# Assignment-1, Week 2

Pallavit Aggarwal, Ms-IS
ID: 22333721

## Question 1

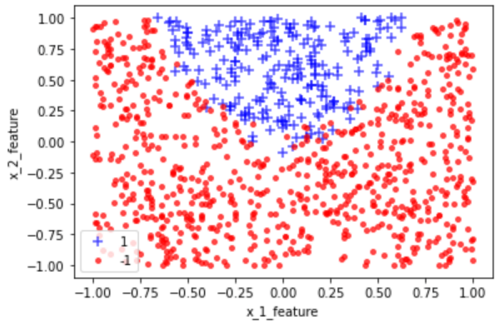
i) The data generated for me corresponds to id -> # id:5--10-5. This is saved in a file named "week2.csv" and has 1000 rows outputted as follows

```
df = pd.read_csv('week2.csv')
print(df)
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X=np.column_stack((X1,X2))
y=df.iloc[:,2]
```

```
        x1    x2  y
0    -0.17 -0.37 -1
1     0.64 -0.71 -1
2     0.95  0.91 -1
3     0.05  0.51  1
4    -0.47  0.40 -1
..     ...   ... ..
994   0.57  1.00  1
995   0.70 -0.86 -1
996  -0.91  0.62 -1
997  -0.78  0.47 -1
998  -0.13 -0.28 -1

[999 rows x 3 columns]
```

This data is visualized in a scatter plot given below with the x axis representing the feature 1, y axis feature 2 and the legends shows that + marker is represented by a blue color and denotes value 1, and o is red and denotes values -1.

```
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
plt.scatter(X1[y==1], X2[y==1], c='blue', marker= '+', s=50, alpha=0.7, label='1')
plt.scatter(X1[y==-1], X2[y==-1],c='red', marker='o',s=15, alpha=0.7, label='-1')
plt.legend()
plt.show()
```




ii)

Now we train a logistic regression classifier on the data using sklearn. First we split the data, in a non-random fashion with 'random_state' as an integer in 'train_test_split' function parameter, in a 80-20 ration.

We use Logistic Regression function and the parameter solver is assigned to 'liblinear' as it is a good choice for smaller datasets and one-versus-rest scheme and supports both l1 and l2 regularizations. Other possible solvers are 'lbfgs', 'sag, 'newton-cg'.

```
In [145]: from sklearn.linear_model import LogisticRegression
          from sklearn.model_selection import train_test_split
          from sklearn import metrics
          X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.20,random_state = 0)

          model = LogisticRegression(solver='liblinear', penalty='l2', C=1.0)
          model.fit(X_train, Y_train)

Out[145]: LogisticRegression(C=100.0, solver='liblinear')
```

Next our penalty parameter decides whether there is regularization and which approach suits us. Regularization normally tries to reduce or penalize the complexity of the model. We use l2.

Now, C value which is the hyperparameter is basically a high value of C tells the model to give high weight to the training data, and a lower weight to the complexity penalty. A low value tells the model to give more weight to this complexity penalty at the expense of fitting to the training data. In other terms, a high C means "Trust this training data a lot", while a low value says "This data may not be fully representative of the real-world data, so if it's telling you to make a parameter really large, don't listen to it". [1]

We proceed report the following result for the intercept and weight of the model, and also accuracy of the model along with confusion matrix.

```
print('intercept \n',(model.intercept_))
print('slope \n',( model.coef_[0]))

print('Model Score \n', model.score(X_train, Y_train))

Y_predict=model.predict(X_test)

cnf_matrix = metrics.confusion_matrix(Y_test, Y_predict)
print('Confusion Matrix \n', cnf_matrix)
```

```
intercept
 [-1.9423667]
slope
 [0.00532466 3.12375707]
Model Score
 0.8012519561815337
Confusion Matrix
 [[111  16]
 [ 19  14]]
```

We try to minimize the false positives and false negatives which are the opposite diagonal and at C=1,
The above values show the intercept, the slope or weight parameters Theta1 and Theta2. The model score is roughly 80% and the FP=16 and FN=19.

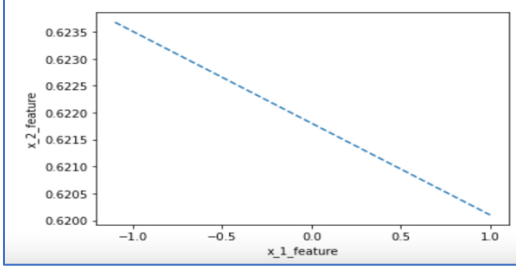
iii) We plot the line obtained. The line has the equation

$$m1*x1 + m2*x2 + c = 0;$$

Which gives us the x2 = -(c + m1x1)/m2.

```
w1, w2 = model.coef_[0]
c = model.intercept_[0]
m1 = w1
m2 = w2
xlist = np.arange(-1.1,1.1,0.1)

def get_line_y(x):
    return (-c-m1*x)/m2

plt.plot(xlist,get_line_y(xlist),'--')
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
```

```
Text(0, 0.5, 'x_2_feature')
```

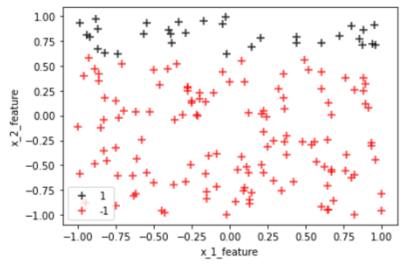



The weight from the coefficient matrix [0.00532466 3.12375707] is used as m1 and m2 denotes weights w1 and w2 respectively.
With m1=0. 00532466 and m2=3.12375707, we calculate x2 = -(c + m1x1)/m2 for every point x from the xlist which is a list of x values from -1 to +1 in steps of 0.1. It starts from -1.1 to factor in -1 as well.

The line plotted is shown in the above figure.

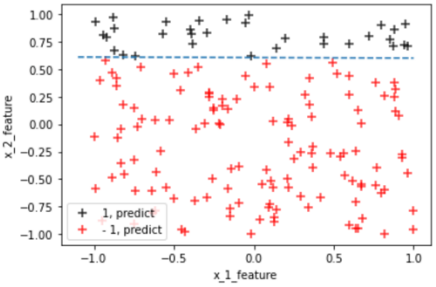
Now, we plot the test data scatter plot.

```
X1test = X_test[:,0]
X2test = X_test[:,1]
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
plt.scatter(X1test[Y_predict==1],X2test[Y_predict==1], c='black', marker= '+', s=50, alpha=0.8, label='1')
plt.scatter(X1test[Y_predict==-1],X2test[Y_predict==-1], c='red', marker='+',s=50, alpha=0.8, label='-1')
plt.legend()
plt.show()
```

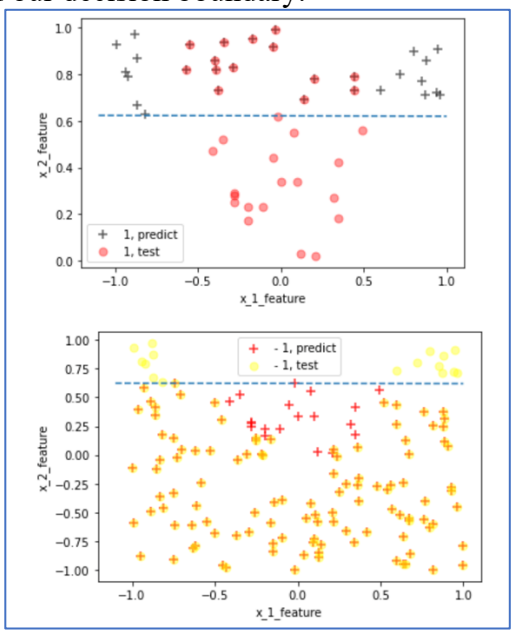



And the super-imposition of the decision boundary on the scatter plot.

```
X1test = X_test[:,0]
X2test = X_test[:,1]
plt.plot(xlist,get_line_y(xlist),'--')
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
plt.scatter(X1test[Y_predict==1],X2test[Y_predict==1], c='black', marker= '+', s=50, alpha=0.8, label='1, predict')
plt.scatter(X1test[Y_predict==-1],X2test[Y_predict==-1], c='red', marker='+',s=50, alpha=0.8, label='- 1, predict')
plt.legend()
plt.show()
```

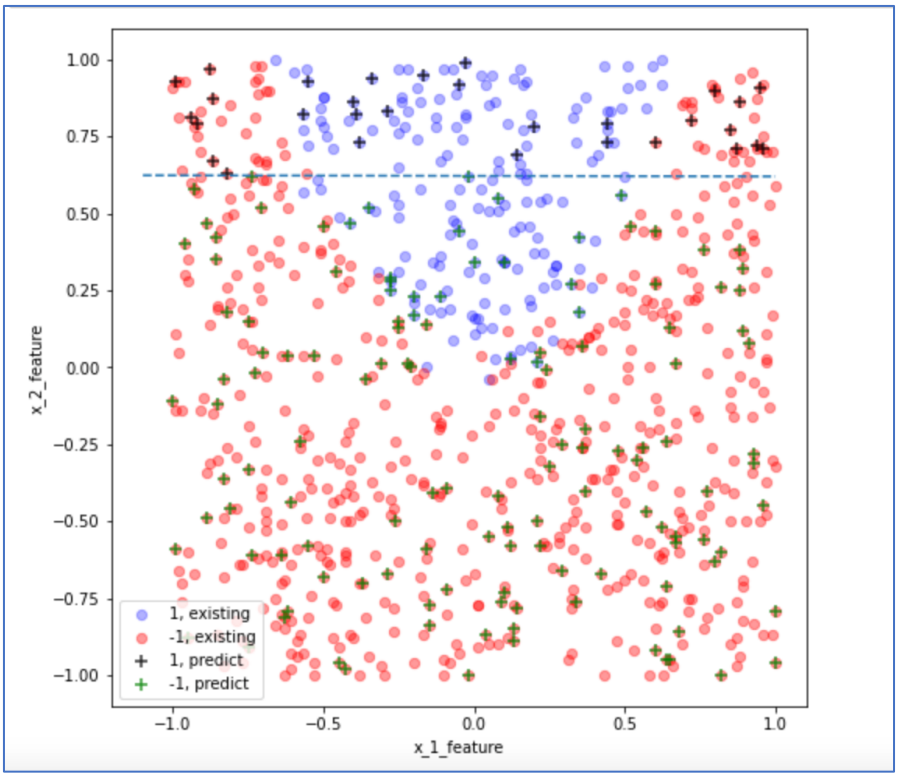



The predict vs test scatter for +1 values are show in the image below in the top plot and the predict vs test scatter for -1 is shows in the bottom one. This basically visualizes the scenarios for FP and FN cases and the accuracy of our decision boundary.




iv) The super imposition of the original scatter maker existing and in blue and red alongside predicted data marked and labeled as +1, -1 predict show the predicted values in black and green +.

The decision boundary is able to correctly segregate the predicted values with some false scenarios




The ‘o’ represent the original scatter for -1 and +1 values and the + represents the predicted values from the test data.

```
plt.subplots(figsize=(8, 8))
X1test = X_test[:,0]
X2test = X_test[:,1]
plt.plot(xlist,get_line_y(xlist),'--')
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
plt.scatter(X1[y==1], X2[y==1], c='blue', marker= 'o', s=40, alpha=0.3, label='1, existing')
plt.scatter(X1[y==-1], X2[y==-1],c='red', marker='o',s=40, alpha=0.4, label='-1, existing')
plt.scatter(X1test[Y_predict==1],X2test[Y_predict==1], c='black', marker= '+', s=50, alpha=0.8, label='1, predict')
plt.scatter(X1test[Y_predict==-1],X2test[Y_predict==-1], c='green', marker='+',s=50, alpha=0.8, label='-1, predict')
plt.legend()
plt.show()
```

The model is fairly accurate, but it’s accuracy can still be improved, by adding additional features.

## Question 2

In this, the LinearSVC is initialized with C=0.01, C=0.1, C=0.5, C=1 and C=100

*Dual* is set to *false* to avoid the *‘Convergence Warning’*

```
['SVM', 'C-value: 0.0001', 'Train accuracy: 0.769', 'Test accuracy: 0.771']
['SVM', 'C-value: 0.001', 'Train accuracy: 0.769', 'Test accuracy: 0.771']
['SVM', 'C-value: 0.01', 'Train accuracy: 0.796', 'Test accuracy: 0.800']
['SVM', 'C-value: 0.1', 'Train accuracy: 0.803', 'Test accuracy: 0.779']
['SVM', 'C-value: 1', 'Train accuracy: 0.801', 'Test accuracy: 0.771']
['SVM', 'C-value: 10', 'Train accuracy: 0.801', 'Test accuracy: 0.771']
['SVM', 'C-value: 100', 'Train accuracy: 0.801', 'Test accuracy: 0.771']
['SVM', 'C-value: 1000', 'Train accuracy: 0.801', 'Test accuracy: 0.771']
['SVM', 'C-value: 10000', 'Train accuracy: 0.801', 'Test accuracy: 0.771']
['SVM', 'C-value: 100000', 'Train accuracy: 0.801', 'Test accuracy: 0.771']
```

The confusion matrix for a few of these are:

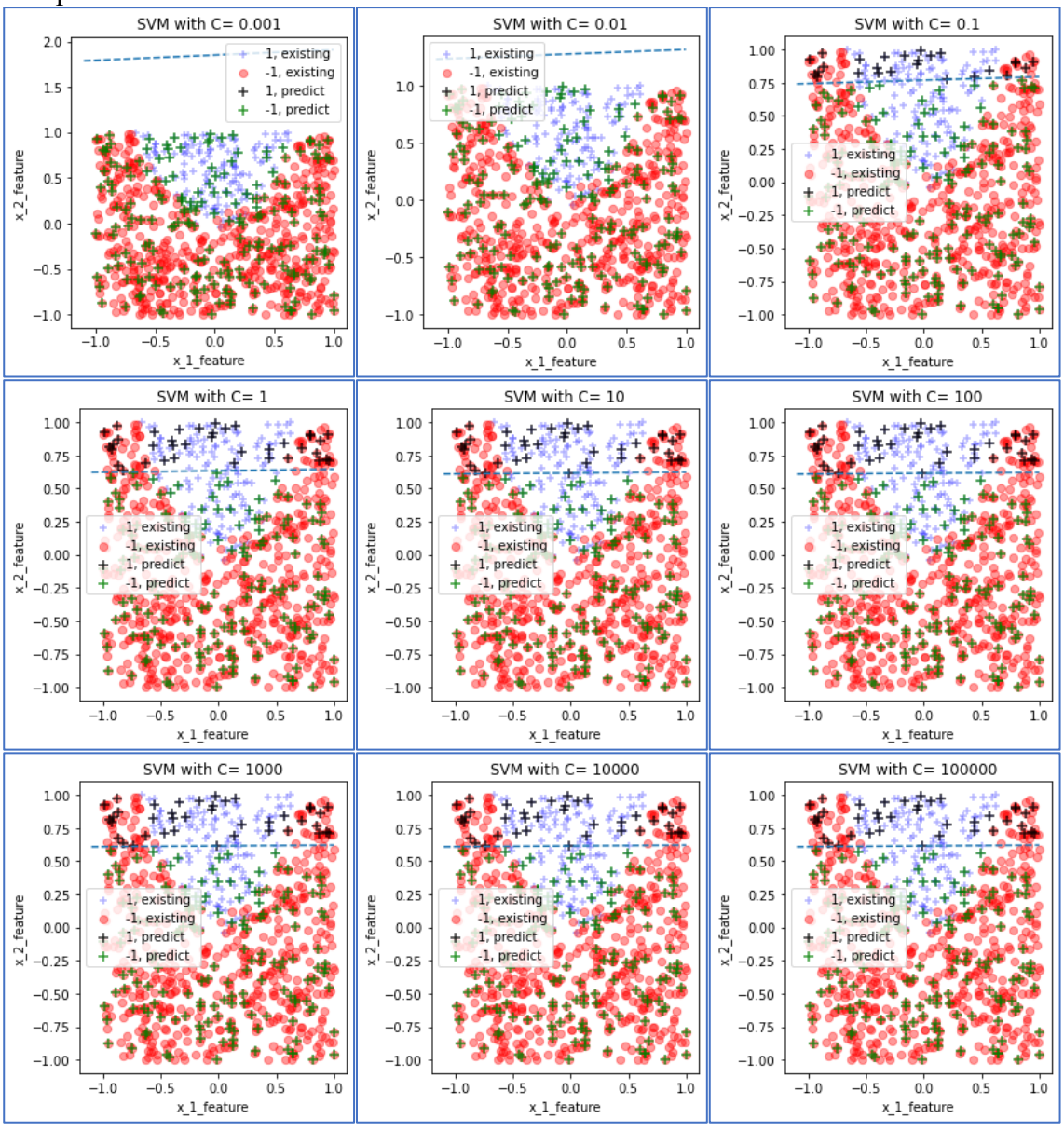
```
Model 1 Confusion Matrix (0.001)
 [[185   0]
 [ 55   0]]
Model 2 Confusion Matrix (0.01)
 [[175  10]
 [ 38  17]]
Model 3 Confusion Matrix (0.5)
 [[161  24]
 [ 31  24]]
Model 4 Confusion Matrix (1)
 [[175  10]
 [ 38  17]]
Model 5 Confusion Matrix (100)
 [[161  24]
 [ 31  24]]
```

Model 4 and 5 with a significant positive value of C have better prediction scores. We can see that a lower value of C skews the prediction model in generating a lot of False-Negatives. C=0.01 and C=1 give the same confusion matrix and C=0.5 and C=100 generate the same confusion matrix in this case.
Here the model is performing best at C=0.01 gives the best test accuracy, and confusion matrix also provides an overview of the FP and FN split.

The decision boundary is the line with equation

i) $m1*x1 + m2*x2 + c = 0;$

Which gives us the x2 = -(c + m1x1)/m2. (As before)
The plots with different C values are as follows:



As smaller C imply stronger regularization and therefor increase in the margin size, we see that the predicted 1 data is not even showing on the plot and that the decision boundary is plotted far off.

As C increase, the margin size decreases and we see the predicted 1 values denoted by a black + marker on the plots.

C=1 is optimally placed , with the decision boundary having a considerable weight for penalizing misclassification.

The SVM with the LinearSVC model and the Logistic Regression mode with solver lbfgs and penalty provide almost similar performances in terms of accuracy of prediction.

```
['SVM', 'C-value: 0.0001', 'Train accuracy: 0.768', 'Test accuracy: 0.775']
['SVM', 'C-value: 0.001', 'Train accuracy: 0.768', 'Test accuracy: 0.775']
['SVM', 'C-value: 0.01', 'Train accuracy: 0.805', 'Test accuracy: 0.795']
['SVM', 'C-value: 0.1', 'Train accuracy: 0.803', 'Test accuracy: 0.775']
['SVM', 'C-value: 1', 'Train accuracy: 0.803', 'Test accuracy: 0.775']
['SVM', 'C-value: 10', 'Train accuracy: 0.805', 'Test accuracy: 0.775']
['SVM', 'C-value: 100', 'Train accuracy: 0.805', 'Test accuracy: 0.775']
['SVM', 'C-value: 1000', 'Train accuracy: 0.805', 'Test accuracy: 0.775']
['SVM', 'C-value: 10000', 'Train accuracy: 0.805', 'Test accuracy: 0.775']
['SVM', 'C-value: 100000', 'Train accuracy: 0.805', 'Test accuracy: 0.775']
------------------------------------------------------------
['LReg', 'C-value: 0.0001', 'Train accuracy: 0.768', 'Test accuracy: 0.775']
['LReg', 'C-value: 0.001', 'Train accuracy: 0.768', 'Test accuracy: 0.775']
['LReg', 'C-value: 0.01', 'Train accuracy: 0.768', 'Test accuracy: 0.775']
['LReg', 'C-value: 0.1', 'Train accuracy: 0.808', 'Test accuracy: 0.795']
['LReg', 'C-value: 1', 'Train accuracy: 0.805', 'Test accuracy: 0.780']
['LReg', 'C-value: 10', 'Train accuracy: 0.805', 'Test accuracy: 0.780']
['LReg', 'C-value: 100', 'Train accuracy: 0.805', 'Test accuracy: 0.775']
['LReg', 'C-value: 1000', 'Train accuracy: 0.805', 'Test accuracy: 0.775']
['LReg', 'C-value: 10000', 'Train accuracy: 0.805', 'Test accuracy: 0.775']
['LReg', 'C-value: 100000', 'Train accuracy: 0.805', 'Test accuracy: 0.775']
```

Also, loss = squared_hinge which is the default value performed better than loss=hinge

## Q3

i) The following two features X1^2 (squared) and X2^2 (squared) are added.

```
df = pd.read_csv('week2.csv')
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X1sq=np.square(X1)
X2sq=np.square(X2)
X=np.column_stack((X1,X2,X1sq,X2sq))
y=df.iloc[:,2]
print(X)
```

```
[[-0.17   -0.37    0.0289  0.1369]
 [ 0.64   -0.71    0.4096  0.5041]
 [ 0.95    0.91    0.9025  0.8281]
 ...
 [ 0.18    0.46    0.0324  0.2116]
 [-0.08    0.43    0.0064  0.1849]
 [-0.13   -0.28    0.0169  0.0784]]
```

Now, the test accuracy for logistic regression improves considerably as compared to logistic regression in (a) and the accuracy scores overall are much better as compared to both (a) and (b) strategies.
Logistic Regression is run for different solvers as given in image above and different C values, along with penalty set to ‘l2’ and max_iter = 10000. The training accuracy and test accuracy scores are listed.

```
['LReg', 'Solver: lbfgs', 'C-value: 0.001', 'Train accuracy: 0.768', 'Test accuracy: 0.775']
['LReg', 'Solver: lbfgs', 'C-value: 0.01', 'Train accuracy: 0.768', 'Test accuracy: 0.775']
['LReg', 'Solver: lbfgs', 'C-value: 0.1', 'Train accuracy: 0.902', 'Test accuracy: 0.885']
['LReg', 'Solver: lbfgs', 'C-value: 1', 'Train accuracy: 0.980', 'Test accuracy: 0.955']
['LReg', 'Solver: lbfgs', 'C-value: 10', 'Train accuracy: 0.978', 'Test accuracy: 0.965']
['LReg', 'Solver: lbfgs', 'C-value: 100', 'Train accuracy: 0.983', 'Test accuracy: 0.965']
['LReg', 'Solver: lbfgs', 'C-value: 1000', 'Train accuracy: 0.985', 'Test accuracy: 0.970']
['LReg', 'Solver: liblinear', 'C-value: 0.001', 'Train accuracy: 0.768', 'Test accuracy: 0.775']
['LReg', 'Solver: liblinear', 'C-value: 0.01', 'Train accuracy: 0.768', 'Test accuracy: 0.775']
['LReg', 'Solver: liblinear', 'C-value: 0.1', 'Train accuracy: 0.913', 'Test accuracy: 0.890']
['LReg', 'Solver: liblinear', 'C-value: 1', 'Train accuracy: 0.980', 'Test accuracy: 0.955']
['LReg', 'Solver: liblinear', 'C-value: 10', 'Train accuracy: 0.978', 'Test accuracy: 0.965']
['LReg', 'Solver: liblinear', 'C-value: 100', 'Train accuracy: 0.983', 'Test accuracy: 0.965']
['LReg', 'Solver: liblinear', 'C-value: 1000', 'Train accuracy: 0.985', 'Test accuracy: 0.970']
['LReg', 'Solver: sag', 'C-value: 0.001', 'Train accuracy: 0.768', 'Test accuracy: 0.775']
['LReg', 'Solver: sag', 'C-value: 0.01', 'Train accuracy: 0.768', 'Test accuracy: 0.775']
['LReg', 'Solver: sag', 'C-value: 0.1', 'Train accuracy: 0.902', 'Test accuracy: 0.885']
['LReg', 'Solver: sag', 'C-value: 1', 'Train accuracy: 0.980', 'Test accuracy: 0.955']
['LReg', 'Solver: sag', 'C-value: 10', 'Train accuracy: 0.978', 'Test accuracy: 0.965']
['LReg', 'Solver: sag', 'C-value: 100', 'Train accuracy: 0.983', 'Test accuracy: 0.965']
['LReg', 'Solver: sag', 'C-value: 1000', 'Train accuracy: 0.985', 'Test accuracy: 0.970']
```

ii)

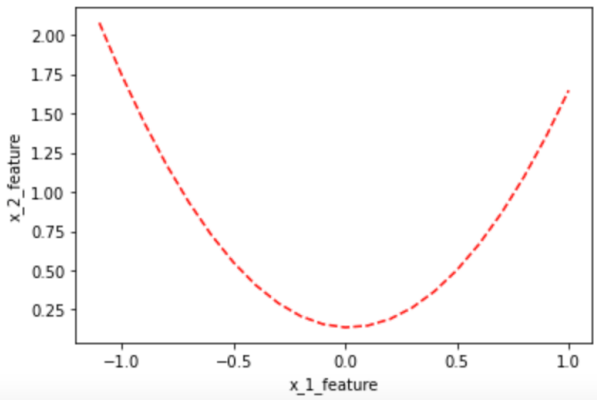
For one such case, solver='lbfgs', penalty='l2', C=1, max_iter=10000, the intercept and slope values are as follows
intercept : [-0.60871864]
slope : [ 0.21895866 4.46035395 -6.9530096 -0.13512954]

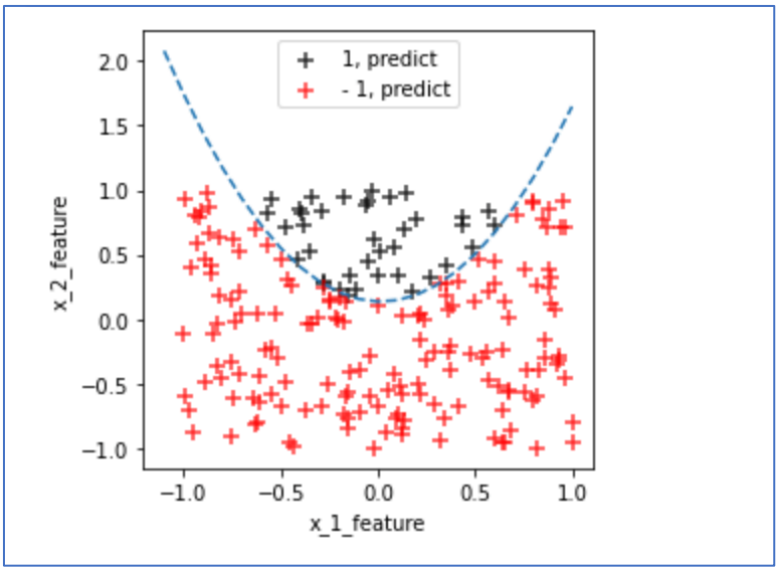
The line formula is :
m1x1 + m2x2 + m3x1sq + m4x2sq + c = 0

The line is calculated with the following formula: -(c + m1*x + m3*x*x)/m2
(ignore m4x2sq completely as the coefficient is low and will not matter in the denominator with m2)

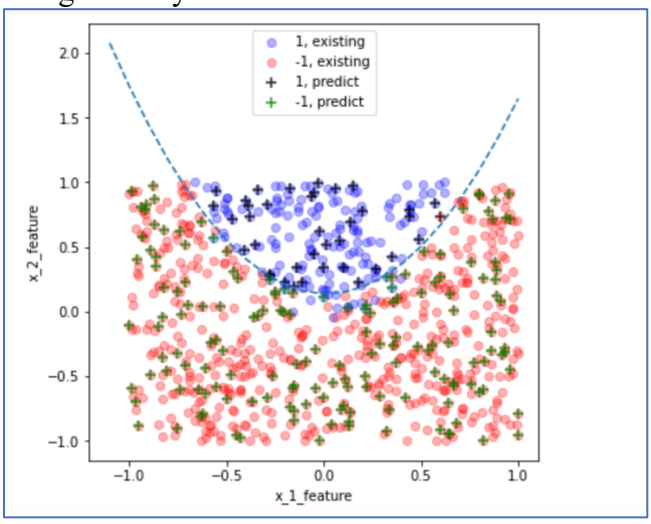
```
w1, w2, w3, w4 = model.coef_[0]
c = model.intercept_[0]
m1 = w1
m2 = w2
m3 = w3
m4 = w4
xlist = np.arange(-1.1,1.1,0.1)

def get_line_y2(x):
    return -(c + m1*x + m3*x*x)/m2

plt.plot(xlist,get_line_y2(xlist),'--',c='red')
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
```

```
Text(0, 0.5, 'x_2_feature')
```






As can be see from the predicted points from the test data plotted, that the decision boundary covers the scatter of the data much gracefully.




As can be seen from the above plot, the accuracy is much better, and the predications are very accurate with the existing complete dataset.

iii) The comparison of Logistic regression with Linear Regression and LinearSVC is given below.

```
---LogisticR with 2 addn squared features---
intercept : [-0.60871864]
slope :  [ 0.21895866  4.46035395 -6.9530096  -0.13512954]
Train Accuracy: 0.9799666110183639
Test Accuracy: 0.955
----
---LinearR with 2 addn squared features---
intercept : [-0.60871864]
slope :  [ 0.21895866  4.46035395 -6.9530096  -0.13512954]
Train Accuracy: 0.5569682645702245
Test Accuracy: 0.49479039432624083
----
---LinearSVC with 2 addn squared features---
intercept : [-0.10340052]
slope :  [ 0.14549376  3.04091035 -5.70468566 -0.22821798]
Train Accuracy: 0.9782971619365609
Test Accuracy: 0.965
----
```

Logistic regression performs good with the following parameters solver='lbfgs', penalty='l2', C=1, max_iter=10000 against no parameters in linear regression and almost similar to slightly average

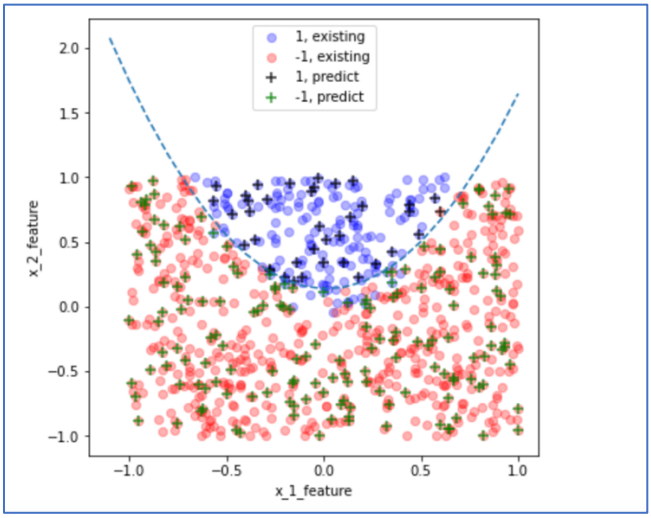
as compared to LinearSVC model with the following parameters C=1, dual=False, max_iter=100000.

iv) The decision boundary can be solved in two ways:

One – we ignore m4X2sq parameter which is the Feature_2 squared and multiplied by the 4th weight value in the array.

Two – we solve with two variables

I have not solved using method Two.



# APPENDIX

## Code For Q1:

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

#Read the csv file
df = pd.read_csv('week2.csv')
print(df)
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X=np.column_stack((X1,X2))
y=df.iloc[:,2]

#Plot the data on a graph
plt.subplots(figsize=(8, 8))
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
plt.scatter(X1[y==1], X2[y==1], c='blue', marker= '+', s=50, alpha=0.7, label='1')
plt.scatter(X1[y==-1], X2[y==-1],c='red', marker='o',s=15, alpha=0.7, label='-1')
plt.legend()

#Train the model
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.20,random_state = 0)

model = LogisticRegression(solver='lbfgs', penalty='l2', C=1, max_iter=200)
model.fit(X_train, Y_train)

#Print intercept, slope, model score, confusion matrix
print('intercept \n',(model.intercept_))
print('slope \n',( model.coef_[0]))
print('Model Score \n', model.score(X_train, Y_train))
Y_predict=model.predict(X_test)
cnf_matrix = metrics.confusion_matrix(Y_test, Y_predict)
print('Confusion Matrix \n', cnf_matrix)

#Plot the line using obtained line formula
plt.subplots(figsize=(8, 8))
w1, w2 = model.coef_[0]
c = model.intercept_[0]
m1 = w1
m2 = w2
xlist = np.arange(-1.1,1.1,0.1)
def get_line_y(x):
    return (-c-m1*x)/m2
plt.plot(xlist,get_line_y(xlist),'--')
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')

#Plot the predicted scatter
```

```
plt.subplots(figsize=(8, 8))
X1test = X_test[:,0]
X2test = X_test[:,1]
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
plt.scatter(X1test[Y_predict==1],X2test[Y_predict==1], c='black', marker= '+', s=50, alpha=0.8, label='1')
plt.scatter(X1test[Y_predict==-1],X2test[Y_predict==-1], c='red', marker='+',s=50, alpha=0.8, label='-1')
plt.legend()
plt.show()


#Plot line on predicted scatter
plt.subplots(figsize=(8, 8))
X1test = X_test[:,0]
X2test = X_test[:,1]
plt.plot(xlist,get_line_y(xlist),'--')
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
plt.scatter(X1test[Y_predict==1],X2test[Y_predict==1], c='black', marker= '+', s=50, alpha=0.8, label='1,
predict')
plt.scatter(X1test[Y_predict==-1],X2test[Y_predict==-1], c='red', marker='+',s=50, alpha=0.8, label='- 1,
predict')
plt.legend()
plt.show()

#Plot FP and FN
plt.subplots(figsize=(8, 8))
X1test = X_test[:,0]
X2test = X_test[:,1]
plt.plot(xlist,get_line_y(xlist),'--')
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
plt.scatter(X1test[Y_predict==1],X2test[Y_predict==1],c='black', marker= '+', s=50, alpha=0.6, label='1,
predict')
plt.scatter(X1test[Y_test==1],X2test[Y_test==1], c='red', marker= 'o', s=50, alpha=0.4, label='1, test')
plt.legend()
plt.show()

plt.subplots(figsize=(8, 8))
X1test = X_test[:,0]
X2test = X_test[:,1]
plt.plot(xlist,get_line_y(xlist),'--')
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
plt.scatter(X1test[Y_predict==-1],X2test[Y_predict==-1],c='red', marker='+',s=50, alpha=0.8, label='- 1,
predict')
plt.scatter(X1test[Y_test==-1],X2test[Y_test==-1], c='yellow', marker='o',s=50, alpha=0.4, label='- 1, test')
plt.legend()
plt.show()


#Plot decision boundary along with predictions on original scatter
plt.subplots(figsize=(8, 8))
X1test = X_test[:,0]
X2test = X_test[:,1]
plt.plot(xlist,get_line_y(xlist),'--')
plt.xlabel('x_1_feature')
```

```
plt.ylabel('x_2_feature')
plt.scatter(X1[y==1], X2[y==1], c='blue', marker= 'o', s=40, alpha=0.3, label='1, existing')
plt.scatter(X1[y==-1], X2[y==-1],c='red', marker='o',s=40, alpha=0.4, label='-1, existing')
plt.scatter(X1test[Y_predict==1],X2test[Y_predict==1], c='black', marker= '+', s=50, alpha=0.8, label='1, predict')
plt.scatter(X1test[Y_predict==-1],X2test[Y_predict==-1], c='green', marker='+',s=50, alpha=0.8, label='-1, predict')
plt.legend()
plt.show()


#Code to iterate over different C values and plot the decision boundary
from sklearn.linear_model import LogisticRegression
solver_arr=['newton-cg','lbfgs','liblinear','sag','saga']
C_values = [0.01, 0.1, 1, 10, 100]
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.25,random_state=0)
models_list2 = []
for solver_itr in solver_arr:
    for c_value in C_values:
        logR_clf_c = LogisticRegression(solver=solver_itr, penalty='l2', C=c_value).fit(X_train, Y_train)
        model = "LReg"
        solvers = "Solver: "+ solver_itr
        C_value = f"C-value: {c_value}"
        train_acc = "Train accuracy: {:.3f}".format(logR_clf_c.score(X_train, Y_train))
        test_acc = "Test accuracy: {:.3f}".format(logR_clf_c.score(X_test, Y_test))
        models_list2.append([model, solvers, C_value, train_acc, test_acc])
        Y_predict=logR_clf_c.predict(X_test)
        w1, w2 = logR_clf_c.coef_[0]
        c = logR_clf_c.intercept_[0]
        m1 = w1
        m2 = w2
        xlist = np.arange(-1.1,1.1,0.1)
        xlist = np.arange(-1.1,1.1,0.1)
        plt.title('SVM with C= ' + str(c_value) + ' and solver ' + solver_itr)
        def get_line_y2(x):
            return (-c-m1*x)/m2
        plt.subplots(figsize=(4,4))
        X1test = X_test[:,0]
        X2test = X_test[:,1]
        plt.xlabel('x_1_feature')
        plt.ylabel('x_2_feature')
        plt.scatter(X1[y==1], X2[y==1], c='blue', marker= '+', s=40, alpha=0.3, label='1, existing')
        plt.scatter(X1[y==-1], X2[y==-1],c='red', marker='o',s=40, alpha=0.4, label='-1, existing')
        plt.scatter(X1test[Y_predict==1],X2test[Y_predict==1], c='black', marker= '+', s=50, alpha=0.8, label='1, predict')
        plt.scatter(X1test[Y_predict==-1],X2test[Y_predict==-1], c='green', marker='+',s=50, alpha=0.8, label='-1, predict')
        plt.plot(xlist,get_line_y(xlist),'--')
        plt.legend()
        plt.tight_layout()
print(*models_list2, sep='\n')
```

## Code for Q2:

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

#Read the file
df = pd.read_csv('week2.csv')
print(df)
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X=np.column_stack((X1,X2))
y=df.iloc[:,2]

#Plot the data
plt.subplots(figsize=(5, 5))
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
plt.scatter(X1[y==1], X2[y==1], c='blue', marker= '+', s=40, alpha=0.7, label='1')
plt.scatter(X1[y==-1], X2[y==-1],c='red', marker='o',s=15, alpha=0.7, label='-1')
plt.legend()
plt.show()

#Train different SVM LinearSVC models for changing C Values
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn import metrics
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.3,random_state=0)
model1 = svm.LinearSVC(C=0.001, dual=False)
model1.fit(X_train, Y_train)
model2 = svm.LinearSVC(C=0.01, dual=False)
model2.fit(X_train, Y_train)
model3 = svm.LinearSVC(C=0.5, dual=False)
model3.fit(X_train, Y_train)
model4 = svm.LinearSVC(C=1, dual=False)
model4.fit(X_train, Y_train)
model5 = svm.LinearSVC(C=100, dual=False)
model5.fit(X_train, Y_train)

#Loop through different C values and quote accuracies
C_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
models_list2 = []
for c_value in C_values:
    svc_clf_c = svm.LinearSVC(C=c_value, dual=False).fit(X_train, Y_train)
    model = "SVM"
    C_value = f"C-value: {c_value}"
    train_acc = "Train accuracy: {:.3f}".format(svc_clf_c.score(X_train, Y_train))
    test_acc = "Test accuracy: {:.3f}".format(svc_clf_c.score(X_test, Y_test))
    models_list2.append([model, C_value, train_acc, test_acc])
print(*models_list2, sep='\n')

#Print confusion matrices
Y_predict1=model1.predict(X_test)
cnf_matrix = metrics.confusion_matrix(Y_test, Y_predict1)
print('Model 1 Confusion Matrix (0.001)\n', cnf_matrix)
```

```
Y_predict2=model2.predict(X_test)
cnf_matrix = metrics.confusion_matrix(Y_test, Y_predict2)
print('Model 2 Confusion Matrix (0.01)\n', cnf_matrix)
Y_predict3=model5.predict(X_test)
cnf_matrix = metrics.confusion_matrix(Y_test, Y_predict3)
print('Model 3 Confusion Matrix (0.5)\n', cnf_matrix)
Y_predict4=model2.predict(X_test)
cnf_matrix = metrics.confusion_matrix(Y_test, Y_predict4)
print('Model 4 Confusion Matrix (1)\n', cnf_matrix)
Y_predict5=model3.predict(X_test)
cnf_matrix = metrics.confusion_matrix(Y_test, Y_predict5)
print('Model 5 Confusion Matrix (100)\n', cnf_matrix)


#Choose one model to proceed with decision boundary
print('Model 2 intercept (0.01)\n',(model2.intercept_))
print('Model 2 slope (0.01)\n',( model2.coef_[0]))

#Plot the decision boundary
w1, w2 = model1.coef_[0]
c = model1.intercept_[0]
m1 = w1
m2 = w2
xlist = np.arange(-1.1,1.1,0.1)
def get_line_y(x):
    return (-c-m1*x)/m2
plt.plot(xlist,get_line_y(xlist),'--')
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')

#Plot the predicted scatter
X1test = X_test[:,0]
X2test = X_test[:,1]
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
plt.scatter(X1test[Y_predict==1],X2test[Y_predict==1], c='black', marker= '+', s=50, alpha=0.8, label='1')
plt.scatter(X1test[Y_predict==-1],X2test[Y_predict==-1], c='red', marker='+',s=50, alpha=0.8, label='-1')
plt.legend()
plt.show()

#Plot for different decision boundaries on different C values
C_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.25,random_state=0)
for c_value in C_values:
    svc_clf_c = svm.LinearSVC(C=c_value, dual=False).fit(X_train, Y_train)
    w1, w2 = svc_clf_c.coef_[0]
    c = svc_clf_c.intercept_[0]
    m1 = w1
    m2 = w2
    Y_predict=svc_clf_c.predict(X_test)
    xlist = np.arange(-1.1,1.1,0.1)
    plt.title('SVM with C= ' + str(c_value))
    def get_line_y(x):
        return (-c-m1*x)/m2
    plt.subplots(figsize=(4,4))
    X1test = X_test[:,0]
    X2test = X_test[:,1]
```

```
    plt.xlabel('x_1_feature')
    plt.ylabel('x_2_feature')
    plt.scatter(X1[y==1], X2[y==1], c='blue', marker= '+', s=40, alpha=0.3, label='1, existing')
    plt.scatter(X1[y==-1], X2[y==-1],c='red', marker='o',s=40, alpha=0.4, label='-1, existing')
    plt.scatter(X1test[Y_predict==1],X2test[Y_predict==1], c='black', marker= '+', s=50, alpha=0.8, label='1,
predict')
    plt.scatter(X1test[Y_predict==-1],X2test[Y_predict==-1], c='green', marker='+',s=50, alpha=0.8, label='-1,
predict')
    plt.plot(xlist,get_line_y(xlist),'--')
    plt.legend()
    plt.tight_layout()


#Check different loss functions in LinearSVC
from sklearn import svm
C_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
loss_arr=['hinge', 'squared_hinge']
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.25,random_state=0)
models_list2 = []
for loss_itr in loss_arr:
    for c_value in C_values:
        svc_clf_c = svm.LinearSVC(loss=loss_itr, C=c_value, max_iter=100000).fit(X_train, Y_train)
        model = "SVM"
        losse = "Loss: "+ loss_itr
        C_value = f"C-value: {c_value}"
        train_acc = "Train accuracy: {:.3f}".format(svc_clf_c.score(X_train, Y_train))
        test_acc = "Test accuracy: {:.3f}".format(svc_clf_c.score(X_test, Y_test))
        models_list2.append([model, losse, C_value, train_acc, test_acc])
print(*models_list2, sep='\n')
```

## Code for Q3:

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

#Read the file and add X1^2 and X2^2 in the columns stack
df = pd.read_csv('week2.csv')
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X1sq=np.square(X1)
X2sq=np.square(X2)
X=np.column_stack((X1,X2,X1sq,X2sq))
y=df.iloc[:,2]
print(X)

#Print accuracies for different models: Logistic, Linear, LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import svm

models_list2 = []
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.25,random_state=0)
model = LogisticRegression(solver='lbfgs', penalty='l2', C=1, max_iter=10000)
```

```
model.fit(X_train, Y_train)
print('---LogisticR with 2 addn squared features---')
print('intercept :',(model.intercept_))
print('slope : ',(model.coef_[0]))
print('Train Accuracy:', model.score(X_train, Y_train))
print('Test Accuracy:', model.score(X_test, Y_test))
print('----')
model2 = LinearRegression()
model2.fit(X_train, Y_train)
print('---LinearR with 2 addn squared features---')
print('intercept :',(model.intercept_))
print('slope : ',(model.coef_[0]))
print('Train Accuracy:', model2.score(X_train, Y_train))
print('Test Accuracy:', model2.score(X_test, Y_test))
print('----')
model3 = svm.LinearSVC(C=1, dual=False, max_iter=100000)
model3.fit(X_train, Y_train)
print('---LinearSVC with 2 addn squared features---')
print('intercept :',(model3.intercept_))
print('slope : ',(model3.coef_[0]))
print('Train Accuracy:', model3.score(X_train, Y_train))
print('Test Accuracy:', model3.score(X_test, Y_test))
print('----')


#Plot the line using the eqn where we ignore the last term in denominator – liney2
w1, w2, w3, w4 = model.coef_[0]
c = model.intercept_[0]
m1 = w1
m2 = w2
m3 = w3
m4 = w4
xlist = np.arange(-1.1,1.1,0.1)

def get_line_y1(x):
    return -m2 + np.roots(m2*m2*x)/(2*m4)

def get_line_y2(x):
    return -(c + m1*x + m3*x*x)/m2

plt.plot(xlist,get_line_y2(xlist),'--',c='red')
plt.plot(xlist,get_line_y1(xlist),'--',c='black')
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')

#Print the line against the original data
X1test = X_test[:,0]
X2test = X_test[:,1]
plt.subplots(figsize=(6,6))
plt.plot(xlist,get_line_y2(xlist),'--')
plt.xlabel('x_1_feature')
plt.ylabel('x_2_feature')
plt.scatter(X1[y==1], X2[y==1], c='blue', marker= 'o', s=40, alpha=0.3, label='1, existing')
plt.scatter(X1[y==-1], X2[y==-1],c='red', marker='o',s=40, alpha=0.3, label='-1, existing')
plt.scatter(X1test[Y_predict==1],X2test[Y_predict==1], c='black', marker= '+', s=50, alpha=0.8, label='1,
predict')
```

```
plt.scatter(X1test[Y_predict==-1],X2test[Y_predict==-1], c='green', marker='+',s=50, alpha=0.8, label='-1, predict')
plt.legend()
plt.show()
```

**Links that helped:**


[1]: https://stackoverflow.com/questions/66045152/how-can-a-lower-c-parameter-value-lead-to-both-better-training-and-testing-score
[2]: https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769
[3]: https://stackoverflow.com/questions/52670012/convergencewarning-liblinear-failed-to-converge-increase-the-number-of-iterati
[4]: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[5]: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[6]: https://stackoverflow.com/questions/62232929/behavior-of-c-in-linearsvc-sklearn-scikit-learn
[7]: https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769