Estimation, Detection and Analysis II

07 - Dimensionality reduction
Principal Component Analysis (PCA)
Linear discriminant analysis
Self organizing map (SOM)

Main Component Analysis

PCA for dataset visualization (visualize the Iris dataset, which has 4D)

Load the Iris dataset:

```
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'])
```

Standardize (scale) the 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)
```

Project the dataset to 2D (instead of 4D)

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

View the 2D projection

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Main Component 1', fontsize = 15)
ax.set_ylabel('Main 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, 'main component 1']
, finalDf.loc[indicesToKeep, 'main component 2']
, c = color
, s = 50)
ax.legend(targets)
ax.grid()
plt.show()
```

Check the explained variance

```
print(pca.explained_variance_ratio_)
# Result:
# [0.72770452 0.23030523]
# 1st component explains 73% of the variance
# 2nd component explains 23%
# Together they explain about 96% of the information in the data
```

PCA to accelerate machine learning algorithms (using the MNIST dataset [1], a database of handwritten characters, with 784 features – 784D, 60,000 examples in the trainset and 10,000 in the testset)

Read the mnist_784 dataset

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

Split the dataset into training and test sets

```
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)
```

Standardize the 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)
```

Import and apply PCA (in trainset)

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

Apply mapping (transformation) to training and test sets

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

Apply Logistic Regression to transformed data

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

Predict for just one observation (image)

```
print("Prediction of image 0: ",logisticRegr.predict(test_img[0].reshape(1,-1)))
```

```
Output:
Image preview 0: [0]
```

Predict multiple observations

```
print("First 10 images prediction: ", logisticRegr.predict(test_img[0:10]))
```

```
Output:
Prediction of the first 10 images: [0 4 1 2 4 7 7 1 1 7]
```

Measure model performance

```
print("Model performance: ", logisticRegr.score(test_img, test_lbl))
```

```
Output:
Model performance: 0.9201
```

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

Linear discriminant analysis

import the libraries

```
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
```

Load the wine datset

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

# Check the dimensions of the dataset
print("dimensions: ", X.shape)

# View a portion of the dataset
print("portion:\n", X.head())

# Check that there are 3 types of wine in the dataset
print("classes: ", wine.target_names)

# Create a dataframe with the features and the target
df = X.join(pd.Series(y, name='class'))
```

Manual LDA

```
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))).

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()
```

automatic LDA

```
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()
```

SOUND

Implement a SOM for the Iris dataset and check the predictions made

```
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)
sound = SOUND(m=3, n=1, dim=2, random_state=1234)

# Fit it to the date
sound.fit(iris_data)

# Assign each datapoint to its predicted cluster
predictions = sound.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')
```