Reconocimiento Estadístico de Patrones - Tarea 1

Rubén Pérez Palacios Lic. Computación Matemática, Profesor: Johan Van Horebeek

May 2, 2023

1 Analisis de vinos por tipo de uvas

1.1 Librerías

Importaremos todas las librerías que se usaran durante toda la tarea, asi como declarar variables globales y configuraciones.

[ ]:
```
import sys
import numpy as np
import pandas as pd
from pprint import pprint

import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
from plotly import figure_factory as ff

from sklearn.decomposition import PCA
from sklearn import manifold
from sklearn_som.som import SOM
from sklearn.manifold import TSNE
from sklearn import neighbors
from sklearn.model_selection import train_test_split

np.set_printoptions(threshold=sys.maxsize)
theme = 'plotly_dark'
```

1.2 Preparación de Datos

1.2.1 Lectura

Se leen los datos a partir del archivo “wines” proporcionado. A su vez se renombraran las columnas por su abrevaicación descrita en el pdf, esto para dar mayor información cuando se esten refiriendo a ellas, claro sin hacer prejucios sobre ellas. Por practicidad se utilizara a la clase de las uvas como un string para evitar escalamientos no deseados.

[ ]:
```
source_data = pd.read_csv(filepath_or_buffer="wines",delim_whitespace=True,)
source_data.columns = [
    "Class",
```

```
    "Alcohol",
    "Malic",
    "Ash",
    "Alcal",
    "Mg",
    "Phenol",
    "Flav",
    "Nonf",
    "Proan",
    "Color",
    "Hue",
    "Abs",
    "Proline"
]
source_data["Class"] = source_data["Class"].map(str)
source_data
```

[ ]:

	Class	Alcohol	Malic	Ash	Alcal	Mg	Phenol	Flav	Nonf	Proan	Color	\
1	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	
2	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	
3	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	
4	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	
5	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	
..	…	…	…	…	…	…	…	…	…	…	…	
174	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	
175	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	
176	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	
177	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	
178	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	

	Hue	Abs	Proline
1	1.04	3.92	1065
2	1.05	3.40	1050
3	1.03	3.17	1185
4	0.86	3.45	1480
5	1.04	2.93	735
..	…	…	…
174	0.64	1.74	740
175	0.70	1.56	750
176	0.59	1.56	835
177	0.60	1.62	840
178	0.61	1.60	560

[178 rows x 14 columns]

1.2.2 Comprobación de completitud

Checamos si hacen falta datos y de ser asi los completamos

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 178 entries, 1 to 178
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Class    178 non-null    object
 1   Alcohol  178 non-null    float64
 2   Malic    178 non-null    float64
 3   Ash      178 non-null    float64
 4   Alcal    178 non-null    float64
 5   Mg       178 non-null    int64
 6   Phenol   178 non-null    float64
 7   Flav     178 non-null    float64
 8   Nonf     178 non-null    float64
 9   Proan    178 non-null    float64
 10  Color    178 non-null    float64
 11  Hue      178 non-null    float64
 12  Abs      178 non-null    float64
 13  Proline  178 non-null    int64
dtypes: float64(11), int64(2), object(1)
memory usage: 20.9+ KB
```

1.2.3 Normalización

Para evitar el cesgo durante nuestro analisis no haremos uso del tipo de clase de uva, ya que esto es lo que se desea predecir. Por último normalizamos nuestros datos para evitar una ponderación en las características.

[ ]:
```
data = source_data.copy(deep=True)
data.pop("Class")
data_norm = (data - data.mean())/data.std()
data_norm
```

[ ]:

	Alcohol	Malic	Ash	Alcal	Mg	Phenol	Flav	\
1	1.514341	-0.560668	0.231400	-1.166303	1.908522	0.806722	1.031908	\
2	0.245597	-0.498009	-0.825667	-2.483841	0.018094	0.567048	0.731565	
3	0.196325	0.021172	1.106214	-0.267982	0.088110	0.806722	1.212114	
4	1.686791	-0.345835	0.486554	-0.806975	0.928300	2.484437	1.462399	
5	0.294868	0.227053	1.835226	0.450674	1.278379	0.806722	0.661485	
..	…	…	…	…	…	…	…	
174	0.873810	2.966176	0.304301	0.300954	-0.331985	-0.982841	-1.420891	
175	0.491955	1.408636	0.413653	1.049555	0.158126	-0.791103	-1.280731	
176	0.331822	1.739837	-0.388260	0.151234	1.418411	-1.126646	-1.340800	
177	0.208643	0.227053	0.012696	0.151234	1.418411	-1.030776	-1.350811	
178	1.391162	1.578712	1.361368	1.498716	-0.261969	-0.391646	-1.270720	

```
          Nonf     Proan     Color       Hue       Abs   Proline
1    -0.657708  1.221438  0.251009  0.361158  1.842721  1.010159
2    -0.818411 -0.543189 -0.292496  0.404908  1.110317  0.962526
3    -0.497005  2.129959  0.268263  0.317409  0.786369  1.391224
4    -0.979113  1.029251  1.182732 -0.426341  1.180741  2.328007
5     0.226158  0.400275 -0.318377  0.361158  0.448336 -0.037767
..         …         …         …         …         …         …
174   1.270726 -0.927563  1.139596 -1.388840 -1.227742 -0.021890
175   0.547563 -0.316058  0.967055 -1.126341 -1.481267  0.009866
176   0.547563 -0.420888  2.217979 -1.607590 -1.481267  0.279786
177   1.351077 -0.228701  1.829761 -1.563840 -1.396759  0.295664
178   1.592131 -0.420888  1.786626 -1.520090 -1.424928 -0.593486

[178 rows x 13 columns]
```

1.3 Analisis exploratorio

1.3.1 Distribuciones

Empezaremos por analizar la distribución marginal de los datos por cada carácteristica y analizar un poco que información nos dan ellas.

[ ]:
```
for characteristic in data.columns:
    fig = ff.create_distplot(
        [source_data[characteristic][source_data["Class"] == wine_class] for␣
 ↪wine_class in source_data["Class"].unique()],
        source_data["Class"].unique(),
        bin_size = [
            (np.array(source_data[characteristic][source_data["Class"] ==␣
 ↪wine_class].max()) - np.
 ↪array(source_data[characteristic][source_data["Class"] == wine_class].
 ↪min())) / 10
            for wine_class in source_data["Class"].unique()
        ]
    )
    fig.update_layout(width=500,height=500, template=theme,␣
 ↪title=characteristic + " Distribution")
    fig.show()
```

Podemos observar que ninguna característica logra separar las distintas clases de uvas, la que mejor los hace es la cantidad de flavonoides que tiene un vino pero hay una intersección significativa. En todo caso, podemos ver que hay características que no nos proporcionan mucha información donde la distribución de las uvas por clase estan muy juntas, como lo son el contenido de magnesio, la cantidad de ceniza, y la alcanilidad de la ceniza; no hay la suficiente evidencia para descartar alguna de estas características por lo que las mantendremos. Por lo que optamos por seguir indagando, ahora por las distirbuciones bivariadas sobre las características.

[ ]:
```
fig = px.scatter_matrix(
    source_data,
    dimensions = [
    "Alcohol",
    "Malic",
    "Ash",
    "Alcal",
    "Mg",
    "Phenol",
    "Flav",
    "Nonf",
    "Proan",
    "Color",
    "Hue",
    "Abs",
    "Proline"
    ],
    color="Class",
    symbol="Class",
    color_discrete_sequence=px.colors.qualitative.D3
)
fig.update_layout(width=1800,height=1800, template=theme, title="Bivariate␣
 ↪Distribution")
fig.show()
```

Nuevamente no encontramos una clasificación importante en ninguna distirbución, pero de los flavonoide con el alcohol y la prolina lográn separar un poco nuestros datos, aunque no lo suficiente. La prolina, el radio de absorción y el color en general logran separar las uvas del tipo 3, del tipo 1 y del tipo 2 respectivamente, por lo que haremos unos metodos de reducción de dimensionalidad para tratar de jusficar estas hipotesis.

1.3.2 Reducción de Dimensionalidad

Graficadores Implementaremos funciones para que nos ayuden a visualizar las proyecciones de nuestros metodos, así como el ver como se comportan la distribución de los tipos de uva y las características de los vinos.

[ ]:
```
def graph_proj(df, proj,coeff=None,labels=None,title="Grafica"):

    fig = go.Figure()
    for wine_class in df['Class'].unique():
        fig.add_scatter(
            x = proj[:,0][df['Class'] == wine_class],
            y = proj[:,1][df['Class'] == wine_class],
            name = "wines class " + wine_class,
            mode="markers"
        )
```

```
    if coeff is not None:
        for direc,label in zip(coeff,labels):
            fig.add_trace(
                go.Scatter(
                    x = [0,direc[0]],
                    y = [0,direc[1]],
                    name = label
                )
            )

    fig.update_layout(
        template=theme,
        title=title,
        width=1000,
        height=1000,

    )
    fig.update_xaxes(title='PC1', showgrid=True, showline=True, zeroline=False)
    fig.update_yaxes(title='PC2', showgrid=True, showline=True, zeroline=False,␣
↪scaleanchor="x", scaleratio=1)

    fig.show()
```

[ ]:
```
def graph_characteristic(df, proj, title="Grafica"):

    x = proj[:,0]
    y = proj[:,1]

    fig = make_subplots(rows=3, cols=5, subplot_titles=data_norm.columns)

    for i,carac in enumerate(data_norm.columns):
        fig.add_scatter(
            row=int(i/5)+1,
            col=int(i%5)+1,
            x = x,
            y = y,
            marker=dict(
                color=df[carac],
                coloraxis="coloraxis"
            ),
            showlegend=False,
            mode="markers"
        )
    fig.update_layout(
        template=theme,
        coloraxis_colorscale='Viridis',
        title=title,
```

```
        width=1800,
        height=1500
    )
    fig.update_xaxes(title='PC1', showgrid=True, showline=True, zeroline=False)
    fig.update_yaxes(title='PC2', showgrid=True, showline=True, zeroline=False,␣
 ↪scaleanchor="x", scaleratio=1)
    fig.show()
```

PCA Empezaremos con nuestro valiente PCA, nuestro caballo de batalla.

[ ]:
```
pca = PCA(2)
pca.proj = pca.fit_transform(data_norm)
print(np.linalg.norm(pca.components_.T * np.sqrt(pca.
 ↪explained_variance_),axis=1))
graph_proj(source_data, pca.proj, pca.components_.T * np.sqrt(pca.
 ↪explained_variance_), data_norm.columns[:], "PCA")
```

```
[0.8259035  0.63971327 0.49946593 0.51942673 0.56485199 0.86228332
 0.91748554 0.64920181 0.68275209 0.85926895 0.78038168 0.85641839
 0.84819186]
```

Podemos notar como nuestra confianza depositada en el siempre rinde sus frutos. En el observamos una mejor seperación de los datos, con uno que otra observación dispersa, y los borde aun no lo suficientemente marcados sobre todo entre las uvas del tipo 1 y 2. No todo es malo podemos ver como los flavonoide en efecto nos ayudan a distinguir entre las uvas del tipo 1 y 3, además es el de mayor magnitud lo cuál nos indica que es una buena característica; la ceniza a pesar de ser de el de menor magnitud estos nos ayudan a diferneciar un poco los uvas de tipo 2. Ahora en convinación una concentración alta de flavonoides, de proantocianidinas, de fenol, de prolines, concentración de magnesio y de alcohol nos describe un vino con uva de clase 1; así como una convinación de concentración de no flavonoides, ácido málico, y un mayor color nos habla de un vino con unva de clase 3; por último una convinación de ausencia de ácido málico, color, prolines, concentración de magnesio y de alcohol nos describe un vino con uva de clase 2.

A pesar de la mejoria en la separación de nuestros datos en este nuevo espacio, no es lo sufciente como para poder hacer una aseveración fuerte en la clasifiación del tipo de uva de un vino, por lo que seguiremos indagando con diferentes tipos de reducciones de dimensionalidad.

ISOMAP

[ ]:
```
iso = manifold.Isomap(n_neighbors=10, n_components=2)
iso.proj = iso.fit_transform(data_norm)
graph_proj(source_data, iso.proj[:,0:2], title="ISOMAP")
```

Tras varias pruebas de distintos parametros para el número de vecinos, me quede con el 10 al ser de los que mayor separación de las clases tiene. Logramos encontrar una mejor separación de los tipos de uva 3 con el resto, pero de nuevo nos encontramos con una dificultad al tratar de diferenciar entre los tipos de uva 1 y 2. Trataremos de obtenemos información sobre las influencia de las características que tienen sobre estas proyecciones.

[ ]:
```
graph_characteristic(data_norm, iso.proj[:,0:2], title="ISOMAP")
```

De nuevo constatamos que relaciones entre tipo de uva y características que el pca nos había arrojado se repiten, aunque en el caso del alcohol con una mayor acentuación, asi como con el tono, el radio de absorción y el color. Intentaremos con un último metodo y con la esperanza de encontrar buenos parámetros

T-SNE

```
[ ]: tsne = TSNE(n_components=2, learning_rate='auto', early_exaggeration=12.0,␣
      ↪perplexity=25, init='random')
     tsne.proj = tsne.fit_transform(data_norm)
     graph_proj(source_data, tsne.proj[:,0:2], title="T-SNE")
```

A pesar de haber realizado diversas pruebas, devido a la sensibilidad de este a sus parámetros no se encontro una buena clasificación. Aunque se encontro una mayor separación entre las uvas de tipo 1 y 2, el cuál no se había podido encontrar, a costa de un poco de difusión entre las uvas de tipo 2 y 3. Proseguiremos a la información de las características.

```
[ ]: graph_characteristic(data_norm, tsne.proj[:,0:2], title="T-SNE")
```

Podemos observar como los phenoles, los flavonoides y los prolines son los que nos ayudan en esta proyección a diferenciar entre tipos de uva 1 y 2.

Debido a la falta de contundencia de estos metodos en encontrar una fuerte separación de los datos, obtamos por seguir analizando los datos.

1.4 Clasificadores

1.4.1 KNN

Iniciaremos por fijarnos en la clasificación que nos da el KNN, para ello dividiremos nuestros datos en dos conjuntos uno de entrenamiento y otro de prueba, donde la relación sea de 1:4. Para evitar diferentes respuestas en la ejecución de esta celda se fijara el random_state con el valor 7. Por último se tomará el valor esperado de la etiqueta de cada observación de nuestro conjunto de prueba, la cuál coincide con la convinación convexa de las etiquetas cuyos coeficientes son las probablidades de pertenecer a esa etiqueta (deben estar todas a la misma distnacia para que no se pondere de mas una).

Para la visualización de los datos haremos uso de los proyecciones que obtuvimos con el pca, sin embargo el entrenamiento del KNN lo haremos en su espacio original.

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(data_norm,␣
      ↪source_data['Class'], test_size=0.2, random_state=7)

     k=33
     knn = neighbors.KNeighborsClassifier(k)
     knn.fit(X=X_train, y=y_train)

     fig = px.scatter(
         pca.transform(X_test), x=0, y=1,
         color=np.squeeze(knn.predict_proba(X_test)@np.array([source_data["Class"].
      ↪map(int).unique()]).T),
```

```
    symbol=y_test, symbol_map={1:'square',2:'circle',3:'diamond'},
    labels={'symbol': 'Wine Class', 'color': 'Expected <br>class score'}
)

fig.update_traces(marker_size=12, marker_line_width=1.5)
fig.update_layout(
    template=theme,
    title="KNN",
    width=1000,
    height=1000,
    legend_orientation='h'
)
fig.update_xaxes(title='PC1', showgrid=True, showline=True, zeroline=False)
fig.update_yaxes(title='PC2', showgrid=True, showline=True, zeroline=False,␣
 ↪scaleanchor="x", scaleratio=1)
fig.show()
```

Observamos que en efecto que el valor esperado de las etiquetas para el conjunto de prueba es muy cercano a su verdadero valor, claro con un poco de confusión en la frontero como esperariamos pero con mejor separación que cualquier de los algoritmos anteriores. Con ello descartamos que en las ejecuciones haya sido mera casualidad su buen etiqueteo. Ahora haremos una serie de corridas con todos los posibles cantidades vecinos para indagar la eficacia del algoritmo, y poder dar una estimación de su poder

[ ]:
```
scores = pd.DataFrame(columns=['k','score'])
for k in range(1, int(data_norm.shape[0]*0.8)):
    for i in range(30):
        X_train, X_test, y_train, y_test = train_test_split(data_norm,␣
 ↪source_data['Class'], test_size=0.2)
        knn = neighbors.KNeighborsClassifier(k)
        knn.fit(X=X_train, y=y_train)
        scores.loc[len(scores)] = {'k': k, 'score':knn.score(X=X_test,y=y_test)}
scores = scores[['k','score']].groupby(['k']).mean()

fig = px.line(x=np.array(range(1, int(data_norm.shape[0]*0.8))), y=np.
 ↪array(scores).squeeze())
fig.update_layout(
    template=theme,
    title="KNN-Scores/neighbours",
    width=1800,
    height=500,
    legend_orientation='h'
)
fig.update_xaxes(title='Neighbours')
fig.update_yaxes(title='Score')
fig.show()
```

Notamos que en varios valores sobre todo con pequeños valores ha podido estar cercano a 1 el score

del algoritmo, por lo que concluimos que KNN clasifica de una muy buena manera nuestros datos.