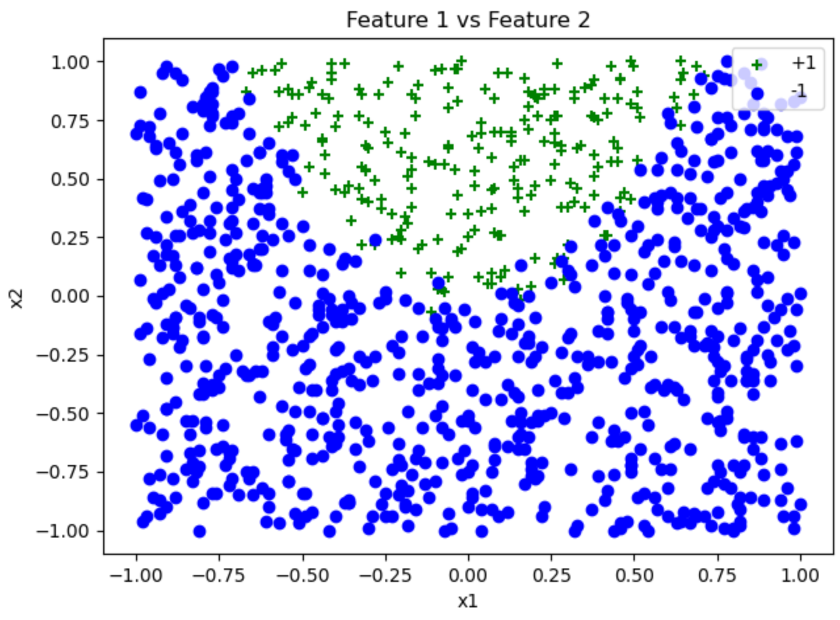
Name: Nupur Rathod Student ID-23336657 Dataset ID: 21--42-21

Machine Learning

Week-2 Assignment Report

A(1)

The given dataset has two features namely x1 and x2 whose value lies between 1 to -1. The two categories (-1 and +1) are represented by green pluses (+1) and blue dots (-1). It can be seen from the figure that data is divided into two different target value.



A(2)

Using sklearn's train_test_split function to partition the dataset into an 80% training set and a 20% testing set. Trained the regression model on the training set. The model, represented by the equation

$$\boldsymbol{y = \theta_0 + \theta_1 * x1 + \theta_2 * x2}$$

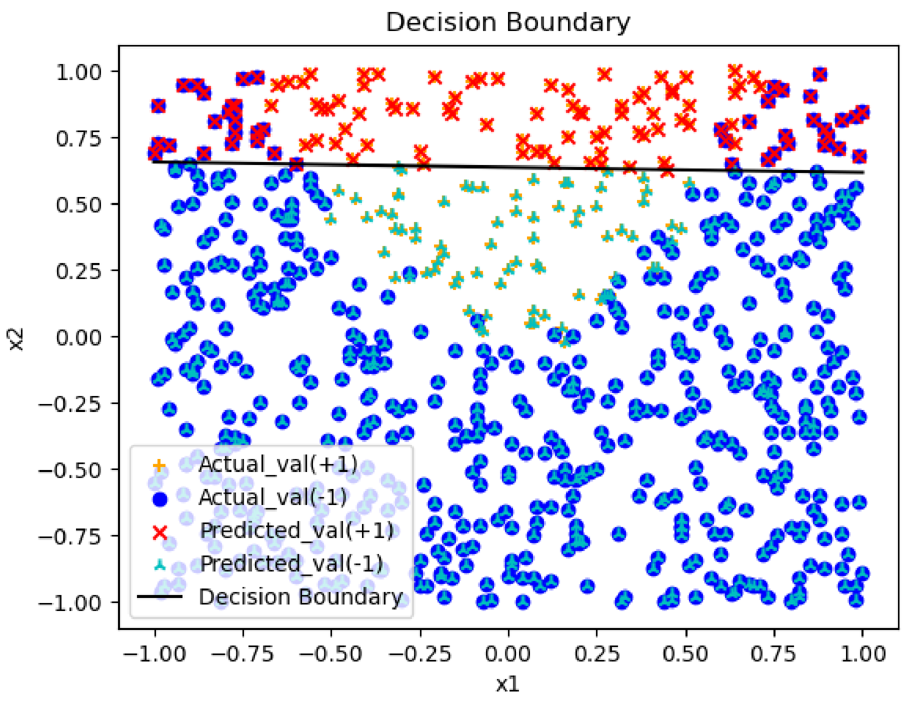
The parameter values of the trained model are θ_0 = -2.10, θ_1 = 0.064 and θ_2 = 3.30.

From the intercept and coefficient obtained it is seen that second feature (x2) has a much stronger influence on the predictions compared to the first feature (x1) as coefficient for x2 is much larger. As both coefficients are positive, it means if x1 or x2 increases, the chance of predicting class 1 also increases. However, there's a negative intercept, which means without considering the features, the model is more inclined to predict class -1.

A(3)

The decision boundary is calculated from the equation and is visualized as a black line on the plot.

Name: Nupur Rathod Student ID-23336657 Dataset ID: 21--42-21



This boundary separates the two classes; points above the line are predicted as class 1, and those below as class -1.

A(4)

The predictions made by the model align well with the training data, as visualized in the scatter plot. The decision boundary, represented by the black line, effectively separates the two classes. However, for points closer to the boundary line, there might be a few misclassifications. The test data set accuracy obtained using sklearn.metrics came to be Accuracy = 0.83.

B. 1

Using LinearSVC function and giving different values of penalty(C) the trained data gives to the model. The parameters obtained were

1.C=0.001

Coefficient=[0.0521,0.3242]

Intercept= -0.36115

It is seen that x2 has a larger coefficient, meaning it has a more influence. Both x1 and x2 have positive coefficients, so as they increase, the model leans more towards a positive classification.

2. C=1

Coefficient=[0.0197,1.2306]

Intercept= -0.7386

For C=1 coefficients are greater, especially for x2, showing that x2 is still the dominant feature. The negative intercept indicates a bias towards negative classification when both features are at zero.

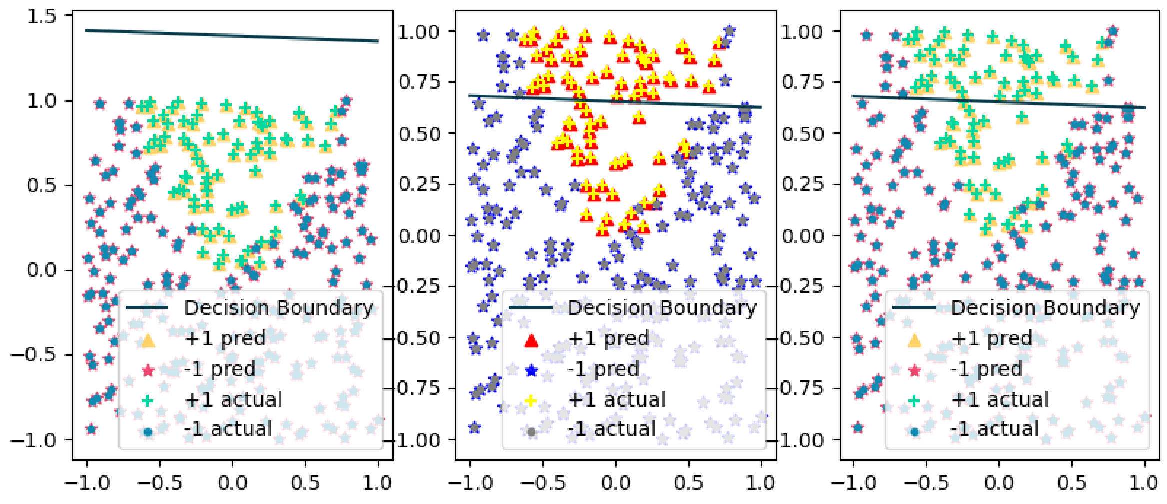
3. C= 100

Coefficient= [-0.06081,1.263]

Intercept= -0.7223

For C=100, the coefficient for x1 turns negative, meaning as x1 increases, the model is inclined towards a negative classification, opposite to the previous cases. The coefficient for x2 remains positive and large, reinforcing its dominance.

B.2



B.3

From the SVM models, it’s evident that the role and influence of x1 and x2 change dynamically with the penalty parameter C. At C=100, x1 changes its role, turning negative, and contradicting its behaviour at lower C values. x2, however, consistently remains a strong influencer, even increasing its effect as C increases.

It is observed that in the logistic regression model, both features consistently contribute towards increasing the predicted value.

B.4

It can evaluated that the accuracy of SVM models for different C values.

When C=0.001, the accuracy is 0.77. For C=1, the accuracy is 0.8266, and for C=100, it's 0.8233.

Notably, both C=1 and C=100 show slightly better accuracy compared to the previous logistic

regression model, which had an accuracy of 0.77.

C. 1.

Extracted the x1 and x2 features using pandas library and add the new features using insert function. The values of new feature is $x1^2$ and $x2^2$ respectively.

Trained the logistic regression model using the new features. The model with new parameter becomes

$$\boldsymbol{y = \theta_0 + \theta_1 * x1 + \theta_2 * x2 + \theta_3 * x1^2 + \theta_4 * x2^2}$$

The result obtained for the above equation is

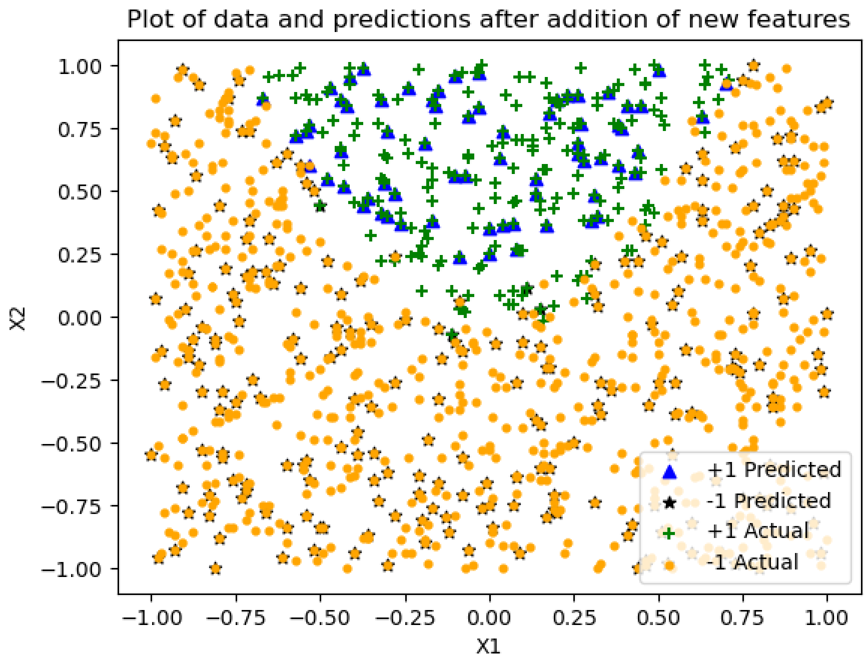
y = 1.8017080253868543 * x1 + 23.947781518755658 * x2 + -49.15955964794653 * x1^2 + -0.4638921441037072 * x2^2 + 0.5080376542644899

C. 2.

Based on x_test data set previously obtained and by adding the new features to it we get new x_test, after using predict function on new dataset we get the following figure:

The Accuracy of the model after addition of new features is 0.9633

The plot makes it clear that predicted results +1 and -1 are in blue triangle and black star while the actual values of +1 and -1 are in green plus and orange circle Therefore, based on the accuracy and the figure 4, compared with the models of a and b, the fitting results of this model are the best i.e., 96.33% accuracy



C.3.

I created a baseline predictor to measure the performance of model by setting all predictions to 1 independent of input value and compared with new model.

The accuracy of baseline model is 0.77 which is lower than that of the model we created with polynomials there so the model of polynomials is better than the baseline model

Code

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

df=pd.read_csv('ML_assign1_dataset.csv',names=['x1','x2','y'],skiprows=1)

print(df.head())
print(df.shape)
print(df.columns)

x1=df.iloc[:,0]
x2=df.iloc[:,1]
X=np.column_stack((x1,x2))
y=df.iloc[:,2]

plus1=df[df['y']==1]
minus1=df[df['y']==-1]
# Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(plus1['x1'],plus1['x2'],marker='+',label='+1',c='g')
plt.scatter(minus1['x1'],minus1['x2'],marker='o',label='-1',c='b')
plt.title('Feature 1 vs Feature 2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
LR_model=LogisticRegression()
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3)
LR_model.fit(X_train,y_train)
y_predict=LR_model.predict(X_test)
y_train_pred = LR_model.predict(X_train)


intercept=LR_model.intercept_[0]
coefficient_x1=LR_model.coef_[0][0]
coefficient_x2=LR_model.coef_[0][1]
print(intercept)
print("Coefficient for x1 is:",coefficient_x1)
print("Coefficient for x2 is:",coefficient_x2)


plt.scatter(X_train[y_train==1,0],X_train[y_train
==1,1],marker='+',label='Actual_val(+1)',c='orange')
plt.scatter(X_train[y_train==-1,0],X_train[y_train==-1,1],marker='o',label='Actual_val(-
1)',c='b')
plt.scatter(X_train[y_train_pred==1,0], X_train[y_train_pred==1,1], marker='x',
label='Predicted_val(+1)', c='r')
plt.scatter(X_train[y_train_pred==-1,0], X_train[y_train_pred==-1,1], marker='2',
label='Predicted_val(-1)', c='c')
x1_val=np.linspace(min(X_train[:, 0]), max(X_train[:, 0]), 100)
x2_val=-(intercept + coefficient_x1 * x1_val) / coefficient_x2
plt.plot(x1_val,x2_val,'k-', label='Decision Boundary')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.title('Decision Boundary')
plt.show()

from sklearn.metrics import accuracy_score
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_predict)
print(f"Training Accuracy: {train_accuracy}")
print(f"Test Accuracy: {test_accuracy}")


from sklearn.svm import LinearSVC


val = [0.001, 1, 100]
svm_models = {}
```

```
for i in val:
    svm_model = LinearSVC(C=i, random_state=42)  # Use 'i' as the C parameter
    svm_model.fit(X, y)
    svm_models[i] = svm_model

    print(f"Model with C={i}:")
    print("Coefficient (w):", svm_model.coef_)
    print("Intercept (b):", svm_model.intercept_)
    print("\n")






from sklearn.svm import LinearSVC

def get_x2(x1, LR_model):
    return (LR_model.coef_[0][0] * x1 + LR_model.intercept_) / - LR_model.coef_[0][1]


model1 = LinearSVC(C=0.001).fit(X_train, y_train)
print('C=0.001', model1.intercept_, model1.coef_[0])
model2 = LinearSVC(C=1).fit(X_train, y_train)
print('C=1', model2.intercept_, model2.coef_[0])
model3 = LinearSVC(C=100, max_iter=10000).fit(X_train, y_train)
print('C=100', model3.intercept_, model3.coef_[0])

pred1 = model1.predict(X_test)
pred2 = model2.predict(X_test)
pred3 = model3.predict(X_test)


x_pos_test = X_test[y_test == 1]
x_neg_test = X_test[y_test == -1]

fig = plt.figure(figsize=(12, 4))


ax1 = fig.add_subplot(1, 3, 1)
dec_x1_1 = np.linspace(-1.0, 1.0, 100)
dec_x2_1 = get_x2(dec_x1_1, model1)
ax1.plot(dec_x1_1, dec_x2_1, color='#073b4c', label='Decision Boundary')

x_pos_1= X_test[y_test == 1]
x_neg_1= X_test[y_test == -1]
ax1.scatter(x_pos_1[:, 0], x_pos_1[:, 1], marker='^', color='#FFD166', label='+1 pred')
ax1.scatter(x_neg_1[:, 0], x_neg_1[:, 1], marker='*', color='#EF476F', label='-1 pred')
ax1.scatter(x_pos_test[:, 0], x_pos_test[:, 1], marker='+', color='#06D6A0', label='+1
actual')
ax1.scatter(x_neg_test[:, 0], x_neg_test[:, 1], marker='o', color='#118ab2', s=10,
label='-1 actual')
ax1.legend(loc=4)
```

```
ax2 = fig.add_subplot(1, 3, 2)
dec_x1_2 = np.linspace(-1.0, 1.0, 100)
dec_x2_2 = get_x2(dec_x1_2, model2)
ax2.plot(dec_x1_2, dec_x2_2, color='#073b4c', label='Decision Boundary')
x_pos_2=X_test[y_test == 1]
x_neg_2=X_test[y_test == -1]
ax2.scatter(x_pos_2[:, 0], x_pos_2[:, 1], marker='^', color='red', label='+1 pred')
ax2.scatter(x_neg_2[:, 0], x_neg_2[:, 1], marker='*', color='blue', label='-1 pred')
ax2.scatter(x_pos_test[:, 0], x_pos_test[:, 1], marker='+', color='Yellow', label='+1
actual')
ax2.scatter(x_neg_test[:, 0], x_neg_test[:, 1], marker='o', color='grey', s=10,label='-1
actual')
ax2.legend(loc=4)


ax3 = fig.add_subplot(1, 3, 3)
dec_x1_3 = np.linspace(-1.0, 1.0, 100)
dec_x2_3 = get_x2(dec_x1_3, model3)
ax3.plot(dec_x1_3, dec_x2_3, color='#073b4c', label='Decision Boundary')
x_pos_3=X_test[y_test == 1]
x_neg_3=X_test[y_test == -1]
ax3.scatter(x_pos_3[:, 0], x_pos_3[:, 1], marker='^', color='#FFD166', label='+1 pred')
ax3.scatter(x_neg_3[:, 0], x_neg_3[:, 1], marker='*', color='#EF476F', label='-1 pred')
ax3.scatter(x_pos_test[:, 0], x_pos_test[:, 1], marker='+', color='#06D6A0', label='+1
actual')
ax3.scatter(x_neg_test[:, 0], x_neg_test[:, 1], marker='o', color='#118ab2', s=10,
label='-1 actual')
ax3.legend(loc=4)
plt.figtext(0.5, 0.95, 'Figure 3: Plot of data and prediction result and decision boundary
of different C', ha='center', va='top')
plt.show()


from sklearn.metrics import accuracy_score


score1 = accuracy_score(model1.predict(X_test), y_test)
score2 = accuracy_score(model2.predict(X_test), y_test)
score3 = accuracy_score(model3.predict(X_test), y_test)

print('C = 0.001: accuracy:', score1)
print('C = 1: accuracy:', score2)
print('C = 100: accuracy:', score3)


x1 = X_train[:, 0]
x2 = X_train[:, 1]
data_new = np.insert(X_train, 2, values = x1 * x1, axis = 1)
data_new = np.insert(data_new, 3, values = x2 * x2, axis = 1)
print(data_new)
x_new = data_new
y_new = y_train
model_new = LogisticRegression()
```

```
model_new.fit(x_new, y_new)
x_test_new = np.insert(X_test, 2, values=X_test[:,0]**2, axis=1)
x_test_new = np.insert(x_test_new, 3, values=X_test[:,1]**2, axis=1)
pred_new = model_new.predict(x_test_new)
x_new_pos = x_test_new[pred_new == 1]
x_new_neg = x_test_new[pred_new == -1]

coefficients = model_new.coef_[0]
intercept = model_new.intercept_[0]

print(f'y = {coefficients[0]} * x1 + {coefficients[1]} * x2 + {coefficients[2]} * x1^2 +
{coefficients[3]} * x2^2 + {intercept}')


plt.scatter(x_new_pos[:, 0], x_new_pos[:, 1], marker='^', color='blue', label='+1
Predicted')
plt.scatter(x_new_neg[:, 0], x_new_neg[:, 1], marker='*', color='black', label='-1
Predicted')
plt.scatter(df[df['y']== 1]['x1'], df[df['y']== 1]['x2'], marker='+', label='+1 Actual',
c='green')
plt.scatter(df[df['y']==-1]['x1'], df[df['y']==-1]['x2'], marker='o', label='-1 Actual',
c='red', s=12)
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend(loc=4)
plt.title("Plot of data on new features")
plt.show()



from sklearn.metrics import accuracy_score
accuracy_new = accuracy_score(pred_new, y_test)

print("The accuracy of the model is : %f" % accuracy_new)


from sklearn.metrics import accuracy_score
logistic_model = LogisticRegression()
logistic_model.fit(np.column_stack((x1, x2)), y_train)

y_pred = logistic_model.predict(np.column_stack((x1, x2)))

classifier_accuracy = accuracy_score(y_train, y_pred)

most_common_class = y_train.mode().values[0]
baseline_predictions = np.full(y_train.shape, most_common_class)

baseline_accuracy = accuracy_score(y_train, baseline_predictions)

print(f"Logistic Regression Classifier Accuracy: {classifier_accuracy:.2f}")
print(f"Baseline Predictor Accuracy: {baseline_accuracy:.2f}")
```

Name: Nupur Rathod Student ID-23336657 Dataset ID: 21--42-21