPython notebook que explique la actividad realizada y las conclusiones del trabajo.
- Se debe preparar una presentación de 20 minutos.
- Se debe mantener un respaldo de cualquier tipo de código utilizado, informe y presentación en Github.
- Deadline: 17 Noviembre.
- Formato de entrega: envío de link Github al correo electrónico del ayudante [**] incluyendo a todos los profesores en copia y especificando asunto: [TallerXXX-CursoXXX-Semestre-Año]

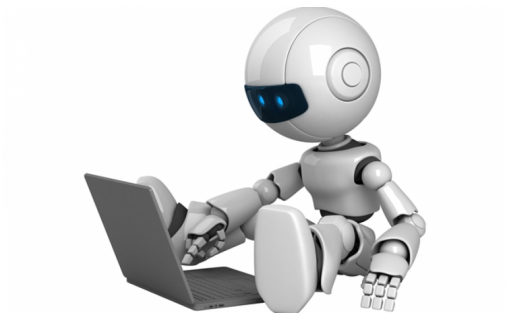
[*] La modalidad de trabajo en equipo nos parece importante, porque en base a nuestra experiencia enriquece significativamente la experiencia de aprendizaje. Sin embargo, esperamos que esto no se transforme en una cruda divisi?n de tareas. Cada miembro del equipo debe estar en condiciones de realizar una presentación y discutir sobre cada punto del trabajo realizado.

[**] francisco.mena.13@sansano.usm.cl

# 1. Tipos de fronteras en Clasificación

Como se ha discutido en clases, los problemas de clasificación pueden ser representados de distintas maneras, en donde estas representaciones definen un espacio de entrada del dominio de los datos ($\mathbb{R}_X$). Los ejemplos dentro del espacio de entrada (*manifold*) pueden tener distintas formas, donde estas indicarán si es que estos ejemplos tendrán fronteras lineales o no. Con el propósito de analizar los distintos tipos de fronteras que definen los algoritmos de clasificación, se trabajará con un *dataset* sintético lo cual es ideal para analizar los diferentes tipos.



Éste *dataset* está en un espacio de 2 dimensiones y es conformado por dos conjuntos de datos, pudiendo ver que la frontera entre ellos claramente no es lineal. Uno de los conjuntos de datos es ovalado, generado a través de una distribución multivariada gaussiana, el otro conjunto de datos es una semi-luna, generado a través de funciones senos y cosenos. Se agrega ruido en los conjuntos para que no sea un problema trivial. El código que los genera es el siguiente:

```
n_samples=500
mean = (0,-4)
C = np.array([[0.3, 0.1], [0.1, 1.5]])
datos1 = np.random.multivariate_normal(mean, C, n_samples)
outer_circ_x = np.cos(np.linspace(0, np.pi, n_samples))*3
outer_circ_y = np.sin(np.linspace(0, np.pi, n_samples))*3
datos2 = np.vstack((outer_circ_x,outer_circ_y)).T
from sklearn.utils import check_random_state
generator = check_random_state(10)
datos2 += generator.normal(scale=0.3, size=datos2.shape)
```

(a) Construya el conjunto de datos (*dataset*) común con los dos conjuntos generados. Luego se realiza un *shift* desde el conjunto 2 al 1, esto se puede ver en la imagen anterior, donde el conjunto de color naranjo (media luna) tiene puntos azules a la derecha pertenecientes al otro conjunto, esto es con el mismo propósito de trabajar con un *dataset* no ideal. Determine cuántos registros contiene cada conjunto y visualícelos.

```
import numpy as np
X = np.concatenate((datos1, datos2), axis=0)
n = 20 #ruido/noise
y1 = np.zeros(datos1.shape[0]+n)
y2 = np.ones(datos2.shape[0]-n)
y =  np.concatenate((y1,y2),axis=0)
```

Para visualizar las fronteras de los distintos algoritmos clasificadores utilice el siguiente código:

```
import matplotlib.pyplot as plt
def visualize_border(model,x,y,title=""):
    fig = plt.figure(figsize=(12,6))
```

```
    plt.scatter(x[:,0], x[:,1], s=50, c=y, cmap=plt.cm.winter)
    h = .02  # step size in the mesh
    x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),np.arange(y_min, y_max, h))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contour(xx, yy, Z, cmap=plt.cm.Paired)
    plt.title(title)
    plt.show()
```

(b) Entrene el clasificador *Linear Discriminant Analysis* (LDA) y visualice la frontera de decisión que define este algoritmo. Analice cualitativamente lo que observa.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
model = LDA()
model.fit(X,y)
visualize_border(model,X,y,"LDA")
```

(c) Entrene el clasificador *Quadratic Discriminant Analysis* (QDA) y visualice la frontera de decisión que define este algoritmo. Analice cualitativamente lo que observa y compare con LDA, en qué difieren y en qué se asemejan ¿Qué distribución de probabilidad asumen cada uno?

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
model = QDA()
model.fit(X,y)
```

(d) Compare cuantitativamente los clasificadores LDA Y QDA en este *dataset* sintético mediante la métrica de error de clasificación.

```
from sklearn.metrics import accuracy_score
print("Miss Classification Loss: %f"%(1-accuracy_score(y_true, y_pred)))
```

Para lo que sigue de la actividad se trabajará con una *widget* interactiva [2] para sintonizar los parámetros de regularización de los distintos algoritmos. Por ello cada vez que se modifique el parámetro en la ventana se entrenará el modelo y se visualizará la frontera en la misma imagen.

```
from ipywidgets import interactive
def visualize_border_interactive(param):
    model = train_model(param)
    visualize_border(model,X,y)
```

(e) Construya una función que entrene/ajuste un modelo de Regresión Logística Regularizado (utilizando como penalizador la norma $l_2$), experimente con distintos valores del parámetro de regularización mediante el gráfico interactivo. Explique el significado y efecto esperado de este parámetro. Analice cualitativamente lo observado.

```
from sklearn.linear_model import LogisticRegression as LR
def train_model(param):
    model=LR() #define your model
    model.set_params(C=param,penalty='l2')
    model.fit(X,y)
    return model
p_min = #define your range
p_max = #define your range
interactive(visualize_border_interactive,param=(p_min,p_max))
```

(f) Construya una función que entrene/ajuste una Máquina de Vectores de Soporte (SVM) Lineal. Mediante la imagen interactiva explore diferentes valores del parámetro de regularización C. Discuta el significado y efecto esperado de este parámetro. Analice cualitativamente lo observado.

```
from sklearn.svm import SVC as SVM #SVC is for classification
def train_model(param):
    model= SVM()
    model.set_params(C=param,kernel='linear')
    model.fit(X,y)
    return model
#use interactive
```

(g) Construya una función que entrene/ajuste una Máquina de Vectores de Soporte (SVM) no Lineal. Mediante la imagen interactiva explore diferentes valores del parámetro de regularización $C$ y con diferentes *kernels*. Discuta el significado y efecto esperado de este parámetro. Analice cualitativamente lo observado.

```
#edit the train_model function
model.set_params(C=param,kernel='rbf') #try poly
```

(h) Construya un Árbol de Decisión de múltiples niveles para la clasificación del problema. Puede utilizar el criterio y la función de partición que prefiera. Mediante la imagen interactiva explore diferentes valores del parámetro de máxima profunidad del árbol. Discuta el significado y efecto esperado de este parámetro. Analice cualitativamente lo observado.

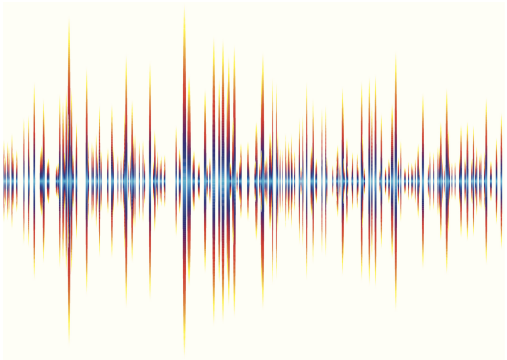
```
from sklearn.tree import DecisionTreeClassifier as Tree
model = Tree() #edit the train_model function
model.set_params(max_depth=param,criterion='gini',splitter='best')
```

(i) Construya un algoritmo $k$-$NN$ para la clasificación del problema. Mediante la imagen interactiva explore diferentes valores del parámetro $k$. Discuta el significado y efecto esperado de este parámetro. Analice cualitativamente lo observado.

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.set_params(n_neighbors=param)
```

## 2. Análisis de audios como datos brutos

Distintos tipos de datos han sido tratados en el área de Machine Learning, donde el análisis de estos y el manejo para poder dejarlos en una representación que se pueda entregar como entrada al algoritmo es crucial. El manejo sobre los datos brutos se denomina pre-procesamiento y existen distintos dependiento del tipo de datos y los distintos diminios de problemas, tales como imágenes, audios, texto.
En esta actividad se trabajará con datos de audios los cuales son directamente extraídos desde datos fuentes *.wav*, lo que corresponde a una señal de sonido en diferentes tiempos.



El *dataset* se denomina *Heartbeat Sounds* [3] y es presentado en la plataforma Kaggle a través del siguiente **link**. Este *dataset* consta de grabaciones de sonidos de latidos cardíacos normales y anormales, con distintas categorías para los latidos anormales.
Para la tarea se trabajará con el **dataset A** presente en la data, el cual corresponde a datos generados desde la vía pública mediante la aplicación de Iphone iStethoscope Pro. El objetivo será el de clasificar cada sonido como latido cardíaco normal o una de las las subcategorías de anormal (*Murmur, Extra Heart Sound, Artifact*), por lo que se trata de un problema de clasificación múltiple con 4 clases. Las distintas clasificaciones para los sonidos son explicadas en el sitio de Kaggle.

Para leer y trabajar los archivos de extensión *.wav* se utilizará el siguiente código:

```
from scipy.io import wavfile
def clean_filename(fname, string):
    file_name = fname.split('/')[1]
    if file_name[:2] == '__':
        file_name = string + file_name
    return file_name
SAMPLE_RATE = 44100
def load_wav_file(name, path):
    s, b = wavfile.read(path + name)
    assert s == SAMPLE_RATE
    return b
```

(a) Construya un dataframe con los datos a analizar. Describa el *dataset* y determine cuántos registros hay por clase.

```
import pandas as pd
import numpy as np
df = pd.read_csv('./heartbeat-sounds/set_a.csv')
```

(b) Lea los archivos *.wav* y transformelos en secuencias de tiempo. Realice un *padding* de ceros al final de cada secuencia para que todas queden representadas con la misma cantidad de elementos, explique la importancia de realizar este paso.

```
def padd_zeros(array,length):
    aux = np.zeros(length)
    aux[:array.shape[0]] = array
    return aux
new_df =pd.DataFrame({'file_name' : df['fname'].apply(clean_filename,string='Aunlabelled')})
new_df['time_series'] = new_df['file_name'].apply(load_wav_file, path='path/to/set_a/')
new_df['len_series'] = new_df['time_series'].apply(len)
new_df['time_series']=new_df['time_series'].apply(padd_zeros,length=max(new_df['len_series']))
```

(c) Manipule los datos y cambie las etiquetas de los audios por otras asignadas por un doctor experto [4], el cual afirma que estos cambios son requeridos. Vuelva a determinar cuántos registros hay por clase. Nótese que ahora son 3 clases ¿Explique la problemática de tener etiquetas mal asignadas en los datos? ¿Un solo dato puede afectar esto?

```
new_labels =[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
             1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2,
             2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1,
             1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 1, 0,
             2, 2, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 2, 1, 0, 1, 1, 1, 1, 1, 2, 0, 0, 0,
             0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
labels = ['artifact','normal/extrahls', 'murmur']
new_df['target'] = [labels[i] for i in new_labels]
```

(d) Codifique las distintas clases a valores numéricos para que puedan ser trabajados por los algoritmos clasificadores.

```
new_df["target"] = new_df["target"].astype('category')
cat_columns = new_df.select_dtypes(['category']).columns
new_df[cat_columns] = new_df[cat_columns].apply(lambda x: x.cat.codes)
```

(e) Desordene los datos, evitando así el orden en el que vienen la gran mayoría de las etiquetas. Cree la matriz que conforma a los datos en sus dimensiones sin preprocesar, es decir, cada ejemplo es una secuencia de amplitudes en el tiempo. ¿Las dimensiones de ésta indica que puede generar problemas? ¿De qué tipo?

```
new_df = new_df.sample(frac=1,random_state=44)
X = np.stack(new_df['time_series'].values, axis=0)
y = new_df.target.values
X.shape
```

(f) Para pre procesar la secuencia en el tiempo realice una transformada de fourier discreta [5] para pasar los datos desde el dominio de tiempos al dominio de frecuencias presentes en la señal de sonido.

```
X_fourier = np.abs(np.fft.fft(X))
```

(g) Para seguir con el pre procesamiento realice un muestreo representativo de los datos a través de una técnica de muestreo especializada en secuencias ¿En qué beneficia este paso? ¿Cómo podría determinar si el muestro es representativo?

```
from scipy import signal
X_resampled = []
for i in range(X_fourier.shape[0]):
    sequence = X_fourier[i,:].copy()
    resampled_sequence = signal.resample(sequence, 100000)
    X_resampled.append(resampled_sequence)
X_resampled = np.array(X_resampled)
X_resampled.shape
```

(h) Genere un conjunto de pruebas mediante la técnica *hold-out validation* para verificar la calidad de los clasificadores. ¿Cuántas clases tiene y de qué tamaño queda cada conjunto?

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y,
                                                    test_size=0.25, random_state=42)
```

(i) Realice un proceso de estandarizar los datos para ser trabajados adecuadamente. Recuerde que solo se debe ajustar (calcular media y desviación estándar) con el conjunto de entrenamiento.

```
from sklearn.preprocessing import StandardScaler
std = StandardScaler(with_mean=True, with_std=True)
std.fit(X_train)
X_train = std.transform(X_train)
X_test = std.transform(X_test)
```

(j) Realice una reducción de dimensionalidad a través de la técnica PCA, para representar los datos en $d = 2$ dimensiones. Recuerde que solo se debe ajustar (encontrar las componentes principales) con el conjunto de entrenamiento. Visualice apropiadamente la proyección en 2 dimensiones.

```
from sklearn.decomposition import PCA
d=2
pca_model = PCA(n_components=d)
pca_model.fit(X_train)
X_pca_train = pca_model.transform(X_train)
X_pca_test = pca_model.transform(X_test)
```

(k) Entrene un modelo de Regresión Logística variando el parámetro de regularizacion $C$ construyendo un gráfico resumen del error en función de este hiper-parámetro. Además entrene una Máquina de Soporte Vectorial (SVM) con kernel lineal, variando el hiper-parámetro de regularizacion $C$ en el mismo rango que para la Regresión Logística, construyendo el mismo gráfico resumen. Compare.

```
Cs = [0.0001,0.01,0.1,1,10,100,1000]
```

(l) Entrene un Árbol de Decisión, con la configuración que estime conveniente, variando el hiper-parámetro regularizador *max depth*, construyendo un gráfico resumen del error en función de este parámetro. Compare con los modelos anteriores.

```
Depths = range(1,30)
```

(m) Experimente con diferentes dimensiones $d$ para la proyección de PCA con el propósito de obtener un modelo con menor error. Construya una tabla o gráfico resumen.

(n) Realice otra reducción de dimensionalidad ahora a través de la técnica LDA, para representar los datos en $d = 2$ dimensiones. Recuerde que sólo se debe ajustar con el conjunto de entrenamiento, si se muestra un *warning* explique el porqué. Visualice apropiadamente la proyección en 2 dimensiones.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
model_lda = LDA(n_components=2)
model_lda.fit(X_train,y_train)
X_pca_train = model_lda.transform(X_train)
X_pca_test = model_lda.transform(X_test)
```

(o) Con el propósito de encontrar el mejor modelo vuelva a realizar el item h) con el i) en el nuevo espacio generado por la representación según las $d$ dimensiones de la proyección LDA. Esta nueva representación ¿mejora o empeora el desempeño? Explique.

(p) Intente mejorar el desempeño de los algoritmos ya entrenados. Diseñe ahora sus propias cracterísticas (*feature crafting*) a partir de los datos brutos (secuencia de amplitudes), puede inspirarse en otros trabajos [6] [7] si desea.

## 3. Análisis de emociones en tweets

El análisis de emociones o sentimientos se refiere al proceso de extraer información acerca de la actitud que una persona (o grupo de ellas) manifiesta, en un determinado medio o formato digital, con respecto a un tópico o contexto de comunicación. Uno de los casos más estudiados corresponde a determinar la *polaridad* de un trozo de texto, es decir, clasificar una determinada evaluación escrita (*review*), en que una persona manifiesta una opinión, como *positiva*, *negativa* o *neutral*. Esto también ha sido extendido a otros medios, como lo es analizar la polaridad de textos en redes sociales.
La conocida red social Twitter tiene una gran cantidad de usuarios, por lo que la información se genera a cada segundo, donde el análisis de texto se ha aplicado fuertemente a estos medios sociales. La dificultad de este problema radica en el carácter altamente ambiguo e informal del lenguaje que utilizan naturalmente las personas así como el manejo de negaciones, sarcasmo y abreviaciones en una frase.


Para esta actividad se trabajará con un *datasets* de *tweets* ofrecidos por CrowdFlower [8]. Cada tweet está asociado a una emoción en particular, donde el conjunto de emociones se trabajarán como **mutuamente excluyentes**, siendo un problema de múltiples clases.
Los datos pueden ser descargados ejecutando el siguiente código en sistema Unix:

```
wget https://www.crowdflower.com/wp-content/uploads/2016/07/text_emotion.csv
```

Para aumentar la eficacia de las características extraídas es conveniente ejecutar algunas técnicas de pre-procesamiento básicas como: pasar todo el texto a minúsculas (*lower-casing*), eliminar signos de puntuación y eliminar palabras sin significado como artículos, pronombres y preposiciones (*stop word removal* [9]). Otra técnica que suele ser útil para obtener buenas características (features) es la lematización [11], es decir la reducción de todas las palabras a su tronco léxico base. Una técnica similar y más utilizada en la práctica es el *stemming* [10]. Varias de éstas están implementadas en la libreria nltk [13] para python.

(a) Construya un dataframe con los datos a analizar. Determine cuántas clases existen, cuántos registros por clase y describa el *dataset*.

```
import pandas as pd
df = pd.read_csv('./emotionanalysis/text_emotion.csv')
```

(b) Construya un conjunto de entrenamiento y otro de pruebas, a través de una máscara aleatoria, para verificar los resultados de los algoritmos.

```
import numpy as np
msk = np.random.rand(len(df)) < 0.8
df_train = df[msk]
df_test = df[~msk]
```

(c) Implemente y explique un pre-procesamiento para los *tweets* para dejarlos en un formato estándarizado en el cual se podrán trabajar.

(d) Haga una reducción binaria al problema, para trabajarlo como un problema de clasificación de dos clases. Para esto, agrupe las distintas emociones como positivas y negativas (defina un criterio), se recomienda codificar las clases como $+1$ y $-1$ respectivamente. Recuerde tener presente que el desbalanceo de los datos puede afectar considerablemente al modelo.

(e) Para construir un clasificador que determine automáticamente la polaridad de un trozo de texto, será necesario representar los tweets $\{t_i\}_{i=1}^n$ disponibles como vectores de características (*features*). El tipo de características más utilizado consiste en contar cuántas veces aparecen ciertos términos/palabras en el texto. Para esto, es necesario un *vocabulario* que, por lo general, se construye mediante la unión de todas las palabras que se observen en los tweets.

Se recomienda utilizar las librerías ofrecidas por *sklearn* de *feature extraction in text* [12] (*CountVectorizer* y *TfidfVectorizer*). Recuerde realizar el ajuste (*fit*) únicamente con el conjunto de entrenamiento, para luego transformar el conjunto de pruebas (con el método *transform*).

(f) Entrene y compare al menos 5 de los diferentes clasificadores vistos en clases para clasificación binaria (por ejemplo: Navie Bayes, Multinomial Naive Bayes, LDA, QDA, Regresión logística, SVM y Arboles de decisión) sobre el conjunto de entrenamiento verificando su desempeño sobre ambos conjuntos (entrenamiento y de pruebas), construyendo un gráfico resumen del error de éstos.

(g) Utilice y explique las métricas que calcula la función *classification report* de la librería sklearn. En base a las distintas métricas calculadas ¿Cuáles clasificadores son los que mejor se comportan?

```
from sklearn.metrics import classification_report
def score_the_model(model,x,y,xt,yt):
  acc_tr = model.score(x,y)
  acc_test = model.score(xt[:-1],yt[:-1])
  print "Training Accuracy: %f"%(acc_tr)
  print "Test Accuracy: %f"%(acc_test)
  print "Detailed Analysis Testing Results ..."
  print(classification_report(y_test, model.predict(X_test), target_names=['+','-']))
```

(h) [Opcional] Visualice las predicciones de algún modelo generativo (probabilístico) definido anteriormente, tomando un subconjunto aleatorio de *tweets* de pruebas y explorando las probabilidades que asigna el clasificador a cada clase.

```
test_pred = model.predict_proba(X_test)
spl = random.sample(xrange(len(test_pred)), 15)
for text, sentiment in zip(df_test.content[spl], test_pred[spl]):
        print sentiment, text
```

(i) Ahora deberá extender el problema a las múltiples clases que tiene presente (las distintas emociones), es decir, su trabajo será el de predecir una de las distintas emociones de cada *tweet*. Para esto utilice el mismo pre-procesamiento realizado en el punto c) y las características generadas mediante las técnicas en el punto e). Recuerde que tendrá que codificar las distintas clases como valores numéricos enteros.

(j) Utilice los clasificadores que son extendidos por defecto a múltiples clases para detectar emociones en cada *tweet*, muestre sus desempeños a través del error de pruebas en un gráfico resumen.

(k) Utilice clasificadores binarios que pueden ser extendidos a través de otras técnicas, tal como One vs One y One vs All/Rest [14]

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.multiclass import OneVsOneClassifier
#example
classif = OneVsRestClassifier(model)
classif.fit(X, Y)
```

(l) Para el caso de la Regresión Logística compare sus dos métodos para ser extendidos a múltiples clases. Uno a través de One vs Rest y otro definiendo que la variable a predecir se distribuye Multinomial.

```
LogisticRegression(multi_class='ovr')
LogisticRegression(multi_class='multinomial')
```

(m) Compare los resultados entre los clasificadores extendidos por defecto y los binarios que son extendidos mediante otras técnicas, construya una tabla o gráfico resumen. Los clasificadores que mejor se comportan en el caso binario ¿Siguen teniendo ese desempeño en múltiples clases?

# Referencias


[1] Hastie, T.; Tibshirani, R., Friedman, J. (2009), The Elements of Statistical Learning, Second Edition. Springer New York Inc.

[2] http://ipywidgets.readthedocs.io/en/stable/examples/Using%20Interact.html

[3] Bentley, P. and Nordehn, G. and Coimbra, M. and Mannor (2011) , Classifying Heart Sounds Challenge, *CHSC2011*, http://www.peterjbentley.com/heartchallenge/index.html

[4] https://www.kaggle.com/toregil/new-labels-for-set-a

[5] https://en.wikipedia.org/wiki/Fourier_transform

[6] https://www.kaggle.com/primaryobjects/voicegender/data

[7] Gamit, M. R., Dhameliya, P. K., & Bhatt, N. S. (2015). Classification Techniques for Speech Recognition: A Review. vol, 5, 58-63.

[8] https://www.crowdflower.com/data-for-everyone/

[9] https://en.wikipedia.org/wiki/Stop_words

[10] https://en.wikipedia.org/wiki/Stemming

[11] https://en.wikipedia.org/wiki/Lemmatisation

[12] http://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction.text

[13] http://www.nltk.org/

[14] http://scikit-learn.org/stable/modules/classes.html#module-sklearn.multiclass


LaTeX