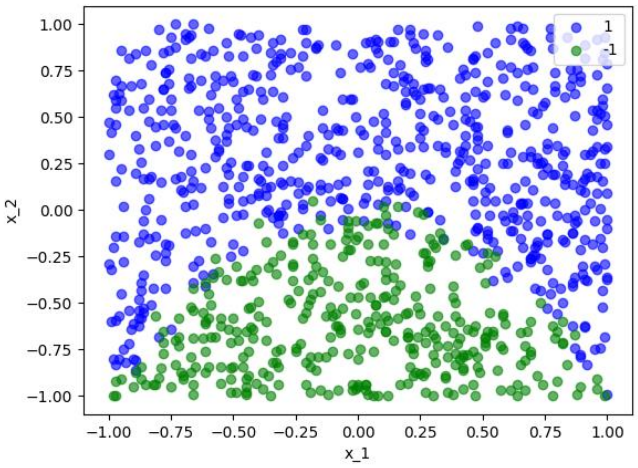
Machine Learning Week 2 Assignment

Jiaming Deng 22302794

a (i) Firstly, I copy the data from the web page into a txt file and use numpy's load_txt function to read it. Secondly, I put the two features of the data in data_A_X, data_A_Y and data_B_X, data_B_Y, where the target is 1 in the first two arrays and the target is -1 in the last two arrays. Lastly, I use matplotlib's scatter function to display them in a graph, where blue indicates points with target of 1 and green indicates points with target of -1.

```python
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import numpy as np

data_txt = np.loadtxt("dataset.txt", delimiter=',')
data_A_X = []
data_A_Y = []
data_B_X = []
data_B_Y = []
for i in range(len(data_txt)):
    if data_txt[i][2] == 1:
        t = data_txt[i]
        data_A_X.append(t[0])
        data_A_Y.append(t[1])
    else:
        t = data_txt[i]
        data_B_X.append(t[0])
        data_B_Y.append(t[1])
print(data_A_X)
plt.xlabel('x_1')
plt.ylabel('x_2')
plt.scatter(data_A_X, data_A_Y, alpha=0.6, c='blue')
plt.scatter(data_B_X, data_B_Y, alpha=0.6, c='green')
plt.legend(['1','-1'],loc='upper right')
plt.show()
```



a (ii) After loading the dataset, I split the dataset into a training set and a test set, with 70% of the training set and 30% of the test set. Then, training the classifier using a logistic regression algorithm through the sklearn framework and obtained an accuracy rate 87%. The slopes [θ_1, θ_2] of the classifier are [0.07636896, 4.95741656]. The intercept is 1.74791901.

According to the principles of logistic regression, the decision boundary is $y = \theta^T \cdot X$. As the slope θ_2 is much more than θ_1, x2 has most influence on the prediction. When x1 or x2 is negative, it causes the prediction to decrease and when x1 or x2 is positive, it causes the prediction to increase. Because the slopes are both positive.

```
/usr/local/bin/python3.8 /Users/dengjiaming/Desktop/ML/A1/a2.py
Accucary: 0.87
Slope:  [[0.07636896 4.95741656]]
intercept:  [1.74791901]
```

```
import ...
data_txt = np.loadtxt("dataset.txt", delimiter=',')
data_X = []
data_Y = []
for i in range(len(data_txt)):
    data_X.append([data_txt[i][0], data_txt[i][1]])
    data_Y.append(data_txt[i][2])

X_train, X_test, Y_train, Y_test = train_test_split(data_X, data_Y, test_size=0.3, random_state=40)

logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

acc = logreg.score(X_test, Y_test)
slope = logreg.coef_
intercept = logreg.intercept_

print('Accucary:', acc)
print('Slope: ', slope)
print('intercept: ', intercept)
```

a (iii) Firstly using the trained logistic regression model, I predict the targets for the test set and plot these predictions in the graph. Secondly to better see if the predictions are correct, points where the predicted target is 1 are plotted in red plus, and points where it is -1 are plotted in black minus.

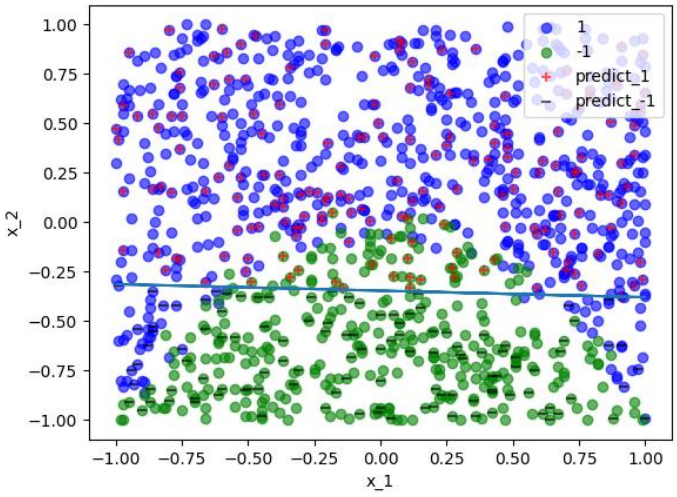
According to the principles of logistic regression, the decision boundary is $\theta^T \bullet$

$X = 0$. This question has two features x1 and x2, so the decision boundary is $\theta^T \bullet$

$X = \theta_0 + \theta_1 X_1 + \theta_2 X_2 = 0$. So deforming the formula to get $X_2 = \frac{-\theta_0 - \theta_1 X_1}{\theta_2}$.

```
intercept = logreg.intercept_[0]
coef_1 = logreg.coef_[0][0]
coef_2 = logreg.coef_[0][1]
y = []
for i in x:
    y.append((-coef_1 * i - intercept) / coef_2)
```

```
for i in range(len(ans)):
    if ans[i] == 1.0:
        pre_A_X.append(X_test[i][0])
        pre_A_Y.append(X_test[i][1])
    else:
        pre_B_X.append(X_test[i][0])
        pre_B_Y.append(X_test[i][1])
```



a(iv) Principlely if the training data is concentrated, the prediction accuracy will be higher. However, if some points in the training data deviate significantly from the range of most points, the prediction accuracy will drop. For this dataset, it is better to use a non-linear decision boundary to classify the data, so the linear decision boundary I used did not get very high prediction accuracy.

b (i) Firstly after load the dataset, I split the dataset into 70% training set and 30% test set. Secondly, using the linear SVM model of the sklearn framework to train the data. By adjusting the value of C, specifically the number from 0.001 to 10 is divided into 100 pieces of data by using the numpy linspace function. The obtained accuracy is partially intercepted as shown in the figure below, you can see the accuracy from 0.01 to 0.011 rise from 83.3% to 88.7% and stay at 88.3% after that. The parameter values are also showed.

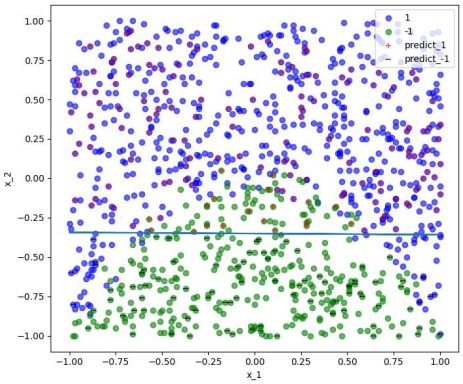
```
import ...
data_txt = np.loadtxt("dataset.txt", delimiter=',')

data_X = []
data_Y = []
for i in range(len(data_txt)):
    data_X.append([data_txt[i][0], data_txt[i][1]])
    data_Y.append(data_txt[i][2])

X_train, X_test, Y_train, Y_test = train_test_split(data_X, data_Y, test_size=0.24, random_state=40)


c = np.linspace(0.001, 0.1, 100)
print("C         acc       w1        w2        b")
for i in c:
    clf = svm.LinearSVC(C=i)
    clf.fit(X_train, Y_train)
    acc = clf.score(X_test, Y_test)
    print('%.3f' % i, '    ', '%.3f' % acc, '    ', '%.3f' % clf.coef_[0][0], '    ', '%.3f' % clf.coef_[0][1], '    ', '%.3f
```

```
/usr/local/bin/python3.8 /Users/dengjiaming/Desktop/ML/A1/b1.py
C         acc       w1        w2        b
0.001     0.833     0.037     0.399     0.198
0.002     0.850     0.045     0.598     0.252
0.003     0.875     0.047     0.717     0.278
0.004     0.875     0.048     0.802     0.296
0.005     0.879     0.049     0.873     0.313
0.006     0.887     0.049     0.933     0.329
0.007     0.887     0.048     0.984     0.343
0.008     0.887     0.048     1.029     0.356
0.009     0.887     0.047     1.068     0.367
0.010     0.887     0.046     1.104     0.378
0.011     0.887     0.045     1.136     0.389
0.012     0.887     0.044     1.165     0.398
0.013     0.887     0.043     1.191     0.406
0.014     0.887     0.042     1.215     0.414
0.015     0.887     0.041     1.237     0.422
0.016     0.887     0.040     1.257     0.429
0.017     0.887     0.039     1.276     0.435
0.018     0.887     0.039     1.293     0.441
0.019     0.887     0.038     1.310     0.447
0.020     0.887     0.036     1.325     0.453
0.021     0.887     0.035     1.340     0.458
0.022     0.887     0.034     1.354     0.463
0.023     0.887     0.034     1.366     0.468
0.024     0.887     0.033     1.378     0.472
```

b (ii) Using the trained SVM linear model, I predict the targets for the test set and plot these predictions in the graph. To better see if the predictions are correct, points where the predicted target is 1 are plotted in red plus, and points where it is -1 are plotted in black minus.

According to the principles of SVM linear model, the decision boundary is $\theta^T \bullet X = 0$. This question has two features x1 and x2, so the decision boundary is $\theta^T \bullet X = \theta_0 + \theta_1 X_1 + \theta_2 X_2 = 0$. So we deform the formula to get $X_2 = \frac{-\theta_0 - \theta_1 X_1}{\theta_2}$.

```
clf = svm.LinearSVC(C=0.1)
clf.fit(X_train, Y_train)
ans = clf.predict(X_test)
b = clf.intercept_
w1, w2 = clf.coef_[0]
for i in x:
    y.append(-w1/w2*i - b/w2)
```



b (iii) When I increase C from 0.001 to 0.008, x is increasing, but after that x is decreasing. y and z keep increasing during the whole time.
C is the penalty parameter. The greater C causes the greater penalty for misclassified samples, so the accuracy in the training samples is higher, but the generalization ability is reduced so the classification accuracy of the test data is reduced. On the contrary if C is reduced, some misclassified samples in the training samples are allowed, and the generalization ability is strong. So the accuracy increases at first and then declines and then remains the same.

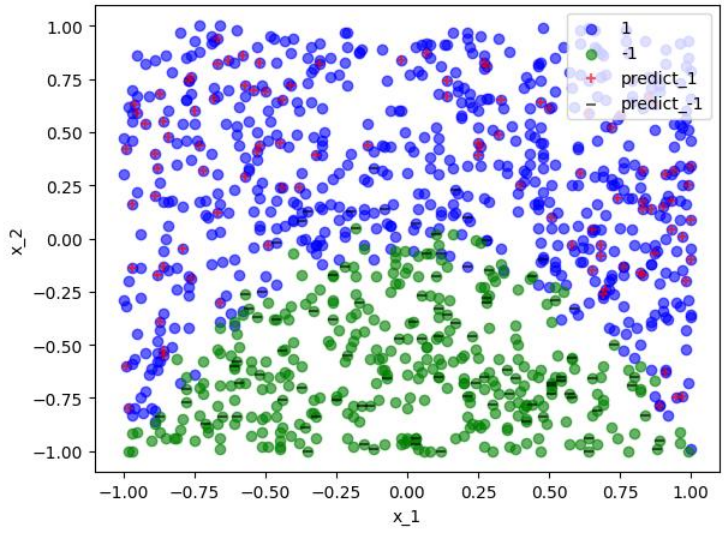
b (iv) When C (penaty parameter of SVM model) is very small, the prediction accuracy of SVM linear model is less than that of logistic regression model, but when C is bigger, the prediction accuracy of SVM linear model is more. The two parameters of SVM linear model are less than that of logistic regression model so the impact of two features of SVM linearr model are less than that of logistic regression

c(i) Firstly, I load dataset using numpy. Secondly I create two additional features by adding the square of each feature. Lastly I train a logistic regression classifier and get the accuracy and parameters of the model. The accuracy is 95%, the slopes are [-0.02297434, 6.0553406, 5.47963709, -1.07327615] and the intercept is 0.56069187.

```
import ...
data_txt = np.loadtxt("dataset.txt", delimiter=',')
data_X = []
data_Y = []
for i in range(len(data_txt)):
    data_X.append([data_txt[i][0], data_txt[i][1], data_txt[i][0]*data_txt[i][0], data_txt[i][1]*data_txt[i][1]])
    data_Y.append(data_txt[i][2])

X_train, X_test, Y_train, Y_test = train_test_split(data_X, data_Y, test_size=0.3, random_state=40)


logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

acc = logreg.score(X_test, Y_test)

print(acc)
```

```
/usr/local/bin/python3.8 /Users/dengjiaming/Desktop/ML/A1/c1.py
accuracy:  0.95
slopes:  [[-0.02297434  6.0553406   5.47963709 -1.07327615]]
intercept:  [0.56069187]
```

c(ii) After adding two parameters, the prediction accuracy of the logistic regression model is 95% which is larger than the accuracy of modle of (a) and (b).



c(iii) I choose DummyClassifier from sklearn as the baseline predictor. The prediction accuracy of A and B are 61.3% and 95% respectively. Therefore, the prediction performance of B is much better than that of A.

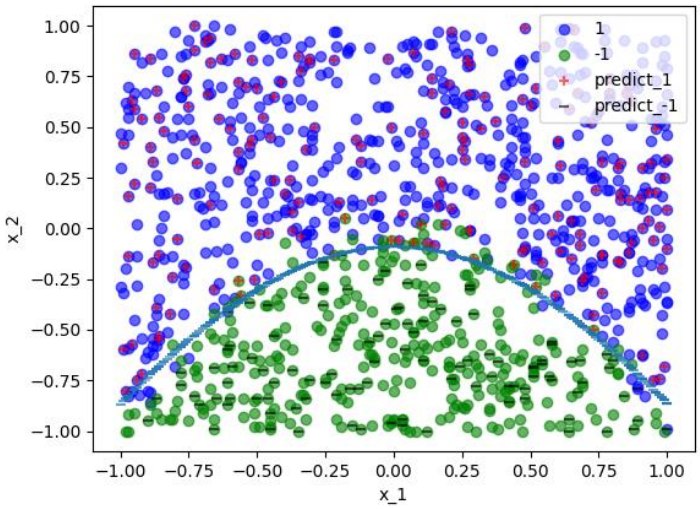
```
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

baseline = DummyClassifier(strategy="most_frequent")
baseline.fit(X_train, Y_train)

acc_1 = logreg.score(X_test, Y_test)

acc_2 = baseline.score(X_test, Y_test)

print('logistic regression model accuracy of prediction: ', acc_1)
print('baseline model accuracy of prediction: ', acc_2)
```

```
/usr/local/bin/python3.8 /Users/dengjiaming/Desktop/ML/A1/c3.py
logistic regression model accuracy of prediction:  0.95
baseline model accuracy of prediction:  0.6133333333333333
```

c(iv) According to the principles of logistic regression model, the decision boundary is $\theta^T \bullet X = 0$. This question has four features x1, x2, x1*x1 and x2*x2 so the decision boundary is $\theta^T \bullet X = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_1^{\ 2} + \theta_4 X_2^{\ 2} = 0$. So according to the formula for finding the root of the quadratic equation of one variable

$$X_2 = \frac{-(\theta_2) \pm \sqrt{(\theta_2)^2 - 4 * \theta_4 * (\theta_3 * (X_1)^2 + \theta_1 X_1 + \theta_0)}}{4 * \theta_4}.$$

```
y = []
b = logreg.intercept_[0]
w = logreg.coef_[0]

for i in x:
    y.append((-w[1] + math.sqrt(w[1]*w[1] - 4*w[3]*(b+w[0]*i+w[2]*i*i))) / (2*w[3]))
```



Appendix

a (i)

```
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import numpy as np

data_txt = np.loadtxt("dataset.txt", delimiter=',')
data_A_X = []
data_A_Y = []
data_B_X = []
data_B_Y = []
for i in range(len(data_txt)):
    if data_txt[i][2] == 1:
        t = data_txt[i]
        data_A_X.append(t[0])
        data_A_Y.append(t[1])
    else:
        t = data_txt[i]
        data_B_X.append(t[0])
        data_B_Y.append(t[1])
print(data_A_X)
plt.xlabel('x_1')
plt.ylabel('x_2')
plt.scatter(data_A_X, data_A_Y, alpha=0.6, c='blue')
plt.scatter(data_B_X, data_B_Y, alpha=0.6, c='green')
plt.legend(['1','-1'],loc='upper right')
plt.show()
```

a (ii)

```
import ...
data_txt = np.loadtxt("dataset.txt", delimiter=',')
data_X = []
data_Y = []
for i in range(len(data_txt)):
    data_X.append([data_txt[i][0], data_txt[i][1]])
    data_Y.append(data_txt[i][2])

X_train, X_test, Y_train, Y_test = train_test_split(data_X, data_Y, test_size=0.3, random_state=40)

logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

acc = logreg.score(X_test, Y_test)
slope = logreg.coef_
intercept = logreg.intercept_

print('Accucary:', acc)
print('Slope: ', slope)
print('intercept: ', intercept)
```

a (iii)

```
import ...
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
data_txt = np.loadtxt("dataset.txt", delimiter=',')
x = []
data_X = []
data_Y = []
for i in range(len(data_txt)):
    data_X.append([data_txt[i][0], data_txt[i][1]])
    data_Y.append(data_txt[i][2])
    x.append(data_txt[i][0])

X_train, X_test, Y_train, Y_test = train_test_split(data_X, data_Y, test_size=0.3, random_state=0)

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

ans = logreg.predict(X_test)
prepro = logreg.predict_proba(X_test)
acc = logreg.score(X_test, Y_test)

intercept = logreg.intercept_[0]
coef_1 = logreg.coef_[0][0]
coef_2 = logreg.coef_[0][1]
print(intercept)
y = []
for i in x:
    y.append((-coef_1 * i - intercept) / coef_2)
```

```
data_A_X = []
data_A_Y = []
data_B_X = []
data_B_Y = []
pre_A_X = []
pre_A_Y = []
pre_B_X = []
pre_B_Y = []
for i in range(len(data_txt)):
    if data_txt[i][2] == 1:
        t = data_txt[i]
        data_A_X.append(t[0])
        data_A_Y.append(t[1])
    else:
        t = data_txt[i]
        data_B_X.append(t[0])
        data_B_Y.append(t[1])
for i in range(len(ans)):
    if ans[i] == 1.0:
        pre_A_X.append(X_test[i][0])
        pre_A_Y.append(X_test[i][1])
    else:
        pre_B_X.append(X_test[i][0])
        pre_B_Y.append(X_test[i][1])

plt.xlabel('x_1')
plt.ylabel('x_2')
plt.scatter(data_A_X, data_A_Y, alpha=0.6, c='blue')
plt.scatter(data_B_X, data_B_Y, alpha=0.6, c='green')
plt.scatter(pre_A_X, pre_A_Y, alpha=0.6, c='red', marker="+")
plt.scatter(pre_B_X, pre_B_Y, alpha=0.6, c='black', marker="_")
plt.plot(x, y)
plt.legend(['1','-1', 'predict_1', 'predict_-1'],loc='upper right')
plt.show()
```

b(i)

```
import ...
data_txt = np.loadtxt("dataset.txt", delimiter=',')

data_X = []
data_Y = []
for i in range(len(data_txt)):
    data_X.append([data_txt[i][0], data_txt[i][1]])
    data_Y.append(data_txt[i][2])

X_train, X_test, Y_train, Y_test = train_test_split(data_X, data_Y, test_size=0.24, random_state=40)


c = np.linspace(0.001, 0.1, 100)
print("C         acc       w1        w2        b")
for i in c:
    clf = svm.LinearSVC(C=i)
    clf.fit(X_train, Y_train)
    acc = clf.score(X_test, Y_test)
    print('%.3f' % i, '    ', '%.3f' % acc, '    ', '%.3f' % clf.coef_[0][0], '    ', '%.3f' % clf.coef_[0][1], '    ', '%.3f' % clf.intercept_)
```

b(ii)

```
import ...
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
data_txt = np.loadtxt("dataset.txt", delimiter=',')
data_X = []
data_Y = []
x = []
y = []
for i in range(len(data_txt)):
    data_X.append([data_txt[i][0], data_txt[i][1]])
    data_Y.append(data_txt[i][2])
    x.append(data_txt[i][0])

X_train, X_test, Y_train, Y_test = train_test_split(data_X, data_Y, test_size=0.24, random_state=40)


clf = svm.LinearSVC(C=0.1)
clf.fit(X_train, Y_train)
ans = clf.predict(X_test)
b = clf.intercept_
w1, w2 = clf.coef_[0]
print(b)
for i in x:
    y.append(-w1/w2*i - b/w2)
```

```
data_A_X = []
data_A_Y = []
data_B_X = []
data_B_Y = []
pre_A_X = []
pre_A_Y = []
pre_B_X = []
pre_B_Y = []
for i in range(len(data_txt)):
    if data_txt[i][2] == 1:
        t = data_txt[i]
        data_A_X.append(t[0])
        data_A_Y.append(t[1])
    else:
        t = data_txt[i]
        data_B_X.append(t[0])
        data_B_Y.append(t[1])
for i in range(len(ans)):
    if ans[i] == 1.0:
        pre_A_X.append(X_test[i][0])
        pre_A_Y.append(X_test[i][1])
    else:
        pre_B_X.append(X_test[i][0])
        pre_B_Y.append(X_test[i][1])

plt.xlabel('x_1')
plt.ylabel('x_2')
plt.scatter(data_A_X, data_A_Y, alpha=0.6, c='blue')
plt.scatter(data_B_X, data_B_Y, alpha=0.6, c='green')
plt.scatter(pre_A_X, pre_A_Y, alpha=0.6, c='red', marker="+")
plt.scatter(pre_B_X, pre_B_Y, alpha=0.6, c='black', marker="_")
plt.plot(x, y)
plt.legend(['1','-1', 'predict_1', 'predict_-1'],loc='upper right')
plt.show()
```

c(i)

```
import ...
data_txt = np.loadtxt("dataset.txt", delimiter=',')
data_X = []
data_Y = []
for i in range(len(data_txt)):
    data_X.append([data_txt[i][0], data_txt[i][1], data_txt[i][0]*data_txt[i][0], data_txt[i][1]*data_txt[i][1]])
    data_Y.append(data_txt[i][2])

X_train, X_test, Y_train, Y_test = train_test_split(data_X, data_Y, test_size=0.3, random_state=40)


logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

acc = logreg.score(X_test, Y_test)

print('accuracy: ', acc)
print('slopes: ', logreg.coef_)
print('intercept: ', logreg.intercept_)
```

c(ii)

```
import ...
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
data_txt = np.loadtxt("dataset.txt", delimiter=',')
data_X = []
data_Y = []
for i in range(len(data_txt)):
    data_X.append([data_txt[i][0], data_txt[i][1], data_txt[i][0]*data_txt[i][0], data_txt[i][1]*data_txt[i][1]])
    data_Y.append(data_txt[i][2])

X_train, X_test, Y_train, Y_test = train_test_split(data_X, data_Y, test_size=0.3, random_state=40)


logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

ans = logreg.predict(X_test)
```

c(iii)

```
import ...

data_txt = np.loadtxt("dataset.txt", delimiter=',')
data_X = []
data_Y = []
for i in range(len(data_txt)):
    data_X.append([data_txt[i][0], data_txt[i][1], data_txt[i][0]*data_txt[i][0], data_txt[i][1]*data_txt[i][1]])
    data_Y.append(data_txt[i][2])

X_train, X_test, Y_train, Y_test = train_test_split(data_X, data_Y, test_size=0.3, random_state=40)


logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

baseline = DummyClassifier(strategy="most_frequent")
baseline.fit(X_train, Y_train)

acc_1 = logreg.score(X_test, Y_test)

acc_2 = baseline.score(X_test, Y_test)

print('logistic regression model accuracy of prediction: ', acc_1)
print('baseline model accuracy of prediction: ', acc_2)
```

c(iv)

```
import ...
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
data_txt = np.loadtxt("dataset.txt", delimiter=',')
data_X = []
data_Y = []
x = []
for i in range(len(data_txt)):
    data_X.append([data_txt[i][0], data_txt[i][1], data_txt[i][0]*data_txt[i][0], data_txt[i][1]*data_txt[i][1]])
    data_Y.append(data_txt[i][2])
    x.append(data_txt[i][0])

X_train, X_test, Y_train, Y_test = train_test_split(data_X, data_Y, test_size=0.3, random_state=40)


logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

ans = logreg.predict(X_test)

y = []
b = logreg.intercept_[0]
w = logreg.coef_[0]

for i in x:
    y.append((-w[1] + math.sqrt(w[1]*w[1] - 4*w[3]*(b+w[0]*i+w[2]*i*i))) / (2*w[3]))
```