1. Week 2 Assignment

```
# id:22-22-22
```

(a) i. Figure 1 shows the training data by placing a marker on a 2D plot for each pair of feature values $(x_1, x_2)$. The x-axis is the value of the first feature $x_1$ and the y-axis is the value of the second feature $x_2$. If the target value $y$ is $+1$, then marker is a green cross, and if $y$ is $-1$, then the marker is a red circle. Seen in Listing 1, the data is read in using `pandas`, and plotted using `matplotlib`.
The decision boundary becomes obvious due to the difference in shape and colour of the markers. The line of the decision boundary looks like it follows a Gaussian distribution.

ii. Using `LogisticRegression()` from $\texttt{sklearn.linear}_{m}odel, thedataisusedtotrainalogisticreg$

```
intercept : [2.12576279], coefficients : [[0.05596919 5.84058369]]
```

The model intercept is given as $b = 2.12576279$ and the model coefficients are given as $\theta_1 = 0.05596919, \theta_2 = 5.84058369$. With two features, we get the following model:

$$\hat{y} = \text{sign}(b + \theta_1 x_1 + \theta_2 x_2)$$

Given that $(\theta_1 > 0) \wedge (\theta_2 > 0)$, both $x_1$ and $x_2$ will cause $\hat{y}$ to increase. $|\theta_1| < |\theta_2|$, therefore $x_2$ has more influence on $\hat{y}$ than $x_1$.

iii. In Listing 3, the trained logistic regression classifier is used to predict the target values of the training data using the `predict()` method. In Figure 2 we plot the $(x_1, x_2)$ data points again, but this time, the colour and shape of the markers depends on the predicted target value $\hat{y}$.
The decision boundary is where the argument of the sign function is 0.

$$b + \theta_1 x_1 + \theta_2 x_2 = 0$$

Solving for $x_2$,

$$x_2 = -\frac{b + \theta_1 x_1}{\theta_2}$$

This equation is used to plot the boundary line. See Listing 3.

iv. It is apparent that the linear decision boundary shown on the graph is not appropriate for the training data. As seen in part (i), the actual target values seem to follow a Gaussian distribution, and the data is not linearly separable. However, it is appropriate for the predicted target values as they follow a linear distribution.

(b) i. Using `LinearSVC` from `sklearn.svm`, linear SVM classifiers are fit to the training data for a wide range of values of the penalty parameter $C$. In this case, $C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]$. Where $b$ is the slope, and $\theta_0$ and $\theta_1$ are the model coefficients, the parameter values of each trained model are given in Figure 4. With two features we get the following model:

$$\hat{y} = \text{sign}(b + \theta_1 x_1 + \theta_2 x_2)$$

The code for this part is so similar to the model fitting in part (a), that I omitted the code from the report.

ii. For each penalty parameter $C$, the models are used to predict the target values of the training data. We plot the $(x_1, x_2)$ data points again, but this time, the colour and shape of the markers depends on the predicted target value $\hat{y}$. See attached "Linear SVM Models" page.
The decision boundary is where the argument of the sign function is 0, and is calculated in the same way as in part (a).

$$b + \theta_1 x_1 + \theta_2 x_2 = 0$$

Solving for $x_2$,

$$x_2 = -\frac{b + \theta_1 x_1}{\theta_2}$$

iii. The parameter $C$ in a linear SVM controls the trade-off between margin size and classification accuracy by determining the strength of $L_2$ regularization (with $1/C$ as the penalty weight). When $C$ is small, regularization is strong, forcing the model parameters to remain small and resulting in a wider margin that may misclassify some training points. This produces a smoother, more generalized decision boundary but can lead to underfitting. As $C$ increases, regularization weakens, allowing the model coefficients to grow in magnitude as the classifier attempts to fit the training data more precisely. Beyond a certain value, the parameters tend to stabilize, indicating that the model has reached its optimal separating hyperplane. Smaller $C$ values prioritize simplicity and generalization, while larger values improve training accuracy at the potential cost of overfitting.

iv. Both models identified $x_1$ as the dominant feature, with much smaller influence from $x_0$. The logistic regression's lack of penalty resulted in larger coefficients due to lack of regularisation. Despite these scale differences, both models learned boundaries oriented in the same direction, suggesting consistency between the two approaches.

(c) i. Two additional features, $x_2$ and $x_3$, are added to the data. Each represents the squared value of it's corresponding feature value. See Listing **??**.

$$x_3 = x_1^2$$

$$x_4 = x_2^2$$

The data including the squared features is now used to train a logistic regression classifier. See Listing 2.

```
intercept: [0.14482452]
coefficients: [[ 0.22474306 21.05084792 21.83333261 -0.48566866]]
```

With four features, we get the following model:

$$\hat{y} = \text{sign}(b + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4)$$

ii. Logistic regression is always linear in parameters, but adding nonlinear transformations of features allows it to produce non-linear decision boundaries. This allows for the perceived curved decision boundary that we can see in Figure **??**. As was mentioned previously in part (a), the perceived decision boundary of the training data resembles a Gaussian distribution. The quadratic nature of the prediction function given by the logistic regression classifier more accurately predicts the target value of the training data than the linear models we tried in part (a) and part (b).

iii. Using `DummyClassfier()` from `sklearn.dummy`, we can fit a baseline model using the data. We then this baseline model to predict $\hat{y}$ given $x$. In the case of the `DummyClassifier()` method, it returns the most frequent the most frequent class label in $y$, which in this case is 1. In order to compare the two models, we can use `accuracy_score()` from `sklearn.metrics`. This method takes the target values and predicted values as input, and returns the fraction of correctly classified samples. See Listing **??**.

```
baseline accuracy score: 0.6856856856856857
logistic regression classifier accuracy score: 0.965965965965966
```

This clearly shows that the trained logistic regression classifier does a much better job than the baseline at accurately predicting target values.

iv. The decision boundary is where the argument of the sign function is 0.

$$b + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 = 0$$

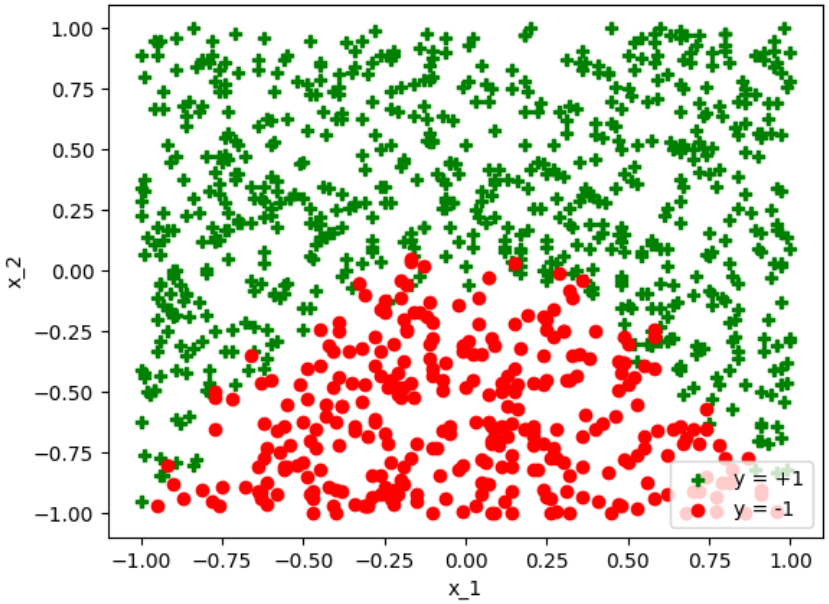



Figure 1: Training data

We can rewrite the function like this if we remember that $x_3$ and $x_4$ are the squared value of $x_1$ and $x_2$.

$$b + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 = 0$$

Plugging in the parameter values of the model that we got in part (i),

$$0.14482452 + 0.22474306 x_1 + 21.05084792 x_2 + 21.83333261 x_1^2 - 0.48566866 x_4^2 = 0$$

Since the coefficients of $x_1^2 =$ and $x_2^2$ have opposite signs, this equation represents a hyperbola.

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

df = pd.read_csv("week2.csv",header=0,names ['X1','X2','Y'])

plus = df[df['Y']>0]
minus = df[df['Y']<0]

plt.figure()

plt.scatter(plus['X1'],plus['X2'],marker="P",color="green",label="y = +1")
plt.scatter(minus['X1'],minus['X2'],marker="o",color="red",label="y = -1")

plt.xlabel('x_1')
```

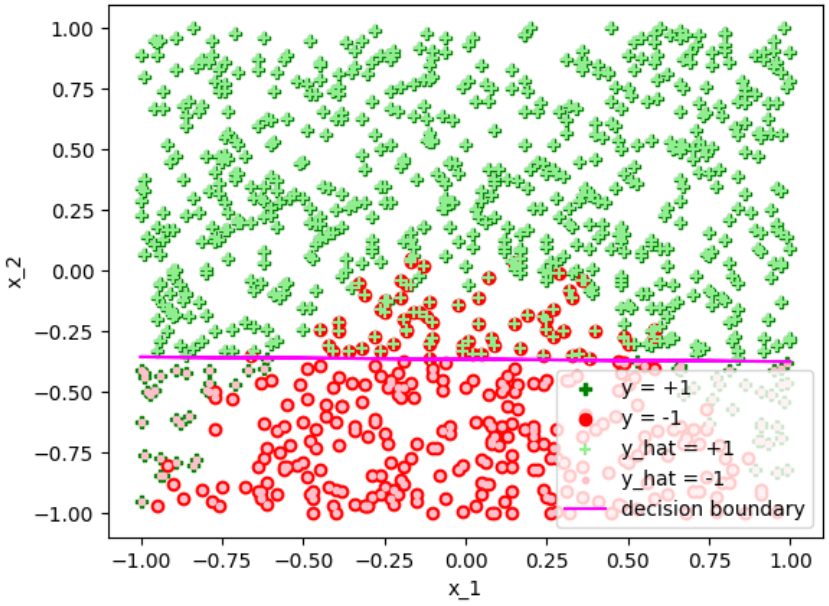



Figure 2: Training data and predicted data

```
plt.ylabel('x_2')
plt.legend()
```

Listing 1: Reading in data file and generating scatterplot

```
from sklearn.linear_model import LogisticRegression

x = df[['X1','X2']]
y = df['Y']

model = LogisticRegression(penalty=None)
model.fit(x,y)

print(f"intercept : {model.intercept_}  slope : {model.coef_}")
```

Listing 2: Logistic Regression Classifier

```
x1 = df['X1']

y_hat = model.predict(x)

predictions = x.copy()
predictions.insert(2, 'Y_hat', y_hat)

plus = predictions[predictions['Y_hat']>0]
minus = predictions[predictions['Y_hat']<0]

plt.scatter(plus['X1'],plus['X2'],marker='+',color='lightgreen',label="
    y_hat = +1")
```

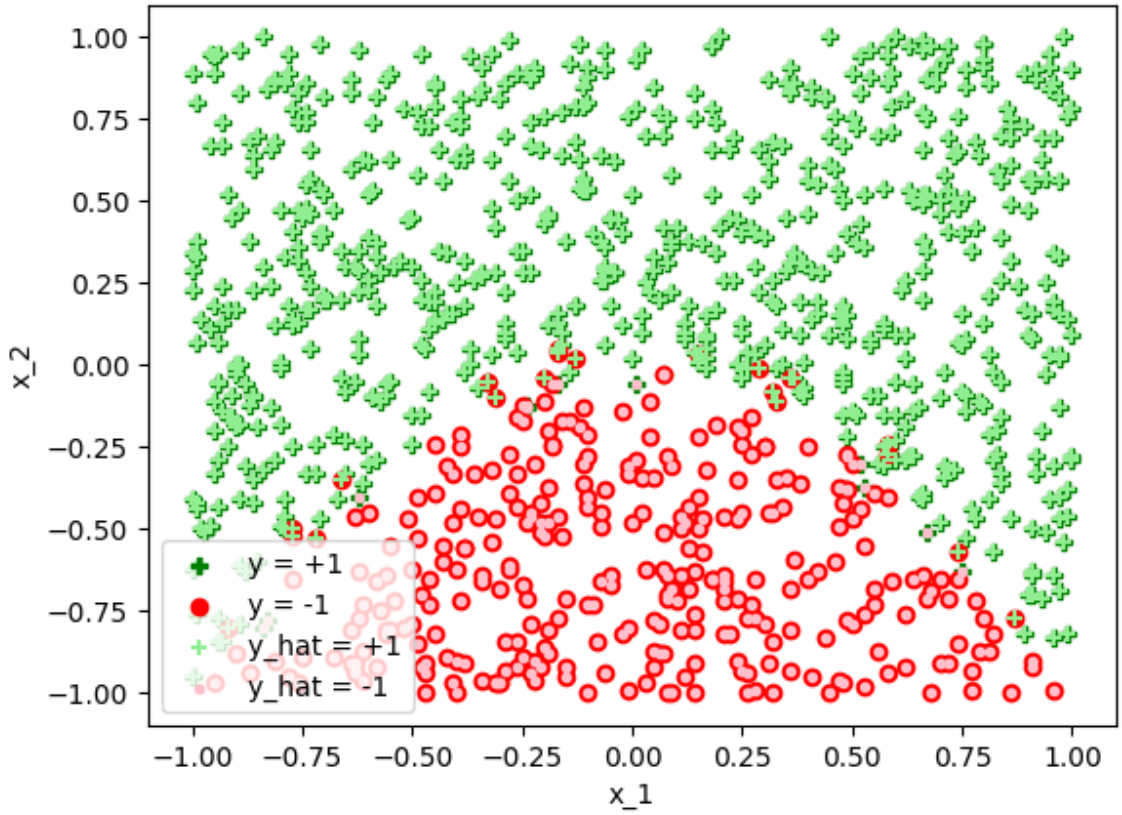



Figure 3: Training data

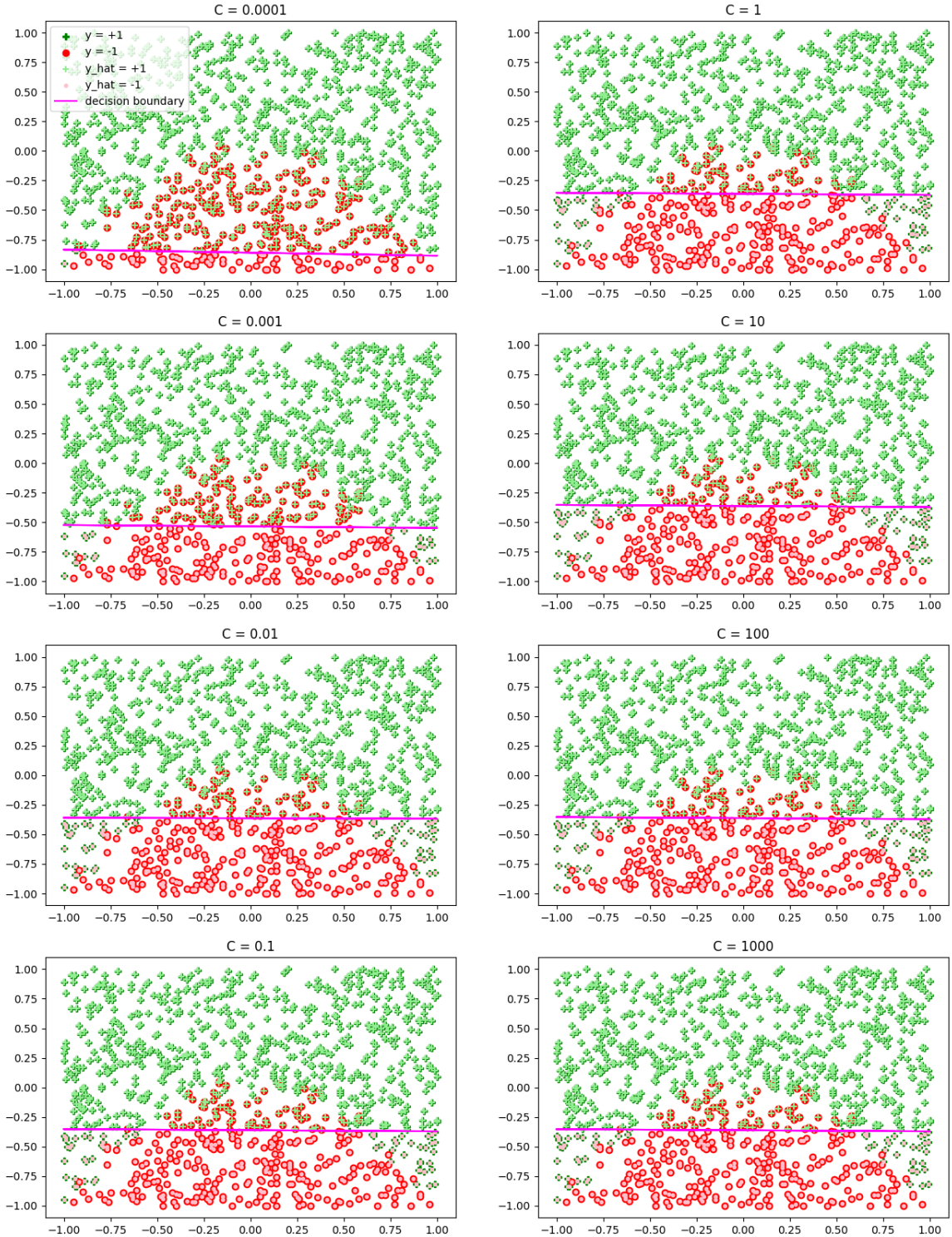
```
plt.scatter(minus['X1'],minus['X2'],marker='.',color='pink',label="y_hat =
    -1")

x2 = -( model.intercept_ + model.coef_[0][0] * x1) / model.coef_[0][1]

plt.plot(x1, x2, color='magenta', label="decision boundary")
```

Listing 3: Predicting target values using Logistic Regression classifier

# Linear SVM Models



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

# (a)
# (i)

df = pd.read_csv("week2.csv", header=0, names=['X1','X2','Y'])

plus = df[df['Y']>0]
minus = df[df['Y']<0]

plt.figure()

plt.scatter(plus['X1'],plus['X2'],marker='P',color='green',label="
y = +1")
plt.scatter(minus['X1'],minus['X2'],marker='o',color='red',label="
y = -1")


plt.xlabel('x_1')
plt.ylabel('x_2')
plt.legend()

plt.savefig('fig1')

# (ii)

from sklearn.linear_model import LogisticRegression

x = df[['X1','X2']]
y = df['Y']

model = LogisticRegression(penalty=None)
model.fit(x,y)

print(f"intercept : {model.intercept_}, coefficients :
{model.coef_}")

# (iii)

plt.figure()
plt.scatter(plus['X1'],plus['X2'],marker='P',color='green',label="
y = +1")
plt.scatter(minus['X1'],minus['X2'],marker='o',color='red',label="
y = -1")

x1 = df['X1']

y_hat = model.predict(x)

predictions = x.copy()
predictions.insert(2, 'Y_hat', y_hat)


plus = predictions[predictions['Y_hat']>0]
minus = predictions[predictions['Y_hat']<0]
```

```
plt.scatter(plus['X1'],plus['X2'],marker='+',color='lightgreen',la
bel="y_hat = +1")
plt.scatter(minus['X1'],minus['X2'],marker='.',color='pink',label=
"y_hat = -1")

x2 = -( model.intercept_ + model.coef_[0][0] * x1) /
model.coef_[0][1]

plt.plot(x1, x2, color='magenta', label="decision boundary")



plt.xlabel('x_1')
plt.ylabel('x_2')
plt.legend()

plt.savefig('fig2')

# (b)
# (i)

from sklearn.svm import LinearSVC


c_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

row = 0
col = 0

fig = plt.figure(figsize=(15, 20))

axs = fig.subplots(4, 2)

for c in c_values:
    model = LinearSVC(C=c, max_iter=10000)
    model.fit(x, y)
    print(f"C = {c}: intercept : {model.intercept_[0]},
coefficients : {model.coef_[0]}")

    plus = df[df['Y']>0]
    minus = df[df['Y']<0]

    y_hat = model.predict(x)
    predictions = x.copy()

    predictions.insert(2, 'Y_hat', y_hat)

    axs[row,col].set_title(f"C = {c}")


axs[row,col].scatter(plus['X1'],plus['X2'],marker='P',color='green
',label="y = +1")

axs[row,col].scatter(minus['X1'],minus['X2'],marker='o',color='red
',label="y = -1")

    plus = predictions[predictions['Y_hat']>0]
    minus = predictions[predictions['Y_hat']<0]
```

```
axs[row,col].scatter(plus['X1'],plus['X2'],marker='+',color='light
green',label="y_hat = +1")

axs[row,col].scatter(minus['X1'],minus['X2'],marker='.',color='pin
k',label="y_hat = -1")

    x2 = -( model.intercept_ + model.coef_[0][0] * x1) /
model.coef_[0][1]

    axs[row,col].plot(x1, x2, color='magenta', label="decision
boundary")

    if(row==3 and col<=1) :
        col=1
        row=0

    elif(row<3): row+=1

axs[0,0].legend()



plt.savefig('fig3')

# (c)
# (i) (ii)
df.insert(2, 'X3', np.square(df['X1']))
df.insert(3, 'X4', np.square(df['X2']))

plus = df[df['Y']>0]
minus = df[df['Y']<0]

x = df[['X1','X2','X3','X4']]
y = df['Y']

model = LogisticRegression(penalty=None)
model.fit(x,y)

print(f"intercept : {model.intercept_}, coefficients :
{model.coef_}")

plt.figure()
plt.scatter(plus['X1'],plus['X2'],marker='P',color='green',label="
y = +1")
plt.scatter(minus['X1'],minus['X2'],marker='o',color='red',label="
y = -1")


y_hat = model.predict(x)

predictions = x.copy()
predictions.insert(2, 'Y_hat', y_hat)


plus = predictions[predictions['Y_hat']>0]
minus = predictions[predictions['Y_hat']<0]
```

```
plt.scatter(plus['X1'],plus['X2'],marker='+',color='lightgreen',la
bel="y_hat = +1")
plt.scatter(minus['X1'],minus['X2'],marker='.',color='pink',label=
"y_hat = -1")

plt.xlabel('x_1')
plt.ylabel('x_2')
plt.legend()
plt.savefig('fig4')


# (iii)

from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score

baseline = DummyClassifier()

baseline.fit(x,y)
baseline_y = baseline.predict(x)


baseline_score = accuracy_score(y, baseline_y)
accuracy_score = accuracy_score(y, y_hat)

print(f"baseline accuracy score: {baseline_score}")
print(f"logistic regression classifier accuracy score:
{accuracy_score}")
```

| $C$ | $b$ | $\theta_0$ | $\theta_1$ |
|---|---|---|---|
| 0.0001 | 0.06186167452546345 | 0.00187932 | 0.07193783 |
| 0.001 | 0.2480068806259195 | 0.00574398 | 0.46274148 |
| 0.01 | 0.43112802740907974 | 0.00474443 | 1.18679822 |
| 0.1 | 0.6263243684442731 | 0.01600643 | 1.73057494 |
| 1 | 0.6807407602706828 | 0.0168163 | 1.87868764 |
| 10 | 0.6879076598469992 | 0.01698143 | 1.89842708 |
| 100 | 0.6886397992348632 | 0.01700404 | 1.90044688 |
| 1000 | 0.6887130988735091 | 0.01700631 | 1.9006491 |

Figure 4: Linear SVM model parameters given their penalty paramenter value $C$