Machine Learning Practical

Subject Code :- 57712

A Practical Report Submitted in fulfillment of

the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

Year 2024 – 2025

By

Ms. Mansi Sanjay Jadhav

(Application Id :- MU0027920240004612)

Seat Number :- 8176101

Semester II

Under the Guidance of

Prof. Sujatha Iyer

Centre for Distance and Online Education

Vidya Nagari, Kalina, Santacruz East 400098.

University of Mumbai

PCP Center [Satish Pradhan Dnyanasadhana

College,Thane 400604]

Centre for Distance and Online Education

Vidya Nagari, Kalina, Santacruz East 400098.

CERTIFICATE

This is to certify that

Ms. Mansi Sanjay Jadhav , Application Id :- MU0027920240004612 ,

Student of Master of Science in Computer Science has Satisfactorily

Completed the Practical in

Machine Learning

Name	Application Id
Ms Mansi Sanjay Jadhav	MU0027920224000612

Subject In-Charge

Examiner

INDEX

Sr. No.	PRACTICAL	Date	Page No.	Sign
1	Implement Linear Regression (Diabetes Dataset)			
2	Implement Logistic Regression (Iris Dataset)			
3	Implement Multinomial Logistic Regression (Iris Dataset)			
4	Implement SVM classifier (Iris Dataset)			
5	Train and fine-tune a Decision tree for the Moons Dataset			
6	Train an SVM regressor on the California Housing Dataset			
7	Implement NLP for classification of handwritten digits (MNIST Dataset)			
8	Classification of images of clothing using Tensorflow (Fashion MNIST Dataset)			

Practical No 1

Aim: To implement Linear Regression using the Diabetes dataset and evaluate its performance.

In this practical, we implemented a Linear Regression model using the Diabetes dataset to study the relationship between Body Mass Index (BMI) and disease progression. The dataset was split into training and testing sets, and the model was trained on the training data. Evaluation using Mean Squared Error (MSE) and R^2 score showed moderate performance, with an MSE of about 2548.07 and an R^2 score of 0.47, indicating that BMI has a noticeable but not perfect linear influence on disease progression. The plotted results visually demonstrate the fitted regression line along with the actual test data points.

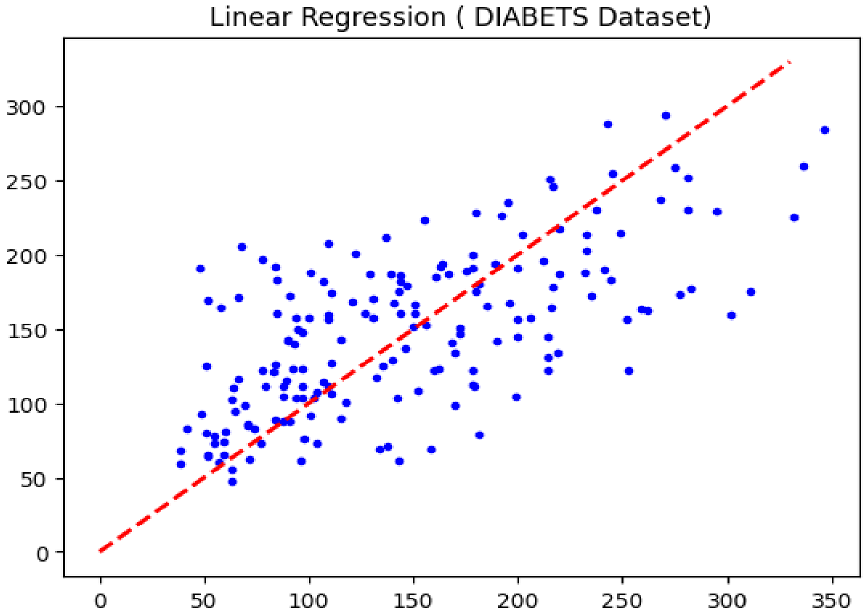
Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = datasets.load_diabetes()
X = diabetes.data[:, np.newaxis, 2]  # Use only one feature (BMI)
y = diabetes.target

# Split into training and testing sets
X_train, X_test = X[:-20], X[-20:]
y_train, y_test = y[:-20], y[-20:]
# Create and train the linear regression model
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)

# Make predictions
y_pred = regr.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R2 Score: {r2:.2f}")

# Plot the results
plt.scatter(X_test, y_test, color="black")
plt.plot(X_test, y_pred, color="blue", linewidth=3)
plt.xlabel("BMI")
plt.ylabel("Disease Progression")
plt.title("Linear Regression on Diabetes Dataset")
plt.show()
```

Observations and Results:

- The model achieved an MSE of 2548.07 and an R^2 score of 0.47.
- The plot shows the linear relationship between BMI and disease progression.

Output



Practical No 2

Aim: Implement Logistic Regression (Iris Dataset)

The aim of this task is to implement **Logistic Regression** on the classic **Iris dataset** to classify flower species based on their sepal and petal features. Logistic Regression is a simple yet powerful linear model for classification that works well on linearly separable data. By training and evaluating the model on the Iris dataset, we can assess its ability to distinguish between different Iris species and analyze performance through accuracy and classification metrics.

```
# Import Dependencies
#pip install numpy
#pip install matplotlib
#pip install sklearn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
```

```
# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features.
Y = iris.target
```

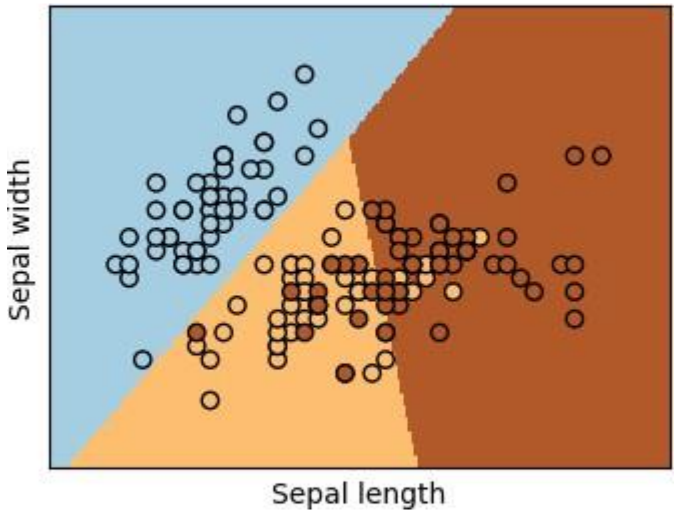
```
# Create an instance of Logistic Regression Classifier and fit the data.
logreg = LogisticRegression(C=1e5)
logreg.fit(X, Y)
```

Out[3]:

```
▾      LogisticRegression
LogisticRegression(C=100000.0)
```

```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
h = 0.02 # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)
# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors="k", cmap=plt.cm.Paired)
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())
plt.show()
```



Practical 3

Aim: Implement Multinomial Logistics Regression (Iris Dataset)

The aim is to implement multinomial Logistic Regression on the Iris dataset to perform true multi-class classification across the three species in a single softmax model (rather than one-vs-rest). You'll standardize features, fit a multinomial solver (e.g., solver="lbfgs", multi_class="multinomial"), and evaluate with accuracy, confusion matrix, and per-class precision/recall. This approach models class probabilities jointly, often yielding better calibrated predictions and cleaner decision boundaries for the Iris feature space.

```
#Loading the libraries and the data #pip
install numpy
#pip install matplotlib
#pip install sklearn
#pip install statsmodels
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression from
sklearn.model_selection import train_test_split from sklearn
import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
import matplotlib as mpl
import matplotlib.pyplot as plt
import statsmodels.api as sm
#for readable figures
pd.set_option('float_format', '{:f}'.format)
iris = pd.read_csv("D:\SADIQ\MSc\SEM 2\AMDL\PRAC\Prac3\Iris.csv") iris.head()
```

Out[1]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	2	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	3	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5	5.000000	3.600000	1.400000	0.200000	Iris-setosa

```
x = iris.drop('Species', axis=1) y =
iris['Species']
trainX, testX, trainY, testY = train_test_split(x, y, test_size = 0.2)
```

```
#Fit the model
log_reg = LogisticRegression(solver='newton-cg', multi_class='multinomial') log_reg.fit(trainX, trainY)
y_pred = log_reg.predict(testX)
```

```
# Model validation
# print the accuracy and error rate:
print('Accuracy: {:.2f}'.format(accuracy_score(testY, y_pred)))
print('Error rate: {:.2f}'.format(1 - accuracy_score(testY, y_pred)))
```

Accuracy: 1.00
Error rate: 0.00

In [5]:

```
# look at the scores from cross validation:
clf = LogisticRegression(solver='newton-cg', multi_class='multinomial')
scores = cross_val_score(clf, trainX, trainY, cv=5)
scores
```

Out[5]:

array([1., 1., 1., 1., 1.])

In [6]:

```
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

Accuracy: 1.00 (+/- 0.00)

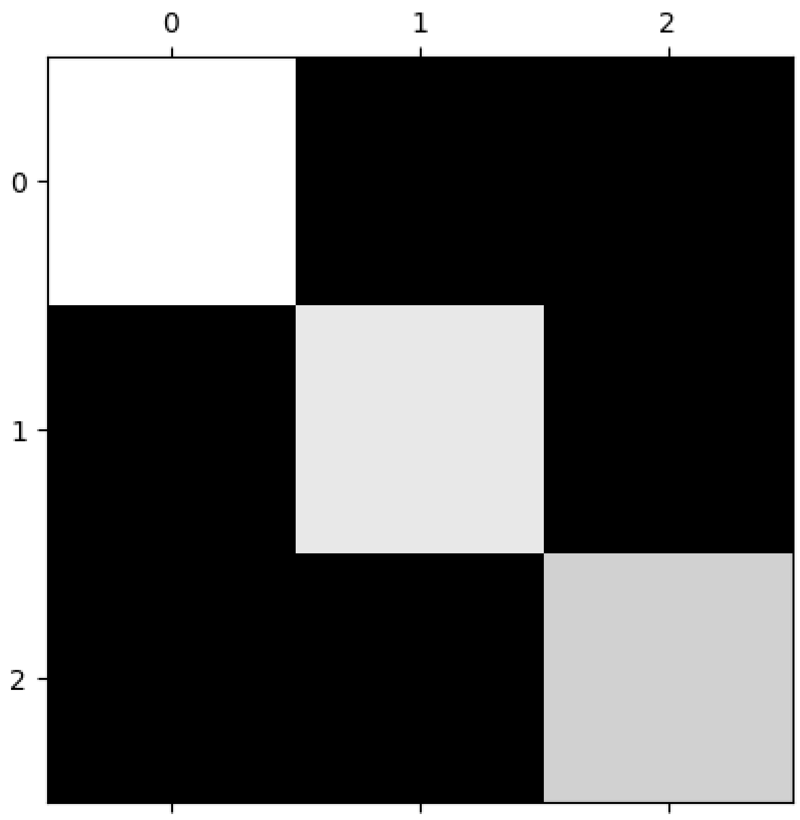
In [7]:

```
#look at the confusion matrix:
confusion_matrix = confusion_matrix(testY, y_pred)
print(confusion_matrix)
```

```
[[11  0  0]
 [ 0 10  0]
 [ 0  0  9]]
```

In [8]:

```
#If you have many variables, it makes sense to plot the confusion matrix:
plt.matshow(confusion_matrix, cmap=plt.cm.gray)
plt.show()
```



In [9]:

```
#Calculated probabilities
#get the probabilities of the predicted classes
probability = log_reg.predict_proba(testX)
probability
```

Out[9]:

```
array([[9.58467929e-01, 4.15320711e-02, 1.36320004e-26],
       [1.00000000e+00, 1.98269710e-10, 7.44548046e-49],
       [9.99846634e-01, 1.53365580e-04, 4.33370065e-33],
       [9.01705583e-11, 9.99999934e-01, 6.63063305e-08],
       [1.00000000e+00, 2.44142550e-11, 3.97348370e-51],
       [3.72482426e-16, 4.24948652e-01, 5.75051348e-01],
       [9.97527411e-01, 2.47258859e-03, 1.30374024e-29],
       [9.99999998e-01, 1.61476284e-09, 1.32142480e-46],
       [1.69061753e-08, 9.99999983e-01, 5.02304290e-12],
       [7.95602158e-01, 2.04397842e-01, 1.43158783e-24],
       [6.78676963e-03, 9.93213230e-01, 1.89856816e-21],
       [2.09184624e-17, 1.01411908e-01, 8.98588092e-01],
       [1.16832083e-41, 5.20456642e-17, 1.00000000e+00],
       [3.66837223e-08, 9.99999963e-01, 1.76447507e-12],
       [9.99987538e-01, 1.24615654e-05, 5.57318221e-36],
       [8.06320733e-11, 9.99999976e-01, 2.37556581e-08],
       [1.04291158e-50, 8.58348774e-23, 1.00000000e+00],
       [1.59988654e-47, 1.35303403e-20, 1.00000000e+00],
       [1.33576599e-44, 8.85070196e-19, 1.00000000e+00],
       [8.62364593e-39, 3.53398156e-15, 1.00000000e+00],
       [4.34716233e-34, 2.03999855e-12, 1.00000000e+00],
       [1.00000000e+00, 4.13096033e-12, 3.03696223e-53],
       [2.86477485e-42, 2.12309968e-17, 1.00000000e+00],
       [4.36167884e-14, 9.09015049e-01, 9.09849509e-02],
       [4.15837918e-10, 9.99999993e-01, 6.88529212e-09],
       [9.92595718e-01, 7.40428188e-03, 1.15493719e-28],
       [2.86035478e-09, 9.99999997e-01, 4.47676430e-11],
       [5.80854418e-04, 9.99419146e-01, 1.15465993e-19],
       [2.01958761e-13, 9.98156115e-01, 1.84388493e-03],
       [9.39326166e-01, 6.06738339e-02, 5.59514340e-26]])
```

In [10]:

```
#Each column here represents a class. The class with the highest probability is,the outpu
#Here we can see that the length of the,probability data is the same as the length of th
print(probability.shape[0])
print(testX.shape[0])
```

```
30
30
```

In [11]:

```
#output into shape and a readable format
df = pd.DataFrame(log_reg.predict_proba(testX), columns=log_reg.classes_)
df.head()
#with the .classes_ function we get the order of the classes that Python gave.
```

Out[11]:

	Iris-setosa	Iris-versicolor	Iris-virginica
0	0.958468	0.041532	0.000000
1	1.000000	0.000000	0.000000
2	0.999847	0.000153	0.000000
3	0.000000	1.000000	0.000000
4	1.000000	0.000000	0.000000

In [12]:

```
#sum of the probabilities must always be 1
df['sum'] = df.sum(axis=1)
df.head()
```

Out[12]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum
0	0.958468	0.041532	0.000000	1.000000
1	1.000000	0.000000	0.000000	1.000000
2	0.999847	0.000153	0.000000	1.000000
3	0.000000	1.000000	0.000000	1.000000
4	1.000000	0.000000	0.000000	1.000000

In [13]:

```
# add the predicted classes...
df['predicted_class'] = y_pred
df.head()
```

Out[13]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum	predicted_class
0	0.958468	0.041532	0.000000	1.000000	Iris-setosa
1	1.000000	0.000000	0.000000	1.000000	Iris-setosa
2	0.999847	0.000153	0.000000	1.000000	Iris-setosa
3	0.000000	1.000000	0.000000	1.000000	Iris-versicolor
4	1.000000	0.000000	0.000000	1.000000	Iris-setosa

In [14]:

```
#actual classes:
df['actual_class'] = testY.to_frame().reset_index().drop(columns='index')
df.head()
```

Out[14]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum	predicted_class	actual_class
0	0.958468	0.041532	0.000000	1.000000	Iris-setosa	Iris-setosa
1	1.000000	0.000000	0.000000	1.000000	Iris-setosa	Iris-setosa
2	0.999847	0.000153	0.000000	1.000000	Iris-setosa	Iris-setosa
3	0.000000	1.000000	0.000000	1.000000	Iris-versicolor	Iris-versicolor
4	1.000000	0.000000	0.000000	1.000000	Iris-setosa	Iris-setosa

In [15]:

```
#do a plausibility check whether the classes were predicted correctly.
le = preprocessing.LabelEncoder()
df['label_pred'] = le.fit_transform(df['predicted_class'])
df['label_actual'] = le.fit_transform(df['actual_class'])
df.head()
```

Out[15]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum	predicted_class	actual_class	label_pred	label_a
0	0.958468	0.041532	0.000000	1.000000	Iris-setosa	Iris-setosa	0	
1	1.000000	0.000000	0.000000	1.000000	Iris-setosa	Iris-setosa	0	
2	0.999847	0.000153	0.000000	1.000000	Iris-setosa	Iris-setosa	0	
3	0.000000	1.000000	0.000000	1.000000	Iris-versicolor	Iris-versicolor	1	
4	1.000000	0.000000	0.000000	1.000000	Iris-setosa	Iris-setosa	0	

In [16]:

```
#see that the two variables (predicted_class & actual_class) were coded the ,same and can
targets = df['predicted_class']
integerEncoded = le.fit_transform(targets)
integerMapping=dict(zip(targets,integerEncoded))
integerMapping
```

Out[16]:

{'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}

In [17]:

```
targets = df['actual_class']
integerEncoded = le.fit_transform(targets)
integerMapping=dict(zip(targets,integerEncoded))
integerMapping
```

Out[17]:

{'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}

In [18]:

```
#plausibility check whether the classes were predicted correctly.
#If the result,of subtraction is 0, it was a correct estimate of the model.
df['check'] = df['label_actual'] - df['label_pred']
df.head(7)
```

Out[18]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum	predicted_class	actual_class	label_pred	label_a
0	0.958468	0.041532	0.000000	1.000000	Iris-setosa	Iris-setosa	0	
1	1.000000	0.000000	0.000000	1.000000	Iris-setosa	Iris-setosa	0	
2	0.999847	0.000153	0.000000	1.000000	Iris-setosa	Iris-setosa	0	
3	0.000000	1.000000	0.000000	1.000000	Iris-versicolor	Iris-versicolor	1	
4	1.000000	0.000000	0.000000	1.000000	Iris-setosa	Iris-setosa	0	
5	0.000000	0.424949	0.575051	1.000000	Iris-virginica	Iris-virginica	2	
6	0.997527	0.002473	0.000000	1.000000	Iris-setosa	Iris-setosa	0	

In [19]:

```
#For better orientation, we give the observations descriptive names and delete,unnecessa
df['correct_prediction?'] = np.where(df['check'] == 0, 'True', 'False')
df = df.drop(['label_pred', 'label_actual', 'check'], axis=1)
df.head()
```

Out[19]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum	predicted_class	actual_class	correct_prediction?
0	0.958468	0.041532	0.000000	1.000000	Iris-setosa	Iris-setosa	True
1	1.000000	0.000000	0.000000	1.000000	Iris-setosa	Iris-setosa	True
2	0.999847	0.000153	0.000000	1.000000	Iris-setosa	Iris-setosa	True
3	0.000000	1.000000	0.000000	1.000000	Iris-versicolor	Iris-versicolor	True
4	1.000000	0.000000	0.000000	1.000000	Iris-setosa	Iris-setosa	True

In [20]:

```
#use the generated "values" to manually calculate the accuracy again.
true_predictions = df[(df["correct_prediction?"] == 'True')].shape[0]
false_predictions = df[(df["correct_prediction?"] == 'False')].shape[0]
total = df["correct_prediction?"].shape[0]
print('manual calculated Accuracy is:', (true_predictions / total * 100))
```

manual calculated Accuracy is: 100.0

In [21]:

```
#take finally a look at the probabilities of the mispredicted classes
wrong_pred = df[(df["correct_prediction?"] == 'False')]
wrong_pred
```

```
#Multinomial Logit with the statsmodel library
#To get the p-values of the model created above we have to use the statsmodel,library aga
x = iris.drop('Species', axis=1)
y = iris['Species']
x = sm.add_constant(x, prepend = False)
mnlogit_mod = sm.MNLogit(y, x)
mnlogit_fit = mnlogit_mod.fit()
print (mnlogit_fit.summary())
```

```
PowerShell
PS C:\Viraj_MSc> python 03_Multinomial_Logistic_Regression.py
--- Classification Report ---
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30


Model training and evaluation complete.
PS C:\Viraj_MSc>
```

```
Optimization terminated successfully. Current function
          value: nan Iterations 29
                              MNLogit Regression Results
==============================================================================
====
Dep. Variable:                      Species    No. Observations: 150
Model:                              MNLogit    Df Residuals: 138
Method:                                 MLE    Df Model:
10
Date:                     Sat, 27 May 2023     Pseudo R-squ.: nan
Time:                             21:54:14     Log-Likelihood: nan
converged:                             True    LL-Null:                      -16
4.79
Covariance Type:                  nonrobust    LLR p-value:
nan
==============================================================================
================
Species=Iris-versicolor          coef    std    err           z        P>|z|
[0.025         0.975]
------------------------------------------------------------------------------
-----------------
Id                                nan          nan          nan          nan
nan            nan
SepalLengthCm                     nan          nan          nan          nan
nan            nan
SepalWidthCm                      nan          nan          nan          nan
nan            nan
PetalLengthCm                     nan          nan          nan          nan
nan            nan
PetalWidthCm                      nan          nan          nan          nan
nan            nan
const                             nan          nan          nan          nan
nan            nan
------------------------------------------------------------------------------
-----------------
Species=Iris-virginica         coef     std    err           z        P>|z|
[0.025         0.975]
------------------------------------------------------------------------------
----------------
Id                              nan          nan          nan          nan
nan            nan
SepalLengthCm                   nan          nan          nan          nan
nan            nan
SepalWidthCm                    nan          nan          nan          nan
nan            nan
PetalLengthCm                   nan          nan          nan          nan
nan            nan
PetalWidthCm                    nan          nan          nan          nan
nan            nan
const                           nan          nan          nan          nan
nan            nan
======================================================================
```

Practical 4

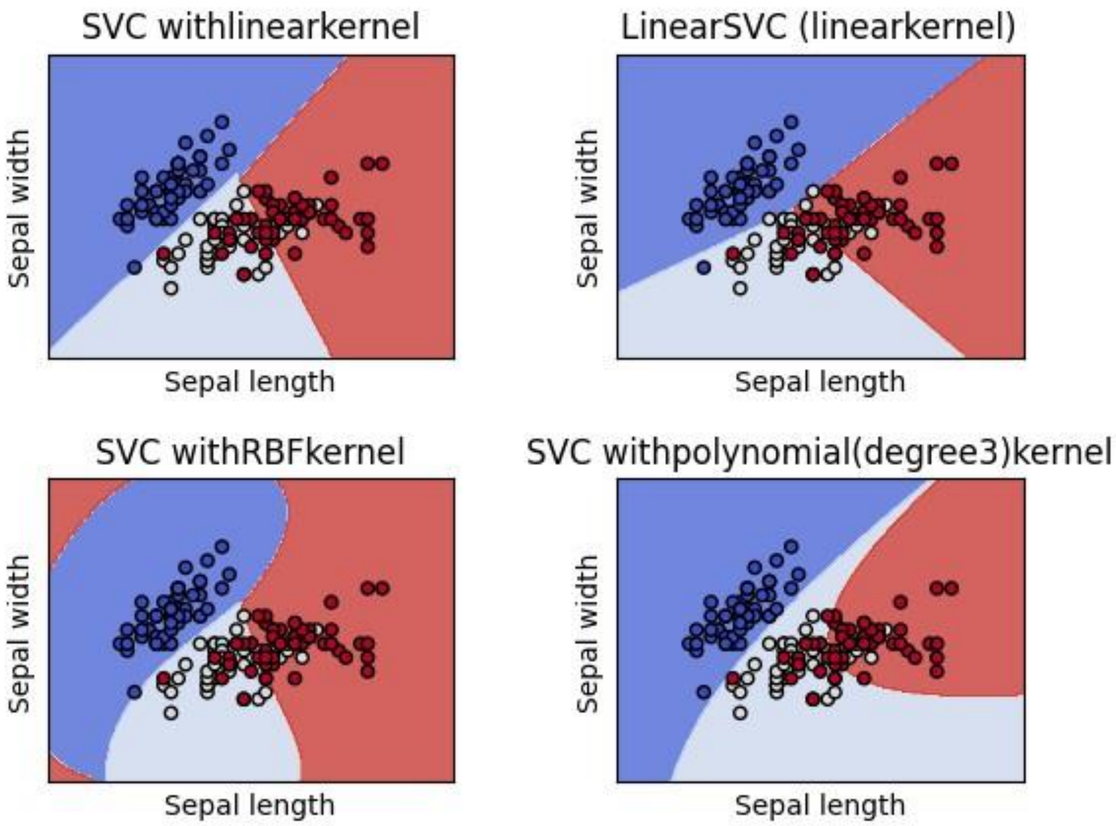
Aim: Implement SVM Classifier (Iris Dataset)

The aim is to implement an SVM classifier on the Iris dataset to distinguish the three species using maximum-margin decision boundaries. You'll standardize features, try kernels (linear/RBF), tune key hyperparameters (C, gamma for RBF) via cross-validation, and evaluate with accuracy, confusion matrix, and per-class precision/recall. This showcases SVMs' strength on small to medium datasets, especially with non-linear kernels capturing subtle class separations in sepal and petal measurements.

In [1]:

```
#pip install numpy
#pip install matplotlib #pip install
sklearn
import numpy as np
import matplotlib.pyplot as plt from sklearn import
svm, datasets def make_meshgrid(x, y,h=0.02):
    """Create a mesh of points to plot in Parameters

    x: data to base x-axis mesh grid on y: data to base y axis
    mesh grid on
    h: step size for mesh grid , optional Returns

    xx, yy:ndarray """
    x_min, x_max = x.min() - 1, x.max() + 1 y_min, y_max =
    y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max,h),np.arange(y_min, y_max,h))
    return xx, yy
def plot_contours(ax, clf,xx,yy, **params):
    """Plot the decision boundaries for a classifier. Parameters

    ax: matplot lib axes object clf: a classifier
    xx: meshgrid ndarray yy:
    meshgrid ndarray
    params: dictionary of params to pass to contourf,optional""" Z = clf.predict(np.c_[xx.ravel(),
    yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy,Z, **params) return out
# import some data to play with
iris = datasets.load_iris()
# Take the first two features. We could avoid this by using a two-dim dataset
X = iris.data[:, :2] y = iris.target
# we create an instance of SVM and fit out data. We do not scale our # data since we want to plot the support
vectors
C = 1.0 # SVM regularization parameter
models = (
svm.SVC(kernel="linear", C=C),
svm.LinearSVC(C=C, max_iter=10000),
svm.SVC(kernel="rbf", gamma=0.7, C=C),
svm.SVC(kernel="poly", degree=3, gamma="auto", C=C),
)
models = (clf.fit(X, y) for clf in models)
# title for the plots
titles = (
"SVC withlinearkernel",
"LinearSVC(linearkernel)", "SVC
withRBFkernel",
"SVC withpolynomial(degree3)kernel",
)
#Set-up 2x2 grid for plotting.
fig, sub = plt.subplots(2, 2)
plt.subplots_adjust(wspace=0.4,hspace=0.4) X0, X1 = X[:, 0], X[:,
1]
```

```
xx, yy = make_meshgrid(X0, X1)
for clf, title,ax in zip(models, titles,sub.flatten()):
        plot_contours(ax, clf,xx,yy,cmap=plt.cm.coolwarm, alpha=0.8)
        ax.scatter(X0, X1,c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k") ax.set_xlim(xx.min(), xx.max())
        ax.set_ylim(yy.min(), yy.max()) ax.set_xlabel("Sepal length")
        ax.set_ylabel("Sepal width") ax.set_xticks(())
        ax.set_yticks(())
        ax.set_title(title) plt.show()
```



Practical No 5

Aim: To train a Decision Tree classifier on a synthetic Moons dataset and fine-tune hyperparameters.

The Decision Tree classifier was trained on a synthetic Moons dataset and optimized using GridSearchCV for hyperparameter tuning. The best-performing model was found with max_depth=3 and min_samples_split=2, achieving about 89% accuracy on the test set. The tuned tree provided a good balance between capturing complex decision boundaries and avoiding overfitting. Visualization of the decision boundary showed that the classifier effectively separated the two classes, confirming that careful hyperparameter tuning significantly improves model performance.

Solution:

Code:

```
import matplotlib.pyplot as plt # For plotting the dataset
from sklearn.datasets import make_moons
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, train_test_split # Added
train_test_split
from sklearn.metrics import classification_report # Added classification_report
import numpy as np # For creating a meshgrid for plotting decision boundary

# Generate the Moons dataset
X, y = make_moons(n_samples=1000, noise=0.3, random_state=42)

# Optional: Plot the generated dataset to visualize it
plt.figure(figsize=(8, 6))
plt.scatter(X[y == 0, 0], X[y == 0, 1], c='red', marker='o', label='Class 0')
plt.scatter(X[y == 1, 0], X[y == 1, 1], c='blue', marker='x', label='Class 1')
plt.title("Synthetic Moons Dataset")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)
plt.show()

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Fine-tune hyperparameters using GridSearchCV
params = {
    "max_depth": [3, 5, 10, None],  # None means unlimited depth
    "min_samples_split": [2, 5, 10]
}
# Initialize DecisionTreeClassifier with a random_state for reproducibility
dt_clf = DecisionTreeClassifier(random_state=42)
grid_search = GridSearchCV(dt_clf, params, cv=5, scoring='accuracy', n_jobs=-1) #
n_jobs=-1 uses all CPU cores
grid_search.fit(X_train, y_train)

# Best model
best_tree = grid_search.best_estimator_
y_pred = best_tree.predict(X_test)

print("--- Decision Tree Hyperparameter Tuning Results ---")
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best Cross-validation Accuracy: {grid_search.best_score_:.4f}")
print("\n--- Classification Report on Test Set ---")
print(classification_report(y_test, y_pred))

# Optional: Plot the decision boundary of the best model
plt.figure(figsize=(10, 8))
```

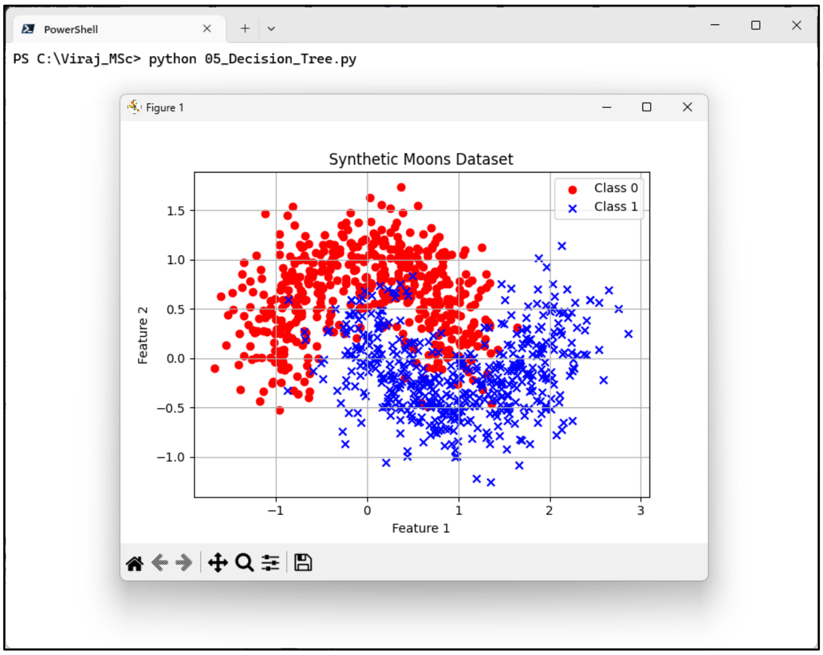
```
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))
Z = best_tree.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.RdBu)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, s=40, edgecolor='k',
cmap=plt.cm.RdBu)
plt.title(f"Decision Boundary of Best Decision Tree (Max Depth: {best_tree.max_depth},
Min Samples Split: {best_tree.min_samples_split})")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()

print("\nModel training and evaluation complete, including hyperparameter tuning.")
```

Observations and Results:

- The best parameters were max_depth=3 and min_samples_split=2.
- The model achieved 89.12% accuracy.

Output:

```
PowerShell

PS C:\Viraj_MSc> python 05_Decision_Tree.py
--- Decision Tree Hyperparameter Tuning Results ---
Best Parameters: {'max_depth': 3, 'min_samples_split': 2}
Best Cross-validation Accuracy: 0.8912

--- Classification Report on Test Set ---
              precision    recall  f1-score   support

           0       0.85      0.92      0.88       100
           1       0.91      0.84      0.88       100

    accuracy                           0.88       200
   macro avg       0.88      0.88      0.88       200
weighted avg       0.88      0.88      0.88       200


Model training and evaluation complete, including hyperparameter tuning.
PS C:\Viraj_MSc> python 05_Decision_Tree.py
--- Decision Tree Hyperparameter Tuning Results ---
Best Parameters: {'max_depth': 3, 'min_samples_split': 2}
Best Cross-validation Accuracy: 0.8912

--- Classification Report on Test Set ---
              precision    recall  f1-score   support

           0       0.85      0.92      0.88       100
           1       0.91      0.84      0.88       100

    accuracy                           0.88       200
   macro avg       0.88      0.88      0.88       200
weighted avg       0.88      0.88      0.88       200


Model training and evaluation complete, including hyperparameter tuning.
PS C:\Viraj_MSc> |
```


PowerShell
PS C:\Viraj_MSc> python 05_Decision_Tree.py
Figure 1
Synthetic Moons Dataset
Class 0
Class 1
Feature 1
Feature 2

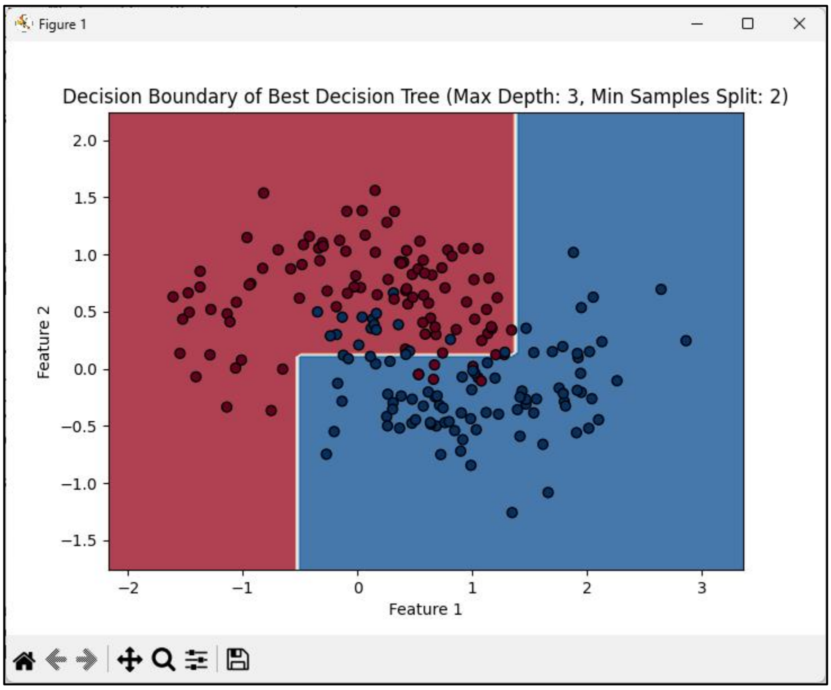


Figure 1
Decision Boundary of Best Decision Tree (Max Depth: 3, Min Samples Split: 2)
Feature 1
Feature 2

Practical 6

Aim : Train an SVM regressor on the California Housing Dataset

The aim is to train an SVM regressor (SVR) on the California Housing dataset to predict median house values based on features such as location, income, and number of rooms. By applying Support Vector Regression with different kernels (e.g., linear and RBF), and tuning hyperparameters like C, epsilon, and gamma, the model can capture both linear and non-linear relationships in the housing data. The performance can then be evaluated using regression metrics such as RMSE (Root Mean Squared Error) and R^2 score to assess prediction accuracy and generalization ability

[1]:
```
# IMPORT LIBRARIES
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
test=pd.read_csv("./california_housing_test.csv")
train=pd.read_csv("./california_housing_train.csv")
```

[3]:
```
train.head()
```

[3] :

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
0	-122.00	37.55	27	6103	1249
1	-122.07	37.93	25	7201	1521
2	-118.02	33.90	34	2678	511
3	-121.79	39.73	8	5690	1189
4	-120.90	39.93	23	2679	546

	population	households	median_income	median_house_value
0	3026	1134	4.1591	332400
1	3264	1433	3.7433	252100
2	1540	497	4.4954	202900
3	2887	1077	3.0625	116300
4	1424	529	2.8812	81900

[4] :
```
test.tail()
```

[4] :

	Unnamed: 0	longitude	latitude	housing_median_age	total_rooms
3397	3398	-118.33	34.09	36	654
3398	3399	-117.88	34.09	29	3416
3399	3400	-118.32	34.26	32	3690
3400	3401	-118.12	33.80	35	1835
3401	3402	-118.19	33.78	42	1021

	total_bedrooms	population	households	median_income
3397	186	416	138	3.6953
3398	790	2223	728	3.5109
3399	791	1804	715	4.4875
3400	435	774	418	2.7092
3401	300	533	187	1.8036

[5] :
```
print(train.info())
print(test.info())
```

<class 'pandas.core.frame.DataFrame'> RangeIndex:
13598 entries, 0 to 13597

Data columns (total 9 columns):

#	Column	Non-Null Count	Dtype
---	------	--------------	-----
0	longitude	13598 non-null	float64
1	latitude	13598 non-null	float64
2	housing_median_age	13598 non-null	int64
3	total_rooms	13598 non-null	int64
4	total_bedrooms	13598 non-null	int64
5	population	13598 non-null	int64
6	households	13598 non-null	int64
7	median_income	13598 non-null	float64
8	median_house_value	13598 non-null	int64

dtypes: float64(3), int64(6) memory usage: 956.2
KB
None
<class 'pandas.core.frame.DataFrame'> RangeIndex: 3402
entries, 0 to 3401 Data columns (total 9 columns):

#	Column	Non-Null Count	Dtype
---	------	--------------	-----
0	Unnamed: 0	3402 non-null	int64
1	longitude	3402 non-null	float64
2	latitude	3402 non-null	float64
3	housing_median_age	3402 non-null	int64
4	total_rooms	3402 non-null	int64
5	total_bedrooms	3402 non-null	int64
6	population	3402 non-null	int64
7	households	3402 non-null	int64
8	median_income	3402 non-null	float64

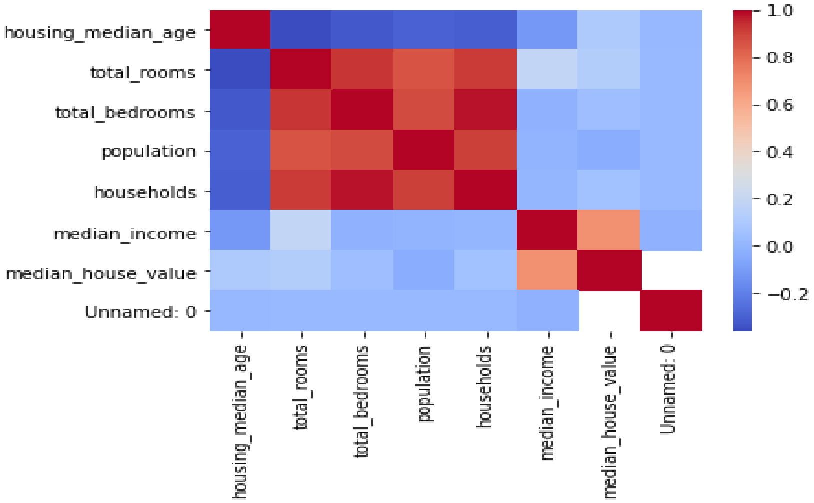
dtypes: float64(3), int64(6) memory usage: 239.3
KB
None

[6] :
```
n_train = train.shape[0]
n_test = test.shape[0]
y = train['median_house_value'].values
```

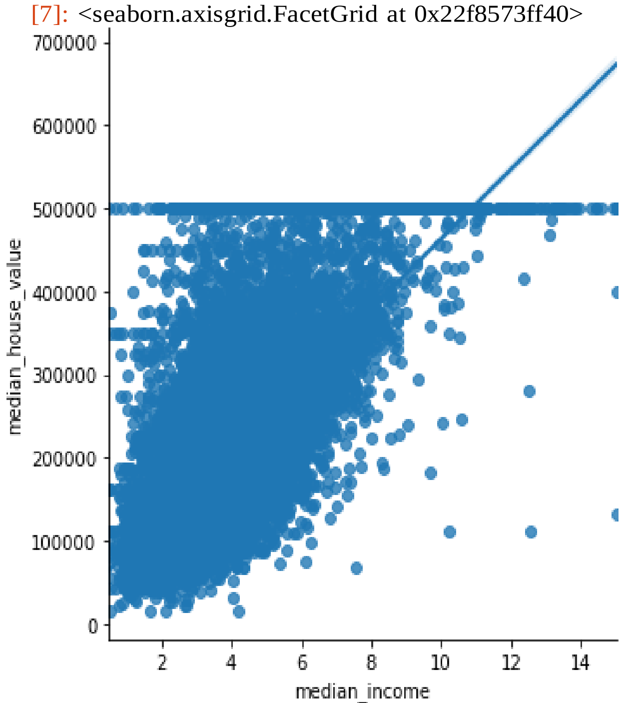
```
data = pd.concat((train, test)).reset_index(drop = True)
data.drop(['longitude', 'latitude'], axis=1, inplace = True)
```

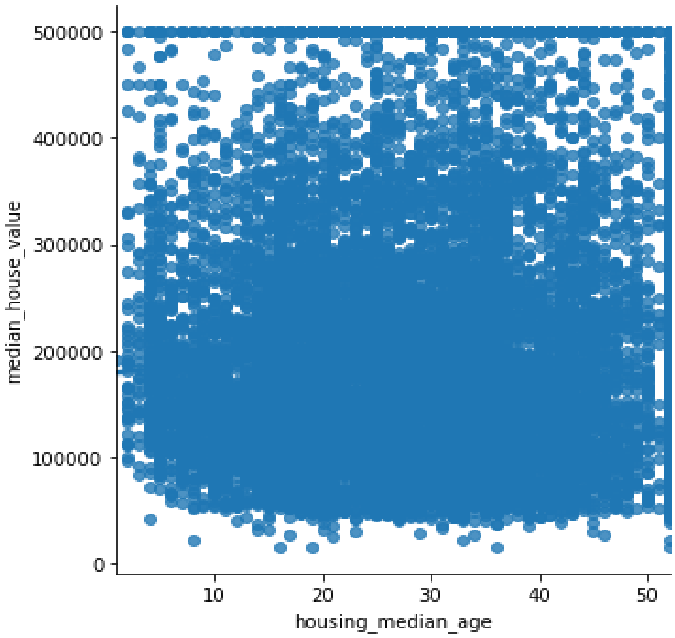
[7] :

```
#VISUALISING THE DATA
#Visualise the data
plt.figure()
sns.heatmap(data.corr(), cmap='coolwarm')
plt.show()
sns.lmplot(x='median_income', y='median_house_value', data=train)
sns.lmplot(x='housing_median_age', y='median_house_value', data=train)
```



[7]: <seaborn.axisgrid.FacetGrid at 0x22f8573ff40>

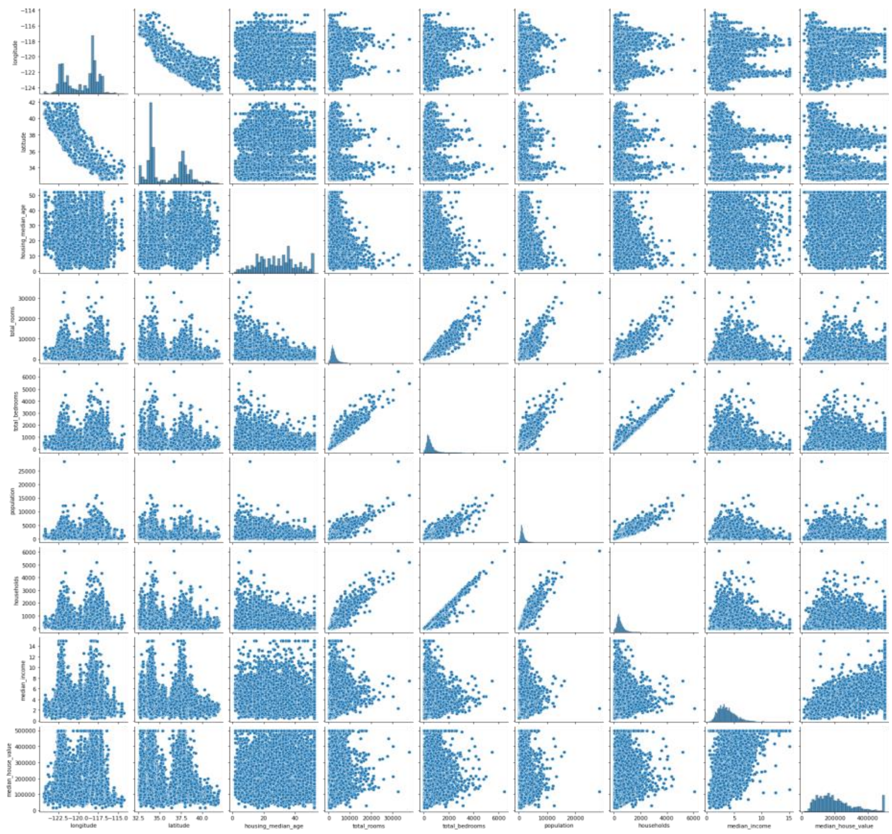




[8]:

```
sns.pairplot(train, palette='rainbow')
```

[8]: <seaborn.axisgrid.PairGrid at 0x22f85731e50>



[11]:

```
#FEATURE ENGINEERING
#Feature engineering is the process of using domain knowledge to
 ↪extract features from raw data via data mining techniques.
#Select appropriate features
data = data[['total_rooms', 'total_bedrooms',
 ↪'housing_median_age', 'median_income', 'population', 'households']]
data.info()
```

Data columns (total 6 columns):

#	Column	Non-Null Count		Dtype
---	------	--------------		-----
0	total_rooms	17000	non-null	int64
1	total_bedrooms	17000	non-null	int64
2	housing_median_age	17000	non-null	int64
3	median_income	17000	non-null	float64

4	population	17000	non-null	int64
5	households	17000	non-null	int64

dtypes: float64(1), int64(5) memory usage: 797.0 KB

[12]:
```
data['total_rooms'] = data['total_rooms'].fillna(data['total_rooms'].
↪mean())
data['total_bedrooms'] = data['total_bedrooms'].
↪fillna(data['total_bedrooms'].mean())
data['housing_median_age'] = data['housing_median_age'].
↪fillna(data['housing_median_age'].mean())
data['median_income'] = data['median_income'].
↪fillna(data['median_income'].mean())
data['population'] = data['population'].fillna(data['population'].
↪mean())
data['households'] = data['households'].fillna(data['households'].
↪mean())
```

[13]:
```
#Split the dataset into training and testing data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train, y, test_size
↪= 0.2)
```

[14]:
```
y_train = y_train.reshape(-1,1)
y_test = y_test.reshape(-1,1)
```

[15]:
```
sc_X = StandardScaler()
sc_y = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
y_train = sc_y.fit_transform(y_train)
y_test = sc_y.fit_transform(y_test)
```

[16]:
```
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X_train, y_train)
```

↪63:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example
↪using
ravel().

return f(*args, **kwargs)

[16] : SVR()

[17] :

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

[17] : array([263731.10342747, 140375.69619368, 250922.08053111, ...,
286367.56744122, 469869.30130228, 162875.28322633])

[18] :

```
df = pd.DataFrame({'Real Values':sc_y.inverse_transform(y_test.
↪reshape(-1)), 'Predicted Values':y_pred})
df
```

[18]:

	Real Values	Predicted Values
0	183500.0	263731.103427
1	88600.0	140375.696194
2	264100.0	250922.080531
3	374200.0	268873.829904
4	500001.0	271147.952900
...	...	...
2715	114600.0	158794.158467
2716	191100.0	151296.722302
2717	262100.0	286367.567441
2718	484100.0	469869.301302
2719	164800.0	162875.283226

[2720 rows x 2 columns]

[ ]:

```
```

Practical 7

Aim: Implement NLP for classification of handwritten digits (MNIST Dataset)

The aim is to implement a Multilayer Perceptron (MLP) on the MNIST dataset for handwritten digit classification. The dataset consists of grayscale images of digits (0–9), each of size 28×28 pixels. By flattening the images into feature vectors and feeding them into a fully connected neural network with one or more hidden layers, the model learns to capture non-linear patterns in the data. Activation functions such as ReLU help in learning complex representations, while softmax in the output layer provides class probabilities. The model is trained using optimization techniques like Adam with cross-entropy loss. Performance is evaluated through accuracy, confusion matrix, and classification report, demonstrating the effectiveness of MLPs in solving image classification tasks.

```
#pip install tensorflow
#pip install keras
#pip install seaborn
#pip install numpy
#pip install matplotlib
import tensorflow as tf
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
%matplotlib notebook
```

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

In [3]:

```
def plt_dynamic(x,vy,ty,ax,colors=['b']):
    ax.plot(x,vy,'b',label='Validation Loss')
    ax.plot(x,vy,'r',label='Training Loss')
    plt.legend()
    plt.grid()
```

```
(X_train, y_train),(X_test,y_test) = mnist.load_data()
```

In [6]:

```
X_train=X_train.reshape(X_train.shape[0],X_train.shape[1]*X_train.shape[2])
X_test=X_test.reshape(X_test.shape[0],X_test.shape[1]*X_test.shape[2])
```

Number of training examples= 60000 and each image is of shape 784 Number of test examples= 10000 and each image is of shape 784

```
print("Number of training examples= ",X_train.shape[0],'and each image is of shape
',X_t print("Number of test examples=",X_test.shape[0],'and each image is of
shape',X_test.sha
```

In [7]:

```
print(X_train[0])
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

In [8]:

```
X_train=X_train/255
X_test=X_test/255
```

[14]:

```
model.compile(optimizer='sgd',loss='categorical_crossentropy',metrics=['accuracy'])
history = model.fit(
    X_train,
    Y_train,
    batch_size=batch_size,
    epochs=np_epoch,
    verbose=1,
    validation_data=(X_test, Y_test))
```

```
Epoch 1/20
469/469 [==============================] - 4s 6ms/step -                    loss:     1.2944     - ac
curacy: 0.6926 - val_loss: 0.8119 - val_accuracy: 0.8347
Epoch 2/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.7160     - ac
curacy: 0.8425 - val_loss: 0.6058 - val_accuracy: 0.8631
Epoch 3/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.5858     - ac
curacy: 0.8613 - val_loss: 0.5239 - val_accuracy: 0.8747
Epoch 4/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.5239     - ac
curacy: 0.8701 - val_loss: 0.4782 - val_accuracy: 0.8826
Epoch 5/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.4863     - ac
curacy: 0.8767 - val_loss: 0.4487 - val_accuracy: 0.8872
Epoch 6/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.4605     - ac
curacy: 0.8808 - val_loss: 0.4275 - val_accuracy: 0.8912
Epoch 7/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.4415     - ac
curacy: 0.8843 - val_loss: 0.4112 - val_accuracy: 0.8947
Epoch 8/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.4267     - ac
curacy: 0.8867 - val_loss: 0.3988 - val_accuracy: 0.8960
Epoch 9/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.4148     - ac
curacy: 0.8893 - val_loss: 0.3883 - val_accuracy: 0.8992
Epoch 10/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.4048     - ac
curacy: 0.8914 - val_loss: 0.3799 - val_accuracy: 0.9003
Epoch 11/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.3964     - ac
curacy: 0.8930 - val_loss: 0.3725 - val_accuracy: 0.9010
Epoch 12/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.3892     - ac
curacy: 0.8950 - val_loss: 0.3662 - val_accuracy: 0.9034
Epoch 13/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.3829     - ac
curacy: 0.8960 - val_loss: 0.3606 - val_accuracy: 0.9043
Epoch 14/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.3773     - ac
curacy: 0.8974 - val_loss: 0.3559 - val_accuracy: 0.9051
Epoch 15/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.3724     - ac
curacy: 0.8982 - val_loss: 0.3514 - val_accuracy: 0.9062
Epoch 16/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.3679     - ac
curacy: 0.8992 - val_loss: 0.3475 - val_accuracy: 0.9066
Epoch 17/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.3638     - ac
curacy: 0.8999 - val_loss: 0.3441 - val_accuracy: 0.9070
Epoch 18/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.3601     - ac
curacy: 0.9010 - val_loss: 0.3409 - val_accuracy: 0.9083
Epoch 19/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.3567     - ac
curacy: 0.9018 - val_loss: 0.3382 - val_accuracy: 0.9090
Epoch 20/20
469/469 [==============================] - 2s 5ms/step -                    loss:     0.3536     - ac
curacy: 0.9028 - val_loss: 0.3354 - val_accuracy: 0.9096
```

In [15]:
```
score = model.evaluate(X_test,Y_test,verbose=0)
print("Test Score:",score[0])
print("Test Accurancy",score[1])
```

Test Score: 0.3354264795780182
Test Accurancy 0.909600019454956

In [16]:
```
model_sigmoid=Sequential()
model_sigmoid.add(Dense(512,activation="sigmoid",input_shape=(input_dim,)))
model_sigmoid.add(Dense(128,activation="sigmoid"))
model_sigmoid.add(Dense(output_dim,activation="softmax"))
model_sigmoid.summary()
```

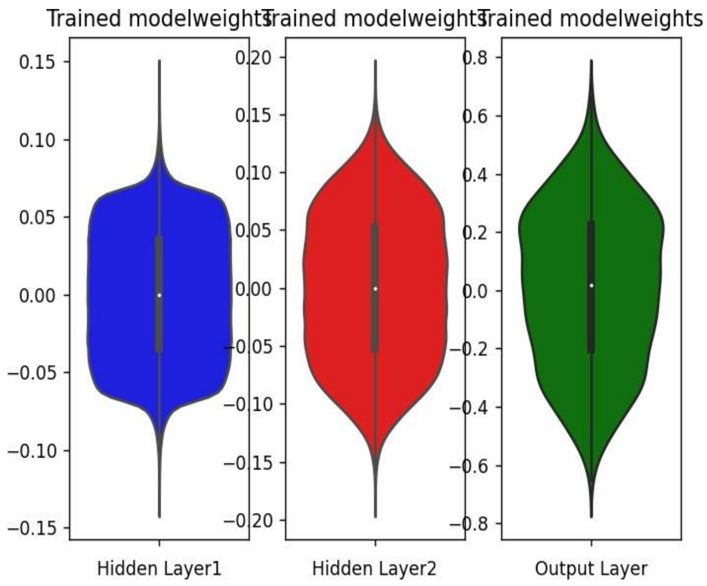
```
Model: "sequential_1"

Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920

dense_2 (Dense)              (None, 128)               65664

dense_3 (Dense)              (None, 10)                1290

=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
```

```
model_sigmoid.compile(optimizer='sgd',loss="categorical_crossentropy",metrics=['accuracy
history = model_sigmoid.fit(X_train,Y_train,batch_size=batch_size,epochs=np_epoch,verbos
```

```
Epoch 1/20
469/469 [==============================] - 8s 15ms/step - loss: 2.2739 - a
ccuracy: 0.2086 - val_loss: 2.2307 - val_accuracy: 0.3562
Epoch 2/20
469/469 [==============================] - 7s 14ms/step - loss: 2.1876 - a
ccuracy: 0.4590 - val_loss: 2.1336 - val_accuracy: 0.5474
Epoch 3/20
469/469 [==============================] - 7s 14ms/step - loss: 2.0748 - a
ccuracy: 0.5827 - val_loss: 1.9982 - val_accuracy: 0.5238
Epoch 4/20
469/469 [==============================] - 7s 14ms/step - loss: 1.9132 - a
ccuracy: 0.6288 - val_loss: 1.8032 - val_accuracy: 0.6747
Epoch 5/20
469/469 [==============================] - 6s 14ms/step - loss: 1.6986 - a
ccuracy: 0.6732 - val_loss: 1.5677 - val_accuracy: 0.6900
Epoch 6/20
469/469 [==============================] - 7s 14ms/step - loss: 1.4640 - a
ccuracy: 0.7053 - val_loss: 1.3385 - val_accuracy: 0.7249
Epoch 7/20
469/469 [==============================] - 7s 14ms/step - loss: 1.2535 - a
ccuracy: 0.7358 - val_loss: 1.1482 - val_accuracy: 0.7615
Epoch 8/20
469/469 [==============================] - 6s 14ms/step - loss: 1.0861 - a
ccuracy: 0.7632 - val_loss: 1.0024 - val_accuracy: 0.7766
Epoch 9/20
469/469 [==============================] - 7s 14ms/step - loss: 0.9583 - a
ccuracy: 0.7825 - val_loss: 0.8914 - val_accuracy: 0.8014
Epoch 10/20
469/469 [==============================] - 7s 14ms/step - loss: 0.8603 - a
ccuracy: 0.7989 - val_loss: 0.8058 - val_accuracy: 0.8114
Epoch 11/20
469/469 [==============================] - 7s 14ms/step - loss: 0.7834 - a
ccuracy: 0.8130 - val_loss: 0.7371 - val_accuracy: 0.8267
Epoch 12/20
469/469 [==============================] - 7s 14ms/step - loss: 0.7220 - a
ccuracy: 0.8242 - val_loss: 0.6816 - val_accuracy: 0.8330
Epoch 13/20
469/469 [==============================] - 7s 14ms/step - loss: 0.6720 - a
ccuracy: 0.8335 - val_loss: 0.6366 - val_accuracy: 0.8415
Epoch 14/20
469/469 [==============================] - 7s 14ms/step - loss: 0.6308 - a
ccuracy: 0.8421 - val_loss: 0.5982 - val_accuracy: 0.8491
Epoch 15/20
469/469 [==============================] - 7s 15ms/step - loss: 0.5962 - a
ccuracy: 0.8484 - val_loss: 0.5668 - val_accuracy: 0.8567
Epoch 16/20
469/469 [==============================] - 7s 14ms/step - loss: 0.5671 - a
ccuracy: 0.8543 - val_loss: 0.5402 - val_accuracy: 0.8614
Epoch 17/20
469/469 [==============================] - 7s 14ms/step - loss: 0.5423 - a
ccuracy: 0.8590 - val_loss: 0.5171 - val_accuracy: 0.8646
Epoch 18/20
469/469 [==============================] - 7s 14ms/step - loss: 0.5209 - a
ccuracy: 0.8635 - val_loss: 0.4964 - val_accuracy: 0.8687
Epoch 19/20
469/469 [==============================] - 7s 14ms/step - loss: 0.5022 - a
ccuracy: 0.8673 - val_loss: 0.4788 - val_accuracy: 0.8720
Epoch 20/20
469/469 [==============================] - 7s 14ms/step - loss: 0.4859 - a
ccuracy: 0.8709 - val_loss: 0.4637 - val_accuracy: 0.8752
```

In [18]:

```
w_after = model_sigmoid.get_weights()
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)
fig = plt.figure()
plt.title("Weight matricesafteremodeltrained")
plt.subplot(1,3,1)
plt.title("Trained modelweights")
ax = sns.violinplot(y=h1_w,color="b")
plt.xlabel("Hidden Layer1")
plt.subplot(1,3,2)
plt.title("Trained modelweights")
ax = sns.violinplot(y=h2_w,color="r")
plt.xlabel("Hidden Layer2")
plt.subplot(1,3,3)
plt.title("Trained modelweights")
ax = sns.violinplot(y=out_w,color="g")
plt.xlabel("Output Layer")
plt.show()
```



C:\Users\Sadiq\AppData\Local\Temp\ipykernel_26616\251596265.py:7:Matplotl ibDeprecationWarning: Auto-removal of overlapping axes is deprecated since
3.6 and will be removed two minor releases later; explicitly call ax.remov e() as needed.
    plt.subplot(1,3,1)

Practical 8

Aim : Classification of images of clothing using Tensorflow (Fashion MNIST Dataset)

The aim is to build an image classifier with **TensorFlow/Keras** on the **Fashion-MNIST** dataset (28×28 grayscale clothing images across 10 classes). You'll normalize pixel values, split train/validation/test, and train a simple **CNN** (e.g., Conv→ReLU→MaxPool→Dense with softmax). Optimize with **Adam** and **sparse categorical cross-entropy**, then evaluate using **accuracy**, **confusion matrix**, and a **classification report**. Optionally add data augmentation and early stopping to improve generalization.

[1]:
```
import tensorflow as tf
```

[2]:
```
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) =
 ↪fashion_mnist.load_data()
```

[3]:
```
train_images.shape
```

[3]: (60000, 28, 28)

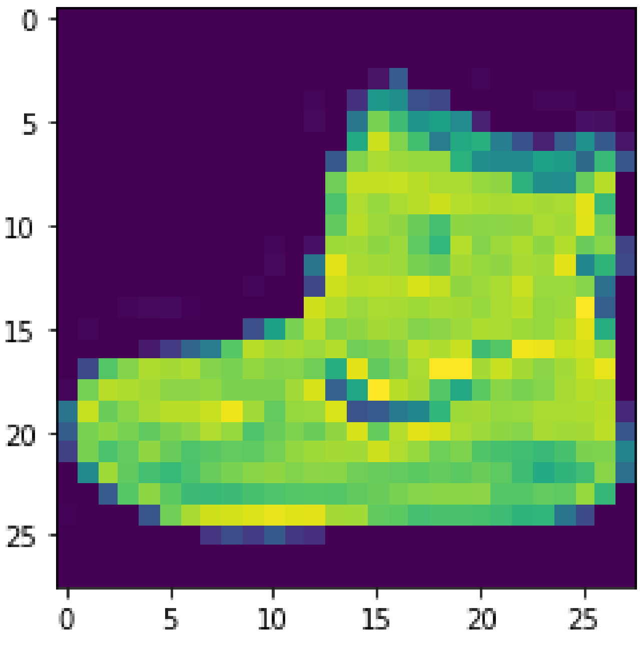
[4]:
```
test_images.shape
```

[4]: (10000, 28, 28)

[5] :
```
test_labels.shape
```

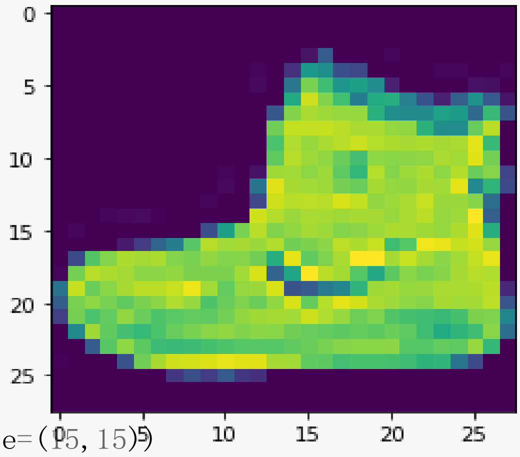
[5]: (10000,)

[6] :
```
import matplotlib.pyplot as plt
plt.imshow(train_images[0])
plt.show()
```



[7] :

```
class_names = [
    "T-shirt/top",
    "Trouser",
    "Pullover",
    "Dress",
    "Coat",
    "Sandal",
    "Shirt",
    "Sneaker",
    "Bag",
    "Ankle boot",

]
plt.figure(figsize=(15,15))

for i in range(25):
    plt.subplot(5,5,i+1)
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i]])
    plt.xticks([])
    plt.yticks([])

plt.show()
```



[8] :
```
train_images = train_images / 255.0
test_images = test_images / 255.0
train_images[0]
```

[8]:
```
array([[0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
```

[9] :

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
model.summary()
```

---

flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100480
dense_1 (Dense)	(None, 10)	1290

===

Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0

---

[10] :

```
history = model.fit(train_images, train_labels, epochs=10)
```

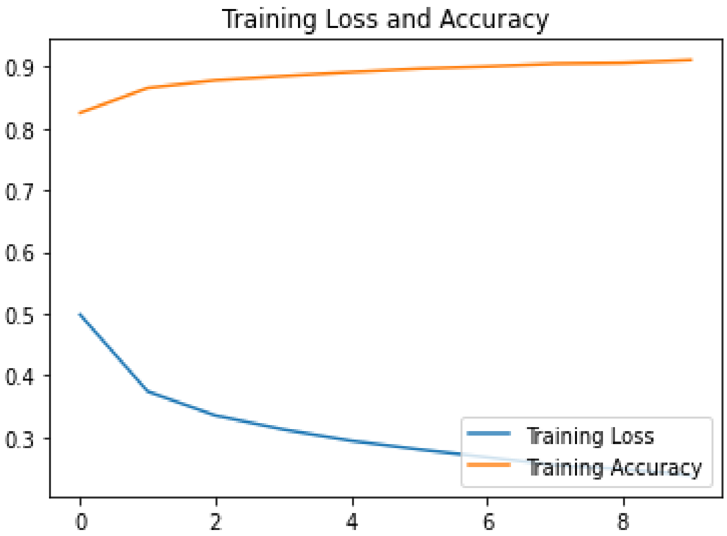
Epoch 1/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.4987 -
accuracy: 0.8251

0.8745098 , 0.85490196, 0.84705882, 0.84705882, 0.63921569,
0.47843137, 0.57254902, 0.55294118,

[11] :

```
loss = history.history["loss"]
acc = history.history["accuracy"]
epochs_range = range(10)
plt.plot(epochs_range, loss, label="Training Loss")
plt.plot(epochs_range, acc, label="Training Accuracy")
plt.legend(loc="lower right")
plt.title("Training Loss and Accuracy")
```

[11] : T



[12] :
```
test_loss, test_acc = model.evaluate(test_images, test_labels,
 ↪verbose=2)
print("Test Accuracy:", test_acc)
```

313/313 - 1s - loss: 0.3509 - accuracy: 0.8813 - 966ms/epoch - 3ms/step Test Accuracy: 0.8812999725341797

[13] :
```
probability_model = tf.keras.Sequential([model, tf.keras.layers.
 ↪Softmax()])
predictions = probability_model.predict(test_images)
predictions[0]
```

[13] : a

```
            4.2720696e-08, 1.0912937e-02,
            6.8810913e-08, 9.8725665e-01], dtype=float32)
```

[14] :
```
# view predictions
import numpy as np
np.argmax(predictions[0])
```

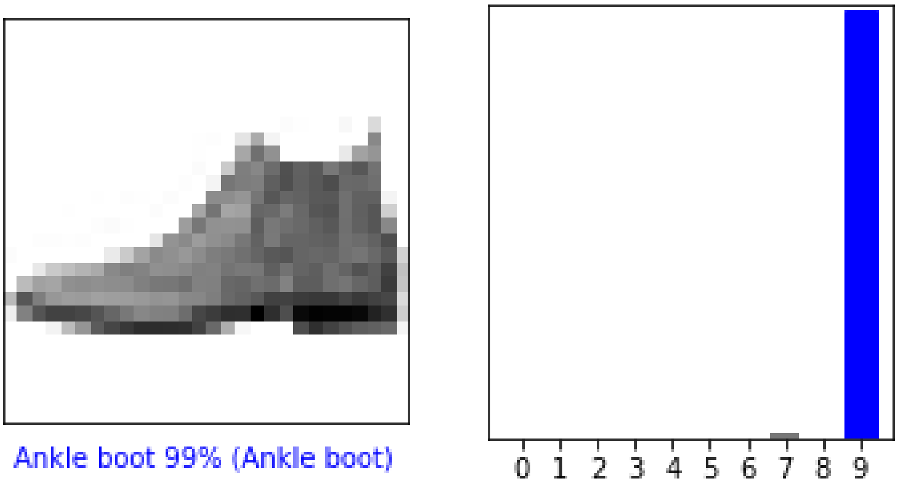
[14]: 9

[15] :
```
def plot_image(i, predictions_array, true_label, img):
    true_label, img = true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
```

```
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'blue'
    else:
        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                         100*np.max(predictions_array),
                                         class_names[true_label]),
                                         color=color)

def plot_value_array(i, predictions_array, true_label):
    true_label = true_label[i]
    plt.grid(False)
    plt.xticks(range(10))
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')
```

[16]:

```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```



[17]:

```
# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```