Clase 15 IMA357

Mg. Alejandro Ferreira Vergara

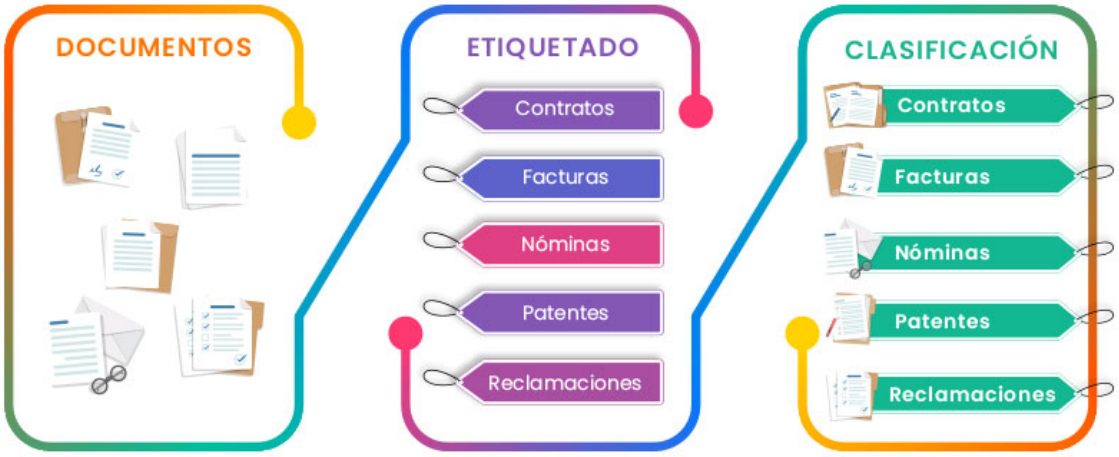
May 15, 2023

1 Análisis de Discriminante Lineal para la Clasificación de Documentos

[1]:
```
from IPython.display import Image

Image(filename=r'Imagenes/15_1.jpg', width=700)
```

[1]:



Usualmente, la clasificación de documentos se puede realizar mediante 3 enfoques:

- **Sistemas basados en reglas:** Este tipo de enfoque aplica un sistema basado en reglas de clasificación. Para esto, se programan directamente reglas lingüísticas construidas manualmente por expertos humanos, basadas en el contenido de los documentos. Luego, un algoritmo puede tomar dichas reglas y clasificar tópicos, detectano elementos relevantes semánticamente de un texto. Cada regla está construida de un patrón que se debe gatillar y un rótulo de clase que se debe asignar.
- **Sistemas basados en Aprendizaje Automático:** Este tipo de enfoque utiliza técnicas de aprendizaje automático supervisado y requiere de un corpus de entrenamiento en dónde cada documento tiene asignado su rótulo de clase correspondiente. Aquí, un modelo de aprendizaje automático no puede entender un texto directamente, por lo que se requiere generar representaciones eficientes de los documentos en el formato de entrada para un algoritmo particular.

- **Sistemas híbridos:** estos métodos son simplemente combinaciones de sistemas basados en reglas y clasificadores que usan aprendizaje automático, lo cual mejora los resultados a medida que se refinan las reglas.

1.1 Análisis de Discriminante Lineal (LDA)

Es un algoritmo de Aprendizaje Supervisado.

Proyecta (linealmente) los datos en un nuevo espacio de características.

LDA realiza la suposición que los datos se distribuyen normalmente. Además, supone que las clases tienen matrices de covarianza idénticas y que las características son estadísticamente independientes entre sí.

Un Algoritmo de entrenamiento de un modelo LDA para la clasificación binaria, a partir de una Matriz TF-IDF:

1. Calcular la posición promedio (centroide) de todos los vectores TF-IDF dentro de la clase positiva (como mensajes SMS de spam).
2. Calcular la posición promedio (centroide) de todos los vectores TF-IDF que no están en la clase positiva (como los mensajes SMS que no son spam).
3. Calcular la diferencia vectorial entre los centroides (la línea que los conecta).
4. Proyectar cada vector TF-IDF sobre la línea recta calculada anteriormente.
5. Definir un umblar de decisión para clasificar.

1.2 Clasificación de Spam

[ ]:
```
import pandas as pd

pd.options.display.width = 120

sms = pd.read_csv(r'Datos/sms-spam.csv', header=0, encoding='utf-8')

index = ['sms{}{}'.format(i, '!'*j) for (i,j) in zip(range(len(sms)), sms.spam)]
sms.index = index

sms.head(6)
```

[ ]:
```
# transformamos el dato del campo spam en tipo INT.
sms['spam'] = sms['spam'].astype(int)

print(len(sms))
```

[ ]:
```
sms.spam.sum()
```

[ ]:
```
len(sms)-sms.spam.sum()
```

[ ]:
```
sms.info()
```

[ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize.casual import casual_tokenize

tfidf_model = TfidfVectorizer(tokenizer=casual_tokenize)
tfidf_docs = tfidf_model.fit_transform(raw_documents=sms.sms).toarray()
tfidf_docs
```

[ ]:
```
tfidf_docs.shape
```

1.2.1 Entrenamiento del Modelo de Clasificación

[ ]:
```
mask = sms.spam.astype(bool).values

spam_centroid = tfidf_docs[mask].mean(axis=0)
nspam_centroid = tfidf_docs[~mask].mean(axis=0)
```

[ ]:
```
spam_centroid.round(3)
```

[ ]:
```
nspam_centroid.round(3)
```

[ ]:
```
spamminess_score = tfidf_docs.dot(spam_centroid - nspam_centroid)
spamminess_score.round(3)
```

[ ]:
```
spamminess_score.shape
```

[ ]:
```
len(spamminess_score)
```

[ ]:
```
import numpy as np

print(np.min(spamminess_score))
print(np.max(spamminess_score))
```

[ ]:
```
from sklearn.preprocessing import MinMaxScaler

sms['lda_score'] = MinMaxScaler().fit_transform(spamminess_score.reshape(-1,1))
sms.lda_score
```

[ ]:
```
sms['lda_predict'] = (sms.lda_score > .5).astype(int)
sms.lda_predict
```

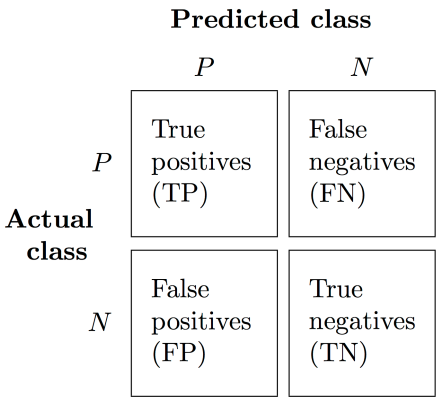
[ ]:
```
sms['spam lda_predict lda_score'.split()].round(2).head(15)
```

1.2.2 Evaluación del Modelo de Clasificación

[ ]:
```
(1. - (sms.spam - sms.lda_predict).abs().sum() / len(sms)).round(3)
```

[2]:
```
Image(filename=r'Imagenes/15_2.png', width=400)
```

[2]:



[ ]:
```
from sklearn.metrics import confusion_matrix

y_real = sms.spam
y_pred = sms.lda_predict

matrix = confusion_matrix(y_real, y_pred)
matrix
```

[ ]:
```
acc = (4135+593)/(4135+593+64+45)
round(acc,3)
```

[ ]:
```
# Para nuestra clase Positiva:    OJO: para sklearn.metrics.confusion_matrix es␣
↪la clase Negativa
acc_spam = 593/(593+45)
round(acc_spam,3)
```

TAREA:

Estudiar las siguientes métricas para la clasificación binaria e implementarlas utilizando Sk-learn.

Error:

$$ERR = \frac{FP + FN}{FP + FN + TP + TN} \quad (1)$$

Exactitud:

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR \quad (2)$$

Tasa de Verdaderos Positivos:

$$TPR = \frac{TP}{P} = \frac{TP}{FN + TP} \quad (3)$$

Tasa de Falsos Positivos:

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} \quad (4)$$

Precisión:

$$PRE = \frac{TP}{TP + FP} \quad (5)$$

Recall:

$$REC = TPR = \frac{TP}{FN + TP} \quad (6)$$

A continuación definiremos:

$$AP = TP + FN, AN = FP + TN \quad (7)$$

$$PP = TP + FP, PN = FN + TN \quad (8)$$

F1-score:

$$F1 = \frac{2 \times TP}{AP + PP} \quad (9)$$

Coeficiente de Correlación de Matthews:

$$MCC = \frac{(TP \times TN - FP \times FN)}{\sqrt{PP \times PN \times AP \times AN}} \quad (10)$$

Coeficiente de Kappa de Cohen:

$$K = \frac{P_o - P_e}{1 - P_e} \quad (11)$$

donde,

$$P_e = P_{pos} + P_{neg} \quad (12)$$

$$P_{pos} = \frac{AP}{N} \times \frac{PP}{N} \quad (13)$$

$$P_{neg} = \frac{AN}{N} \times \frac{PN}{N} \quad (14)$$

P_o es la exactitud alcanza por el modelo y N es el total de datos presentados en la matriz de confusión.

[ ]: