Estimação, Deteção e Análise II

07 – Redução da dimensionalidade

Principal Component Analysis (PCA)

Análise discriminante linear

Self organising map (SOM)

Principal Component Analysis

PCA para visualização de datasets (visualizar o dataset Iris, que tem 4D)

Carregar o dataset Iris:

```
import pandas as pd
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
df = pd.read_csv(url, names=['sepal length','sepal width','petal length','petal
width','target'])
```

Standardizar (escalar) o dataset

```
from sklearn.preprocessing import StandardScaler
features = ['sepal length', 'sepal width', 'petal length', 'petal width']
# Separating out the features
x = df.loc[:, features].values# Separating out the target
y = df.loc[:,['target']].values# Standardizing the features
x = StandardScaler().fit_transform(x)
```

Fazer a projeçao do dataset para 2D (em vez de 4D)

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component
1', 'principal component 2'])
finalDf = pd.concat([principalDf, df[['target']]], axis = 1)
```

Visualizar a projeção 2D

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['target'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
plt.show()
```

Verificar a explained variance

```
print(pca.explained_variance_ratio_)
# Resultado:
# [0.72770452 0.23030523]
# 1ª componente explica 73% da variância
# 2ª componente explica 23%
# Juntas explicam cerca de 96% da informação nos dados
```

PCA para acelerar algoritmos de machine learning (usando o dataset MNIST[1], uma base dados de carateres manuscritos, com 784 features – 784D, 60 mil exemplos no trainset e 10 mil no testset)

Ler o dataset mnist_784

```
import pandas as pd
df          =          pd.read_csv('C:/Users/Catarina/Desktop/Aulas/Materia/DataMining/07-
ReduçãoDimensionalidade/CSV/mnist_784.csv');
```

Separar o dataset em conjuntos de treino e teste

```
from sklearn.model_selection import train_test_split
# test_size: what proportion of original data is used for test set
train_img, test_img, train_lbl, test_lbl = train_test_split(df.iloc[:,:-1], df.iloc[: ,
-1], test_size=1/7.0, random_state=0)
```

Standardizar o dataset

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler() # Fit on training set only.
scaler.fit(train_img) # Apply transform to both the training set and the test set.
train_img = scaler.transform(train_img)
test_img = scaler.transform(test_img)
```

Importar e aplicar PCA (no trainset)

```
from sklearn.decomposition import PCA # Make an instance of the Model
pca = PCA(.95)
pca.fit(train_img)
```

Aplicar o mapeamento (transformação) aos conjuntos de treino e de teste

```
train_img = pca.transform(train_img)
test_img = pca.transform(test_img)
```

Aplicar Logistic Regression aos dados transformados

```
from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression(solver = 'lbfgs')
logisticRegr.fit(train_img, train_lbl)
```

Prever para apenas uma observação (imagem)

```
print("Previsao da imagem 0: ",logisticRegr.predict(test_img[0].reshape(1,-1)))
```

```
Output:
Previsao da imagem 0: [0]
```

Prever múltiplas observações

```
print("Previsao das 10 primeiras imagens: ", logisticRegr.predict(test_img[0:10]))
```

```
Output:
Previsao das 10 primeiras imagens: [0 4 1 2 4 7 7 1 1 7]
```

Medir a performance do modelo

```
print("Performance do modelo: ", logisticRegr.score(test_img, test_lbl))
```

```
Output:
Performance do modelo: 0.9201
```

[1] http://yann.lecun.com/exdb/mnist/

Análise discriminante linear

Importar as bibliotecas

```
from sklearn.datasets import load_wine
import pandas as pd
import numpy as np
np.set_printoptions(precision=4)
from matplotlib import pyplot as plt
import seaborn as sns
sns.set()
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

Carregar o datset wine

```
wine = load_wine()
X = pd.DataFrame(wine.data, columns=wine.feature_names)
y = pd.Categorical.from_codes(wine.target, wine.target_names)

# Verificar as dimensoes do dataset
print("dimensoes: ", X.shape)

# Visualizar uma porção do dataset
print("porcao:\n", X.head())

# Verificar que há 3 tipos de vinho no dataset
print("classes: ", wine.target_names)

# Criar um dataframe com as features e o target
df = X.join(pd.Series(y, name='class'))
```

LDA manual

```
class_feature_means = pd.DataFrame(columns=wine.target_names)
for c, rows in df.groupby('class'):
    class_feature_means[c] = rows.mean()

within_class_scatter_matrix = np.zeros((13, 13))
for c, rows in df.groupby('class'): rows = rows.drop(['class'], axis=1)

s = np.zeros((13, 13))
for index, row in rows.iterrows():
    x, mc = row.values.reshape(13, 1), class_feature_means[c].values.reshape(13, 1)

    s += (x - mc).dot((x - mc).T)

within_class_scatter_matrix += s

feature_means = df.mean()
between_class_scatter_matrix = np.zeros((13, 13))
for c in class_feature_means:
    n = len(df.loc[df['class'] == c].index)

    mc,         m         =         class_feature_means[c].values.reshape(13,         1),
feature_means.values.reshape(13, 1)

    between_class_scatter_matrix += n * (mc - m).dot((mc - m).T)

eigen_values,                                eigen_vectors                                =
np.linalg.eig(np.linalg.inv(within_class_scatter_matrix).dot(between_class_scatter_matri
x))

pairs     =     [(np.abs(eigen_values[i]),     eigen_vectors[:,i])     for     i     in
range(len(eigen_values))]
pairs = sorted(pairs, key=lambda x: x[0], reverse=True)
for pair in pairs:
    print(pair[0])
```

```
eigen_value_sums = sum(eigen_values)
print('Explained Variance')
for i, pair in enumerate(pairs):
    print('Eigenvector {}: {}'.format(i, (pair[0]/eigen_value_sums).real))

w_matrix = np.hstack((pairs[0][1].reshape(13,1), pairs[1][1].reshape(13,1))).real

X_lda = np.array(X.dot(w_matrix))
le = LabelEncoder()
y = le.fit_transform(df['class'])

plt.xlabel('LD1')
plt.ylabel('LD2')
plt.scatter(
    X_lda[:,0],
    X_lda[:,1],
    c=y,
    cmap='rainbow',
    alpha=0.7,
    edgecolors='b'
)
plt.show()
```

LDA automático

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
X_lda = lda.fit_transform(X, y)

print(lda.explained_variance_ratio_)

plt.xlabel('LD1')
plt.ylabel('LD2')
plt.scatter(
    X_lda[:,0],
    X_lda[:,1],
    c=y,
    cmap='rainbow',
    alpha=0.7,
    edgecolors='b'
)
plt.show()

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X, y)

print(pca.explained_variance_ratio_)

plt.xlabel('PC1')
plt.ylabel('PC2')
plt.scatter(
    X_pca[:,0],
    X_pca[:,1],
    c=y,
    cmap='rainbow',
    alpha=0.7,
    edgecolors='b'
)
plt.show()
```

SOM

Implementar um SOM para o dataset Iris e verificar as previsões efetuadas

```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import datasets

from sklearn_som.som import SOM

# Load iris data
iris = datasets.load_iris()
iris_data = iris.data
iris_label = iris.target

# Extract just two features (just for ease of visualization)
iris_data = iris_data[:, :2]

# Build a 3x1 SOM (3 clusters)
som = SOM(m=3, n=1, dim=2, random_state=1234)

# Fit it to the data
som.fit(iris_data)

# Assign each datapoint to its predicted cluster
predictions = som.predict(iris_data)

# Plot the results
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(5,7))
x = iris_data[:,0]
y = iris_data[:,1]
colors = ['red', 'green', 'blue']

ax[0].scatter(x, y, c=iris_label, cmap=ListedColormap(colors))
ax[0].title.set_text('Actual Classes')
ax[1].scatter(x, y, c=predictions, cmap=ListedColormap(colors))
ax[1].title.set_text('SOM Predictions')
plt.savefig('iris_example.png')
```