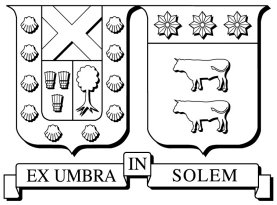

INF-493. Machine Learning. Tarea 2.
Métodos Lineales para Clasificación

Prof. Ricardo Ñanculef & Carlos Valle
jnancu@inf.utfsm.cl

Temas

- Implementación y evaluación de clasificadores en *sklearn*.
- Clasificadores bayesianos ingenuos (Bernoulli, Multinomial).
- LDA versus QDA. Reducción de dimensionalidad para clasificación.
- Regresión Logística. Selección de híper-parámetros estructurales.
- SVMs Lineales. Selección de híper-parámetros estructurales.
- Clasificadores k-NN. Selección de híper-parámetros estructurales.
- Representación vectorial de texto.
- Procesamiento de lenguaje natural.

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: 2 semanas desde publicación.
- 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].

*Cada miembro del equipo debe estar en condiciones de realizar una presentación y discutir sobre cada punto del trabajo realizado

1 Reducción de Dimensionalidad para Clasificación

Como hemos discutido en clases, la reducción de dimensionalidad es extremadamente importante en análisis de datos, no sólo porque nos permite visualizar y por lo tanto explorar los datos a nuestra disposición, y no sólo porque permite reducir el costo computacional de procesarlos, sino porque reduce de modo significativo el riesgo de *overfitting*. En problemas de clasificación, la técnica que hemos discutido en el capítulo 1 (PCA), puede exhibir resultados diametralmente opuestos a aquello que se busca: preservar la información correspondiente a la clase a la que pertenece un ítem.

Para experimentar con este problema, en esta sección trabajaremos con una colección de sonidos fonéticos que deben ser identificados con vocales del inglés británico. Los datos han sido representados en un espacio de $d = 10$ características. Existen 528 datos de entrenamiento y 462 de pruebas, que pueden ser descargados desde el sitio web mantenido por los autores de nuestro texto guía. El mejor desempeño reportado por los autores corresponde a un 56% de accuracy, y es alcanzado por un modelo de vecinos más cercanos y una red neuronal artificial de radio basal.

(a) Construya un dataframe con los datos a analizar descargando los datos desde la URL [3]. Determine cuántos registros contiene el conjunto de entrenamiento y cuántos el conjunto de pruebas. Determine además el número promedio de palabras por ítem en cada clase.

```
import urllib
import pandas as pd
train_data_url = "http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/vowel.train"
test_data_url = "http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/vowel.test"
train_data_f = urllib.urlretrieve(train_data_url, "train_data.csv")
test_data_f = urllib.urlretrieve(test_data_url, "test_data.csv")
train_df = pd.DataFrame.from_csv('train_data.csv',header=0,index_col=0)
test_df = pd.DataFrame.from_csv('test_data.csv',header=0,index_col=0)
train_df.head()
test_df.tail()
```

(b) Construya matrices X e y que contengan las características y las etiquetas correspondientes a los datos de entrenamiento y pruebas. Normalice apropiadamente los datos antes de empezar a trabajar.

```
from sklearn.preprocessing import StandardScaler
X = train_df.ix[:,'x.1':'x.10'].values
y = train_df.ix[:,'y'].values
X_std = StandardScaler().fit_transform(X)
```

(c) Utilizando PCA genere una representación en 2 dimensiones de la data original (10 dimensiones) identificando cada clase con un color distinto (elija una paleta apropiada).

```
from sklearn.decomposition import PCA
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np
sklearn_pca = PCA(n_components=2)
Xred_pca = sklearn_pca.fit_transform(X_std)
cmap = plt.cm.get_cmap('ChooseAnAppropriatePalette')
mclasses=(1,2,3,4,5,6,7,8,9)
mcolors = [cmap(i) for i in np.linspace(0,1,10)]
plt.figure(figsize=(12, 8))
for lab, col in zip(mclasses,mcolors):
    plt.scatter(Xred_pca[y==lab, 0],Xred_pca[y==lab, 1],label=lab,c=col)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
leg = plt.legend(loc='upper right', fancybox=True)
plt.show()
```

(d) Utilizando LDA genere una representación en 2 dimensiones de la data original (10 dimensiones) identificando cada clase con un color distinto (elija una paleta apropiada).

```
from sklearn.lda import LDA
sklearn_lda = LDA(n_components=2)
Xred_lda = sklearn_lda.fit_transform(X_std,y)
cmap = plt.cm.get_cmap('ChooseAnAppropriatePalette')
mclasses=(1,2,3,4,5,6,7,8,9)
mcolors = [cmap(i) for i in np.linspace(0,1,10)]
plt.figure(figsize=(12, 8))
for lab, col in zip(mclasses,mcolors):
    plt.scatter(Xred_lda[y==lab, 0],Xred_lda[y==lab, 1],label=lab,
                c=col)
plt.xlabel('LDA/Fisher Direction 1')
plt.ylabel('LDA/Fisher Direction 2')
leg = plt.legend(loc='upper right', fancybox=True)
plt.show()
```

(e) Compare cualitativamente los resultados obtenidos en 1 y 2. Proponga un método para elegir una técnica de reducción de dimensionalidad.

(f) Construya un clasificador que determine la clase de un dato x aleatoriamente sin considerar las características sino que solamente la probabilidad *a-priori* de cada clase. Por ejemplo, si la clase $y = 0$ ocurre el 25% de las veces, su clasificador debe predecir esta clase para un determinado x con probabilidad 0.25, independiente de los atributos de x.

(g) Compare el desempeño de LDA, QDA y un modelo de *Vecinos Más Cercanos (k-NN)* [†] sin reducir dimensionalidad. ¿Qué técnica se comporta mejor sobre el conjunto de entrenamiento? ¿Sobre el conjunto de pruebas? Describa, utilizando un gráfico, el efecto de cambiar el parámetro de k en el tercer modelo.

```
from sklearn.lda import LDA
from sklearn.qda import QDA
from sklearn.neighbors import KNeighborsClassifier
Xtest = test_df.ix[:,'x.1':'x.10'].values
ytest = test_df.ix[:,'y'].values
X_std_test = StandardScaler().fit_transform(Xtest)
lda_model = LDA()
lda_model.fit(X_std,y)
print lda_model.score(X_std,y)
print lda_model.score(X_std_test,ytest)
qda_model = QDA()
knn_model = KNeighborsClassifier(n_neighbors=10)
```

(g) Utilice PCA para generar una representación de la data en $d' = 1, 2, 3, \ldots, 10$ dimensiones. Para cada caso entrene un modelo LDA, QDA y de k-NN. Construya un gráfico que muestre cómo evoluciona el error de entrenamiento versus d'. Sobreponga a este gráfico el error de pruebas versus d'. Concluya.

(h) Utilice LDA para generar una representación de la data en $d' = 1, 2, 3, \ldots, 10$ dimensiones. Para cada caso entrene un modelo LDA, QDA y de k-NN. Construya un gráfico que muestre cómo evoluciona el error de entrenamiento versus d'. Sobreponga a este gráfico el error de pruebas versus d'. Concluya.

[†]Este clasificador busca los k datos de entrenamiento más similares al patrón que se quiere clasificar y predice la clase más popular entre éstos

2 Análisis de Opiniones sobre Películas

El análisis de sentimiento (o minería de opiniones) 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 (ó *review*), en que una persona manifiesta una opinión, como *positiva*, *negativa* o *neutral*. 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.

Los datos que usaremos para esta actividad corresponden a un subconjunto de los datos publicados en *Kaggle*, en el contexto de una competencia organizada por la Universidad de Stanford [6]. Cada registro disponible corresponderá a una opinión sobre una película, registrada sobre el sitio *Rotten Tomatoes*. Para empezar nos limitaremos a estudiar textos anotados como positivos o negativos, clases que codificaremos como +1 y 0 respectivamente. Para construir un clasificador que determine automáticamente la polaridad de un trozo de texto, vamos a necesitar representar los textos $\{d_i\}_{i=1}^{n}$ 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, necesitaremos un *vocabulario* que, para esta actividad, construiremos mediante la unión de todas las palabras que observemos en los textos que tenemos a disposición. Para aumentar la eficacia de las características extraídas es conveniente ejecutar algunas técnicas de preprocesamiento 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 [7]). Otra técnica que suele ser útil para obtener buenas características (features) es la lematización [9], 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* [8].

(a) Construya un dataframe con los datos a analizar descargando los datos desde la URL local [5]. Determine cuántos registros de cada clase contiene el conjunto de entrenamiento y cuántos el conjunto de pruebas.

```
import urllib
import pandas as pd
train_data_url = "http://www.inf.utfsm.cl/~jnancu/stanford-subset/polarity.train"
test_data_url = "http://www.inf.utfsm.cl/~jnancu/stanford-subset/polarity.dev"
train_data_f = urllib.urlretrieve(train_data_url, "train_data.csv")
test_data_f = urllib.urlretrieve(test_data_url, "test_data.csv")
ftr = open("train_data.csv", "r")
fts = open("test_data.csv", "r")
rows = [line.split(" ",1) for line in ftr.readlines()]
train_df = pd.DataFrame(rows, columns=['Sentiment','Text'])
train_df['Sentiment'] = pd.to_numeric(train_df['Sentiment'])
rows = [line.split(" ",1) for line in fts.readlines()]
test_df = pd.DataFrame(rows, columns=['Sentiment','Text'])
test_df['Sentiment'] = pd.to_numeric(test_df['Sentiment'])
train_df.shape
test_df.shape
```

(b) Construya una función, denominada *word_extractor*, que devuelva una lista de las palabras contenidas en un determinado un trozo de texto. Incorpore en su función las operaciones de lower-casing y stemming. Pruebe la función con las frases sugeridas en el código, invente otras similares y comente. Compare con los resultados obtenidos si no se hace stemming.

```
import re, time
from nltk.corpus import stopwords
from nltk import WordNetLemmatizer, word_tokenize
from nltk.stem.porter import PorterStemmer
def word_extractor(text):
```

```
        wordlemmatizer = WordNetLemmatizer()
        commonwords = stopwords.words('english')
        text = re.sub(r'([a-z])\1+', r'\1\1',text)#substitute multiple letter by two
        words = ""
        wordtokens = [ wordlemmatizer.lemmatize(word.lower()) \
                                for word in word_tokenize(text.decode('utf-8', 'ignore')) ]
        for word in wordtokens:
                if word not in commonwords:
                                words+=" "+word
        return words
word_extractor("I love to eat cake")
word_extractor("I love eating cake")
word_extractor("I loved eating the cake")
word_extractor("I do not love eating cake")
word_extractor("I don't love eating cake")
```

(c) Construya una función, denominada *word_extractor2*, análoga a la función anterior, pero que lematice las palabras en vez de hacer stemming. Pruebe la función con las frases sugeridas en el código anterior y discuta las diferencias que observa.

```
def word_extractor2(text):
        wordlemmatizer = WordNetLemmatizer()
        commonwords = stopwords.words('english')
        text = re.sub(r'([a-z])\1+', r'\1\1',text)#substitute multiple letter by two
        words = ""
        wordtokens = [ wordlemmatizer.lemmatize(word.lower()) \
                                for word in word_tokenize(text.decode('utf-8','ignore')) ]
        for word in wordtokens:
                if word not in commonwords:
                                words+=" "+word
        return words
```

(d) Utilizando la función *CountVectorizer* de la librería *sklearn* y de acuerdo a las directrices mencionadas en la introducción, genere una representación vectorial del texto de entrenamiento y del conjunto que usaremos para realizar pruebas. Explore el vocabulario utilizado y determine cuáles son las palabras más frecuentes en el conjunto de entrenamiento y pruebas.

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
texts_train = [word_extractor2(text) for text in train_df.Text]
texts_test = [word_extractor2(text) for text in test_df.Text]
vectorizer = CountVectorizer(ngram_range=(1, 1), binary='False')
vectorizer.fit(np.asarray(texts_train))
features_train = vectorizer.transform(texts_train)
features_test = vectorizer.transform(texts_test)
labels_train = np.asarray((train_df.Sentiment.astype(float)+1)/2.0)
labels_test = np.asarray((test_df.Sentiment.astype(float)+1)/2.0)
vocab = vectorizer.get_feature_names()
dist=list(np.array(features_train.sum(axis=0)).reshape(-1,))
for tag, count in zip(vocab, dist):
    print count, tag
```

(e) Construya una función que evalúe el desempeño obtenido por un clasificador genérico en el conjunto de entrenamiento y en el conjunto de pruebas. Utilice y explique las métricas que calcula la función *classification_report* de la librería *sklearn*.

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

(f) Construya una función que entrene/ajuste un clasificador *Bayesiano Ingenuo (Binario)* (las características no nulas serán tratadas como 1) y mida el error de predicción obtenido sobre los datos de entrenamiento y pruebas. Utilice esta función con las características extraídas en el punto (d). Mida el efecto de filtrar *stopwords* y de eliminar este paso de pre-procesamiento típico. Determine además, qué representación obtiene un mejor resultado: si aquella obtenida vía lematización o aquella obtenida vía stemming. Finalmente, tome un subconjunto aleatorio de los textos de prueba y analice las predicciones del modelo (explore las predicciones, así como las probabilidades que el clasificador asigna a cada clase).

```
from sklearn.naive_bayes import BernoulliNB, random
def do_NAIVE_BAYES(x,y,xt,yt):
        model = BernoulliNB()
        model = model.fit(x, y)
        score_the_model(model,x,y,xt,yt,"BernoulliNB")
        return model
model=do_NAIVE_BAYES(features_train,labels_train,features_test,labels_test)
test_pred = model.predict_proba(features_test)
spl = random.sample(xrange(len(test_pred)), 15)
for text, sentiment in zip(test_df.Text[spl], test_pred[spl]):
    print sentiment, text
```

(g) Construya una función que entrene/ajuste un clasificador *Bayesiano Ingenuo Multinomial* (las características enteras no nulas no serán reducidas a 1) y mida el error de predicción obtenido sobre los datos de entrenamiento y pruebas. Utilice esta función con las características extraídas en el punto (d). Mida el efecto de filtrar *stopwords* y de eliminar este paso de pre-procesamiento típico. Determine además, qué representación obtiene un mejor resultado: si aquella obtenida vía lematización o aquella obtenida vía stemming. Finalmente, tome un subconjunto aleatorio de los textos de prueba y analice las predicciones del modelo (explore las predicciones, así como las probabilidades que el clasificador asigna a cada clase).

```
from sklearn.naive_bayes import MultinomialNB,random
def do_MULTINOMIAL(x,y,xt,yt):
        model = MultinomialNB()
        model = model.fit(x, y)
        score_the_model(model,x,y,xt,yt,"MULTINOMIAL")
        return model
model=do_MULTINOMIAL(features_train,labels_train,features_test,labels_test)
test_pred = model.predict_proba(features_test)
spl = random.sample(xrange(len(test_pred)), 15)
for text, sentiment in zip(test_df.Text[spl], test_pred[spl]):
    print sentiment, text
```

(h) Construya una función que entrene/ajuste un modelo de *Regresión Logística Regularizado* (utilizando, como penalizador, la norma ℓ_2) y mida el error de predicción obtenido sobre los datos de entrenamiento y pruebas. Incluya en su función la exploración de diferentes valores del parámetro de regularización. Explique el significado y efecto esperado de este parámetro. Utilice la función construida con los atributos extraídos en el punto (d). Mida el efecto de filtrar *stopwords* y de eliminar este paso de

pre-procesamiento típico. Determine además, qué representación obtiene un mejor resultado: si aquella obtenida vía lematización o aquella obtenida vía stemming. Finalmente, tome un subconjunto aleatorio de los textos de prueba y analice las predicciones del modelo (explore las predicciones, así como las probabilidades que el clasificador asigna a cada clase).

```
from sklearn.linear_model import LogisticRegression
def do_LOGIT(x,y,xt,yt):
        start_t = time.time()
        Cs = [0.01,0.1,10,100,1000]
        for C in Cs:
                print "Usando C= %f"%C
                model = LogisticRegression(penalty='l2',C=C)
                model = model.fit(x, y)
                score_the_model(model,x,y,xt,yt,"LOGISTIC")
do_LOGIT(features_train,labels_train,features_test,labels_test)
```

(h) Construya una función que entrene/ajuste una *Máquina de Vectores de Soporte (SVM) Lineal* y mida el error de predicción obtenido sobre los datos de entrenamiento y pruebas. Incluya en su función la exploración de diferentes valores del parámetro de regularización C. Discuta el significado y efecto esperado de este parámetro. Utilice la función construida con los atributos extraídos en el punto (d). Mida el efecto de filtrar *stopwords* y de eliminar este paso de pre-procesamiento típico. Determine además, qué representación obtiene un mejor resultado: si aquella obtenida vía lematización o aquella obtenida vía stemming. Finalmente, tome un subconjunto aleatorio de los textos de prueba y analice las predicciones del modelo (explore las predicciones, así como las probabilidades que el clasificador asigna a cada clase).

```
from sklearn.svm import LinearSVC
def do_SVM(x,y,xt,yt):
        Cs = [0.01,0.1,10,100,1000]
        for C in Cs:
                print "El valor de C que se esta probando: %f"%C
                model = LinearSVC(C=C)
                model = model.fit(x, y)
                score_the_model(model,x,y,xt,yt,"SVM")
do_SVM(features_train,labels_train,features_test,labels_test)
```

(i) Construya un gráfico que permita comparar los resultados obtenidos por los diferentes métodos de clasificación.

References

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

[2] Joshi, M., Das, D., Gimpel, K., Smith, N. A. (2010). Movie reviews and revenues: An experiment in text regression. In the 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (pp. 293-296). Association for Computational Linguistics.

[3] http://statweb.stanford.edu/ tibs/ElemStatLearn/datasets/

[4] https://www.dropbox.com/sh/8r1wrblyfokwuq0/AABUEvgcuMxyZht2-KYyBptUa?dl=0

[5] http://www.inf.utfsm.cl/ jnancu/stanford-subset/

[6] https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews

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

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

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

LaTeX