Assignment Report – Machine Learning

Name: Kumar Naman

Student ID: 25340053

Course: Machine Learning

Date: 10/10/2025

Dataset ID: 12-12-12

I) Introduction

The report provides an implementation and an analysis on two of the most popular classification algorithms used in modern computing - **Support Vector Machines(SVMs)** and the **Logistic Regression Algorithm**. A random dataset with two input variables(x_1, x_2) and an output target variable with values ±1is used, making it a suitable dataset to analyse and implement the above **Binary Classification Algorithms**.

As discussed earlier, we explore the 2 algorithms and plot various graphs to visualise the working of a given algorithm. First we examine a **simple (Linear) Logistic Regression Algorithm**, analysing how well the algorithm works for the given dataset. Second we explore the **linear SVM classifiers** with varying regularization strengths(by controlling the C variable). Finally we explore a Logistic Regression model with polynomial(squared) variables as the parameters and evaluate this against all the above mentioned algorithms.

II) **Data Visualization and Exploration**

This section gives us a basic idea of the dataset provided. There is a basic 2D graph plot giving us a high level overview on the type of data and the relationship between the input dataset and the output data.

II.I) Dataset Identifier and Description

- The dataset belongs to the unique dataset ID 12-12-12 which was obtained from the first line of the file provided.

- The dataset contains 999 samples with two input features (X_1, X_2) and a binary target variable $y \in \{-1, +1\}$.
- Basic Analysis of data provides us with the following information:
    - There are 671 inputs that result into 1 and 328 inputs that result in -1.
    - The values of the input variables x1 and x2 lie between -1 and 1.

II.II) 2D Scatter Plot of Input and Output

Figure 1 shows the spatial distribution of training samples in the two-dimensional feature space.

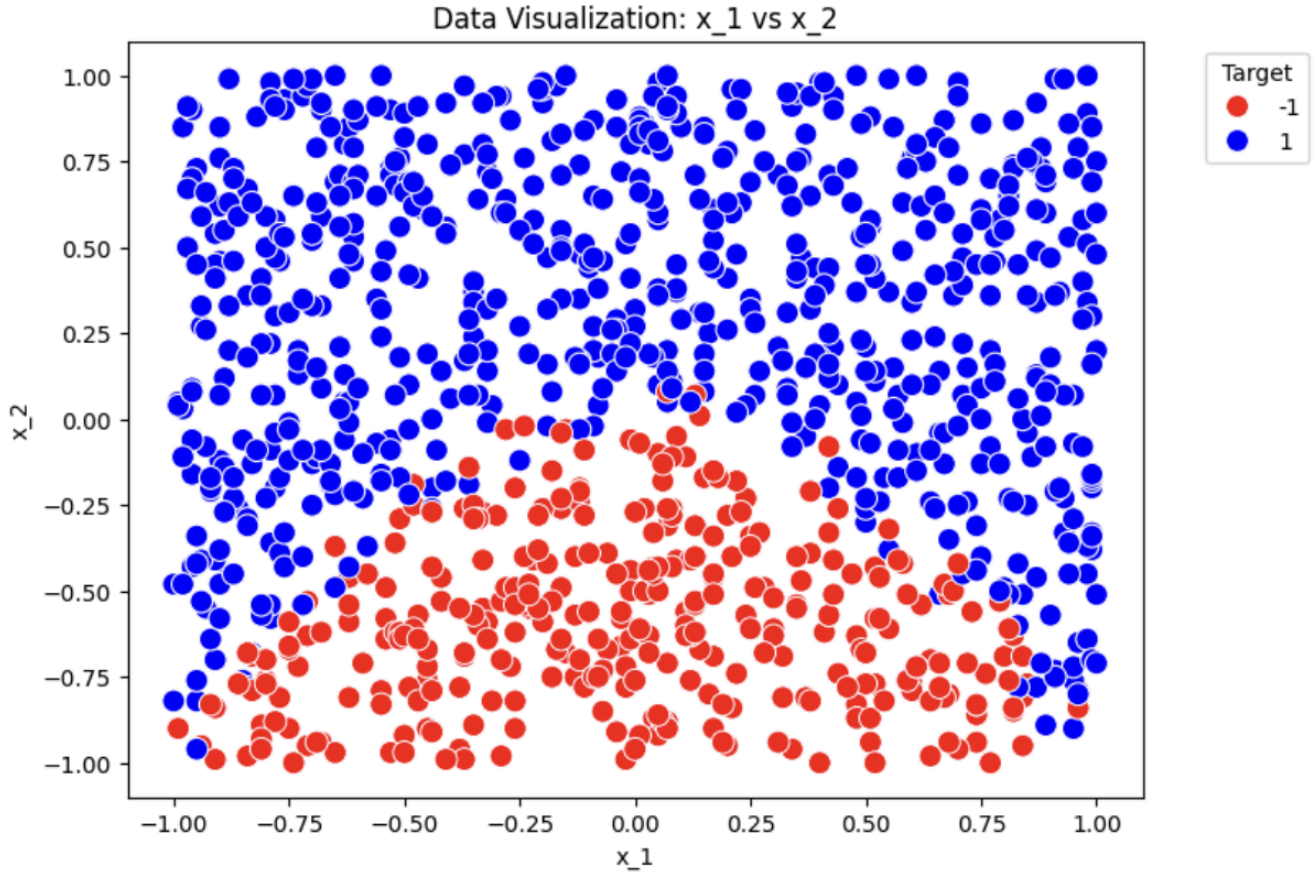


Fig 1. Input parameter vs Output(Target) Variable

II.III) Observations and Insights

- The -1 values form the bottom half of the scatter plot whereas +1 covers the upper half suggesting that a classification algorithm will work well with such a dataset.
- The separation between the two classes is very parabolic and indicate that a polynomial solution might end up working better than a simple linear solution.

III) Logistic Regression Classification

Logistic regression is a classification algorithm that builds upon the logic of simple linear (Ordinary Least Squares) regression but is designed for classification tasks rather than continuous prediction. The logistic regression algorithm leverages the sigmoid function to achieve values between 0 and 1.

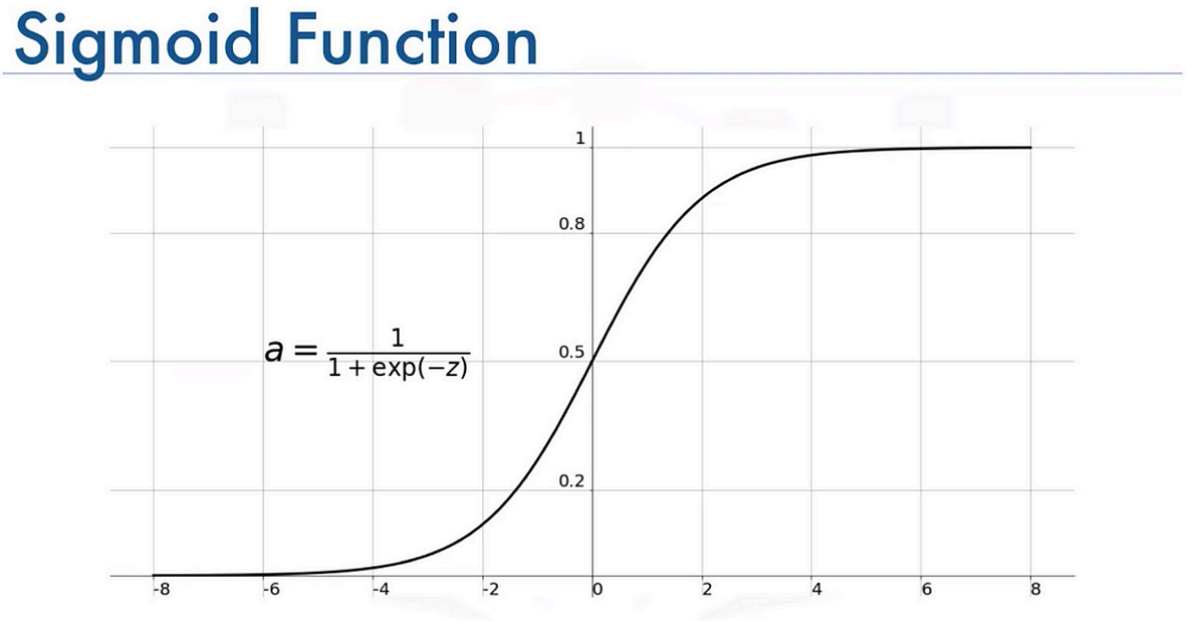


Figure 2. Sigmoid Function[1]

We use the following simple regression equation:$\mathbf{y = b + w^{T}x}$

And pass this as the z parameter in the sigmoid function $\sigma(z) = 1/(1 + e^{-z})$ which gives us the final equation:

$$\mathbf{P(y = +1 \mid x) = \sigma(b + w^{T}x).}$$

The model predicts class +1 when $b + \mathbf{w^{T}x} \geq 0$ (i.e., when $P(y = +1) \geq 0.5$), and class -1 otherwise. This creates a linear decision boundary defined by the hyperplane $b + \mathbf{w^{T}x} = 0$.

In logistic regression, the **decision boundary** is defined by the set of points where the model's predicted probability is 0.5, i.e., w1x1 + w2x2 + ... + b = 0

III.I) Cost Function in Logistic Regression

Unlike linear regression which uses Mean Squared Error (MSE), logistic regression employs a different cost function called the **log-loss** or **cross-entropy loss** to train the model parameters. The cost function for logistic regression is defined as:

$$J(w) = -1/m \sum_{i=1}^{m} [y_i \log(\sigma(b + w^T x_i)) + (1 - y_i) \log(1 - \sigma(b + w^T x_i))]$$

where m is the number of training samples, $y_i \in \{0, 1\}$ is the true label (or {-1, +1} converted to {0, 1}), and $\sigma(b + w^T x_i)$ is the predicted probability. This cost function is derived from the statistical principle of maximum likelihood. During training, sklearn's LogisticRegression minimizes this cost function by using gradient descent and various other algorithms

III.II) Logistic Regression using skLearn

The logistic regression model was trained using sklearn's LogisticRegression class which handles the mathematics under the hood and provides us with the parameter values and interesting insights.

Trained Model Parameters:

- Bias term: b = 1.8072216360101805
- Weight for feature X_1: w_1 = 0.07278969044594163
- Weight for feature X_2: w_2 = 5.06768887804953
- The final prediction model is: **$P(y = +1 \mid x) = \sigma(1.807 + 0.073x_1 + 5.068x_2)$**

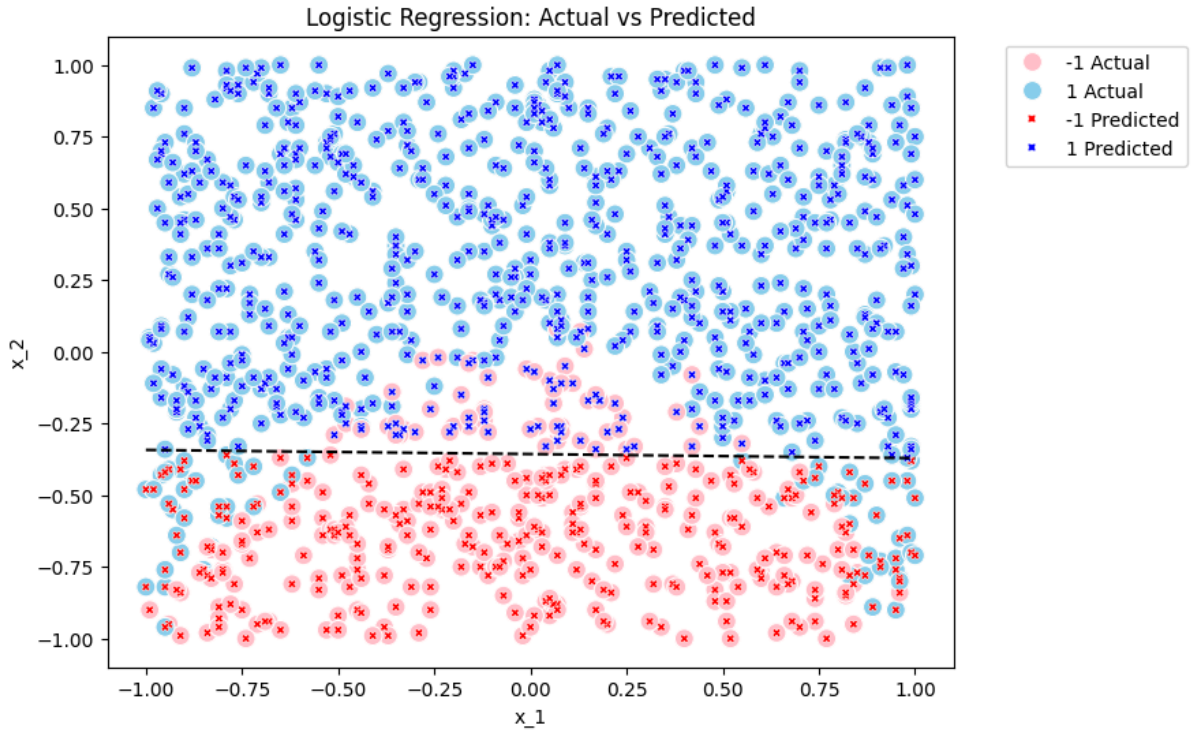


Figure 3. Logistic Regression. Predicted vs Actual Values

III.III) Interpretation of Model Parameters and Feature Importance

Feature X_2 is the dominant predictor, having substantially greater influence on the classification decision. The ratio $|w_2|/|w_1| = 5.068/0.073 \approx 69.5$ indicates that feature X_2 is approximately 70 times more influential than feature X_1 in determining the class prediction. This shows that X2 is the primary determining factor in the values of the output whereas X1 plays a smaller role.

The bias b = 1.807 is **positive and relatively large**, which has important implications:

- When both features equal zero ($x_1 = 0$, $x_2 = 0$), the model computes $\sigma(1.807) \approx 0.859$
- This means the model has a **strong baseline tendency toward predicting the +1 class**
- This suggests potential **class imbalance** in the training data, with more +1 samples than -1 samples, or that the +1 class occupies a larger region of the feature space

III.IV) Conclusions

1. Accuracy: 87.09%: Strong overall performance with 870 out of 999 samples correctly classified, representing a ~20 percentage point improvement over baseline (67.17%).

2. Precision: 90.57%: shows that when the model predicts the positive class, it is correct most of the time , meaning few false positives.

3. Recall: 90.16% : suggests that the model is also effective in identifying actual positives, with few missed cases (false negatives).

4. F1-Score: 90.37% : being the harmonic mean of precision and recall, confirms that the model maintains a good balance between precision and recall, indicating consistent performance.

5. ROC-AUC Score: 85.48%: further demonstrates that the model has strong discriminative power, effectively distinguishing between the two classes

6. Confusion Matrix:

[[(TN)265 (FP)63] TN : True Negative; FP - False Positive

[(FN)66 (TP)605]] FN : False Negative; TP - True Positive;

IV. Linear SVM Classification

Support Vector Machine (SVM) is a powerful classification algorithm that aims to find the optimal hyperplane that maximizes the margin between classes. SVM focuses on geometric margin maximization while handling misclassifications through a penalty parameter (C).

For a binary classification problem with features $\mathbf{x} = [x_1, x_2]^T$ and labels $y \in \{-1, +1\}$, the linear SVM seeks to find a hyperplane defined by:$\mathbf{b + w^T x = 0}$

The SVM prediction function is: $\hat{\mathbf{y}} = \mathbf{sign(b + w^T x)}$

where sign(z) = +1 if $z \geq 0$, and -1 otherwise.

Optimization Objective:

The SVM formulation solves the following optimization problem:

minimize: $(1/2)||\mathbf{w}||^2 + C \sum_{i=1}^{m} \xi_i$

subject to: $y_i(b + \mathbf{w^T x_i}) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

IV.I) Goals of SVM

- **Maximize the margin** : Create the widest possible separation between classes
- **Minimize classification errors** (minimize $\Sigma\xi_i$): Reduce misclassifications and margin violations

The parameter C controls this tradeoff:

- **Large C**: Prioritizes minimizing training errors
- **Small C**: Prioritizes maximizing margin

IV.II) Implementation of SVM for different values of C using sklearn

We train linear SVM classifiers for three different values of the regularization parameter: C = 0.001 (high regularization), C = 1 (moderate regularization), and C = 100 (low regularization) with the following results:

Model for penalty=0.001:

- W1 = 0.003642230169552038
- W2 = 0.465672058117645
- Intercept (b): 0.228691419172776

- Insights: values for W1 and W2 are very small to keep a wide margin, this results in over regularization and leads to an underfit of the decision boundary as can be seen from figure 4.

Model for penalty=1:

- W1 = 0.020392878335181604
- W2 = 1.8947031861798747
- Intercept (b): 0.6578538168254331
- Insights: Weights and intercept rise noticeably, giving a good balance between margin width and classification accuracy.

Model for penalty=100:

- W1 = 0.020848075857074447
- W2 = 1.9155387525690974
- Intercept (b): 0.665901910293073
- Insights: Weights and intercepts increase slightly more, the model becomes highly sensitive to data and may start overfitting.

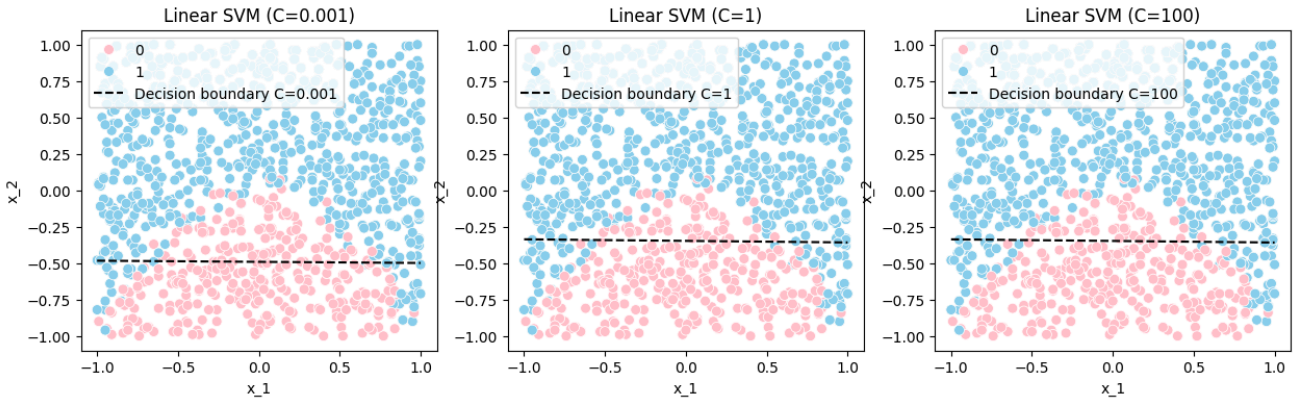


Figure 4. SVM with different values of C

IV.III) Conclusion

Both linear SVM and logistic regression produce nearly identical decision boundaries and achieve comparable classification performance on this dataset, confirming they converge to the same optimal linear separator. While the parameter magnitudes differ due to different optimization objectives,both methods consistently identify feature X_2 as the dominant predictor.

V. Polynomial Logistic Regression

The logistic regression model in Section III achieved 87.09% accuracy using a linear decision boundary based on the original two features (x_1, x_2). While this performance is strong, the 13% error rate and analysis of misclassified samples suggest that a purely linear separator may not fully capture the underlying structure of the data which can also be captured visually from the graphs.

The original logistic regression model had the form: $\mathbf{P(y = +1 \mid x) = \sigma(b + w_1x_1 + w_2x_2)}$

This linear combination of features produces a straight-line decision boundary. However, by adding **quadratic terms** (x_1^2 and x_2^2) as additional features, we create an extended model:

$$\mathbf{P(y = +1 \mid x) = \sigma(b + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2)}$$

V.I) Polynomial Regression using sklearn

We performed the above-mentioned regression by changing the input parameter and passing it to the previously used LogisticRegression class.

We discuss the values of weights and biases obtained in the following section:

Linear Terms (x_1 and x_2):

- $\mathbf{w_1 = 0.055}$: Minimal direct influence from x_1, similar to the simple logistic regression (was 0.073)
- $\mathbf{w_2 = 6.769}$: Strong positive influence from x_2(was 5.068)
- Feature X_2 remains the dominant linear predictor, but its influence has increased

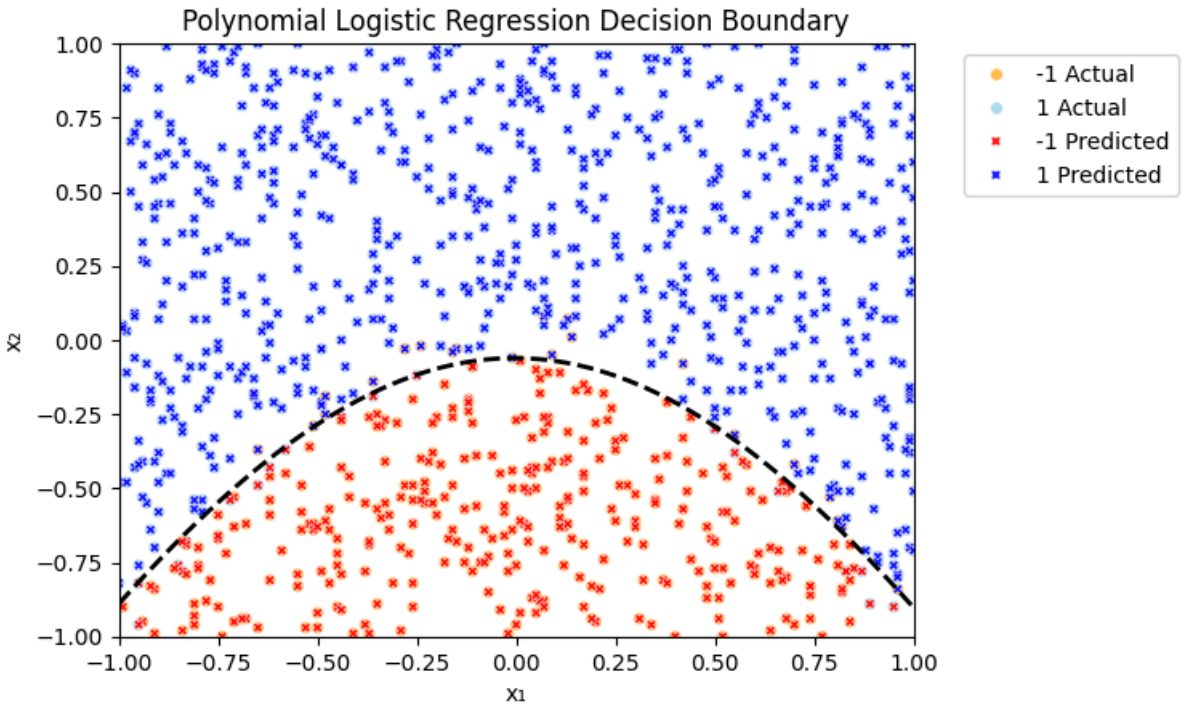
Quadratic Terms (x_1^2 and x_2^2):

- $\mathbf{w_3 = 6.449}$: Despite x_1's minimal linear influence (w_1=0.055), the large positive weight on x_1^2 means samples with extreme x_1 values (either direction) are strongly pushed toward class +1, indicating a U-shaped relationship where distance from the x_1 axis is highly discriminative.
- $\mathbf{w_4 = -0.993}$: The negative weight on x_2^2 counteracts the positive linear effect of x_2 (w_2=6.769), creating an inverted parabola where moderate x_2 values favor class +1 but extreme x_2 values are pushed back toward class -1.

Intercept:

- $\mathbf{w_0 = 0.417}$: Significant decrease from linear models(was 1.807)

- The quadratic terms have absorbed much of the bias, reducing tendency toward +1



V.II) Conclusion

Metric	Linear Logistic Regression	Polynomial Logistic Regression
Accuracy	0.871	0.958
Precision	0.906	0.967
Recall	0.902	0.970
F1-Score	0.904	0.969

ROC-AUC score	0.855	0.952
Confusion Matrix	[[265, 63], [66, 605]]	[[306, 22], [20, 651]]

- Polynomial logistic regression significantly outperforms linear logistic regression across all metrics. Accuracy rises from 87% → 96%, and F1-score improves from ~0.90 → ~0.97.
- Misclassifications are reduced. False positives decrease from 63 → 22, and false negatives from 66 → 20.

VI. APPENDIX

- **Importing all libraries**

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score, confusion_matrix,
accuracy_score

from sklearn.preprocessing import PolynomialFeatures

from sklearn.svm import LinearSVC
```

- **Read and save Data**

```
df = []

with open('week2_dataset.php', 'r') as f:

    i = f.readlines()
```

```
i = i[1:]

data = [line.strip() for line in i]

data = [d.split(",") for d in data]

df = pd.DataFrame(data, columns=['x_1','x_2','output'])
```

- **Logistic Regression**

  a)

  i. Visualise the data you downloaded by placing a marker on a 2D plot for each

  pair of feature values i.e. for each row in the data. On the plot the x-axis should be the value of the first feature, the y-axis the value of the second feature and the marker should be, for example, a + marker when the target value is +1 and a o when the target is −1. Your plot should look similar in style to this (with different data points of course!)

```
df = df.astype(float)

df['output'] = df['output'].astype(int)

df['output_binary'] = df['output'].replace(-1, 0)

df.dtypes

plt.figure(figsize=(8,6))

sns.scatterplot(

    x='x_1', y='x_2',

    hue='output',

    data=df,

    palette={-1:'red', 1:'blue'},

    s=100

)

plt.xlabel('x_1')

plt.ylabel('x_2')

plt.title('Data Visualization: x_1 vs x_2')
```

```
plt.legend(title='Target', bbox_to_anchor=(1.05, 1),
loc='upper left')  # legend outside

plt.show()
```

(ii) Use sklearn to train a logistic regression classifier on the data. Give the logistic regression model for predictions and report the parameter values of the trained model. Discuss e.g. which feature has most influence on the prediction, which features cause the prediction to increase and which to decrease

```
inp = df[['x_1','x_2']] # input parameters

output = df['output_binary']



regression = LogisticRegression(fit_intercept=True)

regression.fit(inp, output)



print("b:", regression.intercept_[0])

print("w1:", regression.coef_[0][0])

print("w2:", regression.coef_[0][1])
```

(iii) Now use the trained logistic regression classifier to predict the target values in the training data. Add these predictions to the 2D plot you generated in part (i), using a different marker and colour so that the training data and the predictions can be distinguished. Show the decision boundary of the logistic regression classifier as a line on the plot (you'll need to use the parameter values of the trained model to figure out what line this should be - explain how you obtain it)

```
df['predicted_output'] = df['predicted_output'].astype(int)



# Decision boundary

x_vals = np.linspace(df['x_1'].min(), df['x_1'].max(), 100)

y_vals = -(regression.coef_[0][0]*x_vals +
regression.intercept_[0])/regression.coef_[0][1]
```

```
plt.figure(figsize=(8,6))


# Original points
sns.scatterplot(
    x='x_1', y='x_2',
    hue='output',
    data=df,
    palette={-1:'pink', 1:'skyblue'},
    s=100,
    legend='full'
)


# Predicted points
sns.scatterplot(
    x='x_1', y='x_2',
    hue='predicted_output',
    data=df,
    palette={-1:'red', 1:'blue'},
    s=20,
    marker='X',
    legend='full',
)


# Decision boundary plot
plt.plot(x_vals, y_vals, 'k--', label='Decision boundary')
```

```
plt.xlabel('x_1')

plt.ylabel('x_2')

plt.title('Logistic Regression: Actual vs Predicted')

handles, labels = plt.gca().get_legend_handles_labels()

plt.legend(handles, graph_labels, bbox_to_anchor=(1.05, 1),
loc='upper left')


plt.show()
```

- **Conclusions - part 1**

```
accuracy = accuracy_score(df['output'], df['predicted_output'])

precision = precision_score(df['output'], df['predicted_output'])

recall = recall_score(df['output'], df['predicted_output'])

f1 = f1_score(df['output'], df['predicted_output'])

roc_auc = roc_auc_score(df['output'], df['predicted_output'])

cm = confusion_matrix(df['output'], df['predicted_output'])

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1-Score:", f1)

print("ROC-AUC score:", roc_auc)

print("Confusion Matrix", cm)
```

- S**VM Classifiers**

Use sklearn to train linear SVM classifiers on your data (nb: be sure to use theLinearSVC function in sklearn, not the SVC function).

(i) Train linear SVM classifiers for a wide range of values of the penalty parameter

C e.g. C = 0.001, C = 1, C = 100. Give the SVM model for predictions and

report the parameter values of each trained mode

```
C_penalties = [0.001, 1, 100]

models = {}

for C in C_penalties:

    svm = LinearSVC(C=C, max_iter=10000, random_state=42)

    svm.fit(inp, output)

    models[C] = svm


    print(f"Model for penalty={C}:")

    print("  W1 = ", svm.coef_[0][0])

    print("  W2 = ", svm.coef_[0][1])

    print("  Intercept (b):", svm.intercept_[0])

    print()
```

ii) Use each of these trained classifiers to predict the target values in the training

data. Plot these predictions and the actual target values from the data, together

with the classifier decision boundary

```
x_vals = np.linspace(inp['x_1'].min(), inp['x_1'].max(), 900)
```

```
plt.figure(figsize=(15,4))


for i, C in enumerate(C_penalties):

    svm = models[C]

    y_vals = -(svm.coef_[0][0]*x_vals +
svm.intercept_[0])/svm.coef_[0][1]


    plt.subplot(1, 3, i+1)


    # Actual data points

    sns.scatterplot(x='x_1', y='x_2', hue='output_binary',
data=df, palette={0:'pink',1:'skyblue'}, s=50)


    # Decision boundary

    plt.plot(x_vals, y_vals, 'k--', label=f'Decision boundary
C={C}')


    plt.title(f'Linear SVM (C={C})')

    plt.xlabel('x_1')

    plt.ylabel('x_2')

    plt.legend(loc='upper left')



plt.show()
```

(c) (i) Now create two additional features by adding the square of each feature (i.e.

giving four features in total). Train a logistic regression classifier. Give the

model and the trained parameter values

```
poly = PolynomialFeatures(degree=2, include_bias=False)

squared = poly.fit_transform(inp)

Squared

squared_input = pd.DataFrame(squared,
columns=poly.get_feature_names_out(['x_1', 'x_2']))

squared_input = squared_input.drop(columns=['x_1 x_2'])

squared_prediction = LogisticRegression(fit_intercept=True)

squared_prediction.fit(squared_input, output)

print("W1 (x_1) =", squared_prediction.coef_[0][0])

print("W2 (x_2) =", squared_prediction.coef_[0][1])

print("W3 (x1_sq) =", squared_prediction.coef_[0][2])

print("W4 (x2_sq) =", squared_prediction.coef_[0][3])

print("Intercept =", squared_prediction.intercept_[0])

w = squared_prediction.coef_[0]

b = squared_prediction.intercept_[0]

df['squared_predictions_binary'] =
squared_prediction.predict(squared_input)

df['squared_prediction'] =
df['squared_predictions_binary'].replace(0, -1)


x1_vals = np.linspace(df['x_1'].min(), df['x_1'].max(), 200)

x2_vals = np.linspace(df['x_2'].min(), df['x_2'].max(), 200)

X1, X2 = np.meshgrid(x1_vals, x2_vals)
```

```
Z = w[0]*X1 + w[1]*X2 + w[2]*X1**2 + w[3]*X2**2 + b



# Plot actual points
sns.scatterplot(
    x='x_1', y='x_2',
    hue='output_binary',
    data=df,
    palette={0:'orange', 1:'skyblue'},
    s=30,
    alpha=0.7
)

# Plot predicted points
sns.scatterplot(
    x='x_1', y='x_2',
    hue='squared_predictions_binary',
    data=df,
    palette={0:'red', 1:'blue'},
    s=20,
    marker='X',
    alpha=0.9
)

# Plot the decision boundary (Z=0)
```

```
plt.contour(X1, X2, Z, levels=[0], colors='k', linewidths=2,
linestyles='--')



plt.title("Polynomial Logistic Regression Decision Boundary")

plt.xlabel("x₁")

plt.ylabel("x₂")

handles, labels = plt.gca().get_legend_handles_labels()

plt.legend(handles, graph_labels, bbox_to_anchor=(1.05, 1),
loc='upper left')

plt.show()
```

##Conclusions

```
print("accuracy of baseline predictor = ", max(num_pos,
num_neg)/(num_neg+ num_pos))



squared_accuracy = accuracy_score(df['output'],
df['squared_predictions'])

squared_precision = precision_score(df['output'],
df['squared_predictions'])

squared_recall = recall_score(df['output'],
df['squared_predictions'])

squared_f1 = f1_score(df['output'],
df['squared_predictions'])

squared_roc_auc = roc_auc_score(df['output'],
df['squared_predictions'])

squared_cm = confusion_matrix(df['output'],
df['squared_predictions'])

print("Accuracy:", squared_accuracy)

print("Precision:", squared_precision)

print("Recall:", squared_recall)
```

```
print("F1-Score:", squared_f1)

print("ROC-AUC score:", squared_roc_auc)

print("Confusion Matrix", squared_cm)
```