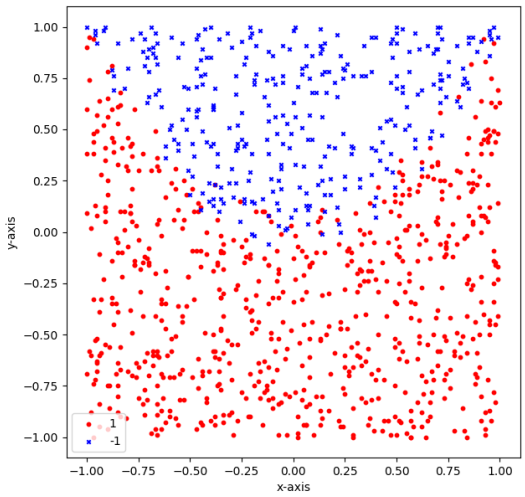
Xiangyu Zheng 21331025 CS7CS4 Machine Learning

(a)(i)

I use four arrays to record the x and y coordinates of -1 and 1:

```
px,py,nx,ny = [],[],[],[]
for d in dataArr[1:]:
    nums = d.split(',')
    if nums[2] == '1':
        px.append(float(nums[0]))
        py.append(float(nums[1]))
    else:
        nx.append(float(nums[0]))
        ny.append(float(nums[1]))
```

This is the distribution graph of all -1s and 1s:



(a)(ii)

My model uses the following parameters:

1. penalty='l2', To prevent over-fitting, the model features few, so choose L2
2. multi_class='ovr', Two classification
3. max_iter=1000, Algorithm convergence maximum number of iterations
4. fit_intercept=True, Add variance to the model
5. tol=0.0001, Convergence error range
6. coefficient of x:0.13313
7. coefficient of y:-4.50365
8. intercept:1.65408

   ....

   The coefficients of x and y are obtained through the cost function:

$$\frac{1}{m}\sum_{i=1}^{m}\log(1+e^{-y(i)\theta^T x^{(i)}})/\log(2)$$

According to the decision boundary formula

$$x = w_0 + w_1 x_1 + \ldots + w_n x_n$$

the larger the feature coefficient, The greater the impact on the forecast.

Features with too many or too few dimensions will lead to reduced predictions, because learning too many useless features will lead to overfitting, and too few dimensions will lead to underfitting. So features with moderate dimensions will increase predictions

(a)(iii)

We have the sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}}$$

We bring the feature values into the sigmoid function to get:

$$\hat{p} = \sigma\left(\theta^{\mathrm{T}} \cdot \mathrm{x_b}\right) = \frac{1}{1 + e^{-\theta^T \cdot \mathrm{x_b}}}$$

We can analyze $\theta^{\mathrm{T}} \cdot \mathrm{x_b}$'s value range:

$$\hat{y} = \begin{cases} 1, & \hat{p} \geq 0.5 \quad \theta^T \cdot x_b \geq 0 \\ -1, & \hat{p} < 0.5 \quad \theta^T \cdot x_b < 0 \end{cases}$$

There are two features in this sample: x1,x2, So we have decision boundaries:

$$\theta_0 + \theta_1 \cdot x_1 + \theta_z \cdot x_2 = 0$$

According to the above formula, we can get:

$$x_2 = \frac{-\theta_0 - \theta_1 \cdot x_1}{\theta_2}$$

We can first set the x coordinates of 100 decision boundaries:

```
x1_plot = np.linspace(-1, 1,100)
```

We can get the coefficient array and intercept through the trained model:

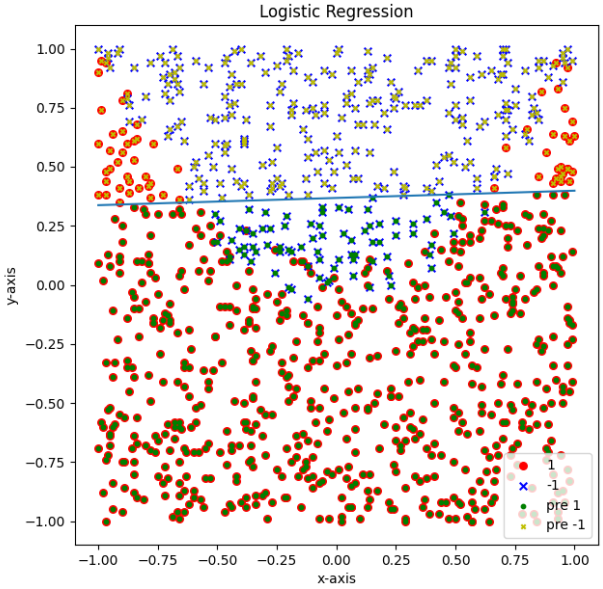
```
lr = LR(penalty='l2', max_iter=1000)
lr.coef_
lr.intercept_[0]
```

Putting the coefficient and intercept into the formula, we can calculate the y coordinate of the decision boundary:

```
x2_plot = (-lr.coef_[0][0] * x1_plot - lr.intercept_[0]) /
lr.coef_[0][1]
```

Next, we just need to draw it:

```
plt.plot(x1_plot, x2_plot)
```



(a)(iv)

As we can see from the graph in (a)(i), it is obvious that the dividing line between 1 and -1 is a curve, but the decision boundary of my model is a straight line, so when forecasting, the data from upper left corner , upper right corner and below the middle of the decision boundary are all predicted incorrectly. So I feel that my model classification effect is not good for these 'special data'.

(b) (i)

Three models were used for this task:

1.parameters:

(a) penalty='l2' Use l2 regularization
(b) loss='squared_hinge' loss function
(c) dual=True Turned into a dual problem
(d) C=0.001 Penalty coefficient
(e) fit_intercept=True Calculate the intercept
(f) max_iter=1000 The maximum number of iterations
(g) coefficient of x:0.01109
(h) coefficient of y:-0.38334
(i) intercept:0.23246

2.parameters:

(a) penalty='l2' Use l2
(b) loss='squared_hinge'
(c) dual=True
(d) C=1
(e) fit_intercept=True

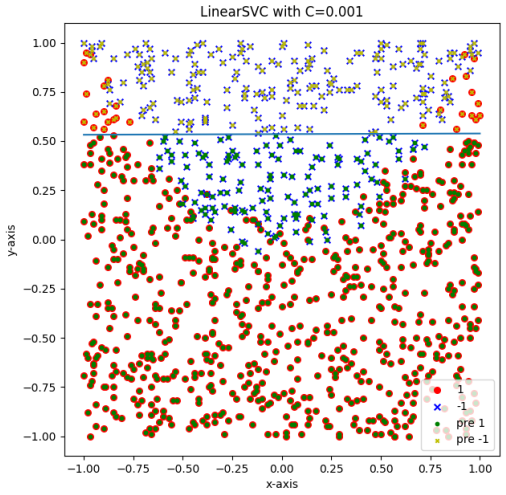
(f) max_iter=1000
(g) coefficient of x:0.05306
(h) coefficient of y:-1.69412
(i) intercept:0.60767

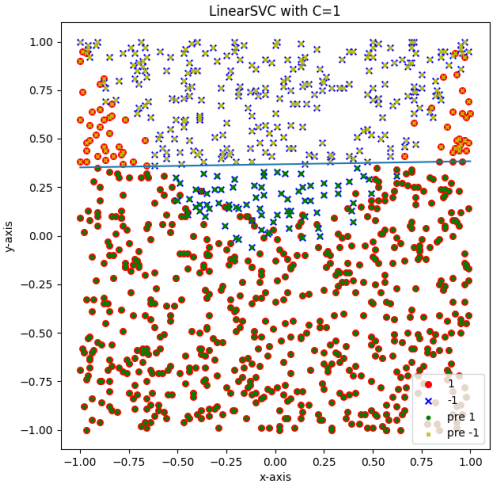
3.parameters:

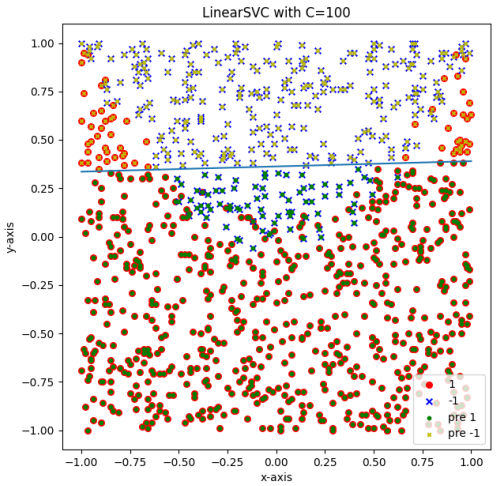
(a) penalty='l2'
(b) loss='squared_hinge'
(c) dual=True
(d) C=100
(e) fit_intercept=True Calculate the intercept
(f) max_iter=1000 The maximum number of iterations
(g) coefficient of x:0.02683
(h) coefficient of y:-1.83350
(i) intercept:0.58653

(b) (ii)

The following are three LinearSVC diagrams with different Cs, and the decision boundary. The calculation method of the decision boundary is the same as in (a) (iii)







(b) (iii)

According to the svm cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} max\left(0,1 - y^{(i)\theta^T}x^{(i)}\right) + \theta^T\theta/C$$

The bigger C is, the smaller the influence on θ, the smaller C is, the bigger the influence on θ's calculation.

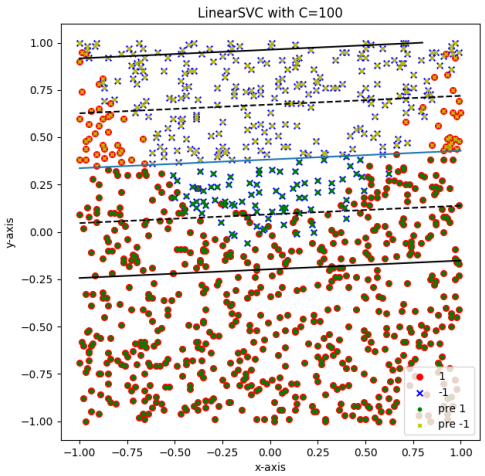
According to the hard-spaced margin line formula:

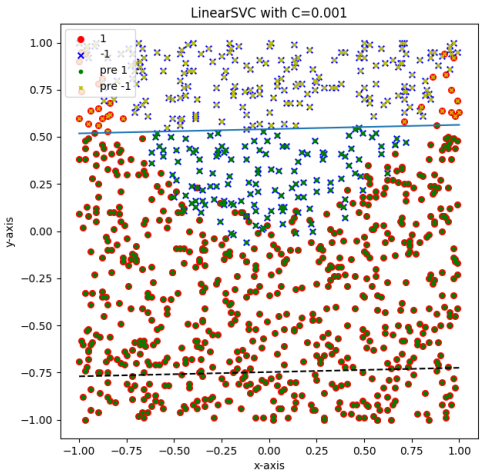
$$\begin{cases} w^T x^{(i)} + b \geq 1, & \forall y^{(i)} = 1 \\ w^T x^{(i)} + b \leq -1, & \forall y^{(i)} = -1 \end{cases}$$

And soft margin formula:

$$y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \zeta_i, \qquad \zeta \geq 0$$

Lines with soft and hard margin are drawn as follow when C=100 and C=0.001:





We can see that when C=0.001, the soft margin is much larger than when C=100, which shows that the model has a large tolerance for sample errors, which will reduce the accuracy of the model's prediction. When C is relatively large, it will cause over-fitting, but it is not reflected in the above figure.

(b) (iv)

Through the three linearSVC and logistic regression graphs, we can get that the coefficient of x and intercept in the logistic regression model are larger, and the proportion of x is more when it predicts. In the linearSVC model, as the value of C becomes larger, the coefficient of x and intercept become larger and larger, while the coefficient of y becomes smaller and smaller, the prediction accuracy of the model seems to be getting better and better.

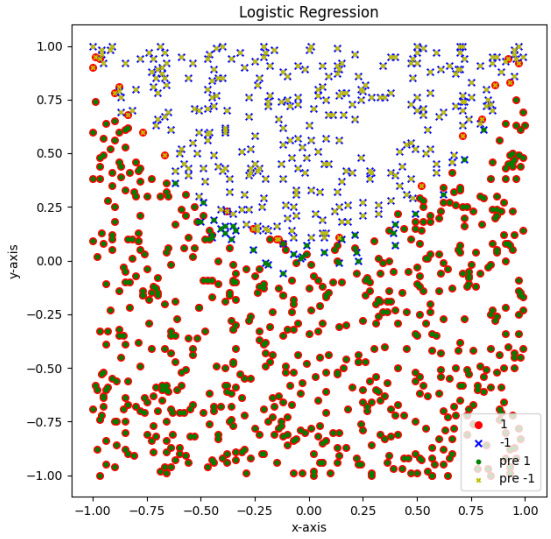
(c) (i)

The following are the parameters of the trained model:

1. penalty='l2', To prevent over-fitting, the model features few, so choose L2
2. multi_class='ovr', Two classification
3. max_iter=1000, Algorithm convergence maximum number of iterations
4. fit_intercept=True, Add variance to the model
5. tol=0.0001, Convergence error range
6. coefficient of x:-0.03959
7. coefficient of x:-6.11030
8. coefficient of x^2:5.57693
9. coefficient of y^2:-0.96831
10. intercept:0.45464

(ii)

The following is the point distribution diagram and the predicted distribution diagram after adding 2 features. Basically, the classification is successful, and the dividing line becomes a curve, which basically fits the data perfectly.



(iii)

I chose the logistic regression model in (a) for comparison.

First print out the classification reports of the two models:

```
print(classification_report(y_test, lr.predict(X_test)))
```

	precision	recall	f1-score	support
-1.0	0.81	0.71	0.76	86
1.0	0.86	0.91	0.88	164
accuracy			0.84	250

Logistic regression with 2 features

	precision	recall	f1-score	support
-1.0	0.96	0.90	0.93	81
1.0	0.95	0.98	0.97	169
accuracy			0.96	250

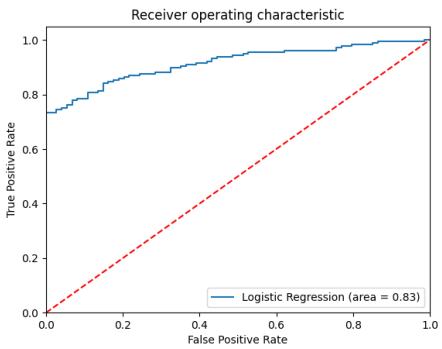
Logistic regression with 4 features

First we pass 4 parameters:

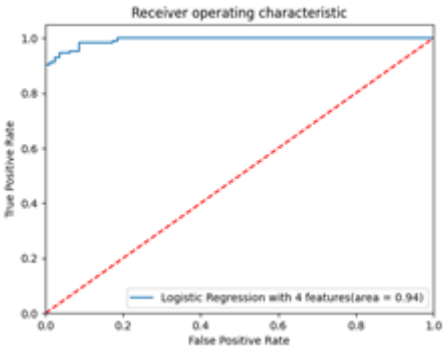
1. precision
   Indicates the actual number of positive samples among the samples predicted to be positive
   $$precision(P) = \frac{TP}{\mathrm{TP} + \mathrm{FP}}$$
2. recall
   Indicates the proportion of samples that are actually positive that are judged to be positive
   $$recall(R) = \frac{TP}{\mathrm{TP} + \mathrm{FN}}$$
3. fi-score
   Weighted harmonic average of Precision and Recall
   $$F1(R) = 2 \times \frac{P \times R}{\mathrm{P} + \mathrm{R}}$$
4. accuracy
   It refers to the ratio of the sample book correctly classified by the classifier to the total number of samples for a given test data set, that is, the probability of correct prediction.
   $$Accuracy = \frac{TP + FN}{\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN}}$$

   We can see that no matter which number it is, the performance of the logistic regression model with four feature is much better

   Next we look at the ROC diagram:



Logistic regression with 2 features



Logistic regression with 4 features

The area under the curve is also shown in the figure. The logistic regression model with 4 features is larger, indicating better performance.

(iv)

First we added 2 features, so our decision boundary formula is:

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 = 0 \quad (1)$$

Because x_3 and x_4 are the squares of x_1 and x_2, we have:

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 = 0 \quad (2)$$

Because we know the x coordinate, we need to solve for the y coordinate. The y coordinate is the value of x_2. We want the equation to have no square of x_2, so we first solve for the value of x_2^2:

$$\theta_4 x_2^2 = -\theta_0 - \theta_1 x_1 - \theta_2 x_2 - \theta_3 x_1^2 \quad (3)$$

We substitute formula 3 into formula 2 to get:

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + (-\theta_0 - \theta_1 x_1 - \theta_2 x_2 - \theta_3 x_1^2) = 0 \quad (4)$$

After simplifying the formula, we get:

$$\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 = 0 \quad (5)$$

What we are asking for is x_2, so there is:

$$\theta_2 x_2 = -\theta_1 x_1 - \theta_3 x_1^2 \quad (6)$$

Finally we have:

$$x_2 = \frac{-\theta_1 x_1 - \theta_3 x_1^2}{\theta_2} \quad (7)$$

Next we draw the decision boundary

First, we first generate 200 x coordinates, the interval is -1 to 1:

```
x1_plot = np.linspace(-1, 1, 200)
```

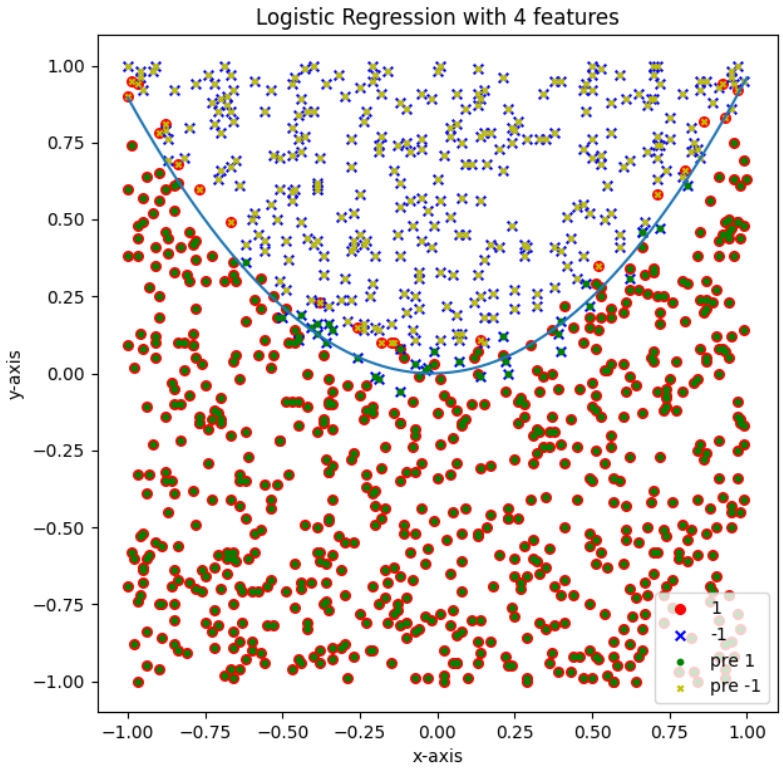
Then we calculate the coordinate array of x2 according to the formula:

```
x2_plot=[]
for i,v in enumerate(x1_plot):
```

```
    a = (-lr.coef_[0][0]*v-lr.coef_[0][2]*math.pow(v,2)) /
lr.coef_[0][1]
    x2_plot.append(a)
x2_plot = np.array(x2_plot)
```

Finally, we draw the decision boundary and get the picture::

```
plt.plot(x1_plot, x2_plot)
```



Appendix

code

(a)(i)

```
1.  import numpy as np
2.  import pandas as pd
3.  import matplotlib.pyplot as plt
4.
5.  # Read data
6.  data = open("data")
7.  dataArr = str(data.read()).split('\n')
8.
9.  #Divide the data into x, y arrays of 1 and xy arrays of -1 according to -
    1 and 1
10. px,py,nx,ny = [],[],[],[]
11. for d in dataArr[1:]:
```

```
    nums = d.split(',')
    if nums[2] == '1':
        px.append(float(nums[0]))
        py.append(float(nums[1]))
    else:
        nx.append(float(nums[0]))
        ny.append(float(nums[1]))

#Draw a picture
fig,square = plt.subplots(figsize=(7, 7))
square.scatter(px,py,s=10,c='r',marker ='o',label = '1')
square.scatter(nx,ny,s=10,c='b',marker ='x',label = '-1')
square.legend()
square.set_xlabel("x-axis")
square.set_ylabel("y-axis")
plt.show()
```

(a)(ii and iii)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression as LR
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, classification_report, roc_auc_score


# Read data
dataText = open("data")
dataArr = str(dataText.read()).split('\n')

# Divide the data into x, y arrays of 1 and xy arrays of -1 according to -1 and 1
px, py, nx, ny = [], [], [], []
data, target = [], []
for d in dataArr[1:]:
    nums = d.split(',')
    dxy = []
    dxy.append(float(nums[0]))
    dxy.append(float(nums[1]))
    data.append(dxy)
    target.append(float(nums[2]))
    if nums[2] == '1':
        px.append(float(nums[0]))
        py.append(float(nums[1]))
    else:
        nx.append(float(nums[0]))
        ny.append(float(nums[1]))

# Divide the data set into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(data, target)
# Training model
lr = LR(penalty='l2', max_iter=1000, tol=0.0001)
lr.fit(X_train, y_train)

# Prediction data set
y_pred = lr.predict(data)
y_test_pred = lr.predict(X_test)

# Print classification report
print(classification_report(y_test, y_test_pred))
fig, square = plt.subplots(figsize=(7, 7))

# Get accuracy and other data
y_prob = lr.predict_proba(X_test)[:, 1]
logit_roc_auc = roc_auc_score(y_test, lr.predict(X_test))
```

```
fpr_lr, tpr_lr, threshold_lr = roc_curve(y_test, y_prob)

# Draw the ROC curve graph
plt.figure()
plt.plot(fpr_lr, tpr_lr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()

# Draw a dot plot
y_pred = list(y_pred)
ppx, ppy, pnx, pny = [], [], [], []
for i, v in enumerate(data):
    if y_pred[i] == 1:
        ppx.append(v[0])
        ppy.append(v[1])
    else:
        pnx.append(v[0])
        pny.append(v[1])
square.scatter(px, py, s=30, c='r', marker='o', label='1')
square.scatter(nx, ny, s=30, c='b', marker='x', label='-1')
square.scatter(ppx, ppy, s=10, c='g', marker='o', label='pre 1')
square.scatter(pnx, pny, s=10, c='y', marker='x', label='pre -1')

# Draw the decision boundary
x1_plot = np.linspace(-1, 1, 100)
x2_plot = (-lr.coef_[0][0] * x1_plot - lr.intercept_[0]) / lr.coef_[0][1]
plt.plot(x1_plot, x2_plot)

square.legend()
plt.title("Logistic Regression")
square.set_xlabel("x-axis")
square.set_ylabel("y-axis")
plt.show()
```

(b)(i)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split

# LinearSVC related classes
# Read data
dataText = open("data")
dataArr = str(dataText.read()).split('\n')

# Divide the data into x, y arrays of 1 and xy arrays of -1 according to -1 and 1
px, py, nx, ny = [], [], [], []
data, target = [], []
for d in dataArr[1:]:
    nums = d.split(',')
    dxy = []
    dxy.append(float(nums[0]))
    dxy.append(float(nums[1]))
    data.append(dxy)
    target.append(float(nums[2]))
```

```
    if nums[2] == '1':
        px.append(float(nums[0]))
        py.append(float(nums[1]))
    else:
        nx.append(float(nums[0]))
        ny.append(float(nums[1]))

# Divide the data set into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(data, target)

# Training model
LinearSVC = LinearSVC(penalty='l2', loss='squared_hinge', dual=True, C=100,
fit_intercept=True, max_iter=1000)
LinearSVC.fit(X_train, y_train)

# Prediction data set
y_pred = LinearSVC.predict(data)
y_pred = list(y_pred)
y_test_pred = LinearSVC.predict(X_test)

# Print classification report
print(classification_report(y_test, y_test_pred))

# Draw a dot plot
ppx, ppy, pnx, pny = [], [], [], []
for i, v in enumerate(data):
    if y_pred[i] == 1:
        ppx.append(v[0])
        ppy.append(v[1])
    else:
        pnx.append(v[0])
        pny.append(v[1])

fig, square = plt.subplots(figsize=(7, 7))
square.scatter(px, py, s=30, c='r', marker='o', label='1')
square.scatter(nx, ny, s=30, c='b', marker='x', label='-1')
square.scatter(ppx, ppy, s=10, c='g', marker='o', label='pre 1')
square.scatter(pnx, pny, s=10, c='y', marker='x', label='pre -1')

# Draw the decision boundary
x1_plot = np.linspace(-1, 1, 200)
x2_plot = (-
LinearSVC.coef_[0][0] * x1_plot - LinearSVC.intercept_[0]) / LinearSVC.coef_
[0][1]
plt.plot(x1_plot, x2_plot)

# Draw the soft margin and hardmargin of LinearSVC
# w0*x0 + w1*x1 + b = 1
h_up_y = -
LinearSVC.coef_[0][0] / LinearSVC.coef_[0][1] * x1_plot - LinearSVC.intercep
t_[0] / LinearSVC.coef_[0][
    1] + 1 / LinearSVC.coef_[0][1]
# w0*x0 + w1*x1 + b = -1
h_down_y = -
LinearSVC.coef_[0][0] / LinearSVC.coef_[0][1] * x1_plot - LinearSVC.intercep
t_[0] / LinearSVC.coef_[0][
    1] - 1 / LinearSVC.coef_[0][1]
up_index = (h_up_y >= -1) & (h_up_y <= 1)
down_index = (h_down_y >= -1) & (h_down_y <= 1)
plt.plot(x1_plot[up_index], h_up_y[up_index], color='black')
plt.plot(x1_plot[down_index], h_down_y[down_index], color='black')
# w0*x0 + w1*x1 + b = 1-
h_up_y = -
LinearSVC.coef_[0][0] / LinearSVC.coef_[0][1] * x1_plot - LinearSVC.intercep
t_[0] / LinearSVC.coef_[0][
    1] + 0.5 / LinearSVC.coef_[0][1]
```

```
# w0*x0 + w1*x1 + b = -1
h_down_y = -
LinearSVC.coef_[0][0] / LinearSVC.coef_[0][1] * x1_plot - LinearSVC.intercep
t_[0] / LinearSVC.coef_[0][
    1] - 0.5 / LinearSVC.coef_[0][1]
up_index = (h_up_y >= -1) & (h_up_y <= 1)
down_index = (h_down_y >= -1) & (h_down_y <= 1)
plt.plot(x1_plot[up_index], h_up_y[up_index], color='black', linestyle='--
')
plt.plot(x1_plot[down_index], h_down_y[down_index], color='black', linestyle
='--')

square.legend()
plt.title("LinearSVC with C=0.001")
square.set_xlabel("x-axis")
square.set_ylabel("y-axis")
plt.show()
```

(c)

```
import math
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression as LR
from sklearn.metrics import classification_report, roc_auc_score, roc_curve

from sklearn.model_selection import train_test_split

# Logistic regression 4 features
# Read data
dataText = open("newdata.txt")
dataArr = str(dataText.read()).split('\n')

# Divide the data into x, y arrays of 1 and xy arrays of -1 according to -
1 and 1
px, py, nx, ny = [], [], [], []
data, target = [], []
for d in dataArr:
    nums = d.split(',')
    print(nums)
    dxy = []
    dxy.append(float(nums[0]))
    dxy.append(float(nums[1]))
    dxy.append(float(nums[2]))
    dxy.append(float(nums[3]))
    data.append(dxy)
    target.append(float(nums[4]))
    if nums[4] == '1':
        px.append(float(nums[0]))
        py.append(float(nums[1]))
    else:
        nx.append(float(nums[0]))
        ny.append(float(nums[1]))

# Divide the data set into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(data, target)

# Training model
lr = LR(penalty='l2', max_iter=1000, tol=0.0001)
lr.fit(X_train, y_train)

# Prediction data set
y_pred = lr.predict(data)

# Print classification report
```

```
print(classification_report(y_test, lr.predict(X_test)))
fig, square = plt.subplots(figsize=(7, 7))

# Get accuracy and other data
y_prob = lr.predict_proba(X_test)[:,1]
logit_roc_auc = roc_auc_score(y_test, lr.predict(X_test))
fpr_lr,tpr_lr,threshold_lr = roc_curve(y_test,y_prob)

# Draw the ROC curve graph
plt.figure()
plt.plot(fpr_lr, tpr_lr, label='Logistic Regression with 4 features(area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()

# Draw a dot plot
y_pred = list(y_pred)
ppx, ppy, pnx, pny = [], [], [], []
for i, v in enumerate(data):
    if y_pred[i] == 1:
        ppx.append(v[0])
        ppy.append(v[1])
    else:
        pnx.append(v[0])
        pny.append(v[1])
square.scatter(px, py, s=30, c='r', marker='o', label='1')
square.scatter(nx, ny, s=30, c='b', marker='x', label='-1')
square.scatter(ppx, ppy, s=10, c='g', marker='o', label='pre 1')
square.scatter(pnx, pny, s=10, c='y', marker='x', label='pre -1')

# Draw the decision boundary
x1_plot = np.linspace(-1, 1, 200)
x2_plot=[]
for i,v in enumerate(x1_plot):
    a = (-lr.coef_[0][0]*v-lr.coef_[0][2]*math.pow(v,2)) / lr.coef_[0][1]
    x2_plot.append(a)
x2_plot = np.array(x2_plot)
plt.plot(x1_plot, x2_plot)

square.legend()
plt.title("Logistic Regression with 4 features")
square.set_xlabel("x-axis")
square.set_ylabel("y-axis")
plt.show()
```