1BM21CS235

INDEX

Name VAISHNAVI KAMATH Std Sec

Roll No. Subject ML LAB School/College

School/College Tel. No. Parents Tel. No.

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1.	21/3/24	LAB-1	A	
2.	28/3/24	LAB-2	A	
3	4/4/24	LAB-3	A	
4	18/4/24	LAB-4	A	
5.	25/4/24	Insurance system		
6.	9/5/24.	LAB-6.	A	
7	23/5/24	LAB-7	A	
8.	23/5/24	LAB-7		
9.	30/5/24.	LAB-8. PCA	A	
10.	30/5/24	LAB-8 Grocerys.		

Date 21/03/24

Page

LAB-1

```
> import pandas as pd
  url = "https://....."
  df = pd.read_csv(url, names=['sepal_len', 'sepal_width', 'petal_leng',
       'petal_width', 'class'])

  df.head()

> df.to_csv("cleaned_data.csv")
```

	sepal_len	sepal_width	petal_len	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa

PANDAS COURSE

1. Creating, Reading & Writing.

```
import pandas as pd.
* pd.DataFrame({'Yes': [1,2], 'No': [3,4]}, index=['A','B'])
* df

* pd.Series([30,40,50], index=['2015 Sales', '2016 Sales', '2017 Sales'], name='Product A')
```

2. Indexing, Selecting & Assigning.

```
> reviews.country (or) reviews['country']
> reviews.iloc[[0,1,2],0]
> reviews.iloc[-5:]
        ↳ index based sel.
> reviews.loc[:, ['a','b','c']]
```

- reviews.set_index('NI')
- reviews.loc[(reviews.country == 'Italy') & (reviews.points >= 90)]
- reviews.loc[reviews.country.isin(['Italy', 'France'])]
- reviews.loc[reviews.price.notnull()]

3. Summary f^n's & Maps
   - df.describe()
   - df.col.unique()
   - df['A'].value_counts()
   - df.points.map(lambda p: p - mean)
4. Grouping & sorting
   - reviews.groupby(['country', 'province']).apply(lambda df: df.loc[df.points.idxmax()])
   - reviews.groupby(['country']).price.agg([len, min, max])

Date __/__/____

Page _____

```
> countries_reviewed.reset_index()
> df.sort_index()
```

5. Data Types and Missing Values

```
> df.dtypes
> df.points.astype('float64')
> reviews[pd.isnull(reviews.country)]
> reviews['A'].replace('A', 'B')
```

6. Renaming & Combining

```
> reviews.rename(columns={'points': 'score'})
> reviews.rename_axis("wines", axis='rows')
    .rename_axis("fields", axis='columns')
> pd.concat([df1, df2])
> left.join(right, lsuffix='_CAN',
            rsuffix='_UK')
```

End to End Project

28/3/24

Date 28/3/24
Page

```
import os
import tarfile
import urllib.

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

def fetch_housing_data(housing_url=HOUSING_URL,
                       housing_path=HOUSING_PATH):
    os.makedirs(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()

fetch_housing_data()
import pandas as pd
```

```
def lhd(housing_path = HOUSING_PATH):
    data_path = os.path.join(housingpath,
                housing.csv")
    return pd.read_csv(data_path)
housing = load_housing_data()
housing.head()
housing['ocean_proximity'].value_counts()

import matplotlib.pyplot as plt
import seaborn as sns
housing.hist(bins=50, figsize=(20,15))
plt.show()
```

CREATING TEST SET

```
import numpy as np
def split(data, test_ratio = 0.2):
    si = np.random.permutation
         (len(data))
    test = int(len(data)*test_ratio)
    test_ind = si[ : test]
    train_ind = si[test : ]
    return data.iloc[train_ind],
           data.iloc[test_ind]

train, test = split_train_test(data =
              housing)

len(train_set), len(test_set)
```

HASH STRATEGY

```
from zlib import crc32

def test_set_check(identifier, test_ratio):
    total_size = 2**32
    hex_rep = crc32(np.int64(identifier)) & 0xffffffff
    in_test = hex_rep < test_ratio * total_size
    return in_test

def split_train_test_by_id(data, test_ratio, id_column):
    ids = data[id_column]
    in_test = ids.apply(lambda id_: test_set_check(id_, test_ratio))
    return(data.loc[~in_test_set], data.loc[in_test_set])

housing_with_id = housing.reset_index()

train_test = split_train_test_by_id(data = housing_with_id, test_ratio=0.2, id_column = "index");

def z_to_N(z):
    if (z>=0):
        n = 2*z
    else:
```

Date __/__/__
Page

```
n = 2*3.
else
n = -2*(3-1)
```

3. Discover & visualise Data

```
strat_train_set.shape, strat_test_set.shape.
strat_test_set.reset_index().
    to_feather('path=1')

housing = strat_train_set.copy()

housing.plot(kind='scatter'
x='long', y='lat', alpha=0.1)

corr = housing.corr()

corr_matrix['median_h_v'].sort_values
(ascending=False)
```

ATTRIBUTE COMBINATIONS

```
housing['rooms_p_h'] = housing['tot_rooms']
                       housing['households']
```

4. PREPARE DATA FOR ML ALGOS

```
h = strat_tr_set.drop("med_house_value
                      axis=1)
h_labels = strat_train_set['median_
           house_value'].copy()
h.shape, h_labels.shape
```

Data cleaning

```
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.index,columns=housing_num.columns)

housing_tr.head()

housing_cat=housing[['ocean_proximity']]

housing_cat.head(10)

housing_cat['ocean_proximity'].value_counts()
```

Date __/__/__

Page _____

```
from sklearn.preprocessing import
ordinalEncoder

ordinal_encoder = OrdinalEncoder()

housing_cat_encoded[:10]

ordinal_encoder.categories_
```

* from sklearn.preprocessing import OneHotEncoder

```
one_hot_encoder = OneHotEncoder()

housing_cat_1hot = one_hot_encoder.
                   fit_transform(housing_
cat.values)

housing_cat_1hot.toarray()

one_hot_encoder.categories_
```

Date
Page

→ df.values returns numpy representation of a given array.

4/11/24

PART-B

→ Import data
df = pd.read_c
df.head()

→ df.describe()

→ sns.distplot(df[
plt.show()

→ plt.scatter(df
plt.show()

SPLIT DATA

→ X = df.iloc[:
y = df.iloc[

→ X_train, X_test
= train_test_

→ reg = LinearRegres
reg.fit(X_trai

→ y_pred_test =
y_pred_train =

print(reg.coef_
print(reg.inter

4/11/24

Date / /
Page 4

PART-B

Aq

→ Import data
df = pd.read_csv('data.csv')
df.head()

→ df.describe()

→ sns.distplot(df['Salary'])
plt.show()

→ plt.scatter(df['years of'], df['Sal'])
plt.show()

SPLIT DATA

→ X = df.iloc[:, :1]
y = df.iloc[:, 1:]

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

→ reg = LinearRegression()
reg.fit(X_train, y_train)

→ y_pred_test = reg.predict(X_test)
y_pred_train = reg.predict(X_train)

print(reg.coef_)
print(reg.intercept_)

MULTIPLE LINEAR REG.

Date
Page

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

df.describe()

sns.distplot(df['Profit'])

plt.scatter(df['RSD spend'],
            df['Profit'])

plt.show()

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

#One hot encoding of categorical data.
ct = ColumnTransformer(
transformers = [('encoder', OneHot
Encoder(), [3])], remainder = 'passthrough')

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

reg = LinearRegression()
reg.fit(x_train, y_train)

y_pred = reg.predict(x_test)
```

```
res = np.concatenat
(len(y_pred), 1),
(len(y_test), 1))
```

Date / /

Page

→ res = np.concatenate((y_pred.reshape(len(y_pred), 1), y_test.reshape(len(y_test), 1)), 1)

18/4/24

Date
Page

Decision Tree

```
url = "..."
df = pd.read_csv(url, header=None,
     names=['sepallength' ...])

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)

df = DecisionTreeClassifier(max_depth=3,
   min_samples_leaf=10,
   random_state=1)
dt.fit(X_train, y_train)

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
    names = features)
graph = pydotplus.graph_from
     dot_data(dot_data)
Image(graph.create_png())
```

Hyperparameter tuning

```
dt = DecisionTreeCla

params = {'max_
 'min_samples_sp
 'min_samples_

gsearch = GridSear
     param_grid

gsearch.fit(X, y)
gsearch.best_para

dt2 = DecesionTree
   (**gsearch.
    random_
dt2.fit(X_train
y_pred_train = dt
y_pred_train =
   (X_train)[

y_pred = dt2.p
y_prob = dt2.pre

print(accuracy_sc

print(accuracy
```

Date __/__/__
Page

Hyperparameter tuning.

→ 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_
```

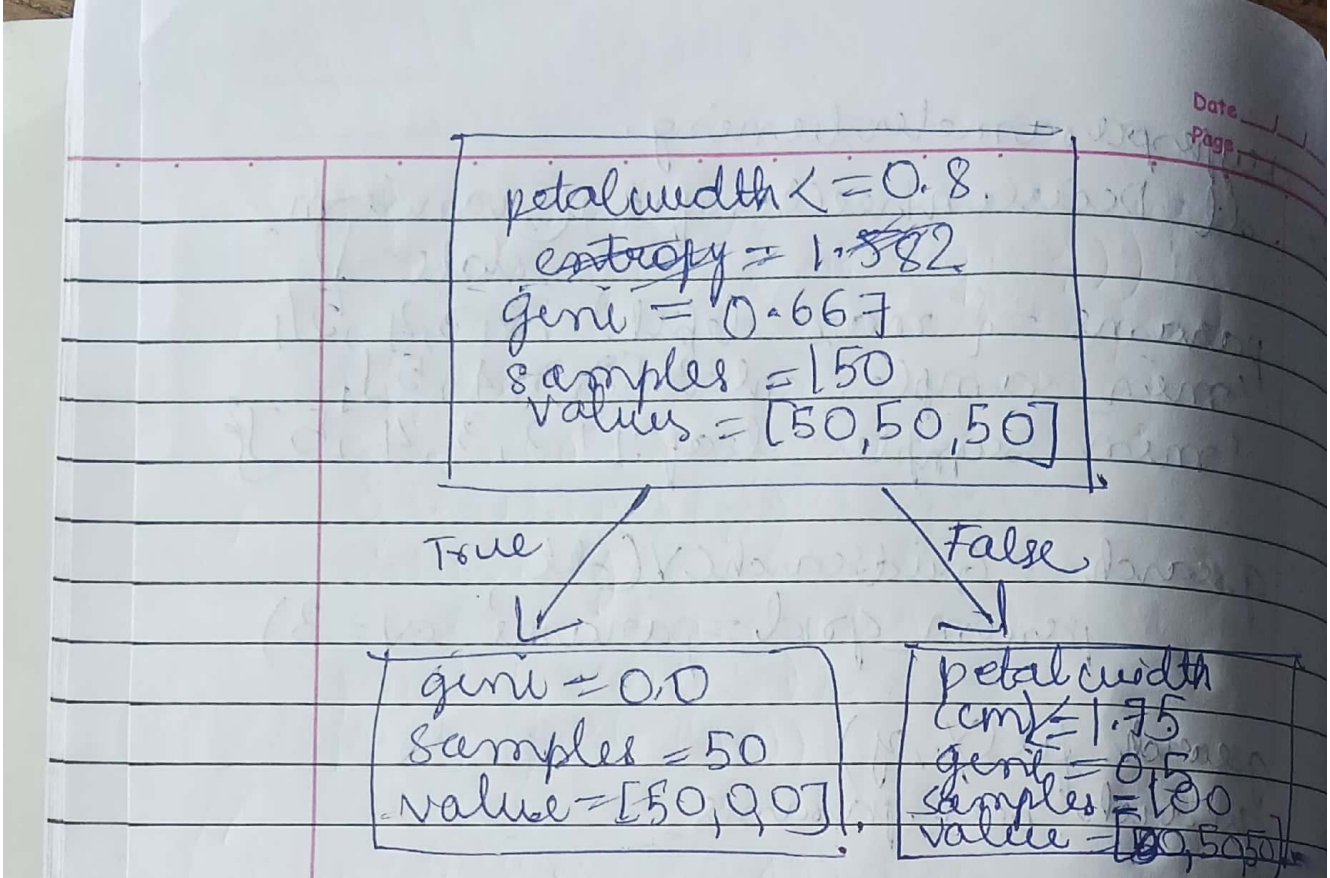
→ dt2 = DecisionTreeClassifier
(**gsearch.best_params_,
random_state=1)

```
dt2.fit(X_train, y_train)
y_pred_train = dt2.predict(X_train)
y_pred_train = dt2.predict_proba
        (X_train)[:,1]
```

```
y_pred = dt2.predict(X_test)
y_prob = dt2.predict_proba(X_test)
                [:,1]
```

→ print(accuracy_score(y_pred_train,
y_train)

→ print(accuracy_score(y_pred,
y_test)


petal width <= 0.8
entropy = 1.582
gini = 0.667
samples = 150
value = [50, 50, 50]
True
False
gini = 0.0
samples = 50
value = [50, 0, 0]
petal width (cm) <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]

25/4/24

Date __/__/__

Page _____

Insurance System

```
import pandas as pd
from matplotlib.pyplot import plt
from sklearn.model_selection
  import train_test_split
from sklearn.linear_model import
  LogesticRegression

df = pd.read_csv('id.csv')

x_train, X_test, y_train, y_test
=train_test_split(df[['age']],
        df.bought_insurance
    test_size=0.2)

model = LogesticRegression()
model.fit(X_train, Y_train)
y_predicted = model.predict(X_test)
print(y_predicted)

print(model.predict_probA(X_test))
print(model.score(X_test, y_test))
```

Accuracy:

0.5

```
def sigmoid(z):
    return 1/(1+Math.exp(-z))

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

print (predi(35))
print (predi(43))
```

Predictions : 0.485
0.5685

9/5/24

SVM - Iris Dataset

```
df = pd.read_csv('IRIS.csv')
sns.pairplot(data = df, hue = 'species',
             palette = 'Set2')

from sklearn.model_selection
    import train_test_split

x = df.iloc[:, :-1]
y = df.iloc[:, 4]
x_train, x_test, y_train, y_test
    = train_test_split(x, y, test_size = 0.3)

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

O/P:

Accuracy : 0.98
Precision : 0.98
Recall : 0.98

KNN

Date
Page

```
from sklearn.neighbours import
  KNeighboursClassifier
from sklearn.metrics import
  confusion_matrix, accuracy_score

cls = KNeighboursClassifier(n_neighbours
                            =3)
cls.fit(x_train, y_train)
y_pred = cls.predict(x_test)

print(confusionmatrix(y_test, y_pred))

print(accuracyscore(y_test, y_pred))
```

CROSS VALIDATIONS

~~MSE = [1-x for x in cv_scores]~~

```
k_list = list(range(1, 50, 2))
cv_scores = []

for k in k_list:
knn = KNeighboursClassifier(n_neighbou
                            =k)
scores = cross_val_score(knn,
    x_train, y_train, cv=10,
    scoring = 'accuracy')

MSE = [1-x for x in cv_scores]
plt.figure()

best_k = k_list[MSE.index(min(MSE))]
print(best_k)
```

O/P:
Optimal no. of neigh
Accuracy = 97.78%

Date / /
Page

O/P:
Optimal no. of neighbours = 7
Accuracy = 97.78%

23/5/24

8. Back propagation A 23-5-24 Date / / Page

→ db = np.loadtxt("duke-breast-cancer.txt")
print(np.shape(db))

→ np.random.shuffle(db)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

→ hidden_layer = np.zeros(72)
weights = np.random.random((len(x[0]), 72))

output_layer = np.zeros(2)
hidden_weights = np.random.random((72, 2))

def back_propagation(hidden_layer, output_layer, one_hot_encoding, learning_rate, x):

op_derivative = np.zeros(2)
op_gradient = np.zeros(2)

for i in range(0, len(output_layer)):
output_derivative[i] = (1.0 - output_layer[i]) * output_layer[i]

for i in range(0, len(op
op_grad[i] = op_derivat
(one_hot_en[i] - output

hidden_derivative = np.ze
hidden_gradient = np.ze

for i in range(0, len(hi
hidden_derivative[i] = (1.
layer[i]) * (1.0 + hidden_

for i in range(0, len(h
sum = 0
for j in range(0, le

sum += output_g
hidden_weights[i]

recalculate_weights(learnin
hidden_weights, output_g
hidden_layer)

for i in range (0, len (output layer)):
op grad[i] = op derivative[i] *
(one hot enc[i] - output layer[i])

hidden derivative = np.zeros(72)
hidden_gradient = np.zeros(72)

for i in range (0, len(hidden layer)):
hidden derivative[i] = (1.0 - hidden
layer[i]) * (1.0 + hidden_layer[i])

for i in range (0, len(hidden layer)):
error = 0
for j in range (0, len(output grad)):
error += output grad[j] *
hidden_weights[i][j]

recalculate_weights (learning_rate,
hidden_weights, output_gradient,
hidden_layer)

23/5/24

9a) & 9b) Random forest & Adaboost

Date
Page

→ dataset = pd.read_csv('faulti.csv')

→ corr_matrix = input_x.corr().abs()

upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

to_drop = [col for col in upper.columns if any(upper[column] > 0.95)]

print(to_drop)

→ x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

→ n_classes = 7
n_estimators = 100
RANDOM_SEED = 13.

names = ['RF classifier', 'AdaBoostClassifier']
models = [RandomForestClassifier(n_estimators = n_estimators), AdaBoostClassifier(DecisionTreeClassifier(max_depth=None), n_estimators = n_estimators)]
for counter, model in enumerate(models):

model.fit(X_train,
y_pred = model.pre
print(names[counter
metrics.accuracy_s

RF = 0.7685
AdaBoostClassifier =

Date __/__/__

Page _____

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

RF = 0.7684

AdaBoostClassifier = 0.7344

30/5/24
Date
Page

K-Means clustering (Iris)

```
→ iris = pd.read_csv('iris.csv')
→ iris.info()
  iris[0:10]
→ iris_outcome = pd.crosstab(
      index = iris['species'],
      columns = 'count')

# Finding optimum no. of clusters
  for kMeans.
from sklearn.cluster import kmeans
wcss = []

for i in range(1,11):
    km = KMeans(n_clusters = i
init = 'kmeans++', max_iter = 300,
n_init = 10, random_state = 0)

kmeans.fit(x)
wcss.append(kmeans.inertia_)

→ kmeans = KMeans(n_clusters = 3
init = 'kmeans++' max_iter = 300,
n_init = 10, random_state = 0)

y_kmeans = kmeans.fit_predict(x)
```

30/5/24

Date

Page

PCA (Breast cancer dataset)

```
cancer = load_breast_cancer()
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
pca.components_
df_comp = pd.DataFrame(pca.components_, columns = cancer['feature_names'])
plt.figure(figsize=(12,6))
sns.heatmap(df_comp, cmap='plasma')
```