Ritika Sharma
MSc Computer Science - Intelligent Systems
sharmari@tcd.ie
21334470

WEEK 2 ASSIGNMENT

Dataset Used - # id:18-36--18

a) i) I have chosen a scatter plot for data visualisation, as scatter plot is useful in observing and showing relationships between two features. The points in scatter plot reports the values of individual points and also helps in detecting any patterns when the plot is seen as a whole.

- The x-axis represents the first column, *X1*, of the dataset. On Y-axis, the second column *X2*. The target value, *y,* is represented as the points on the plot. (Blue "+" marker is used for *Positive* target values. Green "o" for *Negative* target values. A legend is included in the plot for reference. The data is more positive than negative.

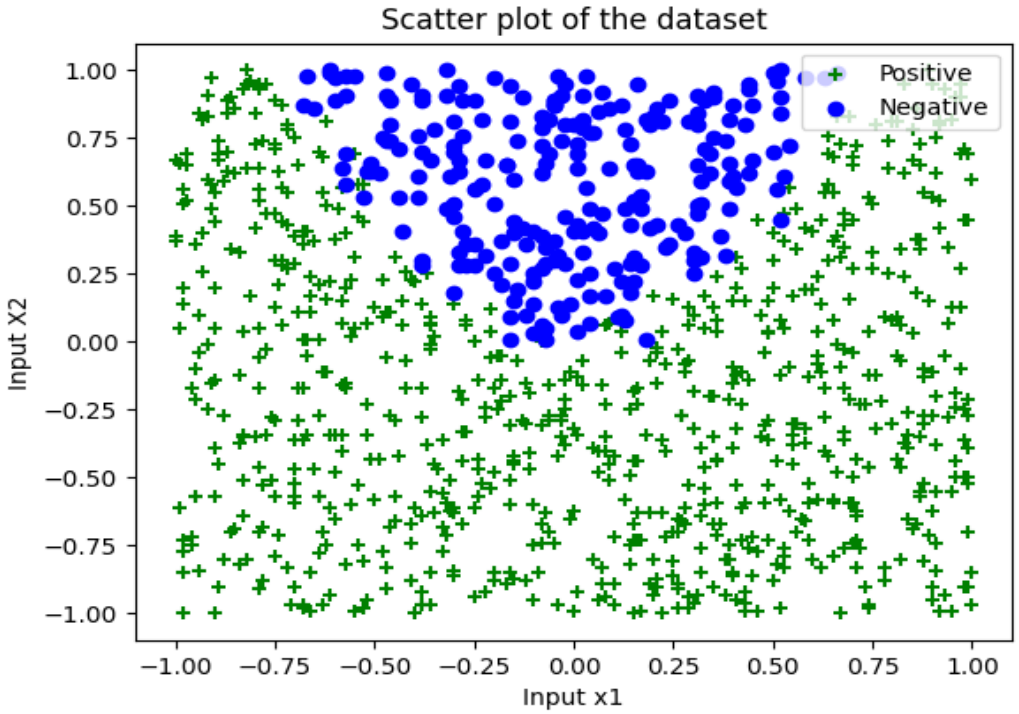


Fig 1. Scatter plot of the dataset provided

a) ii) LogisticRegression class of the library sklearn is being used here.

```
# evaluate the data
y_pred = model.predict(X)

# Retrieve parameters of the model.
i = model.intercept_[0]
c1, c2 = model.coef_.T
print(model.intercept_, model.coef_)
score_ = accuracy_score(y, y_pred)
conf_m = confusion_matrix(y, y_pred)
```

```
report = classification_report(y, y_pred)
print(score_,conf_m,report)
```

- An instance, *model,* of LogisticRegression is created to feed dataset to the model.
- X is a two dimensional input training data created by stacking both X1 and X2 using *np.column_stack()*. *y* is the training data output value. The statement *model.fit(X,y)* fits (trains) the model. Helpful in training the data and finding coefficients to attain the best value of the cost function.
- For prediction, the training data input value has been used again. The statement, *model.predit(X)* helps in predicting the output value for the test data.
- The remaining code obtains values like *accuracy_score, classification_report, confusion_matrix* and most importantly *intercept* (constant value) *and coefficient*(weights of input feature) for the cost function.
- The results are – Intercept - [2.08598017] and Coefficients - [[-0.1269703 -3.581926]]
- $y = a + c_1x_1 + c_2x_2$.
- According to the above equation, the highest absolute value will affect the decision among all the parameters. Here we see c2 = -3.581926. Since, the value of c2>c1, c2 will affect the equation more and decide if the value is positive or negative. As c2 is attached to X2, it becomes the important feature.
- The accuracy of the model is 0.81.

a) iii) The below scatter plot depicts the predictions done by the model. The plot is built on the top of the original dataset to provide a better comparison between the original values and predicted values.

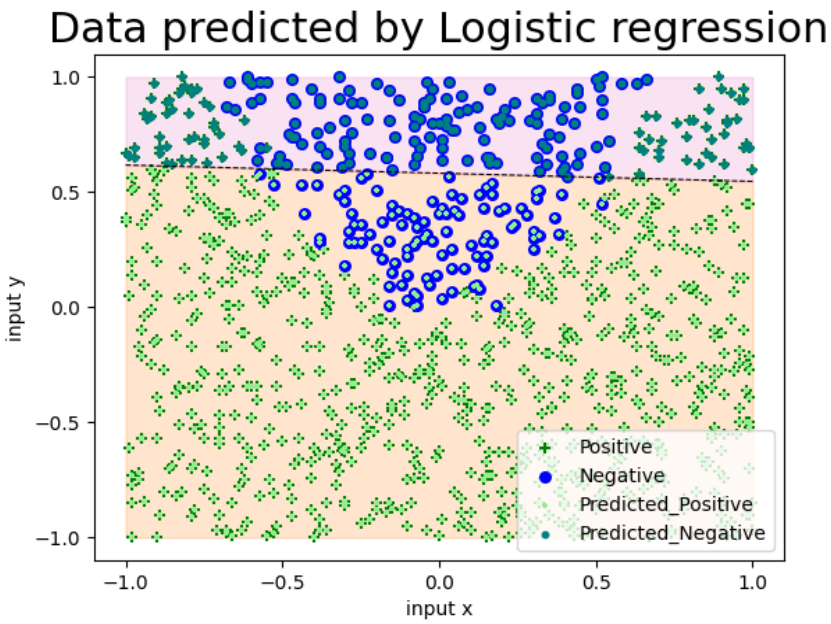


Fig 2. Scatter plot depicting predicted and original values

- Values in "green +" are original positive values, "lightgreen +" depicts predicted positives.
- Values in "blue o" are original negative values, "teal o" depicts predicted negatives.
- The decision boundary is a black dashed line separating the plot.
- To obtain the decision boundary –

This is a linear equation so the decision boundary would eb a straight line (y=mx+c). To obtain the value of slope, m and intercept, c, solve the equation for 2d data in logistic regression - θ0+θ1x1+θ2x2=0

Solving this, the value for $x2=-c1/c2*x1-i/c2$.

Substituting these equations with i, c1, c2, the results are - m = -0.03544749 and c = 0.58236272. With x values ranging from [-1,1], a decision boundary is drawn.

a) iv) The accuracy of the prediction is 0.81, so it's safe to assume that the model didn't predict all the values correctly. The reason for this is the training data has a curved boundary whereas the predicted data has a straight line which leads to the wrong prediction and underfitting of the data.

b) i) In the code below,

```
model = LinearSVC(verbose=0, C=j)
    model.fit(X, y)
    y_pred = model.predict(X)
    intercept1 = model.intercept_[0]
    c1, c2 = model.coef_.T
    results.append( {
        'C' : j,
        'conf_m' : confusion_matrix(y, y_pred),
        'score' : accuracy_score(y, y_pred),
        'intercept' : model.intercept_,
        'coeff' : model.coef_,
    } )
```

- The model is trained using a SVM Classifier algorithm. The *LinearSVC* class is used from sklearn library. It generates optimal hyperplane in a manner to reduce error.
- The model is trained using different values of Penalty parameter, C [0.001, 100] and the original training data. The intercept and coefficients are recorded.
- The SVM algorithm uses hinge loss function which minimises the error.

ii) Predict the model as shown in the above code and generate model parameters to find the accuracy score generated by different values of penalty parameter, C.

- The model is predicted using *model.predict(X)* function. And corresponding values are retrieved from the instance model like,
- Confusion matrix – defines false negatives, true negatives, false positives and true positives.
- Accuracy Score – determines the accuracy of the prediction model
- Intercept and coefficient to calculate the decision boundary as the same way done in part a)

Model parameters –

{'C': 0.001, 'conf_m': array([[0, 240], [0, 759]], dtype=int64), 'score': 0.7597597597597597, 'intercept': array([0.34632901]), 'coeff': array([[0.00882218, -0.33610823]])}

{'C': 0.01, 'conf_m': array([[109, 131], [68, 691]], dtype=int64), 'score': 0.8008008008008008, 'intercept': array([0.53027735]), 'coeff': array([[-0.0159417 , -0.80167851]])}

Ritika Sharma sharmari@tcd.ie
MSc Computer Science - Intelligent Systems 21334470

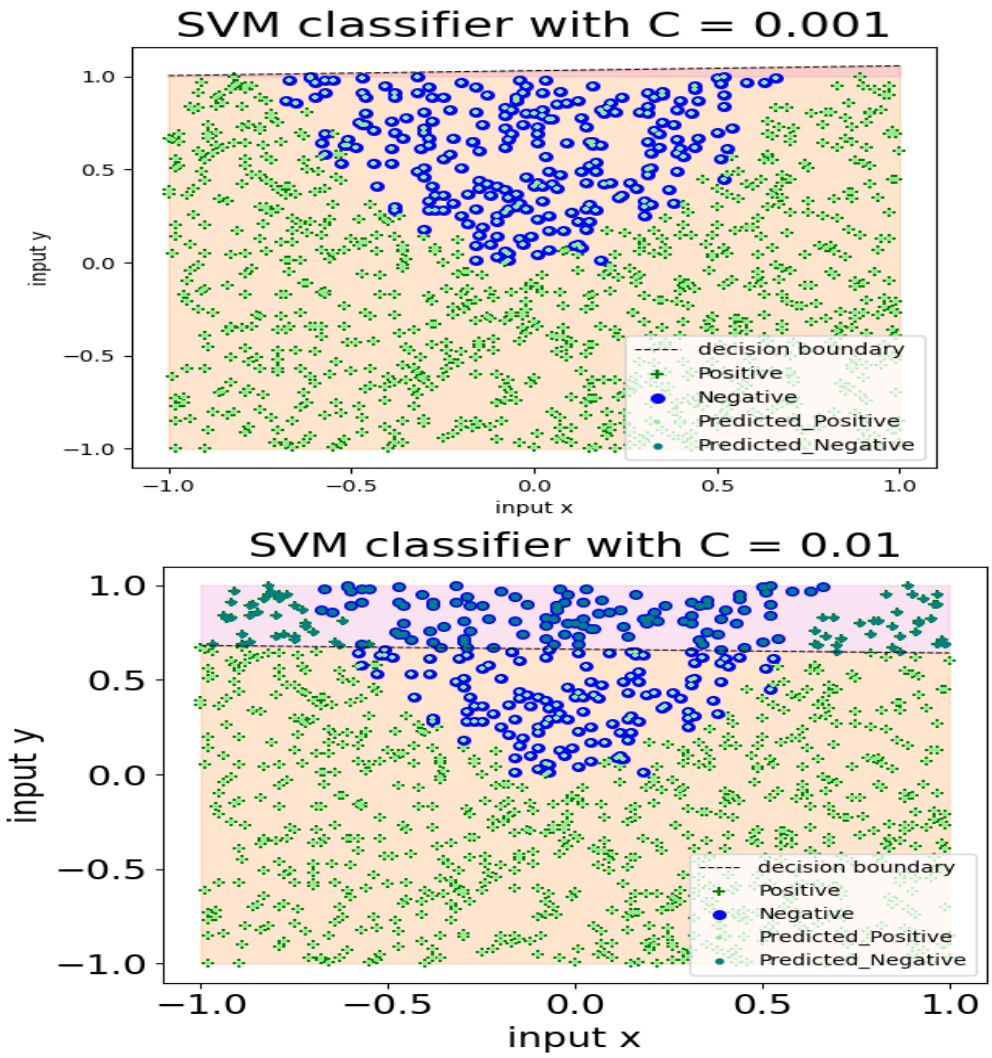
{'C': 0.1, 'conf_m': array([[136, 104], [85, 674]], dtype=int64), 'score': 0.8108108108108109, 'intercept': array([0.6626214]), 'coeff': array([[-0.04545115, -1.1382976]])}

{'C': 1.0, 'conf_m': array([[137, 103], [87, 672]], dtype=int64), 'score': 0.8098098098098098, 'intercept': array([0.69503937]), 'coeff': array([[-0.05149167, -1.21184503]])}

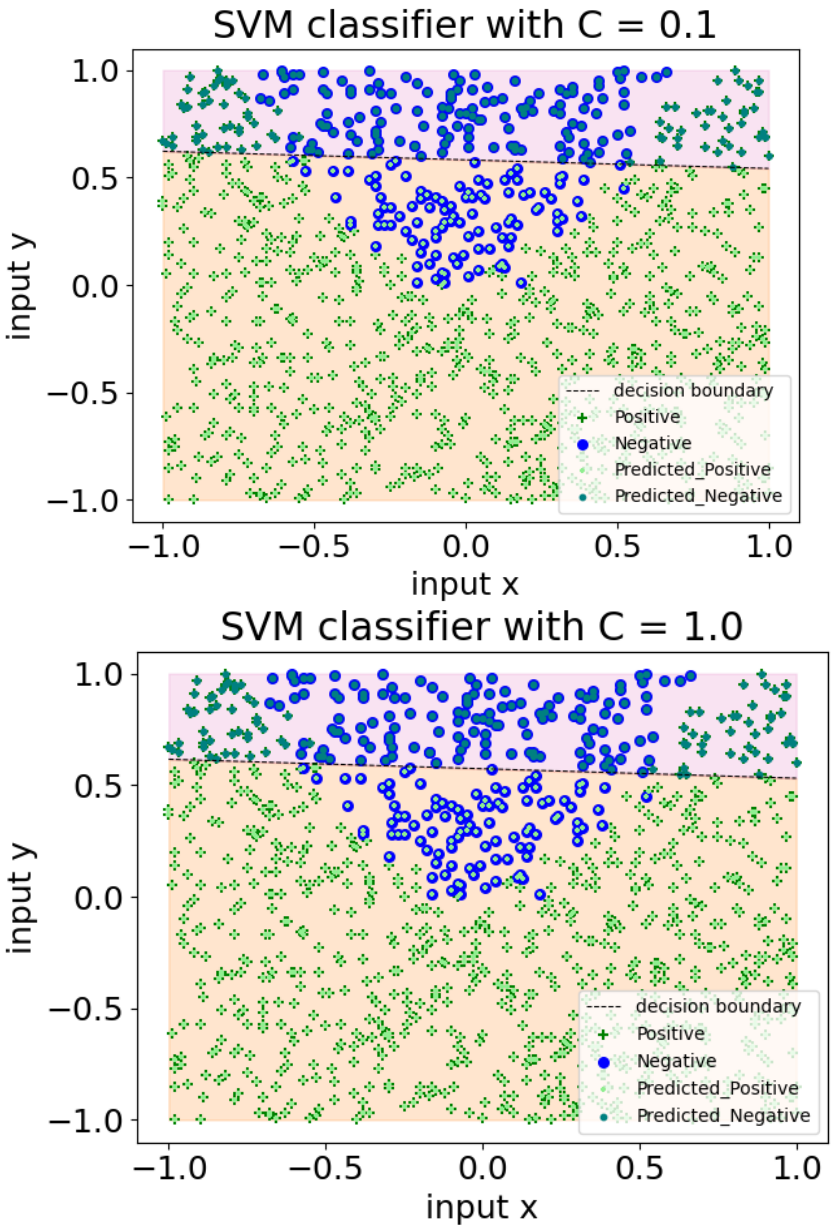
{'C': 10.0, 'conf_m': array([[137, 103], [87, 672]], dtype=int64), 'score': 0.8098098098098098, 'intercept': array([0.69880937]), 'coeff': array([[-0.05214119, -1.22025669]])}

{'C': 100.0, 'conf_m': array([[124, 116], [72, 687]], dtype=int64), 'score': 0.8118118118118118, 'intercept': array([0.75925088]), 'coeff': array([[-0.08925515, -1.22548559]])}

- Below are the graphs for some of the values of C, drawn on the original data and predicted data.



Ritika Sharma
MSc Computer Science - Intelligent Systems
sharmari@tcd.ie
21334470



Ritika Sharma
MSc Computer Science - Intelligent Systems
sharmari@tcd.ie
21334470

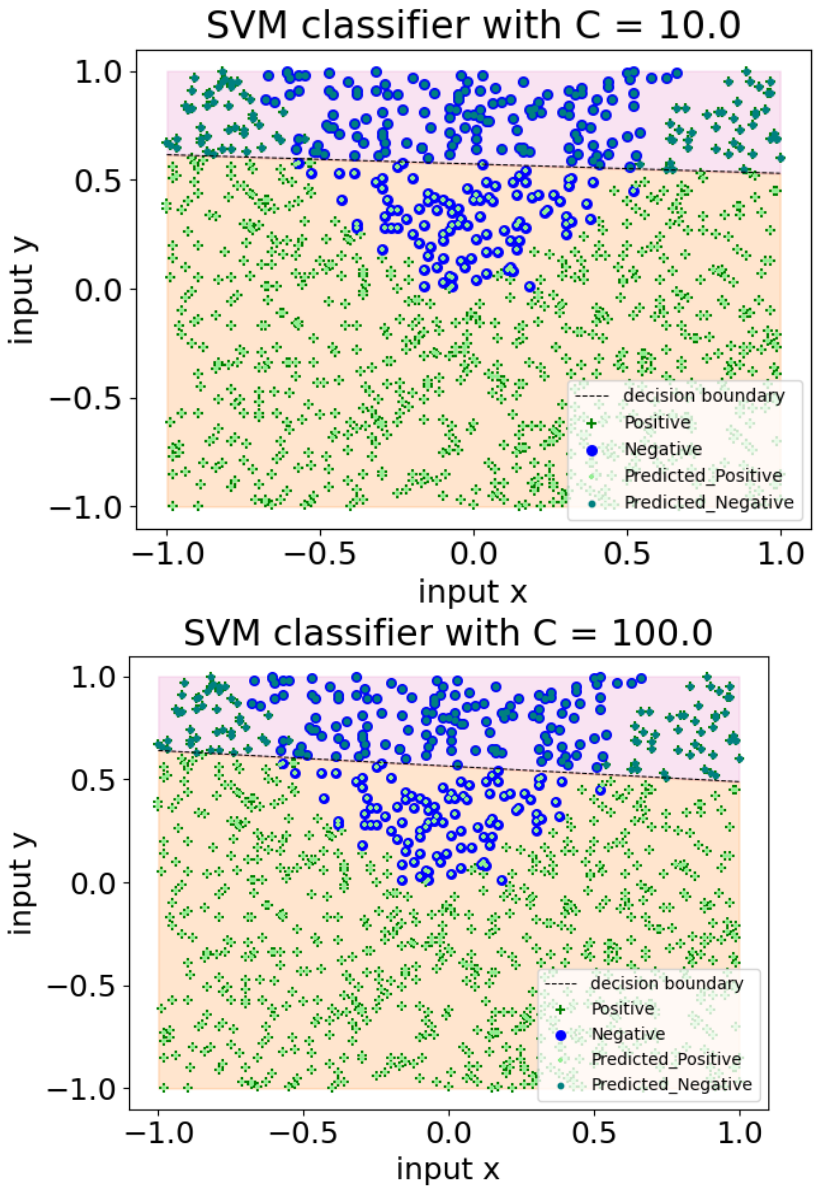


Fig 3. Scatter plots with different values of C(penalty parameter)

b) iii) The penalty parameter is used to keep a penalty on the theta values. The value of C is inversely proportional to the cost function. Thus, for larger C, cost function is minimised and the errors are reduced.

- The highest accuracy is obtained by C=100.0
- Higher values of C generate better optimised output
- The decision boundaries are not separating the positives and negatives correctly when the value of C is low.
- As can be seen in the graphs, for C 0.001, the decision boundary formed had more errors and almost all negatives were predicted as positives. Whereas, for C = 100, the accuracy was similar to the model made in part a)

b) iv) Both the models had almost similar accuracy. There was not much difference in the model parameters as well. But in SVM, we can change the value of C, thus changing theta which can bring difference in the accuracy as well. Thus, SVM has better optimised cost function. And it can reduce errors which is not the case in logistic regression.

Ritika Sharma
MSc Computer Science - Intelligent Systems
sharmari@tcd.ie
21334470

c) i) In the below code,

```
X3 = X1**2
X4 = X2**2
X_new = np.column_stack((X1, X2, X3, X4))
model = LogisticRegression(penalty='none', solver='lbfgs')
model.fit(X_new, y)

# Retrieve parameters of the model.
print(model.intercept_, model.coef_)
```

- Creates two new features (*X3, X4*) by squaring the existing features (*X1, X2)* respectively.
- To train the data, combine all these features into a new array *X_new.*
- Apply Logistic regression model to train the new data.
- Model parameters values –
  Intercept = [0.29707214] ,
  Coefficient = [[1.08234095 -27.27990671 53.35955198 0.11627308]]
  Accuracy Score - 0.9819819819819819

c) ii) Fig 3 shows the scatter plot generated after taking squares of the features. The plot contains the training data and the predicted data, differentiated by different colors and markers. Due to the introductions of squares of the features, the equation became quadratic and the data is fitting more accurately as the decision boundary is no longer a straight line, rather it's a curve this time, giving us the accuracy score of 0.98.

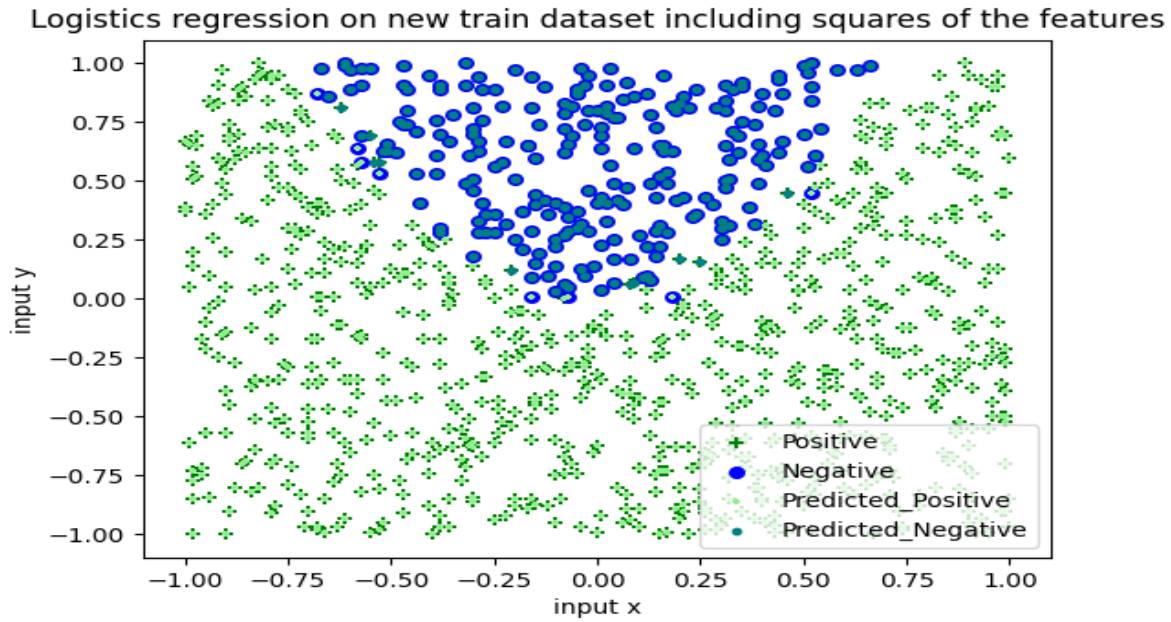


Fig 4. Scatter Plot with squared features

c) iii) A baseline predictor algorithm is the one that gives us the minimum accuracy required. A baseline predictor will serve as the threshold value and tell if the changes done are bringing any significant change or not. So, in this case, since 2/3rd dataset is positive, and assuming the baseline predictor will predict all the values as positive, we receive an accuracy score of 75% approximately.

```
# baseline predictor considering all values positive
y_base = np.empty(y.shape, dtype = int)
y_base.fill(1)
baseline = accuracy_score(y, y_base)
print(baseline)
```

APPENDIX

Part a) code –

```
# import packages

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_
score

# get data
df = pd.read_csv('week2_dataset1.csv')
print(df.head())
X1=df.iloc[ : , 0 ]
X2=df.iloc[ : , 1 ]
y=df.iloc[ : , 2]
X = np.column_stack((X1,X2))

# a) i)
# visualise the data
plt.scatter(X1[y==1], X2[y==1], c=['green'], marker='+', label = "Positive")
plt.scatter(X1[y==-1], X2[y==-1], c=['blue'], label="Negative")
plt.title('Scatter plot of the dataset')
plt.xlabel("Input x1")
plt.ylabel("Input X2")
plt.legend(loc="upper right")
plt.show()


# a) ii)
# train the data

model = LogisticRegression(penalty='none', solver='lbfgs')
model.fit(X, y)

# Retrieve parameters of the model.
i = model.intercept_[0]
c1, c2 = model.coef_.T
```

```
print(model.intercept_, model.coef_)

# evaluate the data
y_pred = model.predict(X)
score_ = accuracy_score(y, y_pred)
conf_m = confusion_matrix(y, y_pred)
report = classification_report(y, y_pred)
print(score_,conf_m,report)

# a) iii)
# plotting the prediction

# Calculate the intercept and gradient of the decision boundary.
c = -i/c2
m = -c1/c2
print("m = ",m)
print("c = ", c)
# Plot the data and the classification with the decision boundary.
x_min, x_max = -1, 1
y_min, y_max = -1, 1
x_d = np.array([x_min, x_max])
y_d = m*x_d + c
plt.plot(x_d, y_d, 'k', lw=0.75, ls='--')
plt.fill_between(x_d, y_d, y_min, color='tab:orange', alpha=0.2)
plt.fill_between(x_d, y_d, y_max, color='tab:pink', alpha=0.2)

plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True

plt.scatter(X1[y==1], X2[y==1], s=30, c=['green'], marker='+', label = "Positi
ve")
plt.scatter(X1[y==-1], X2[y==-1], s=30, c=['blue'], label="Negative")
plt.scatter(X1[y_pred==1], X2[y_pred==1], s=10, c=['lightgreen'], marker='+',
label = "Predicted_Positive")
plt.scatter(X1[y_pred==-1], X2[y_pred==-
1], s=10, c=['teal'], label="Predicted_Negative")
plt.title("Data predicted by Logistic regression")
plt.xlabel("input x"); plt.ylabel("input y")
plt.legend(loc="lower right", prop={"size":10})
plt.show()
```

Part B) code

```
# import packages

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

```
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix, accuracy_score


# get data
df = pd.read_csv('week2_dataset1.csv')
print(df.head())
X1=df.iloc[ : , 0 ]
X2=df.iloc[ : , 1 ]
y=df.iloc[ : , 2]
X = np.column_stack((X1,X2))


# b) i)
# Linear SVM

results = []
j=0.001

# train classifiers for a large set of C, penalty parameter
while (j<=100):
    model = LinearSVC(verbose=0, C=j)
    model.fit(X, y)
    y_pred = model.predict(X)
    intercept1 = model.intercept_[0]
    c1, c2 = model.coef_.T
    results.append( {
        'C' : j,
        'conf_m' : confusion_matrix(y, y_pred),
        'score' : accuracy_score(y, y_pred),
        'intercept' : model.intercept_,
        'coeff' : model.coef_,
    } )

    # find m and c for decision boundary
    intercept1 = model.intercept_[0]
    c1, c2 = model.coef_.T
    coef = -intercept1/c2
    m = -c1/c2
    x_min, x_max = -1, 1
    y_min, y_max = -1, 1
    x_d = np.array([x_min, x_max])
    y_d = m*x_d + coef


    # Plot the data and the classification with the decision boundary.

    plt.plot(x_d, y_d, 'k', lw=0.75, ls='--', label="decision boundary")
    plt.fill_between(x_d, y_d, y_min, color='tab:orange', alpha=0.2)
```

```
    plt.fill_between(x_d, y_d, y_max, color='tab:pink', alpha=0.2)

    plt.rc('font', size=18)
    plt.rcParams['figure.constrained_layout.use'] = True

    plt.scatter(X1[y==1], X2[y==1], s=30, c=['green'], marker='+', label = "Po
sitive")
    plt.scatter(X1[y==-1], X2[y==-1], s=30, c=['blue'], label="Negative")
    plt.scatter(X1[y_pred==1], X2[y_pred==1], s=10, c=['lightgreen'], marker='
+', label = "Predicted_Positive")
    plt.scatter(X1[y_pred==-1], X2[y_pred==-
1], s=10, c=['teal'], label="Predicted_Negative")
    plt.title('SVM classifier with C = '+ str(j))
    plt.xlabel("input x"); plt.ylabel("input y")
    plt.legend(loc="lower right", prop={"size":10})
    plt.show()

    j = j *10.0


for i in results:
    print(i)
```

Part c) Code –

```
# import packages

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_
score

# get data
df = pd.read_csv('week2_dataset1.csv')
print(df.head())
X1=df.iloc[ : , 0 ]
X2=df.iloc[ : , 1 ]
y=df.iloc[ : , 2]
X = np.column_stack((X1,X2))

# c) i)

# create two new features
X3 = X1**2
X4 = X2**2
X_new = np.column_stack((X1, X2, X3, X4))
```

```
model = LogisticRegression(penalty='none', solver='lbfgs')
model.fit(X_new, y)

# Retrieve parameters of the model.
print(model.intercept_, model.coef_)

# evaluate the data
# p_pred = model.predict_proba(X)
y_pred = model.predict(X_new)
score_ = accuracy_score(y, y_pred)
conf_m = confusion_matrix(y, y_pred)
report = classification_report(y, y_pred)
print(score_)

plt.scatter(X1[y==1], X2[y==1], s=30, c=['green'], marker='+', label = "Positi
ve")
plt.scatter(X1[y==-1], X2[y==-1], s=30, c=['blue'], label="Negative")
plt.scatter(X1[y_pred==1], X2[y_pred==1], s=10, c=['lightgreen'], marker='+',
label = "Predicted_Positive")
plt.scatter(X1[y_pred==-1], X2[y_pred==-
1], s=10, c=['teal'], label="Predicted_Negative")
plt.title("Logistics regression on new train dataset including squares of the
features")
plt.xlabel("input x"); plt.ylabel("input y")
plt.legend(loc="lower right", prop={"size":10})

plt.show()

# baseline predictor considering all values positive
y_base = np.empty(y.shape, dtype = int)
y_base.fill(1)
baseline = accuracy_score(y, y_base)
print(baseline)
```