LAB-1

→ Importing and Exporting data using Pandas library functions.

① Importing a CSV file using read_csv() function

```
import pandas as pd
data = pd.read_csv("C:\\Users\\Admin\\cleaned_iris_data.csv")
data.head()
```

Output:-

```
Unnamed: 0   sepal_length_in_cm   sepal_width_in_cm   class
    0
    1
    2
    3
    4
```

② Reading data from URL

```
import pandas as pd
url = "https://.... .data"
col_names = ["sepal_length_in_cm", "sepal_width_in_cm",
  "petal_length_in_cm", "petal_width_in_cm", "class"]
iris_data = pd.read_csv(url, names=col_names)
iris_data.head()
```

Output:

```
sepal_length_in_cm | sepal_width_in_cm | petal_length_in_cm |
                                     petal_width_in_cm | cl
0
1
2
3
```

⑧ Exporting the Data frame to a CSV file

iris_data.to_csv("cleaned_iris_data.csv")

Output:- Exports the file to the current working directory

21/3/24

WEEK-2

End to end machine learning Project

Main steps

1. Framing the problem & looking at the big picture
2. Get the data
3. Explore the data
4. Data preparation
5. Shortlisting promising models
6. Fine tuning the model & combine them into a great sol^n
7. Present the solution

---

1. The data includes features such as
   - Population
   - Median income
   - Median housing price for each block grp in California

Problem:- Prediction will serve as an input to model that attempts to increase the company's ROI

Performance measure: RMSE

2. Get the Data

Downloading the data:

```
import os
import tarfile
import urllib

DOWNLOAD_ROOT = "https://raw...../"
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.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 load_housing_data(housing_path = HOUSING_PATH):
    data_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(data_path)
```

③ Discover & Visualise the Data

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

housing = strat_train_set.copy(); housing.shape
```

Visualizing Geographical Data

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

Anu 28-3-2024

4. 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')
housing_num = housing.drop("ocean_proximity", axis=1)
```

5. Select & Train Model

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X=housing_prepared, y=housing_labels)
```

Cross-validation

```
scores = cross_val_score(estimator=lin_reg, X=housing_prepared, y=housing_label,
                         scoring='neg_mean_squared_error', cv=10)
```

6. Fine Tune your Model

```
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 = GridSearchCV(estimator=forest_reg,
        param_grid, scoring='neg_mean_squared_
        error', cv=5, return_train_score=True,
        n_jobs=-1)

grid_search.fit(X=housing_prepared, y=housing_labels)
```

WEEK-3

Linear Regression

① df-sal = pd.read_csv(r'C:\Users\STUDENT ... csv')
df-sal.head()

② plt.title('Salary Distribution Plot')
sns.distplot(df-sal['Salary'])
plt.show()

③ plt.scatter(df-sal['YearsExperience'], df-sal['Salary'], color='lightcoral')

plt.xlabel('Years of Exp')
plt.ylabel('Salary')
plt.box(False)
plt.show()

④ Splitting variables
x = df-sal.iloc[:, :1]
y = df-sal.iloc[:, 1:]

⑤ x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state=0)

⑥ regressor.fit(x_train, y_train)

⑦ Prediction

y-pred-test = regressor.predict(x_test)
y-pred-train = regressor.predict(x_train)

Steps
① Import libraries ② Import data
③ Analyze data ④ Split data
⑤ Predict results ⑥ Visualize predictions

4/4/24

Multiple Linear Regression

① df-start = pd.read_csv('/content/50_Startup.csv')
df-start.head()

② plt.title('Profit Distribution Plot')
sns.displot(df-start['Profit'])
plt.show()

③ Splitting
x = df-start.iloc[:, :-1].values
y = df-start.iloc[:, -1].values

④ X-train, X-test, y-train, y-test = train-train-split(X, y,
test-size = 0.2, random-state = 0)

⑤ regressor = LinearRegression()
regressor.fit(X-train, y-train)

⑥ Predict
y-pred = regressor.predict(X-test)

Steps :-
→ Import libraries
→ Import data
→ Analyze data
→ Split into independent / Dependent variables
→ Predict Results
→ Compare predictions

WEEK-4

Decision Tree

savitha-bms

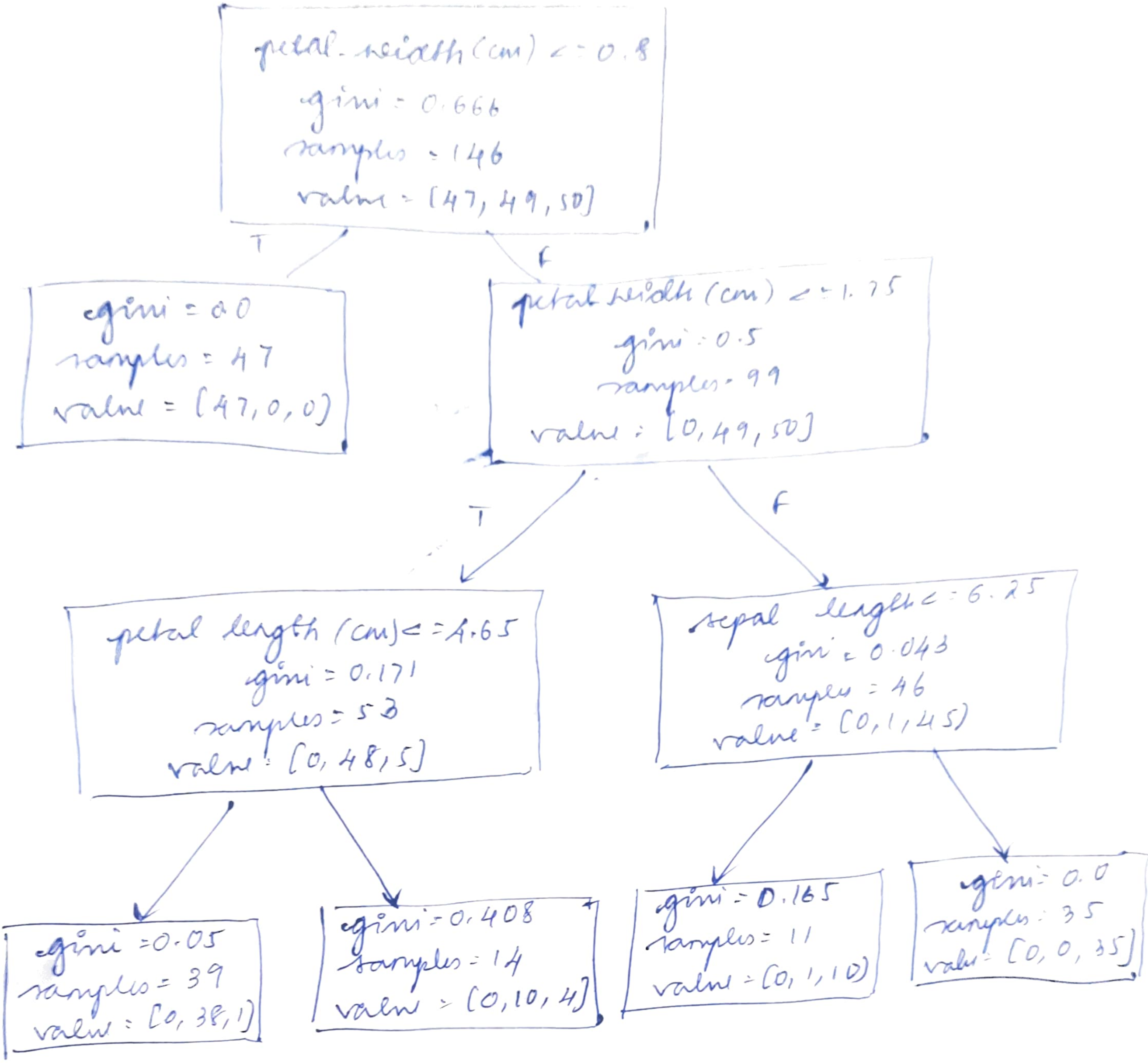
1.
```
cols = df.columns[0:-1]
for i in cols:
    sns.boxplot(y = df[i])
    plt.show()
```

2.
```
dt = DecisionTreeClassifier(max_depth = 3, min_samples_leaf = 10,
                            random_state = 1)

dt.fit(x, y)

    x = df.drop("species", axis = 1)
    y = df["species"]
```

3.
```
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())
```

4.
```
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)
```

5.
```
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))
```


petal width (cm) <= 0.8
gini = 0.666
samples = 146
value = [47, 49, 50]
T
F
gini = 0.0
samples = 47
value = [47, 0, 0]
petal width (cm) <= 1.75
gini = 0.5
samples = 99
value = [0, 49, 50]
T
F
petal length (cm) <= 4.65
gini = 0.171
samples = 53
value = [0, 48, 5]
sepal length <= 6.25
gini = 0.043
samples = 46
value = [0, 1, 45]
gini = 0.05
samples = 39
value = [0, 38, 1]
gini = 0.408
samples = 14
value = [0, 10, 4]
gini = 0.165
samples = 11
value = [0, 1, 10]
gini = 0.0
samples = 35
value = [0, 0, 35]

WEEK 5

Logistic regression

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

file_path = r"C:\Users ... .csv"
df = pd.read_csv(file_path)

df.head()

from sklearn.model_selection import train_test_split

plt.scatter(df.age, df.bought_insurance, marker = '', colo

x_train, x_test, y_train, y_test = train_test_split(df[['age']],
            df.bought_insurance

print(x_test)

model = LogisticRegression()
model.fit(x_train, y_train)
print(x_test)

model = LinearRegression()
model.fit(X_train, y_train)
print("Coefficient (m):", model.coef)
print("Intercept (b):", model.intercept)


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

x=0
sigmoid_value = sigmoid(x)
print("Sigmoid value at x=", x, ":", sigmoid_value)

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

```
def prediction_function(age):
    m = 0.042
    b = -1.53

    z = m * age + b

    y = sigmoid(z)
    return y

predicted probability = prediction_function(35)
print("Predicted probability of buying insurance for
        age 35:", predicted_probability)
```

Dvab
30-5-2024

WEEK-6

KNN Classifier

P1)
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

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

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 average_information(df, attribute):
    target = df.keys()[-1]
    target_variables = df[target].unique()
    variables = df[attribute].unique()
    entropy2 = 0
for variable in variables:
    entropy = 0
    for target_variable in target_variables:
        num = len(df[attribute][df[attribute] == variables]
                [df[target] == target_variable])
        fraction = num/(den + eps)
        entropy += -fraction * log(fraction + eps)
return abs(entropy2)
```

```
def buildTree(df, tree=None):
    target = df.keys()[-1]

node = find_winner(df)
attValue = np.unique(df[node])
if tree is None:
    tree = {}
    tree[node] = {}

for value in attValue:
    subtable = get_subtable(df, node, value)
    clValue, counts = np.unique(subtable[target], return_
                                            counts = True)

    if len(counts)==1
        tree[node][value] = clValue[0]
    else:
        tree[node][value] = buildTree(subtable)

return tree
```

3.5M

p2) SVM

```
plt.scatter(df.age, df.bought_insurance, marker='+', color='red')
x_train, x_test, y_train, y_test = train_test_split(df[['age']]
            df.bought_insurance, train_)

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)

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

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

Week

ANN model with back propagation

```
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

wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))
bh = np.random.uniform(size=(1, hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons, output_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):
    hinp1 = np.dot(X, wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1 = np.dot(hlayer_act, wout)
    outinp = outinp1 + 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 * hiddengrad
wout += hlayer_act.T.dot(d_output) * lr
wh += X.T.dot(d_hiddenlayer) * lr
```

p2) Random Forest Algorithm

```
iris = load_iris()
x = iris.data
y = iris.target
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                   test_size = 0.5, random_state = 32)

rf_classifier = RandomForestClassifier()
rf_classifier.fit(x_train, y_train)
y_pred = rf_classifier.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```

p3) Adaboost Algorithm

```
iris = load_iris()
x = iris_data
y = iris.target
adaboost_clf = AdaBoostClassifier(n_estimators = 30, learning_rate = 1.0,
                random_state = 42)
adaboost_clf.fit(x_train, y_train)
y_pred = adaboost_clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

30-5-2024

WEEK-8

K-Means algorithm to cluster

```
import numpy as np
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
Path = '/myDrive/ML-dataset/Iris.csv'
df = pd.read_csv(Path)
df
X = df.iloc[:, :-1].values

class kmeans:
    def __init__(self, n_clusters=4):
        self.K = n_clusters

    def fit(self, X):
        self.centroids = X[np.random.choice(len(X), self.K,
                                            replace=False)]
        self.initial_centroids = self.centroids
        self.prev_label, self.labels = None, np.zeros(len(X))
        while not np.all(self.labels == self.prev_label):
            self.prev_label = self.labels = None, np.zeros(len(X))
            self.labels = self.predict(X)
            self.update_centroid(X)
        return self

    def predict(self, X):
        return np.apply_along_axis(self.compute_label, 1, X)

    def compute_label(self, x):
        return np.argmin(np.sqrt(np.sum((self.centroids - x)**2,
                                        axis=1)))

fig = plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X[:,1], c=y)
```

PCA

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

data = load_breast_cancer()
data.keys()

print (data['target_names'])
print (data ['feature_names'])

df1 = pd.DataFrame (data ['data'], columns = data ['feature_names'])

scaling = StandardScaler()

scaling.fit(df1)
Scaled_data = scaling.transform (df1)

principal = PCA (n_components = 3)
principal.fit (scaled_data)
x = principal.transform (scaled_data)


plt.figure (figsize = (10,10))
plt.scatter (x [:0], x [:1], c= data = ['target'], cmap = 'plasma')
plt.xlabel ('pc1')
plt.ylabel ('pc2')
```

7/5/22