Machine Learning with Python

Exercises

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 Exercise

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

# Plot the data
fig, ax = plt.subplots()
groups = data.groupby('target')
for name, group in groups:
    ax.plot(group.iloc[:, 0], group.iloc[:, 1], marker='o', linestyle='',
        label=name)
ax.set_xlabel('Sepal length')
ax.set_ylabel('Sepal width')
ax.legend(iris['target_names'])

# Split the data
X = data.iloc[:, 0:2]
y = data.iloc[:, 4]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
    random_state=42)

# Train the model
model = DecisionTreeClassifier(criterion="gini", min_samples_split=2,
    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, target_names = iris['target_names
    ']))

# Plot the tree
dot_data = export_graphviz(model, out_file=None, feature_names=X_train.
    columns.values, \
                                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()
```

2. Classification with SVM

2.1 Exercise

```
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 C 100
model = SVC(kernel='rbf', gamma=1, C=100)
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="C=100")

# RBF kernel with C 0.1
model = SVC(kernel='rbf', gamma=1, C=1)
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="C=1")
```

```python
# RBF kernel with C 10000
model = SVC(kernel='rbf', gamma=1, C=10000)
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="C=10000")

# Find training and testing error
Cvalues = [0.01, 1, 100, 10000]
trainingerror, testingerror = [], []
for C in Cvalues:
    model = SVC(kernel='rbf', gamma=1, C=C)
    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(Cvalues)))
ax.set_xticklabels(Cvalues)
ax.set_xlabel("C")
ax.legend()

# Find best C using cross validation
accuracies = []
for C in Cvalues:
    model = SVC(kernel='rbf', gamma=1, C=C)
    scores = cross_val_score(model, X_train, y_train, cv=10)
    accuracies.append(np.mean(scores))

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

plt.show()
```

3. Polynomial Regression

3.1 Exercise

```python
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_absolute_error, mean_squared_error, r2_score
from mpl_toolkits.mplot3d import Axes3D

# Read data
data = pd.read_csv("ad_data_3d.csv", sep=';')

# Plot data
fig = plt.figure()
ax = Axes3D(fig)
a, b, c = data.columns.values
for index, row in data.iterrows():
    ax.scatter(row[a], row[b], row[c], color='b')
ax.set_xlabel(a)
ax.set_ylabel(b)
ax.set_zlabel(c)
plt.show()

# Apply linear regression model
X = data.iloc[:, 0:2]
y = data.iloc[:, 2]
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

# Show equation for first model
print('Equation: y = ' + \
        str(model.intercept_) + ' + ' + \
        str(model.coef_[0]) + ' * x ' + ' + ' + \
        str(model.coef_[1]) + ' * y ')

# 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))

# Apply polynomial regression model
degree = 2
poly_features = PolynomialFeatures(degree = degree, include_bias = False)
model = LinearRegression()
X_poly = poly_features.fit_transform(X)
model.fit(X_poly, y)
y_pred = model.predict(X_poly)
```

```
# Show equation for second model
print('\nEquation: y = ' + str(model.intercept_) + ' + ' + \
        str(model.coef_[0]) + ' * x ' + ' + ' + \
        str(model.coef_[1]) + ' * y ' + ' + ' + \
        str(model.coef_[2]) + ' * x^2 ' + ' + ' + \
        str(model.coef_[3]) + ' * xy ' + ' + ' + \
        str(model.coef_[4]) + ' * y^2 ')

# 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))
```

4. Feature Selection

4.1 Exercise

```
import numpy as np
import pandas as pd
from sklearn import datasets
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, chi2

# 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(chi2, 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(chi2, 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(chi2, 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))
```

5. Clustering with K-Means

5.1 Exercise

```python
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_ex.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])

# Apply kmeans
kmeans = KMeans(n_clusters=4)
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([])
#plt.show()
```

```python
# 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()
```

6. HierarchicalClustering

6.1 Exercise

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

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

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

# Plot the dendrogram given a threshold
threshold = 30 # 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)
ax.axhline(y = threshold, c = 'k')
fig.subplots_adjust(bottom=0.25)

# 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/