INDEX

Name RIA RATHOD Std Sec

Roll No. Subject School/College

School/College Tel. No. Parents Tel No

Sl. No.	Date	Title	Page No	Teacher Sign / Remarks
1.	21/3/24	import and export using pandas library functions		
2	26/3/24	end-end project		
3	28/3/24	end-end project		
4	16/4/24	Linear & Multiple Linear regression		
5	20/4/24	Decision tree ID3		
6	9/5/24	KNN & Logistic regression		
7	23/5/24	K Means clustering, SVM, PCA		
8	30/5/24	Random forest - Not uploaded		

28/2/24 (W)eek-2

Hands on Machine Learning with Scikit Learn Keras and Tensor Flow

1. Get the Data

Download import OS
the import tarfile
Data import urllib
def fetch_housing_data (housing_url = HOUSING_URL
housing_path = HOUSING_PATH):
urllib.request.urlretrieve (url, housing_tgz = tarfile.open (name = tgz_path)
housing_tgz.extractall (path = housing_path)
housing_tgz.close()
fetch_housing_data().

Load the data

import pandas as pd.
def load_housing_data (housing_path = HOUSING_PATH)
data_path = os.path.join
return pd.read_csv (data_path)
housing.head()
housing.info()
housing['ocean_proximity'].value_counts()
housing.describe()

Create a Test Set

def split_train_test (data, test_ratio = 0.2)
shuffled_indices = np.random.permutation (len(data))
test_set_size = int (len (data) * test_ratio)
test_indices = shuffled_indices [:test_set_size]
return data.iloc [train_indices]

def test_set_check (identifier, test_ratio = 2)
total_size = 2^5 * 32
in_test_set < (test_ratio * total_size)

```
def cantor_pairing(n1, n2)
    n = ((n1+n2) * (n1+n2+1)) / 2) + n2
    return n

def lat_lon_to_index(lat, lon)
    lat, lon = int(lat * 100), int(lon * 100)
    lat, lon = from_2_to_N(lat), from_2_to_N(lon)
    index = cantor_pairing(lat, lon)
    return np.int64(index)

def from_2_to_N(z)
    if z >= 0
        n = z * z;
    else
        n = -z * z - 1
    return n
```

2. Discover and visualize the data to gain insights.

```
strat_train_set_shape, strat_test_set_shape
strat_test_set.sort_index(), to_feather
                                (name = 'data/01')
housing = strat_train_set.copy()    strat_test
```

Visualizing geographical data

```
housing.plot(kind='scatter', x='longitude', y='latitude')
plt.show()

housing.plot(kind='scatter', x='longitude', alpha=0.1)
plt.show()
```

3 Prepare the Data for Machine learning Algorithms

housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
housing.shape, housing_labels.shape

→ Data Cleaning -

from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='median')
imputer.fit(housing_num)
imputer.statistics_
X.shape
housing_num.median().values
housing_tr.head().

→ Custom transformer.

from sklearn.base import TransformerMixin, BaseEstimator
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
Self.add_bedrooms_per_room =
add_bedrooms_per_room
def fit(self, X, y=None):
rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
housing_extra_attribs = attr_adder.transform(housing.values)

Transformation Pipelines -

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])
```

Select and Train a Model

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
some_data = housing.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
from sklearn.tree import DecisionTreeRegressor
tree_reg = DecisionTreeRegressor()
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Fine Tune Your Model

Grid Search

```
from sklearn.model_selection import GridSearchCV
param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10],
     'max_features': [2, 3, 4]},
  ]
forest_reg = RandomForestRegressor()
```

grid_search.best_score_
grid_search.best_estimator_
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)

6 Launch, Monitor and Maintain your system.

week 3

① Linear Regression-

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x,y):
    n = np.size(x)
    m_x = np.mean(x)
    m_y = np.mean(y)
    SS_xy = np.sum(x*y) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x
    b_1 = SS_xy/SS_xx
    b_0 = m_y - b_1*m_x
    return(b_0,b_1)
def plot_regression_line(x,y,b):
    plt.scatter(x,y,color="m",marker="o",s=30)
    y_pred = b[0] + b[1]*x
    plt.plot(x,y_pred,color="g")
    plt.xlabel('x')
    plt.ylabel('y')
def main():
    x = np.array([0,1,2,3,4,5,6,7,8,9])
    y = np.array([1,3,2,5,7,8,8,9,10,12])
    b = estimate_coef(x,y)
print("Estimated coefficients:\nb_0 = {}\
      \nb_1 = {}".format(b[0],b[1]))
```

Output-

(b_0, b_1) = (1.2, 1.1, 1.0363)

Date __/__/__
Page _____

(8) Multiple Linear Regression

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics

data_url = 'url'
raw_df = pd.read_csv(data_url, sep=' ', skiprows=23
                     (header=None)
X = np.hstack([raw_df.values[::2, :], raw_df
                values[1::2, :2]])
y = raw_df.values[1::2, 2]
X_train, X_test, y_train, = train_test_split
    (X, y, test_size=0.4, random_state=1)

reg = linear_model.LinearRegression()
reg.fit(X_train, y_train)
print("coefficients:", reg.coef_)
print("variance score: {}".format(reg.score,
                                  (X_test, y_test))
plt.style.use('fivethirtyeight')
plt.scatter(reg.predict(X_train), reg.predict(X_train)
            y_test, color='blue', s=10, label='train_data')
plt.hlines(y=0, xmin=0, xmax=50, linewidth=2)
plt.legend(loc='upper_right')
plt.title('Residual errors')
plt.show()
```

Week-4

① Decision tree ID3

```
import numpy as np
import pandas as pd
df = pd.read_csv('v1')
df.head()
df.info()
df.describe()
def find_entropy(df):
    target = df.keys()[-1]
    entropy = 0
    values = df[target].unique()
    for value in values:
        fraction = df[target].value_counts()[value]/
                   (len(df[target])
        entropy += fraction * np.log2(fraction)
    return entropy

def build_tree(df, tree=None):
    target = df.keys()[-1]
    node = find_winner(df)
    attr = np.unique(df[node])
    if tree is None:
        tree = {}
        tree[node] = {}
    for value in attr:
        sub = get_subtable(df, node, value)
    clvalue, counts = np.unique(sub[target],
                                return_counts=True)
    if len(counts) == 1:
        tree[node][value] = clvalue[0]
    else:
```

```
tree = buildTree(df)
import pprint
pprint.pprint(tree)
```

DecisionTree (sklearn)

② Decision tree (sklearn):

```
import pandas as pd
import numpy as np
import sklearn.model_selection import DecisionTree
from sklearn.tree import plotTree

df = pd.read_csv('url')
df.head()
df.info()
df.isnull().sum()
cols = df.columns[0:-1]
for i in cols:
    sns.boxplot(y=df[i])
        plt.show()
X = df.drop('species', axis=1)
y = df['species']
dt = DecisionTree(criterion='gini', max_depth=3)
dt.fit(X,y)
y_pred_train = dt.predict(X_train)
y_pred = dt.predict(X_test)
accuracy = score(y_pred, y_test)
```

Week-5

Logistic Regression -

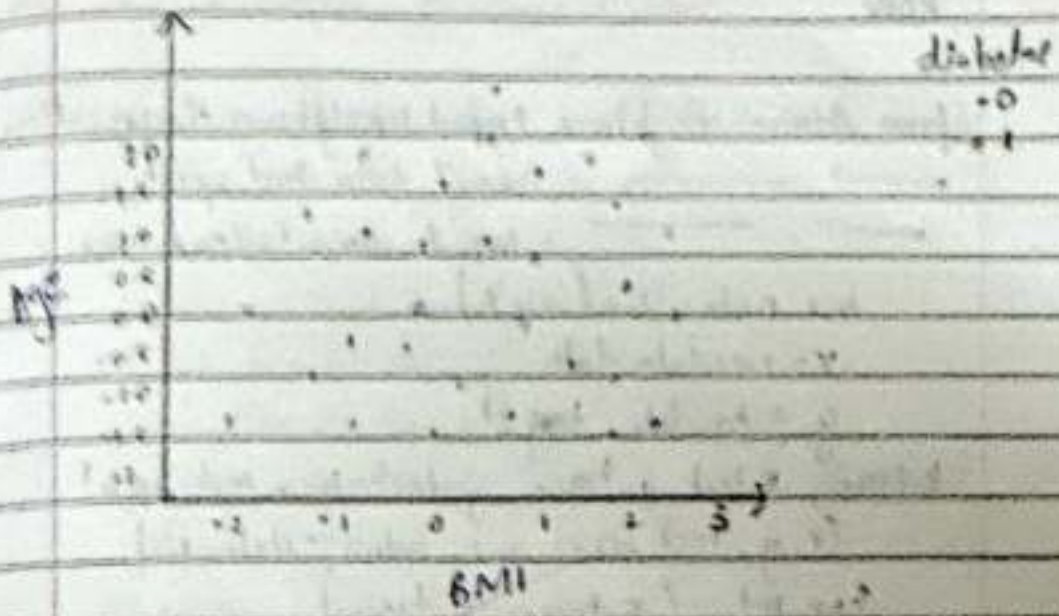
```
import numpy as np
import pandas as pd
import sklearn.datasets import load_diabetes
from sklearn.linear_model import logistic regression
from sklearn.preprocessing import standard scale
from sklearn.metrics import accuracy_score,
    classification_report, confusion_matrix

diabetes = load_diabetes()
    X, y = diabetes.data, diabetes.target
    y_binary = (y > np.median(y)).astype(int)
    X_train, X_test, y_train, y_test = train_test_split
    (X, y_binary, test_size = 0.2, random_state = 42)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    model = LogisticRegression()
    model.fit(X_train, y_train)
print("Accuracy : {:.2f} %".format(accuracy*100))
print("confusion matrix:", confusion_matrix(y_test, y_pred))
print("Classification report:", classification_report(y_test, y_pred))
```

Confusion matrix

```
[[36 13]
 [11 29]]
```

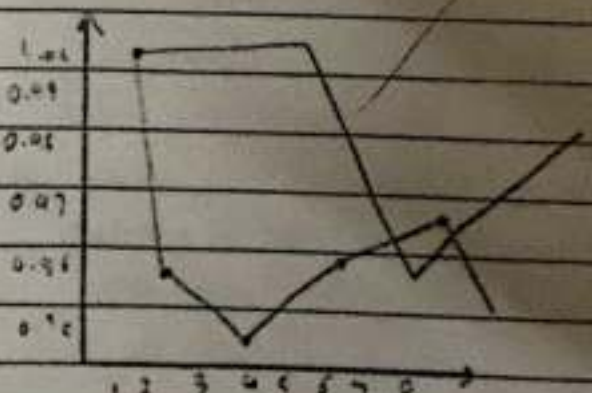
```
sns.scatterplot(x=X_test[:,2], y=X_test[:,8], hue=y_test,
    palette={0:'blue', 1:'red'}, marker='o')
plt.legend(title='Diabetes', loc='upper right')
plt.show()
```


diabetes
0
1
Age
BMI

Date / /
Page

KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
iris_data = load_iris()
x = iris_data.data
y = iris_data.target
x_train, x_test, y_train, y_test = train_test_split
(x, y, test_size = 0.2, random_state = 42)
knn.fit(x_train, y_train)
print(knn.predict(x_test))
neighbors = np.arange(1, 9)
for i, k in enumerate(neighbors):
knn = KNeighborsClassifier(n_neighbors = k)
knn.fit(x_train, y_train)
train_accuracy[i] = knn.score(x_train, y_train)
test_accuracy[i] = knn.score(x_test, y_test)
plt.plot(neighbors, test_accuracy, label = 'Testing dataset
                                            accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset
                                            accuracy')
plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```



23/5/24

* Support Vector Machine (SVM)

```
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.svm import SVC
cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target
SVM = SVC(kernel = "rbf", gamma = 0.5, C = 10)
SVM.fit(X, y)
plt.scatter(X[:, 0], X[:, 1], c = y, s = 20)
plt.show()
```

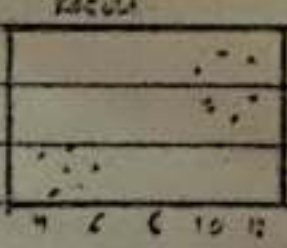
* PCA using sklearn -

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
    pca.fit(2)
X_pca = pca.transform(2)
df_pca = pd.DataFrame(X_pca, columns = ['PC1',
                format('i+1') for i in range(n_components)])
print(df_pca)
```

Output :

	PC1	PC2
0	9.1641	1.9468

* K Means Clustering -




```
from sklearn.cluster import KMeans
data = list(zip(x, y))
inertias = []
for i in range(1, 11):
    kMeans = KMeans(n_clusters = 2)
    kMeans = fit(data)
plt.show()
```

PCA -

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
data.keys()

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

```
scaling = StandardScaler()
scaling.fit(data)
principal = PCA(n_components = 3)
X = principal.transform(scale_data)
```

```
plt.figure(figsize=(12,10))
plt.scatter(x[:,0], [:,1], c=data['target'])
plt.xlabel('x')
plt.ylabel('y')
```

Random Forest -

```
import pandas as pd
import sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
ds = datasets.load_iris()
type(ds)
x = pd.DataFrame(ds.data, columns=ds.feature_names)
y = ds.target
model = RandomForestClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
accuracy_score(y_test, y_pred)
```

Ensemble boosting -

```
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
x = ds.data
y = ds.target
x_train, x_test, y_train, y_test =
    train_test_split(x, y, test_size=0.4)
adaboost = AdaBoostClassifier(base_estimator=
                              30, learning_rate=1)
adaboost.fit(x_train, y_train)
y_pred = adaboost.predict(x_test)
accuracy_score(y_test, y_pred)
```

30.05.21