Machine Learning with Python

Manual by Themistoklis Diamantopoulos

Machine Learning with Python
Notes for a tutorial compiled by Themistoklis Diamantopoulos

For more information check website:
`https://thdiaman.github.io/machinelearning/`
or check the following QR code:

1. Classification with Decision Trees

1.1 Example 1 - Categorical Data

```
import pydotplus
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_graphviz

# Read the data
data = pd.read_csv("weather.csv", sep=';', dtype='category')

# Transform the data
#print(data)
data = data.apply(lambda x: pd.factorize(x)[0])

# Split the data
X_train = data.iloc[:, 0:3]
Y_train = data[["Play"]]

# Train the classifier
model = DecisionTreeClassifier(criterion="gini", min_samples_split=2,
    min_samples_leaf=1)
model.fit(X_train, Y_train)

# Plot the tree
dot_data = export_graphviz(model, out_file=None, feature_names=X_train.
    columns.values, \
                           class_names=["No", "Yes"], proportion=True,
                              rounded=True)
graph = pydotplus.graph_from_dot_data(dot_data)
# Visualize using IPython
from IPython.display import Image, display
display(Image(graph.create_png()))
# Visualize using matplotlib
#from io import BytesIO
#import matplotlib.pyplot as plt
#import matplotlib.image as mpimg
#plt.figure()
#img = mpimg.imread(BytesIO(graph.create_png()))
#imgplot = plt.imshow(img,  aspect='equal')
#plt.axis('off')
#plt.show()

# Make a new prediction
print(model.predict([[0, 0, 0]]))
```

1.2 Example 2 - Numerical Data

```
import pydotplus
import numpy as np
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.metrics import classification_report, confusion_matrix

# Read the data
data = pd.read_csv("weatherfull.csv", sep=';', dtype='float')

# Plot the data
fig, ax = plt.subplots()
groups = data.groupby('Play')
for name, group in groups:
    ax.plot(group.iloc[:, 0], group.iloc[:, 1], marker='o', linestyle='',
        label=name)
ax.set_xlabel('Temperature')
ax.set_ylabel('Humidity')
ax.legend()

# Split the data
X = data.iloc[:, 0:2]
y = data.iloc[:, 2]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
    random_state=4)

# Train the model
model = DecisionTreeClassifier(criterion="gini", min_samples_split=20,
    min_samples_leaf=1)
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Plot the tree
dot_data = export_graphviz(model, out_file=None, proportion=True, \
                           feature_names=X_train.columns.values,rounded=True)
graph = pydotplus.graph_from_dot_data(dot_data)
# Visualize using IPython
#from IPython.display import Image, display
#display(Image(graph.create_png()))
# Visualize using matplotlib
from io import BytesIO
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
plt.figure()
img = mpimg.imread(BytesIO(graph.create_png()))
imgplot = plt.imshow(img,  aspect='equal')
plt.axis('off')
plt.show()
```

2. Classification with Naive Bayes

2.1 Example 1 - Categorical Data

```
import pandas as pd
from sklearn.naive_bayes import BernoulliNB

# Read the data
data = pd.read_csv("traffic.csv", sep=';', dtype='category')

# Transform the data
data = data.apply(lambda x: pd.factorize(x)[0])

# Split the data
X = data.iloc[:, 0:2]
Y = data.iloc[:, 2]

# Train the model (set also alpha for smoothing)
model = BernoulliNB()
model.fit(X, Y)

# Make predictions
print(model.predict([[0, 0]]))
print(model.predict_proba([[0, 0]]))
```

2.2 Example 2 - Numerical Data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
    roc_curve, precision_recall_curve, auc

# Read the data
data = pd.read_csv("trafficfull.csv", sep=';')
X = data.iloc[:, 0:-1]
y = data.iloc[:, -1]

# Plot the data
fig, ax = plt.subplots()
groups = data.groupby('HighTraffic')
for name, group in groups:
    ax.plot(group.iloc[:, 0], group.iloc[:, 1], marker='o', linestyle='',
        label=name)
ax.set_xlabel('Temperature')
ax.set_ylabel('Wind')
ax.legend()

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
    random_state=42)
```

```
# Train the model
model = GaussianNB()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
y_pred_prob = model.predict_proba(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Plot the Precision Recall Curve
#precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob
    [:, 1])
#precision = np.insert(precision, 0, 0)
#recall = np.insert(recall, 0, 1)
#plt.plot(precision, recall, 'b')
#plt.ylabel('Precision')
#plt.xlabel('Recall')
#plt.show()

# Plot the ROC curve
fpr, tpr, threshold = roc_curve(y_test, y_pred_prob[:, 1])
plt.figure()
plt.plot(fpr, tpr, 'b')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
print(auc(fpr, tpr)) # print the AUC
```

3. Classification with SVM

3.1 Example

```
import numpy as np
import pandas as pd
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, cross_val_score

# Read the data
data = pd.read_csv("loans.csv", sep=';')
columns = data.columns.values
X = data.iloc[:, 0:2]
y = data.iloc[:, 2]

# Split to training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
    random_state=42)
traindata = pd.DataFrame(data=np.c_[X_train, y_train], columns=columns)

# Plot data
fig, ax = plt.subplots()
groups = traindata.groupby(columns[2])
for name, group in groups:
    ax.plot(group.iloc[:, 0], group.iloc[:, 1], marker='o', linestyle='',
        label=name)
ax.set_xlabel(columns[0])
ax.set_ylabel(columns[1])
ax.legend()

# Create meshgrid
x0_min, x0_max = X.iloc[:, 0].min() - 1, X.iloc[:, 0].max() + 1
x1_min, x1_max = X.iloc[:, 1].min() - 1, X.iloc[:, 1].max() + 1
x0, x1 = np.meshgrid(np.arange(x0_min, x0_max, 0.02), np.arange(x1_min,
    x1_max, 0.02))

# RBF kernel with gamma 1
model = SVC(kernel='rbf', gamma=1, C=1.0)
model.fit(X_train, y_train)
Z = model.predict(np.c_[x0.ravel(), x1.ravel()])
Z = Z.reshape(x0.shape)
CS = ax.contour(x0, x1, Z, colors=['blue'])
labels = ax.clabel(CS, fmt="gamma=1")

# RBF kernel with gamma 0.01
model = SVC(kernel='rbf', gamma=0.01, C=1.0)
model.fit(X_train, y_train)
Z = model.predict(np.c_[x0.ravel(), x1.ravel()])
Z = Z.reshape(x0.shape)
CS = ax.contour(x0, x1, Z, colors=['red'])
labels = ax.clabel(CS, fmt="gamma=0.01")
```

```
# RBF kernel with gamma 100
model = SVC(kernel='rbf', gamma=100, C=1.0)
model.fit(X_train, y_train)
Z = model.predict(np.c_[x0.ravel(), x1.ravel()])
Z = Z.reshape(x0.shape)
CS = ax.contour(x0, x1, Z, colors=['gray'])
labels = ax.clabel(CS, fmt="gamma=100")

# Find training and testing error
gammavalues = [0.01, 0.1, 1, 10, 100, 1000, 10000, 100000, 1000000]
trainingerror, testingerror = [], []
for gamma in gammavalues:
    model = SVC(kernel='rbf', gamma=gamma, C=1.0)
    model.fit(X_train, y_train)
    trainingerror.append(1 - accuracy_score(y_train, model.predict(X_train)))
    testingerror.append(1 - accuracy_score(y_test, model.predict(X_test)))

# Plot training and testing error
fig, ax = plt.subplots()
ax.plot(trainingerror, label="Training Error")
ax.plot(testingerror, label="Testing Error")
ax.set_xticks(range(len(gammavalues)))
ax.set_xticklabels(gammavalues)
ax.set_xlabel("gamma")
ax.legend()

# Find best gamma using cross validation
accuracies = []
for gamma in gammavalues:
    model = SVC(kernel='rbf', gamma=gamma, C=1.0)
    scores = cross_val_score(model, X_train, y_train, cv=10)
    accuracies.append(np.mean(scores))

# Plot accuracy vs gamma
fig, ax = plt.subplots()
ax.plot(accuracies)
ax.set_xticks(range(len(gammavalues)))
ax.set_xticklabels(gammavalues)
ax.set_xlabel("gamma")
ax.set_ylabel("Accuracy")

plt.show()
```

4. Regression with KNN

4.1 Example 1 - Classification using KNN

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Read the data
data = pd.read_csv("salary.csv", sep=';')
X = data.iloc[:, 0:-1]
y = data.iloc[:, -1]

# Convert problem to classification
y = np.digitize(y, [1500])
data.iloc[:, -1] = y

# Plot the data
fig, ax = plt.subplots()
groups = data.groupby('Salary')
for name, group in groups:
    ax.plot(group.iloc[:, 0], group.iloc[:, 1], marker='o', linestyle='',
        label=name)
ax.set_xlabel(data.columns.values[0])
ax.set_ylabel(data.columns.values[1])
ax.legend()
plt.show()

# Split to training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
    random_state=42)

# Train the model
model = KNeighborsClassifier(n_neighbors=4)
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

4.2 Example 2 - Regression using KNN

```
import numpy as np
import pandas as pd
from math import sqrt
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Read the data
data = pd.read_csv("salary.csv", sep=';')
X = data.iloc[:, 0:-1]
y = data.iloc[:, -1]

# Plot the data
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(X.iloc[:, 0], X.iloc[:, 1], y)
ax.set_xlabel(data.columns.values[0])
ax.set_ylabel(data.columns.values[1])
ax.set_zlabel(data.columns.values[2])
plt.show()

# Split to training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
    random_state=42)

# Train the model
model = KNeighborsRegressor(n_neighbors=4)
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
print(sqrt(mean_squared_error(y_test, y_pred)))
print(r2_score(y_test, y_pred))
```

5. Linear Regression

5.1 Example

```
import pandas as pd
from math import sqrt
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Read data
data = pd.read_csv("houses.csv", sep=';')
X = data.iloc[:, 0].values.reshape(-1, 1)
y = data.iloc[:, 1]

# Fit model to data
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

# Plot data and model
fig, ax = plt.subplots()
ax.scatter(X, y, color='black')
ax.plot(X, y_pred, color='blue', linewidth=3)
ax.set_xlabel(data.columns.values[0])
ax.set_ylabel(data.columns.values[1])
#plt.show()

# Show equation
print('Equation: y = %.2f + %.2f * x' %(model.intercept_, model.coef_))

# Calculate metrics
print("Mean Absolute Error: %.2f" %mean_absolute_error(y, y_pred))
print("Mean Squared Error: %.2f" %mean_squared_error(y, y_pred))
print("Root Mean Squared Error: %.2f" %sqrt(mean_squared_error(y, y_pred)))
print("Coefficient of Determination: %.2f" %r2_score(y, y_pred))
```

6. Polynomial Regression

6.1 Example

```
import numpy as np
import pandas as pd
from math import sqrt
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Read data
data = pd.read_csv("ac_data.csv", sep=';')
X = data.iloc[:, 0].values.reshape(-1, 1)
y = data.iloc[:, 1]

# Plot the data
#plt.plot(X, y, 'o')
#plt.xlabel(data.columns.values[0])
#plt.ylabel(data.columns.values[1])

# Create test space
X_test = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)

# Apply linear regression model
model = LinearRegression()
model.fit(X, y)
y_pred_d1 = model.predict(X_test)

# Show equation for first model
print('Equation: y = ' + str(model.intercept_) + ' + ' + str(model.coef_))

# Apply polynomial regression model
degree = 4
poly_features = PolynomialFeatures(degree = degree, include_bias = False)
model = LinearRegression()
model.fit(poly_features.fit_transform(X), y)
y_pred_d4 = model.predict(poly_features.fit_transform(X_test))

# Show equation for second model
print('Equation: y = ' + str(model.intercept_) + ' + ' + \
        ' + '.join(str(model.coef_[d]) + ' * x^' + str(d+1) for d in range(
            degree)))

# Plot the two models
plt.figure()
plt.plot(X, y, 'o')
plt.plot(X_test, y_pred_d1, linewidth=3, label="degree=1")
plt.plot(X_test, y_pred_d4, linewidth=3, label="degree=" + str(degree))
plt.legend()
plt.xlabel(data.columns.values[0])
plt.ylabel(data.columns.values[1])
plt.show()
```

```
# Calculate RMSE for different polynomial degrees
errors = []
degrees = list(range(1, 11))
for degree in degrees:
        poly_features = PolynomialFeatures(degree = degree)
        X_poly = poly_features.fit_transform(X)
        model = LinearRegression()
        model.fit(X_poly, y)
        y_pred = model.predict(X_poly)
        errors.append(sqrt(mean_squared_error(y, y_pred)))

plt.figure()
plt.plot(degrees, errors)
plt.xlabel('Degree')
plt.ylabel('RMSE')
plt.show()
```

7. Feature Selection

7.1 Example 1 - Feature Selection using Correlation

```
import numpy as np
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Read the data
iris = datasets.load_iris()
data = pd.DataFrame(data=np.c_[iris['data'], iris['target']], columns=iris['
    feature_names'] + ['target'])
X, y = data.iloc[:, 0:4], data.iloc[:, 4]

# Print the correlation matrix
print(X.corr())

# Plot the data
axes = pd.plotting.scatter_matrix(X)
plt.show()
```

7.2 Example 2 - Feature Selection using Mutual Information

```
import numpy as np
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, mutual_info_classif

# Read the data
iris = datasets.load_iris()
data = pd.DataFrame(data=np.c_[iris['data'], iris['target']], columns=iris['
    feature_names'] + ['target'])

# Split the data
trainingdata, testdata = train_test_split(data, test_size=0.30, random_state
    =2)
X_train, y_train = trainingdata.iloc[:, 0:4], trainingdata.iloc[:, 4]
X_test, y_test = testdata.iloc[:, 0:4], testdata.iloc[:, 4]

# Apply classifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))

# Keep 3 out of 4 features
fselector = SelectKBest(mutual_info_classif, k=3)
fselector.fit(X_train, y_train)
# and apply classifier
model = DecisionTreeClassifier()
model.fit(fselector.transform(X_train), y_train)
y_pred = model.predict(fselector.transform(X_test))
print(classification_report(y_test, y_pred))

# Keep 2 out of 4 features
fselector = SelectKBest(mutual_info_classif, k=2)
fselector.fit(X_train, y_train)
# and apply classifier
model = DecisionTreeClassifier()
model.fit(fselector.transform(X_train), y_train)
y_pred = model.predict(fselector.transform(X_test))
print(classification_report(y_test, y_pred))

# Keep 1 out of 4 features
fselector = SelectKBest(mutual_info_classif, k=2)
fselector.fit(X_train, y_train)
# and apply classifier
model = DecisionTreeClassifier()
model.fit(fselector.transform(X_train), y_train)
y_pred = model.predict(fselector.transform(X_test))
print(classification_report(y_test, y_pred))
```

8. Feature Extraction

8.1 Example 1 - Feature Extraction

```
import numpy as np
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D

# Read data
pdata = pd.read_csv("pdata.csv", sep=';')

# Plot data
fig = plt.figure()
ax = Axes3D(fig)
for index, row in pdata.iterrows():
    ax.scatter(row['X'], row['Y'], row['Z'], color='b')
    ax.text(row['X'], row['Y'], row['Z'], " x" + str(index + 1))
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

# Apply principal component analysis
pca = PCA(n_components=2)
pca.fit(pdata)
print(pca.explained_variance_ratio_)

# Transform the data
pdata_transformed = pca.transform(pdata)
pdata_transformed = pd.DataFrame(data={'PC1': pdata_transformed[:, 0], 'PC2':
    pdata_transformed[:, 1]})

# Plot data
fig, ax = plt.subplots()
for index, row in pdata_transformed.iterrows():
    ax.scatter(row['PC1'], row['PC2'], color='b')
    ax.text(row['PC1'], row['PC2'], " x" + str(index + 1))
ax.set_xlabel('PC1')
ax.set_xlabel('PC2')
plt.show()
```

8.2 Example 2 - Classification using Feature Extraction

```
import numpy as np
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Read data
pdata = pd.read_csv("wine.csv", sep=';')

# Split the data
trainingdata, testdata = train_test_split(pdata, test_size=0.30, random_state
    =2)
X_train, Y_train = trainingdata.iloc[:, 0:-1], trainingdata.iloc[:, -1]
X_test, Y_test = testdata.iloc[:, 0:-1], testdata.iloc[:, -1]

# Scale the data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Apply classifier
model = KNeighborsClassifier(n_neighbors=4)
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)
print(classification_report(Y_test, Y_pred))

# Apply PCA
pca = PCA(n_components=X_train.shape[1])
pca.fit(X_train)

# Plot variance ratio
variance = pca.explained_variance_ratio_
xticks = list(range(len(variance)))
plt.bar(xticks, variance)
plt.xticks(xticks, ['PC' + str(i+1) for i in xticks])
plt.xlabel('Principal Components')
plt.ylabel('Percentage of Variance')
plt.show()

# Transform data
X_train_transformed = pca.transform(X_train)
X_test_transformed = pca.transform(X_test)

# Apply classifier using first 4 PCs
PCs = 4
model = KNeighborsClassifier(n_neighbors=4)
model.fit(X_train_transformed[:, :PCs], Y_train)
Y_pred = model.predict(X_test_transformed[:, :PCs])
print(classification_report(Y_test, Y_pred))
```

9. Clustering with K-Means

9.1 Example

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

# Read data
kdata = pd.read_csv("gre.csv", sep=';')

# Plot the data
fig, ax = plt.subplots()
ax.scatter(kdata.iloc[:, 0], kdata.iloc[:, 1])
plt.xlabel(kdata.columns.values[0])
plt.ylabel(kdata.columns.values[1])
plt.show()

# Apply kmeans
kmeans = KMeans(n_clusters=3)
kmeans.fit_predict(kdata)
labels = kmeans.labels_
centers = kmeans.cluster_centers_

# Plot clustering result
fig, ax = plt.subplots()
ax.scatter(kdata.iloc[:, 0], kdata.iloc[:, 1], c=labels)
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', marker='+', s=100);
plt.xlabel(kdata.columns.values[0])
plt.ylabel(kdata.columns.values[1])

# Silhouette
silhouette = silhouette_score(kdata, labels)
print(silhouette)

# Silhouette plot
silhouette_values = silhouette_samples(kdata, labels)
fig, ax = plt.subplots()
y_lower = 10
for i in range(len(centers)):
    ith_cluster_silhouette_values = silhouette_values[labels == i]
    ith_cluster_silhouette_values.sort()
    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i
    ax.fill_betweenx(np.arange(y_lower, y_upper), 0,
       ith_cluster_silhouette_values)
    ax.text(-0.025, y_lower + 0.5 * size_cluster_i, str(i))
    y_lower = y_upper + 10
ax.axvline(x=silhouette, color="red", linestyle="--")
ax.set_yticks([])
```

```
# Select number of clusters
fig, ax = plt.subplots()
cluster_nums = [2, 3, 4, 5, 6, 7, 8]
silhouettes = []
for n_clusters in cluster_nums:
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit_predict(kdata)
    labels = kmeans.labels_
    silhouettes.append(silhouette_score(kdata, labels))
ax.plot(cluster_nums, silhouettes)
ax.set_xlabel("Number of clusters")
ax.set_ylabel("Silhouette")

plt.show()
```

10. Hierarchical Clustering

10.1 Example 1 - Single and Complete

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

# Read the data
X = np.array([[50,30], [24,10], [85,70], [71,80], [60,78], [70,55], [28,91]])
data = pd.read_csv("data.csv", sep=';')
indexes = ["x" + str(i + 1) for i in range(len(data))]

# Plot the data with labels
fig, ax = plt.subplots()
for index, row in data.iterrows():
    ax.scatter(row['X'], row['Y'], color='b')
    ax.text(row['X'], row['Y'], " x" + str(index + 1))
ax.set_xlabel('X')
ax.set_ylabel('Y')

# Apply hierarchical clustering
Z = linkage(X, method='single') # or complete

# Plot the dendrogram
fig, ax = plt.subplots()
dendrogram(Z, labels = indexes, color_threshold=100)

# Get the labels for 3 clusters
labels = fcluster(Z, 3, criterion='maxclust')
print(labels)

# Plot the data with labels
fig, ax = plt.subplots()
for index, row in data.iterrows():
    ax.scatter(row['X'], row['Y'], color=['b', 'r', 'g'][labels[index] - 1])
    ax.text(row['X'], row['Y'], " x" + str(index + 1))
ax.set_xlabel('X')
ax.set_ylabel('Y')
plt.show()
```

10.2 Example 2 - Scaling

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

# Read the data
data = pd.read_csv("europe.csv", sep=';')
countries = data.iloc[:, 0].tolist()
X = data.iloc[:, 1:]

# Scale the data
X = MinMaxScaler().fit_transform(X)

# Apply hierarchical clustering
Z = linkage(X, method='complete')

# Plot the dendrogram given a threshold
threshold = 0.4 # alternatively use the number of clusters, i.e. k = 6, and
    then threshold = Z[-(k-1), 2]
fig, ax = plt.subplots()
dendrogram(Z, labels = countries, color_threshold = threshold, leaf_rotation
    = 90)
ax.axhline(y = threshold, c = 'k')
fig.subplots_adjust(bottom=0.35)

# Print the labels given the same threshold
labels = fcluster(Z, threshold, criterion='distance') # or fcluster(Z, k,
    criterion='maxclust')
print(labels)

plt.show()
```

References

Any references used to create this tutorial can be found at the following address:
https://thdiaman.github.io/machinelearning/modules/references/references/