21/3/24 A-24-3-24

Week-1

Impost & Expost a csv file with pandas in python:

Code:

```
import pandas as pd
df = pd.import_csv("file_path.csv")
df.head()
```

OUTPUT:

Id	Sepal Length	Sepal Width	Petal Length	..
0	.	.	.	.
1	.	.	.	.
2	.	.	.	.
3	.	.	.	.
4	.	.	.	.

Expost code:

```
df.to_csv("path\new_name.csv")
```

Kaggle Completed: 3 chapters out of 6

1) Creating, Reading, Writing

→ pd.DataFrame({'Yes': [0,1], 'No': [1,2]})

→ pd.DataFrame ({ }, index = [ ])

→ pd.Series ([ ], index = [ ],
name = ' ')

→ pd.read_csv (" ")

→ new_file.shape()

→ new_file.head()

→ new_file.to_csv(' ')

2) Indexing, Selecting, Assigning

→ df.iloc [ ]

→ df.iloc [[row_index], [col_index]]

→ df.loc [[row_name/index],
[col_name/index]]

→ df.set_index (" ")

→ df.specific_col_name == 'name'

→ df.loc [df.specific_col_name == 'name']

→ df.specific_col_name.isin ([ ])

→ | → or , & → and

→ not null to make sure if a row is not null.

3) Summary Functions & Maps:

→ df.specific_col_name.describe()
* gives count, unique, top & most frequent.

→ df.number_col.mean()

→ df.col.unique()

→ df.col.value_counts()
* To see unique sets & how often they appear.

→ df.head (number)

→ df_mean = df.col.mean()
df.col.map (lambda p: p - df_mean)

28/3/24

Week-2

End-End ML Project:

Get the Data:

```
import os
import tarfile
import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/
                 handson-ml2/master/"

HOUSING_PATH = os.path.join("data", "01")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.
                               tgz"

def fetch_housing_data(housing_url = HOUSING_URL,
                       housing_path=HOUSING_PATH):
    os.makedir(name = housing_path, exist_ok = True)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(url = housing_url,
                               filename = tgz_path)
    housing_tgz = tarfile.open(name=tgz_path)
    housing_tgz.extractall(path= housing_path)
    housing_tgz.close()

# to download data, call the function
fetch_housing_data()
```

3. Discover & Visualize the data to gain insights.

```
#latitute/longitude info can be represented
 using .plot()
housing.plot(kind='scatter', x='longitude',
             y='latitude')
plt.show()
# To estimate densities, use 'alph' parameter
housing.plot(kind='scatter', x='longitude'
             y='latitude', alpha=*0.1)
plt.show()

# To check the relation between two & more
 data info, 'corr' method is used.
housing[['population', 'median_house_value']].
                                    corr()
# This generates a weak correlation b/w the
 specified parameters.
# For a better correlation, we need to do,
corr_matrix = housing.corr()
corr_matrix['median_house_value'].
                    sort_values(ascending
                    =False)
```

4. Prepare Data for ML Algorithms.

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='median')
# this can handle only numbers
housing_num = housing.drop("ocean_proximity",
                           axis=1)

# Now we fit imputer over our data
# fitting is basically training.
imputer.fit(housing_num)

# Nowe we can use this to transform the
numbers by replacing their missing
values with corresponding medians.
X = imputer.transform(housing_num)

# X is a numpy array with the transform
-d features. We add it back to the
dataframe.
housing_tr = pd.DataFrame(data=X, index=
                housing_num.index,
                columns=housing_num
                .columns)
```

5. Select & Train a Model:

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X=housing_prepared, y=housing_labels)
some_data = housing.iloc[:5] # lets take
some_labels = housing_labels.iloc[:5] small amount
                                      of the data
some_data_prep= full_pipeline.transform(some_
                                        data)
# some_labels.tolist() & lin_reg.predict(some_data_prep)
  we use another method:

from sklearn.metrics import mean_squared_
                            error.

housing_pred = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels,
                             housing_predictions
                             )
lin_rmse = np.sqrt(lin_mse)
lin_rmse # this is the root, mean
           squared error which is
           more accurate.

# linear regression has better performance but
  lower accuracy when compared to
  Decision tree & Random forest.
# linear regression is represented as y=mx+c
```

11 Random Forest is a combination of multiple Decision Trees. So, it is better at capturing non-linear relationships.

4/4/24

Week-9

Simple Linear Regression: 4/4/24

```
import numpy as np
import matplotlib.pyplot as plt.
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from pandas.core.common import random_state
from sklearn.linear_model import LinearRegression

df_sal = pd.read_csv("/content/Sal_Data.csv")
df_sal.head()
df_sal.describe()
plt.title("Salary Distribution Plot")
sns.displot(df_sal['Salary'])
plt.show()

plt.scatter(df_sal['Years Exp'], df_sal['Sal'],
            color='lightcoral')
plt.title('Sal vs Exp')
plt.xlabel('Exp')
plt.ylabel('Salary')
 plt.box(False)
 plt.show()
```

Split Data:

```
x = df_sal.iloc[:,:1]
 y = df_sal.iloc[:, 1:]
```

Split into Train test sets:

X_train, X_test, y_train, y_test
= tt train_test_split(X, y, test_size = 0.2,
random_state = 0)

Train Model:

regressor = LinearRegression()
regressor.fit(X_train, y_train)

Predict Results:

y_pred_test = regressor.predict(X_test)
y_pred_train = regressor.predict(X_train)

Visualize Predictions:

plt.scatter(X_train, y_train, 'lightcoral')
plt.plot(X_train, y_pred_train, color='firebrick')
plt.title('Salary vs Exp (Training Set)')
plt.xlabel('Years of Exp')
plt.ylabel('Salary')
plt.legend(['X_train / Pred(y_test)',
'X_train / y_train'], title='Sal/Exp',
loc='best', facecolor='white')
plt.box(False)
plt.show()

plt.scatter(X_test, y_test, color='light coral')
plt.plot(X_train, y_pred_train, color='firebrick')
plt.title('Sal vs Exp (Test Set)')
plt.xlabel('Years of Exp')
plt.ylabel('Sal')

plt.show()

Coefficient and Intercept

print(f'Coeff: {regressor.coef_}')
print(f'Intercept: {regressor.intercept_}')

Output:

Coeff: [[9312.57]]
Intercept: [26780.099]

Multiple Linear Regression:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seabon as sns
from sklearn.model_selection import
                                train_test_split as
                                tts
from sklearn.linear_model import LinearRegression

df_start = pd.read_csv ('/content/startup.csv')
df_start.head()
df_start.describe()
```

Distribution:

```
plt.title ('Profit Distribution Plot')
sns.distplot (df_start ['Profit'])
plot.show()
```

Relation b/w profit & R&D spend

```
plt.scatter (df_start['R&D Spend'], df_start['Profit'],
             color='light coral')
plt.title ('Profit vs R&D Spend')
plt.xlabel ('R&D spend')
plt.ylabel ('Profit')
plt.box (False)
plt.show()
```

Split into Independent/Dependent variables:

```
x = df_start.iloc[:, :, -1].values
y = df_start.iloc[:, :, -1].values
```

One-hot encoding:

```
ct = ColumnTransformer (transformers = [('encoder',
        OneHotEncoder(), [3])], remainder='
        'passthrough')

x = np.array (ct.fit_transform (x))
```

Split into train/test Data:

```
X_train, X_test, y_train, y_test = ~~train~~tts (x, y,
                                   test_size =0.2,
                                   random_state = 0)
```

Train Model:

```
regressor = LinearRegressor()
regressor.fit (X_train, y_train)
```

Predict Results:

```
y_pred = regressor.predict (X_test)
```

Compare Predictions:

```
np.set_print options (precision=2)
result = np.concatenate ((y_pred.reshape (len(
                          y_pred),1),
         y_test.reshape (len (y_test),1)),1)
```

18/4/24

Week-4

Decision Tree

Prelab/24

Import Libraries:

```
from sklearn.datasets import make_classification, load_iris
from sklearn import tree
from sklearn.model_selection import train_test_split as tts
from numpy as np
```

Load the dataset

```
iris = load_iris()
X = iris.data[:, 2:]
y = iris.target
X_train, y_train, X_test, y_test =
    tts(X, y, test_size = 0.3, shuffle = True,
        random_state = 1)
```

DecisionTree Classifier

```
model = tree.DecisionTreeClassifier()
model = model.fit(X_train, y_train)
```

Predicting:

```
predicted_value = mode.predict(X_test)
print(f" Pred Val : {predicted_value}")
```

Pred Val : [0 1 1 0 2 1 0 0 2 1 0 2 1 1 0 1
1 0 0 1 1 2 0 2 1 0 0 1 2 1 2 1 2
2 0 1 0 1 2 2 0 1 2 1 7

Plot the Tree graph:

```
tree.plot_tree(model)
zeros = np.sum(y_train == 0)
ones = np.count_nonzero(y_train)
val = 1 - ((zeros/70)**2 + (ones/70)**2)
print(f'No. of zeros : {zeros}\n
      No. of non-zeros : {ones}\n
      Gini : {val}")
```

O/P:

No. of zeros : 36
No. of non-zeros : 69
Gin: -0.23612244897959179

Accuracy:

```
match = sum([1 for i in range(30) if
             predicted_value[i] ==
             y_test[i]])
acc = match/30
print(f" Accuracy : {acc}")
```

O/P:

a Accuracy is 0.967

25/4/24 Week-5

Logistic Regression and Sigmoid Functions

Imports

```
# import pandas as pd
  from matplotlib. import pyplot as plt
  from sklearn.model_selection import train_test_split as tts.
from sklearn.linear_model import LogisticRegression as LR
  import math
% matplotlib. inline
```

Read the Dataset (csv file)

```
df = pd. read_csv("insurance_data.csv")
df.head()
```

Data Exploration & Visualization:

```
plt.scatter (df.age, df.bought_insurance,
             marker = "+", color = 'red')
# x    y
```

Perform Split:

```
X_train, X_test, y_train, y_test = tts(df [['age']],
        df.bought_insurance, train_size = 0.6,
        random_state = 42)
```

Create & fit the Model:

```
model = LR()
model.fit (X_train, y_train)
X_test  # display.
```

Predict

```
y_predicted = model.predict (X_test)
y_predicted # display
```

Accuracy:

```
model.predict_proba (X_test)
model.score ( X_test, y_test)
```

→ 0.9090

Sigmoid functions:

```
model.coef_
```

→ array ([[0.12335856]])

```
model.intercept_
```

→ array ([-4.87081951])

Analyze Sigmoid:

```
def sigmoid (x):
    return 1 / (1 + math.exp(-x))
```

Predict function :

```
def predict_function (age):
    z = 0.042 * age - 1.53
    y = sigmoid (z)
    return y
```

Check Value with probability :

```
age = 35
prediction_function (age)
```

→ 0.485

```
age = 88
Prediction_function (age)
```

→ 0.89715

9/5/24

Week 8

KA 4/5

KNN & SVM

```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
.|. matplotlib inline
```

Read The Dataset

```
df = pd.read_csv ("/content/Iris.csv")
```

Show the Plot :

```
plt.show()
plt.figure (figsize = (10, 6))
sns.scatterplot (data = df, x = df ["Petal Length"]
                 y = df ["Sepal Width"]
                 legend = False, hue = df ["Species"]
                 )
```

Add Labels & Title to Plot :

```
plt.xlabel ("Petal Length")
plt.ylabel ("Sepal Width")
plt.title ("Petal Length VS Sepal Width")
plt.show()
```

KNN :

```
from sklearn.model_selection import test_train_split as tts
from sklearn.neighbours import KNeighbourClassifier
X_train, X_test, Y_train, Y_test = tts (X, y, 0.6, 0)
neigh = KNeighbourClassifier (n_neighbours = 2)
neigh.fit (X, y)
```

```
y_pred = neigh.predict(X_test)
acc = accuracy_score(y_pred, y_test)
print(acc)
```

→ 1.0

SVM :

```
from sklearn.svm import SVC
model = SVC()
model.fit(X, y)
y_pred = model.predict(X_test)
acc = accuracy_score(y_pred, y_test)
print(acc)
```

→ 1.0

23/5/24 — Thu 23-5-24 — Week-7

ANN :

```
import numpy as np
X = np.array(([2,9],[1,5],[3,6]), dtype=float)
y = np.array(([92],[86],[89]), dtype=float)
X = X/np.amax(X, axis=0)
y = y/100
```

Initialize All Variables:

```
epoch = 100
lr = 0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
```

Initialize Weights and Bias :

```
wh = np.random.uniform(size=(inp_neu, hid_neu))
bh = np.random.uniform(size=(1, hid_neu))
wout = np.random.uniform(size=(hid_neu, out_neu))
bout = np.random.uniform(size=(1, out_neu))
```

Sigmoid Function :

```
def sigmoid(x):
    return 1/1+(np.exp(-x))
#derivative
def derivatives_sigmoid(x):
    return x * (1-x)
```

Driver Code:

```
for i in range(epoch):
    hinp1 = np.dot(X, wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    out_p1 = np.dot(hlayer_act, wout)
    out_p = out_p1 + bout
    output = sigmoid(out_p)

    EO = y - output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)

    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hidlay = EH * hiddengrad
//end for
wout += hlayer_act.T.dot(d_output) *lr
wh += X.T.dot(d_hidlay) *lr
print(f"Input: {X} \n
      Actual Output: {y} \n
      Predicted Output: {output}")
```

Output:

Input:

```
[[0.66  1.
  0.33  0.55
  1.0   0.66]]
```

Actual Output:

```
[[0.92]
 [0.86]
 [0.89]]
```

Predicted Output:

```
[[0.84552]
 [0.8406]
 [0.8460]]
```

Random Forest :

```
from sklearn.ensemble import RandomForestClassifier as
                                                    rfc
import pandas as pd
from sklearn.metrics import accuracy_score as acc
```

LOAD DATASET :

```
df = pd.read_csv("/content/drive/MyDrive/
                    melb_data.csv")
data = df.dropna(axis=0)
# Choose target & features
y = data.Price
m_features = ['Rooms', 'Bathroom', 'Landsize',
'BuildingArea', 'Year Built', 'Lattitude',
'Longitude']
X = data[m_features]
```

Driver Code :

```
from sklearn.model_selection import train_test_spli
                                        as tts
X_train, X_test, y_train, y_test = tts(X, y, train_size=0.
                                    random_state=0)
model = rfc()
model.fit(X, y)
y_pred = model.predict(X_test)
a = acc(y_pred, y_test)
print(a)
```

→ 0.9935

AdaBoost :

#Load Data same as Random Forest

X_train, X_test, y_train, y_test = tts (X, y, train_size = 0.7, random = 0)

from sklearn.ensemble import AdaBoostClassifier as ada

modul = ada(n_estimators = 50)

modul.fit (X, y)

y_pred = modul.predict (X_test)

acc = acc (y_pred, y_test)

print(a)

→ 1.0 ≈.

30/5/24

Week-8

K-means

A 30-5-24

Imports:

import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

LOAD the Dataset:

iris = datasets.load_iris ()
x = pd. DataFrame(iris.data)
X.columns = ['Sepal Length', 'Petal Length', 'Sepal Width', 'Petal Width']
y = pd. DataFrame (iris.target)
y. columns = ['Targets']

fit the model:

modul = KMeans (n_clusters = 3)
modul.fit (X)

Plot the figures:

plt. figure (figsize = (14,14))
colormap = np. array (['red', 'lime', 'black'])

```
plt.subplot(2,2,1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets])
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.subplot(2,2,2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_])
plt.title('K Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

PCA

Imports:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
```

Import Dataset:

```
from sklearn.datasets import load_breast_cancer
cancer.keys()
df = pd.DataFrame(cancer['data'], columns =
                  cancer['feature_names'])
```

PCA Visualization:

```
from sklearn.preprocessing import StandardScaler
                                  as SS
scaler = SS()
scaler.fit(df)
scaled_data = scaler.transform(df)
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(scaled_data)
```

```
x_pca = pca.transform(scaled_data)
scaled_data.shape
x_pca.shape
```

Plot the Graph:

```
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0], x_pca[:,1],
            c=cancer['target'], cmap='plasma')
plt.xlabel('First Principal component')
plt.ylabel('Second Principal Component')
```