INDEX

NAME: SNEHA SANTHOSH BHAT STD: 6th SEC: D ROLL NO: 1BM21CS213

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	21/3/24	Importing and Exporting Data using Pandas library functions	10	
2.	28/3/24	End to End Machine Learning Project	10	
3.	04/04/24	Simple Linear Regression and Multiple Linear Regression	10	
4.	18/4/24	Decision Tree	10	
5.	25/4/24	Logistic Regression	10	
6.	09/05/24	Build KNN Classification model for a given dataset	10	
7.	09/05/24	Build Support Vector machine model for a given dataset	10	
8.	23/05/24	Build Artificial Neural Network model with back propagation for a given dataset	10	
9.	23/05/24	Random Forest, Adaboost	10	
10.	30/05/24	K-Means Algorithm to cluster data	10	
11.	30/05/24	Dimensionality Reduction using PCA method	4	

Date 21/3/24
Page

WEEK - 1

TO import and export data using Pandas library functions

(i) importing data

```
import pandas as pd
airbnb_data = pd.read_csv("data/listings-austin.csv")
airbnb_data.head()
```

(ii) Reading data from URL

```
url = "https://archive.ics.uci.edu/ml/machine-
      learning-databases/iris/iris.data"
col_names = ["sepal_length_in_cm", "sepal_width_in_cm
             "petal_length_in_cm", "petal_width_in_cm"
             "class"]
iris_data = pd.readcsv(url, names = col_names)
iris_data.head()
```

(iii) Exporting dataframe to csv file

```
iris_data.to_csv("cleaned_iris_data.csv")
```

OUTPUT :

	sepal_length_in_cm	sepal_width_in_cm	petal_length_in_cm	petal_width_in_cm	cla
0	5.1	3.5	1.4	0.2	iris. setosa
1					
2					
3					
4					

4/4/24

28/3/24

WEEK - 2

End to End Machine Learning Project

1. Look at the Big Picture
2. Get the Data

(i) Download the data

```
import os
import tarfile
import urllib
DOWNLOAD_ROOT = "https://raw.github.com"
housing_path = os.path.join("data", "01")
housing_url = download.root + "datasets/housing/housing.tag"
def fetch_housing(housing=url = housing_url,
                  housing - path = housing_path):
    os.makedirs(name = housing-path, exit_ok=True
    tgz_path = os.path.join(housing_path,
                            "housing.tgz")
    fetch_housing_data()
import matplotlib.pyplot as plt
import seaborn as sns
housing.hist(bins = 50, figsize = (20,15))
plt.show
housing['income_cat'] = pd.cut(x=housing['median-income'],
                               bins=[0,1.5,3,4.5,6,np.inf],
                               labels=[1,2,3,4,5])
housing('income_cat').hist()
```

3. Discover and visualize the data to gain insights

```
strat_train_set.shape, strat_test_set.shape
strat_test_set.reset_index().to_feather(fname =
                                        'data/01/strat_test_set.ft')
```

```
housing = strat_train_set.copy();
housing.shape
housing.plot(kind='scatter', x='longitude', y='latitude')
plt.show()
corr_matrix = housing.corr()
corr_matrix['median_house_value'].sort_values
                                    (ascending=False)

from pandas.plotting import scatter_matrix
attributes = ['median_house_value', 'median_income',
              'total_income', 'housing_median_age']
scatter_matrix(frame = housing[attributes], figsize=(12,8))
plt.show()
```

4. Prepare the Data for Machine Learning Algorithms

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='median')
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
imputer.statistics_
housing_num.median().values
X = imputer.transform(housing_num)
X.shape
housing_tr = pd.DataFrame(data = X, index=housing_num
                                    columns=housing_num.columns
housing_tr.head()
housing_cat = housing_cat[['ocean_proximity']]
housing_cat.head()
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
num_pipeline = Pipelearn([
        ('imputer', SimpleImputer(strategy='median')),
        ('attributes_adder', CombinedAttributesAdder()),
        ('std_scalar', StandardScaler())
    ])
```

housing_num_tr = num_pipeline.fit transform(housing_num)
housing_num_tr.shape

5. Select and Train a model:

from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X = housing_prepared, y = housing_labels)
housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_label,
housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse

tree_reg = DecisionTreeRegresson()
tree_reg.fit(X = housing_prepared, y = housing_labels)
forest_reg = RandomForestRegresson()
forest_reg.fit(X = housing_prepared, y = housing_labels)
forest_reg.fi

6. Fine - Tune Your Model:

grid_search.best_params_
grid_search.best_estimator_
cat_encoder = full_pipeline.named_
transformers_['cat']
final_model = grid_search.best_estimator_
X_test_prepared = full_pipeline.transform(X=X_test)
final_rmse = np.sqrt(final_mse)
final_rmse
squared_errors = (y_test_final_predictions)
**2

7. Launch, Monitor and Maintain your system

Date 4/4/24
Page

WEEK - 3

Simple Linear Regression and Multiple Linear Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
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(r'C:\Users\STUDENT\Downloads\
                                    Salary_Data.csv
df_sal.head()
df_sal.describe()

plt.title('Salary Distribution Plot')
sns.distplot(df_sal['salary'])
plt.show()

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

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

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

y_pred_test = regressor.predict(X_test)
y_pred_train = regressor.predict(X_train)
print(f'Coefficient: {regressor.coef_}')
print(f'Intercept: {regressor.intercept_}')
```

```
df_start = pd.read_csv(r'C:\Users\STUDENT\Downloads\
df_start.head()                    50_Startups.csv')
df_start.describe()

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

regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

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

Date 18/04/24
Page

WEEK - 4

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample

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

import sklearn.datasets as datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, roc_auc_score
                                              roc_curve
from sklearn.tree import plot_tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV,
                                   RandomizedSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

url = " ... "
df = pd.read_csv (url, names = ['sepal length(cm)', ...]
df.head()

X = df.drop ("Species", axis = 1)
y = df ["Species"]
X_train, X_test, y_train, y_test = train_test_split
                    (X, y, test_size = 0.3
                     random_state=1)

dt = DecisionTreeClassifier (max_depth = 3, min_samples_leaf
                                          =10,
dt.fit (X, y)                             random_state = 1)
```

```
from IPython.display import Image
from sklearn.tree import export_graphviz
!pip install pydotplus
import pydotplus
features = X.columns
dot_data = export_graphviz(dt, out_file = None,
                           feature_name = features)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

```
dt = DecisionTreeClassifier(random_state = 1)
dt.fit(X_train, y_train)
y_pred_train = dt.predict(X_train)
y_pred = dt.predict(X_test)
y_prob = dt.predict_proba(X_test)
```

```
print('Accuracy of Decision Tree-Train:', accuracy_score
                                          (y_pred_train,
                                           y_train))
print('Accuracy of Decision Tree-Test:', accuracy_score
                                          (y_pred, y_test))
print(classification_report(y_test, y_pred))
```

```
dt = DecisionTreeClassifier(random_state = 1)
params = {'max_depth': [2,3,4,5],
          'min_samples_split': [2,3,4,5],
          'min_samples_leaf': [1,2,3,4,5]}
gsearch = GridSearchCV(dt, param_grid = params,
                       cv = 3)
gsearch.fit(X, y)
gsearch.best_params_
```

Date
Page

```
dt = DecisionTree Classifier (** gsearch.best_paranc_,
                                random_state=1)
dt.fit (X_train, y_train)
y_pred_train = dt.predict (X_train)
y_prob_train = dt.predict_proba (X_train) [:, 1]
y_pred = dt.predict (X_test)
y_prob = dt.predict_proba (X_test) [:, 1]
```

Output:

25/4/24

WEEK - 5

Predicting if a person would buy life insurance based on his age using logistic regression

```
import pandas as pd
from matplotlib import pyplot as plt
% matplotlib inline
```

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

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

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split
                (df[['age']], df.bought_insurance,
                                    train_size = 0.8)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit (X_train, y_train)
```

```
y_predicted = model.predict (X_test)
model.predict_proba (X_test)
model.score (X_test, y_test)

model.coef_

model.intercept_

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

```
def prediction - function (age):
    z = 0.042 * age - 1.53  # 0.04150133 ~ 0.042 and
    y = sigmoid (z)          -1.52726963 ~ 1.53
    return y
```

age = 35

prediction - function (age)

Output :-

0.4850044983805899

age = 43

prediction - function (age)

Output :-

0.5685652990777705

09/05/2024

WEEK - 6

Build KNN Classification model for a given dataset

```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
import matplotlib.pyplot as plt

df = pd.read_csv('teleCust1000t.csv')
df.head()

df['custcat'].value_counts()

x = df.drop('custcat', axis=1)
x = x.values
x[0:5]

y = df['custcat'].values
y[0:5]

from sklearn.preprocessing import StandardScaler

x = StandardScaler().fit(x).transform(x.astype(float))
x[0:5]

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split
                                   (x, y, test_size=0.2,
                                    random_state=4)
print('Train set:', x_train.shape, y_train.shape)
print('Test set:', x_test.shape, y_test.shape)
```

Date
Page

```
Ks = 10
mean_acc = []
for n in range (1, Ks):
        neigh = K Neighbors Classifiers (n_neighbors = n) fit
                                        (x_train, y_train)
        yhat = neigh. predict (x_test)
        mean_acc. appe (metrics. accuracy_score
                                        (y_test, yhat))
mean_acc


plt.plot (range (1, Ks), mean_acc)
plt. xlabel ('No of K\'s')
plt. ylabel ('Accuracy')
plt. show ()


print ("Best accuracy was {} with k = {}". format
        (max (mean_acc), max (i+1 for i, _ in
        enumerate(mean_acc))))


neigh = KNeighbours Classifier (n_neighbors = 9). fit (x_train,
                                                y_train)
neigh


yhat = neigh. predict (x_test)
yhat [0:5]


print ("Train set Accuracy :", metrics. accuracy_score(y_train,
                        neigh. predict (x_train))
print ("Test set Accuracy :", metrics. accuracy_score (y_test, yhat))
```

Output :-

Train set Accuracy : 0.5025

Test set Accuracy : 0.34

09/05/2024

WEEK - 7

Build Support vector machine model for a given dataset

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

iris = pd.read_csv('IRIS.csv')

iris.head()

sns.pairplot(data = iris, hue = 'species', palette = 'Set2')

from sklearn.model_selection import train_test_split

x= iris.iloc[:,:-1]
y = iris.iloc[:,4]
x_train, x_test, y_train, y_test = train_test_split
                                   (x, y, test_size = 0.30)

from sklearn.svm import SVC
model = SVC()

model.fit(x_train, y_train)

pred = model.predict(x_test)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

Date
Page

```
model2 = SVC (C = 5, kernel = 'rbf')
model2.fit (x_train, y_train)
pred = model2.predict (x_test)
from sklearn.metrics import classification_report, confusion_matrix
print (confusion_matrix (y_test, pred))
print (classification_report (y_test, pred))


from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score
classifier = KNeighborsClassifier (n_neighbors = 3)
classifier.fit (x_train, y_train)
y_pred = classifier.predict (x_test)


cm = confusion_matrix (y_test, y_pred)
cm


accuracy = accuracy_score (y_test, y_pred) * 100
print ('Accuracy of our model is equal'
                    + str (round (accuracy, 2)) + '%.')


k_list = list (range (1, 50, 2))
cv_scores = []
for k in k_list:
    knn = KNeighborsClassifier (n_neighbors = k)
    scores = cross_val_score (knn, x_train, y_train,
                              cv = 10, scoring = 'accuracy')
    cv_scores.append (scores.mean())
```

Scanned with CamScanner

```
MSE = [1 - x for x in cv_scores]
plt.figure()
plt.figure(figsize = (15, 10))
plt.title('The optimal number of neighbors',
                    fontsize = 20, fontweight = 'bold')
plt.xlabel('Number of Neighbors K', fontsize=15)
plt.ylabel('Misclassification Error', fontsize=15)
sns.set_style("whitegrid")
plt.plot(k_list, MSE)
plt.show()

best_k = k_list[MSE.index(min(MSE))]
print("The optimal number of neighbors is %d."
                    % best_k)
```

Output :-

Accuracy of our model is equal 97.78 %

The optimal number of neighbors is 1

Date 23/05/24
Page

WEEK - 8

8. Build Artificial Nemal Network model with back propagation on a given dataset

```
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

epoch = 5000
lr = 0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1

wh = np.random.uniform(size = (inputlayer_neurons,
                               hiddenlayer_neurons))
bh = np.random.uniform(size = (1, hiddenlayer_neurons))
wout = np.random.uniform(size = (hiddenlayer_neurons,
                                 outputlayer_neurons))
bout = np.random.uniform(size = (1, output_neurons))

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

for i in range(epoch):
    hinpl = np.dot(x, wh)
    hinp = hinpl + bh
    hlayer_act = sigmoid(hinp)
    outinpl = np.dot(hlayer_act, wout)
    outinp = outinpl + bout
    output = sigmoid(outinp)
```

```
EO = y - output
outgrad = derivatives_sigmoid (output)
d_output = EO * outgrad
EH = d_output . dot (wout.T)
hiddengrad = derivatives_sigmoid (hlayer_act)
d_hiddenlayer = EH * ddergrad

wout += hlayer_act.T.dot (d_output) * lr
wh += X.T. dot (d_hiddenlayer) * lr

print ("Input: \n" +str(X))
print ("Actual Output: \n" +str(y))
print ("Predicted Output: \n", output)
```

Output :-

Input :

```
[[ 0.66666667   1.        ]
 [0.3333333    0.55555556]
 [1.           0.66666667]]
```

Actual Output :

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

Predicted Output :

```
[ [0.76630321]
[ 0.75772239]
[ 0.76682594]]
```

Date 23/05/24
Page

WEEK - 9

9a, 9b. Random Forest, Adaboost

```
import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'https%3A...'

KAGGLE_INPUT_PATH = '/kaggle/input'
KAGGLE_WORKING_PATH = '/kaggle/working'
KAGGLE_SYMLINK = 'kaggle'

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
print('Data source import complete')

import numpy as np
import pandas as pd
import numpy as np
from sklearn.svm import SVC, SVR
from sklearn import preprocessing
import matplotlib.pyplot as plt
```

```
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.ensemble import (RandomForestClassifier,
                              ExtraTreesClassifier,
                              AdaBoostClassifier)
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

import os
print(os.listdir("../input"))

dataset = pd.read_csv("../input/faults.csv")
dataset[0:6]

corr_matrix = input_x.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape),
                                  k=1)
to_drop = [column for column in
           upper.columns if any(upper[column] > 0.95)]
                                .astype(np.bool_))
print(to_drop)

X_train, X_test, y_train, y_test = train_test_split
                                   (input_sc_norm, Y,
                                    test_size = 0.3)

n_classes = 7
n_estimators = 100
RANDOM_SEED = 13
names = ['DecisionTree Classifier', 'RandomForest Classifier',
         'ExtraTrees Classifier', 'AdaBoost Classifier']
models = [DecisionTreeClassifier(max_depth=None),
          RandomForestClassifier(n_estimators=n_estimators),
          ExtraTreesClassifier(n_estimators=n_estimators),
          AdaBoostClassifier(DecisionTreeClassifier(max_depth
                                                    =None),
                             n_estimators=n_estimators)]
```

```
for counter, model in enumerate (models):
    model.fit (X_train, y_train)
    y_pred = model.predict (X_test)
    print ("Accuracy " + names [counter] + ":",
           metrics.accuracy_score (y_test, y_pred))
```

```
score = model.evaluate (X_test, y_test, batch_size=128)
print (score)
print (model.metrics_names)
```

```
Y_predicted = model.predict (X_test, batch_size = 32,
                             verbose = 0)
percent = 0
for i in range (0, len (Y_predicted)):
    class_id_predicted = np.argmax (Y_predicted [i])
    class_id_real = np.argmax (y_test [i])
    if class_id_predicted == class_id_real:
        percent += 1
print ('val accuracy : ', percent / len (Y_predicted))
```

Output :-

val accuracy : 0.7409948542024014

30/5/24

WEEK - 10

Build K - Means algorithm to cluster a set of data stored in a .csv file

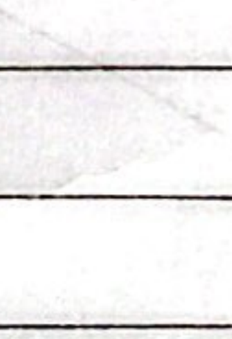
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans

data = pd.read_csv('Iris.csv')

X = data[['SepalLengthCm', 'SepalWidthCm',
          'PetalLengthCm', 'PetalWidthCm']]
model = KMeans(n_clusters = 3)
model.fit(X)

plt.figure(figsize = (14,7))

plt.subplot(1,2,2)
plt.scatter(X['PetalLengthCm'], X['PetalWidthCm'],
            c= model.labels_, cmap
            ='viridis', s= 40)
plt.title('K-Means Clustering')
plt.xlabel('Petal length')
plt.ylabel('Petal Width')
plt.show()
```



Date 30/05/24
Page

WEEK - 11

Implement Dimensionality reduction using Principle Component Analysis (PCA) method

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

from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()

cancer.keys()

print(cancer['DESCR'])

df = pd.DataFrame(cancer['data'], columns=cancer['feature_names'])

df.head()

cancer['target_names']

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

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
```

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

```
pca.components_
```

```
df_comp = pd.DataFrame(pca.components_,
                       columns=cancer
                       ['feature_names'])
df_comp
```

```
plt.figure(figsize=(12,6))
sns.heatmap(df_comp, cmap='plasma')
```

30/9/22