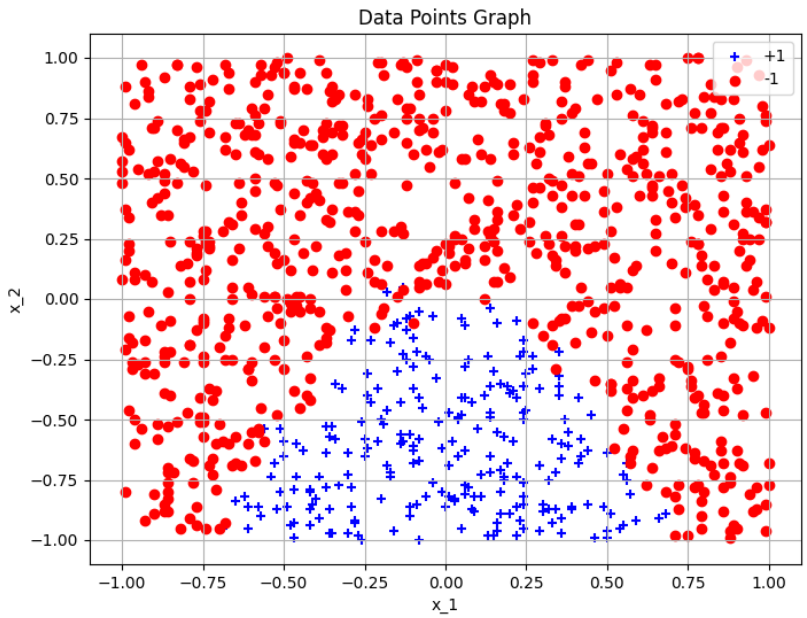
Assignment Week 2

Name: Patrick Farmer Student Number 201331828

Dataset: 16–32–16

A)i)

The data was visualised on a 2D plot for each pair of feature values. As seen in the below graph:



A)ii)

A logistic regression classifier was trained on the data by using the following code:

```
model = LogisticRegression()
model.fit(X, y)

coefficients = model.coef_[0]
intercept = model.intercept_[0]
print("Model coefficients:", coefficients)
print("Model intercept:", intercept)
```

The result of what was printed to terminal is as follows:

```
Model coefficients: [-0.05951135 -3.56226871]
Model intercept: -2.1026321370873173
```

We can see from this that the magnitude of the second feature is much greater than the magnitude of the first feature. This means that the second coefficient has a much greater effect on the prediction.

A)iii)

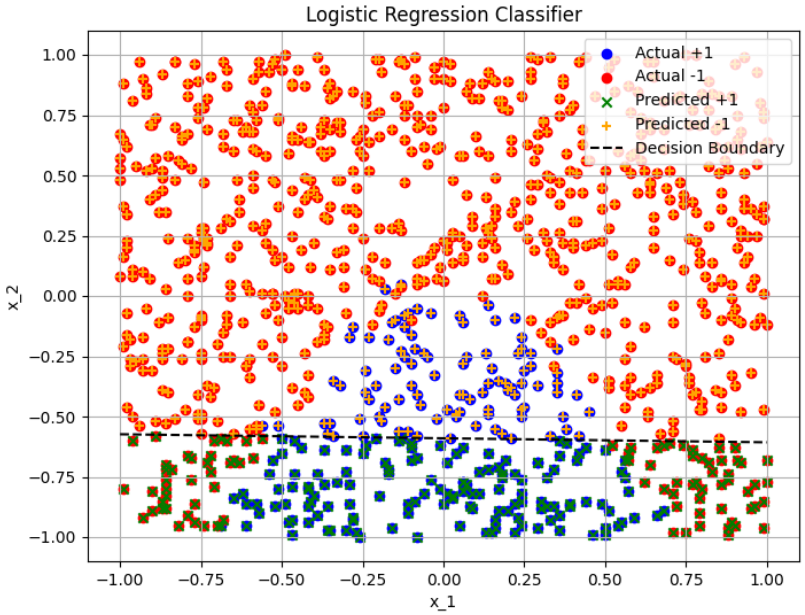
The trained logistic regression classifier was then used to predict the class of the data and this was then plotted onto the graph. This was done using the following formula: (where beta refers to the coefficients and beta_0 refers to the intercept)

$$y_p = \beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$$

this was done with the following code: *(The original graph format had to be slightly diverged from to remain clear but a legend is provided)*

```
predictions = model.predict(X)
plt.figure(figsize=(8, 6))
plt.scatter(X[y == 1, 0], X[y == 1, 1], marker='o', color='blue', label='Actual +1')
plt.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', color='red', label='Actual -1')
plt.scatter(X[predictions == 1, 0], X[predictions == 1, 1], marker='x', color='green',
    label='Predicted +1')
plt.scatter(X[predictions == -1, 0], X[predictions == -1, 1], marker='+', color='orange',
    label='Predicted -1')
x_values = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
decision_boundary = -(coefficients[0] * x_values + intercept) / coefficients[1]
plt.plot(x_values, decision_boundary, color='black', linestyle='--',
    label='Decision Boundary')

plt.xlabel('x_1')
plt.ylabel('x_2')
plt.legend()
plt.title('Logistic Regression Classifier')
plt.grid(True)
plt.savefig(os.path.join(fileDirectory, 'A(iii).png'))
```

From this code the following graph was generated: *(The original graph format had to be slightly diverged from to remain clear but a legend is provided)*



A)iv)

The classifier does not do a good job at predicting the class as we can see from the amount of incorrect guesses on the graph. This is due to the fact that the data is not linearly separable. The line that was generated by the logistic regression is the best possible prediction for a linear classifier but a squared classifier would be much better suited for this data.

B)i)

Using LinearSVC a model was trained for a range of penalty parameters C. The following code was used to train the models:

```
C_values = [0.01, 1, 100, 1000]
models = []
for C in C_values:
    model = make_pipeline(StandardScaler(), LinearSVC(C=C, max_iter=1000000))
    model.fit(X, y)
    models.append((C, model))

for C, model in models:
    print(f"Model trained with C={C}")
    coefficients = model.named_steps['linearsvc'].coef_
```

```
    intercept = model.named_steps['linearsvc'].intercept_
    print("Model coefficients:", coefficients)
    print("Model intercept:", intercept)
```

The coefficients and intercepts for each model were printed to the terminal and are as follows:

```
C_values = [0.01, 1, 100, 1000]
models = []
for C in C_values:
    model = make_pipeline(StandardScaler(), LinearSVC(C=C, max_iter=1000000))
    model.fit(X, y)
    models.append((C, model))

for C, model in models:
    print(f"Model trained with C={C}")
    coefficients = model.named_steps['linearsvc'].coef_
    intercept = model.named_steps['linearsvc'].intercept_
    print("Model coefficients:", coefficients)
    print("Model intercept:", intercept)
```

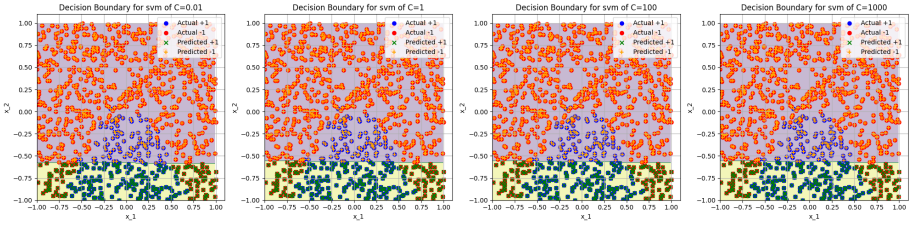
B)ii)

The trained models were then used to predict the class of the data and this was then plotted onto the graph. This was done using the following code:

```
x_min, x_max = -1, 1
y_min, y_max = -1, 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))

fig, axes = plt.subplots(1, len(C_values), figsize=(20, 5))

for ax, (C, model) in zip(axes, models):
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, alpha=0.3)
    ax.scatter(X[y == 1, 0], X[y == 1, 1], marker='o', color='blue', label='Actual +1')
    ax.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', color='red', label='Actual -1')

    predictions = model.predict(X)
    ax.scatter(X[predictions == 1, 0], X[predictions == 1, 1], marker='x', color='green',
        label='Predicted +1')
    ax.scatter(X[predictions == -1, 0], X[predictions == -1, 1], marker='+', color='orange'
        label='Predicted -1')
```

```
    ax.set_title(f'Decision Boundary for svm of C={C}')
    ax.set_xlabel('x_1')
    ax.set_ylabel('x_2')
    ax.legend()
    ax.grid(True)

plt.tight_layout()
plt.savefig(os.path.join(fileDirectory, 'B(ii).png'))
```

From this code the following graph was generated which shows the classifier decision boundary for each model:



B)iii)

The graphs look very much the same except for some small differences but do not tell use much. The coefficients and intercepts printed to terminal do differ somewhat though. As C gets larger the coefficients get larger and converge to the best set of coefficients for the training data. This is not always wanted though as the model can easily overfit. The smaller values for C are just trying to follow the trend of the data. This is difficult to observe in this case though due to the LinearSVC model being used not fitting the data well.

B)iv)

The predictions much like with the linear regression do not do a very good job and predicting the class of the data. This is due to the fact that the data is not linearly separable. The decision boundary is converging towards the best possible prediction for the training data with a linear classifier but a squared classifier would be much better suited for this data.

C)i)

Another classifier was trained by creating two additional features by adding the square of each feature to give four features in total. The following code was used to train the model:

```
X_squared = np.column_stack((X, X**2))
model = LogisticRegression()
model.fit(X_squared, y)
```

```
predictions = model.predict(X_squared)
coefficients = model.coef_[0]
intercept = model.intercept_[0]

print("Model coefficients:", coefficients)
print("Model intercept:", intercept)
```

The coefficients and intercept were printed to the terminal and are as follows:

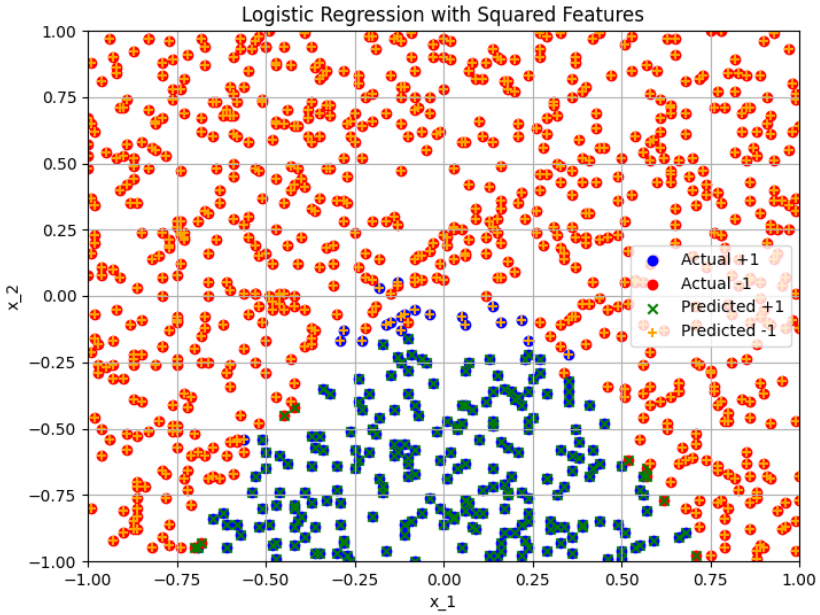
```
Model coefficients: [-4.16949497e-03 -5.33206340e+00 -8.37072162e+00 -1.41269352e-01]
Model intercept: -0.6249218989499781
```

C)ii)

The trained model was then used to predict the class of the data. This was then plotted onto the graph using the following code:

```
X_squared = np.column_stack((X, X**2))
model = LogisticRegression()
model.fit(X_squared, y)

predictions = model.predict(X_squared)
coefficients = model.coef_[0]
intercept = model.intercept_[0]

print("Model coefficients:", coefficients)
print("Model intercept:", intercept)

plt.figure(figsize=(8, 6))
plt.scatter(X[y == 1, 0], X[y == 1, 1], marker='o', color='blue', label='Actual +1')
plt.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', color='red', label='Actual -1')
plt.scatter(X[predictions == 1, 0], X[predictions == 1, 1], marker='x', color='green',
    label='Predicted +1')
plt.scatter(X[predictions == -1, 0], X[predictions == -1, 1], marker='+', color='orange',
    label='Predicted -1')

plt.xlabel('x_1')
plt.ylabel('x_2')
plt.legend()
plt.title('Logistic Regression with Squared Features')
plt.grid(True)
plt.xlim(-1, 1)
plt.ylim(-1, 1)
plt.savefig(os.path.join(fileDirectory, 'C(i).png'))
```

From this code the following graph was generated:



The prediction of this model in comparison to all of the previous models generated is much better. As mentioned with those previous models a squared feature classifier was much better suited for this data as can be seen from the graph where there is a much larger amount of data correctly predicted.

C)iii)

A baseline model was trained on the new data with the squared features to use as a comparison. It will always just predict the most common class. The following code was used to train the model:

```
baseline_model = DummyClassifier(strategy='most_frequent')
baseline_model.fit(X_squared, y)
baseline_predictions = baseline_model.predict(X_squared)
baseline_accuracy = accuracy_score(y, baseline_predictions)

logistic_accuracy = accuracy_score(y, predictions)

print("Baseline model accuracy:", baseline_accuracy)
print("Logistic regression model accuracy:", logistic_accuracy)
```

The accuracy of the baseline model and the logistic regression model were printed to the terminal and are as follows:

```
Baseline model accuracy: 0.7595190380761523
Logistic regression model accuracy: 0.9709418837675351
```

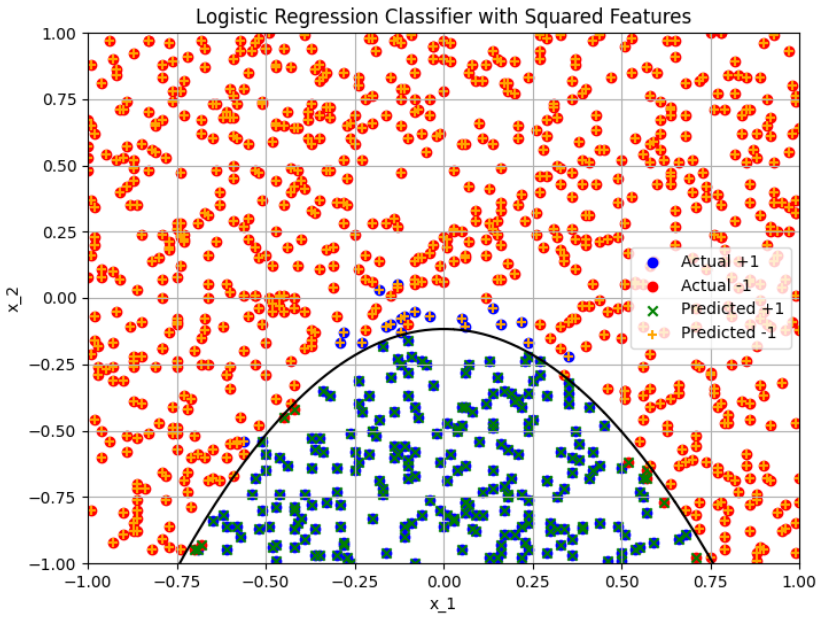
As we can see the logistic regression model is massively more accurate than the baseline model.

C)iv)

Finally a decision boundary was plotted for the logistic regression model with the squared features. This was done using the following code:

```
x1_sorted = np.sort(X[:, 0])
x2_sorted = np.sort(X[:, 1])

y_values = (-model.coef_[0][0] * x1_sorted - model.coef_[0][2] * x1_sorted**2 - model.interc

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))

Z = model.predict(np.column_stack((xx.ravel(), yy.ravel(), (xx**2).ravel(), (yy**2).ravel()))
Z = Z.reshape(xx.shape)

plt.plot(x1_sorted, y_values, 'k-')
plt.title('Logistic Regression Classifier with Squared Features')
plt.savefig(os.path.join(fileDirectory, 'C(iv).png'))
```

From this code the following graph was generated:



Appendicies

All of the code has already been included in the report in the relevant sections but the full code can be found below:

logistic_regression.py

```
import numpy as np
import pandas as pd
import os
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

fileDirectory = os.path.dirname(os.path.abspath(__file__))
## Assumes only one csv
csv_file = [f for f in os.listdir(fileDirectory) if f.endswith('.csv')][0]
csv_path = os.path.join(fileDirectory, csv_file)
df = pd.read_csv(csv_path, skiprows=1)

# Extract features and target
X1 = df.iloc[:, 0]
```

```
X2 = df.iloc[:, 1]
X = np.column_stack((X1, X2))
y = df.iloc[:, 2]

model = LogisticRegression()
model.fit(X, y)

coefficients = model.coef_[0]
intercept = model.intercept_[0]
print("Model coefficients:", coefficients)
print("Model intercept:", intercept)
predictions = model.predict(X)
plt.figure(figsize=(8, 6))
plt.scatter(X[y == 1, 0], X[y == 1, 1], marker='o', color='blue', label='Actual +1')
plt.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', color='red', label='Actual -1')
plt.scatter(X[predictions == 1, 0], X[predictions == 1, 1], marker='x', color='green', label
plt.scatter(X[predictions == -1, 0], X[predictions == -1, 1], marker='+', color='orange', la
x_values = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
decision_boundary = -(coefficients[0] * x_values + intercept) / coefficients[1]
plt.plot(x_values, decision_boundary, color='black', linestyle='--', label='Decision Boundar

plt.xlabel('x_1')
plt.ylabel('x_2')
plt.legend()
plt.title('Logistic Regression Classifier')
plt.grid(True)
plt.savefig(os.path.join(fileDirectory, 'A(iii).png'))

X_squared = np.column_stack((X, X**2))
model = LogisticRegression()
model.fit(X_squared, y)

predictions = model.predict(X_squared)
coefficients = model.coef_[0]
intercept = model.intercept_[0]

print("Model coefficients:", coefficients)
print("Model intercept:", intercept)

baseline_model = DummyClassifier(strategy='most_frequent')
baseline_model.fit(X_squared, y)
baseline_predictions = baseline_model.predict(X_squared)
baseline_accuracy = accuracy_score(y, baseline_predictions)

logistic_accuracy = accuracy_score(y, predictions)
```

```
print("Baseline model accuracy:", baseline_accuracy)
print("Logistic regression model accuracy:", logistic_accuracy)

plt.figure(figsize=(8, 6))
plt.scatter(X[y == 1, 0], X[y == 1, 1], marker='o', color='blue', label='Actual +1')
plt.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', color='red', label='Actual -1')
plt.scatter(X[predictions == 1, 0], X[predictions == 1, 1], marker='x', color='green', label
plt.scatter(X[predictions == -1, 0], X[predictions == -1, 1], marker='+', color='orange', la

plt.xlabel('x_1')
plt.ylabel('x_2')
plt.legend()
plt.title('Logistic Regression with Squared Features')
plt.grid(True)
plt.xlim(-1, 1)
plt.ylim(-1, 1)
plt.savefig(os.path.join(fileDirectory, 'C(i).png'))

x1_sorted = np.sort(X[:, 0])
x2_sorted = np.sort(X[:, 1])

y_values = (-model.coef_[0][0] * x1_sorted - model.coef_[0][2] * x1_sorted**2 - model.interc

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))

Z = model.predict(np.column_stack((xx.ravel(), yy.ravel(), (xx**2).ravel(), (yy**2).ravel())
Z = Z.reshape(xx.shape)

plt.plot(x1_sorted, y_values, 'k-')
plt.title('Logistic Regression Classifier with Squared Features')
plt.savefig(os.path.join(fileDirectory, 'C(iv).png'))
```

svm.py

```
import numpy as np
import pandas as pd
import os
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
import matplotlib.pyplot as plt

fileDirectory = os.path.dirname(os.path.abspath(__file__))
```

```
## Assumes only one csv
csv_file = [f for f in os.listdir(fileDirectory) if f.endswith('.csv')][0]
csv_path = os.path.join(fileDirectory, csv_file)
df = pd.read_csv(csv_path, skiprows=1)
X1 = df.iloc[:, 0]
X2 = df.iloc[:, 1]
X = np.column_stack((X1, X2))
y = df.iloc[:, 2]

plt.figure(figsize=(8, 6))
plt.scatter(X[y == 1, 0], X[y == 1, 1], marker='+', color='blue', label='+1')
plt.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', color='red', label='-1')

plt.xlabel('x_1')
plt.ylabel('x_2')
plt.legend()
plt.title('Data Points Graph')
plt.grid(True)
plt.savefig(os.path.join(fileDirectory, 'A(i).png'))

C_values = [0.01, 1, 100, 1000]
models = []
for C in C_values:
    model = make_pipeline(StandardScaler(), LinearSVC(C=C, max_iter=1000000))
    model.fit(X, y)
    models.append((C, model))

for C, model in models:
    print(f"Model trained with C={C}")

x_min, x_max = -1, 1
y_min, y_max = -1, 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))

fig, axes = plt.subplots(1, len(C_values), figsize=(20, 5))

for ax, (C, model) in zip(axes, models):
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, alpha=0.3)
    ax.scatter(X[y == 1, 0], X[y == 1, 1], marker='o', color='blue', label='Actual +1')
    ax.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', color='red', label='Actual -1')

    predictions = model.predict(X)
```

```
        ax.scatter(X[predictions == 1, 0], X[predictions == 1, 1], marker='x', color='green', la
        ax.scatter(X[predictions == -1, 0], X[predictions == -1, 1], marker='+', color='orange',

        ax.set_title(f'Decision Boundary for svm of C={C}')
        ax.set_xlabel('x_1')
        ax.set_ylabel('x_2')
        ax.legend()
        ax.grid(True)

    plt.tight_layout()
    plt.savefig(os.path.join(fileDirectory, 'B(ii).png'))
```