Clase 12 IMA357

Alejandro Ferreira Vergara

April 24, 2023

1 Descomposición en Valores Singulares y Análisis de Componentes Principales

1.1 Machine Learning o Aprendizaje Automático

El aprendizaje automático o aprendizaje automatizado o aprendizaje de máquinas (del inglés, machine learning) es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan. Se dice que un agente aprende cuando su desempeño mejora con la experiencia; es decir, cuando la habilidad no estaba presente en su genotipo o rasgos de nacimiento y luego se adquiere. Los modelos o programas resultantes deben ser capaces de generalizar comportamientos e inferencias para un conjunto más amplio (potencialmente infinito) de datos. (Russell, Stuart; Norvig, Peter (2009). Inteligencia Artificial: Un Enfoque Moderno (3rd edición)).

El ser humano adquiere su aprendizaje de los conceptos cotidianos a partir de su experiencia inmediata y no mediante definiciones abstractas. O sea: Aprende a partir de datos. Las técnicas de Aprendizaje Automático son basadas en **Datos**.

1.1.1 Paradigmas de Aprendizaje

Aprendizaje Supervisado

- Cada dato tiene asociada una etiqueta o valor correcto que se quiere poder predecir
- Es el caso más desarrollado
- Requiere un esfuerzo o conocimiento previo que viene codificado en las etiquetas ya asignadas

Aprendizaje no supervisado

- Los datos no tienen información asociada que se persigue replicar
- En general se busca identificar patrones o información desconocida que se manifiesta en los datos
- Es probablemente la manera más común en que se presentan y operan datos.

Aprendizaje reforzado

- La información sobre el objetivo a aprender no viene acompañando al dato, sino que se obtiene como respuesta a acciones en un entorno prefijado
- Los datos son acciones de interacción en ese entorno, que llevan a un mejor resultado
- Se aprende basicamente a prueba y error, y en general subyacen a un contexto interacción dinámica.

1.2 Análisis Semántico Latente (LSA)

El análisis semántico latente se basa en la técnica más antigua y más utilizada para la reducción de dimensiones, la **descomposición de valores singulares o SVD**. SVD descompone una matriz en tres matrices, una de las cuales es diagonal.

TEOREMA 1

Para cada matriz real A de dimensión $n \times n$, existen 2 matrices ortogonales U y V y una matriz diagonal D que cumplen $A = VDU^T$, donde D tiene la forma:

$$D = \begin{pmatrix} \sigma_1 & & \dots & \\ & \sigma_2 & \dots & \\ \vdots & \vdots & \ddots & \vdots \\ & & \dots & \sigma_n \end{pmatrix}$$

y donde $\sigma_1, \dots, \sigma_r$ son los valores singulares de f, es decir, las raices positivas de los autovalores distintos de 0 de A^TA y AA^T, y $\sigma_{r+1} = \dots = \sigma_n = 0$. Las columnas de U son los vectores propios de A^TA y las columnas de V son los vectores propios de AA^T.

El Teorema anterior se puede generalizar para matrices no cuadradas.

TEOREMA 2

Para cada matriz real A de dimensión $m \times n$, existen 2 matrices ortogonales U $(n \times n)$ y V $(m \times m)$ y una matriz diagonal D $(m \times n)$ que cumplen $A = VDU^T$, donde D tiene la forma:

$$D = \begin{pmatrix} \sigma_1 & & \dots & \\ & \sigma_2 & \dots & \\ \vdots & \vdots & \ddots & \vdots \\ & & \dots & \sigma_n \\ 0 & \vdots & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \vdots & \dots & 0 \end{pmatrix}, D = \begin{pmatrix} \sigma_1 & & \dots & & 0 & \dots & 0 \\ & \sigma_2 & \dots & & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & 0 & \vdots & 0 \\ & & \dots & \sigma_m & 0 & \dots & 0 \end{pmatrix}$$

y donde $\sigma_1, \dots, \sigma_r$ son los valores singulares de f, es decir, las raices positivas de los autovalores distintos de 0 de A^TA y AA^T y $\sigma_{r+1} = \dots = \sigma_p = 0$, donde $p = min(m, n)$. Las columnas de U son los vectores propios de A^TA y las columnas de V son los vectores propios de AA^T.

Podemos usar **SVD** para descomponer una matriz de vectores TF-IDF en tres matrices más simples, que al multiplicarlas obtendríamos la matriz original.

Ventajas:

1. Las matrices U, V y D revelan propiedades de nuestra matriz de vectores TF-IDF que podemos explotar para simplificarla. Podríamos por ejemplo truncar estas matrices (quitarles filas ó columnas) antes de multiplicarlas nuevamente con lo que reduciríamos la dimensión de la matriz TF-IDF al multiplicarlas.
2. Estas matrices truncadas no nos otorgarán la matriz TF-IDF con la que comenzamos (como dijimos, su dimensión se verá reducida), sino que podrían otorgarnos una mejor. Esta nueva

representación de los documentos contiene la esencia, la "**semántica latente**" de esos documentos.

Cuando usamos **SVD** de esta manera en NLP, lo llamamos **análisis semántico latente**; esto, porque LSA intenta descubrir la semántica latente o significado de las palabras.

Esta técnica matemática es usada para encontrar la "mejor" forma de transformar (rotar y estirar) linealmente cualquier conjunto de vectores (como los vectores TF-IDF o los vectores de Bag of Words).

En muchas aplicaciones esto consiste en alinear los ejes (dimensiones) en los nuevos vectores con la mayor "dispersión" o variación en las frecuencias de palabras. Luego, podemos eliminar esas dimensiones en el nuevo espacio vectorial.

LSA usa SVD para encontrar las combinaciones de palabras que son responsables, juntas, de la mayor variación en los datos. Cada una de las dimensiones se convierte en una combinación de frecuencias de palabras en lugar de una sola frecuencia de palabras, de modo que pensamos en **las dimensiones** como la **combinacion lineal de palabras que componen varios "temas"** encontrados a lo largo del corpus.

[ ]:
```
from nltk.tokenize import TreebankWordTokenizer

tokenizer = TreebankWordTokenizer()

docs = ["La UFRO está en Temuco, y yo estudio en la ufro."]
docs.append("La Ufro es una universidad estatal.")
docs.append("Facultad de Ingeniería y Ciencia, Ufro.")
docs.append("Departamento de Ingeniería Matemática, Ufro.")
docs.append("Departamento de Ciencias Físicas, Ufro.")
print(docs)
```

[ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = docs
vectorizer = TfidfVectorizer(min_df=1, encoding='utf-8')
model = vectorizer.fit_transform(corpus)
X = model.todense()

print(X.shape)
print(X)
```

1.3 Descomposición en Valores Singulares

Independientemente de si descomponemos con SVD una matriz de documentos vectorizados TF-IDF o Bag of Words, SVD nos será útil para encontrar las combinaciones de palabras que formarán un tema.

SVD realiza esto calculando la correlacion del uso de tokens entre documentos y los documentos en sí. Luego, se calculan combinaciones lineales entre los token que tengan mayor frecuencia, los que se podrían interpretar como los "temas" en el corpus.

Finalmente, nos quedaremos solo con los temas que retengan la mayor cantidad de información.

Recordemos: Con **Teorema 2** podemos escribir $X = VDU^T$

La matriz V:

- Corresponde a la **matriz de términos (o tokens) y temas** que informa sobre "los vecinos de las palabras en el espacio vectorial semántico". Contiene la correlación cruzada entre palabras y temas, basada en la coincidencia de palabras en el mismo documento.
- La matriz más importante para el análisis semántico.
- Contiene todos los vectores de temas para cada palabra del corpus como columnas.

[ ]:
```
import numpy as np

V, s, Ut = np.linalg.svd(X.T)

print(V.shape)
print(V)
```

Implementación con Numpy. **OJO**: AQUI X de debe transponer

La matriz D o σ:

- Matriz diagonal cuadrada.
- Contiene los valores singulares.
- Los valores singulares indican cuánta información captura cada dimensión en el nuevo espacio vectorial semántico.
- Como es diagonal, posee una gran cantidad de 0's. Para ahorrar memoria Numpy lo entrega como un vector, sin embargo, es sencillo obtener su forma matricial usando *numpy.fill_diagonal*

[ ]:
```
D = np.zeros((len(V), len(Ut)))
np.fill_diagonal(D, s)

print(D.shape)
print(D)
```

La matriz U^T:

- Mide con que frecuencia los documentos usan los mismos temas en el nuevo modelo semántico de documentos.
- Solo se utiliza para verificar la precisión de los vectores de temas para recrear los vectores orginales de documentos que utilizamos para crearla.

[ ]:
```
print(Ut.shape)
print(Ut)
```

```
[ ]: Xp = V * D * Ut

     dif = X - Xp.T
     print(dif.round(10))
```

Al igual que la matriz D, ignoraremos la matriz U^T siempre que estemos transformando nuevos vectores de documentos en un espacio vectorial de temas.

Hasta aquí, **no hemos disminuído la dimensión del problema**; para esto es necesario truncar las matrices. Al realizar este proceso, estamos realizando lo que se conoce como **SVD Truncado**.

Importante: Podemos ignorar la matriz D porque las filas y columnas de la matriz V ya están organizadas de modo que los temas más importantes con los **valores singulares más grandes** están a la izquierda. Otra razón por la que podemos ignorar D es que la mayoría de los vectores de documentos que queramos usar con este modelo (como los TF-IDF) ya están normalizados.

1.4 Análisis de Componentes Principales

PCA intenta encontrar los ejes de componentes ortogonales de máxima varianza en un conjunto de datos. Por otro lado, LDA busca encontrar el subespacio de características que optimiza la separabilidad de clases.

1.4.1 Fundamentos matemáticos

- El objetivo es identificar patrones en los datos y comprender la estructura de varianza-covarianza incluida en estos.
- Esto es útil para las siguientes tareas:
    1. Reducción de las variables explicativas: a menudo, gran parte de la variabilidad de los datos se puede explicar por un número menor de componentes principales.
    2. Interpretación: PCA puede mostrar relaciones que no se sospechaban previamente.

TEOREMA 3

Sea X una matriz de dimensión $n \times d$ de puntos $X_1, \dots, X_n$ y denotemos por μ al centroide de los puntos $X_i's$. Si $X - \mu = VDU^T$ es la descomposición SVD de $X - \mu$ y la diagonal principal de D posee los valores singulares $\sigma_1 \leq \sigma_2 \leq \cdots \leq \sigma_d$, entonces los puntos centrados $Y_1, \dots, Y_d$, donde:

$$Y_k = (X - \mu)u_k = \ k\text{-ésima columna de } VD$$

y u_k es la k-ésima columna de U, son las d componentes principales de X. Además,

$$var(Y_k) = \frac{\sigma_k^2}{n-1}$$

y $cov(Y_h, Y_k) = 0$, cuando $h \neq k$ and $1 \leq k, h \leq d$.

1.5 Acercamiento a la Modelación de Tópicos

[ ]:
```
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)
```

[ ]:
```
for i in range(6):
    print(sms.iloc[i,1])
    print('------------------------')
```

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

tfidf = TfidfVectorizer(tokenizer=casual_tokenize)
tfidf_docs = tfidf.fit_transform(raw_documents=sms.sms).toarray()
len(tfidf.vocabulary_)
```

[ ]:
```
tfidf_docs = pd.DataFrame(tfidf_docs)
# X - \mu
tfidf_docs = tfidf_docs - tfidf_docs.mean()
tfidf_docs.shape
```

1.5.1 PCA

[ ]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=16)
pca = pca.fit(tfidf_docs)
pca_topic_vectors = pca.transform(tfidf_docs)
pca_topic_vectors.shape
```

[ ]:
```
columns = ['topic{}'.format(i) for i in range(pca.n_components)]
pca_topic_vectors = pd.DataFrame(pca_topic_vectors, columns=columns,index=index)
pca_topic_vectors.round(3).head(6)
```

[ ]:
```
column_nums, terms = zip(*sorted(zip(tfidf.vocabulary_.values(), tfidf.
 ↪vocabulary_.keys())))
print(len(terms))
```

[ ]:
```
weights = pd.DataFrame(pca.components_, columns=terms, index=['topic{}'.
 ↪format(i) for i in range(16)])

pd.options.display.max_columns = 8
```

```
weights.head(6).round(3)
```

[ ]:
```
pd.options.display.max_columns = 12
deals = weights['! ;) :) half off free crazy deal only $ 80 %'.split()].
↪round(3) * 100

deals
```

1.5.2 SVD Truncado

- Enfoque más directo de LSA que omite el modelo de PCA.
- Maneja de mejor manera las matrices dispersas, por lo que en grandes conjuntos de datos es una buena opción a PCA.
- Descarta las dimensiones que contengan la menor información sobre la matriz TF-IDF o BOW.

[ ]:
```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=16, n_iter=100)
svd_topic_vectors = svd.fit_transform(tfidf_docs.values)

svd_topic_vectors = pd.DataFrame(svd_topic_vectors, columns=columns,index=index)
svd_topic_vectors.round(3).head(6)
```

Se puede observar que **los vectores de tema de TruncatedSVD son exactamente los mismos que se produjeron con PCA**, esto se debe a la gran cantidad de iteraciones y además a que nos hemos preocupado de encontrar el centroide de nuestros vectores TF-IDF.

[ ]:
```
svd_topic_vectors = (svd_topic_vectors.T / np.linalg.norm(svd_topic_vectors,␣
↪axis=1)).T
svd_topic_vectors.iloc[:10].dot(svd_topic_vectors.iloc[:10].T).round(1)
```

[ ]:
```
sms.iloc[1,1]
```

[ ]:
```
sms.iloc[3,1]
```

[ ]:
```
sms.iloc[0,1]
```

[ ]:
```

```