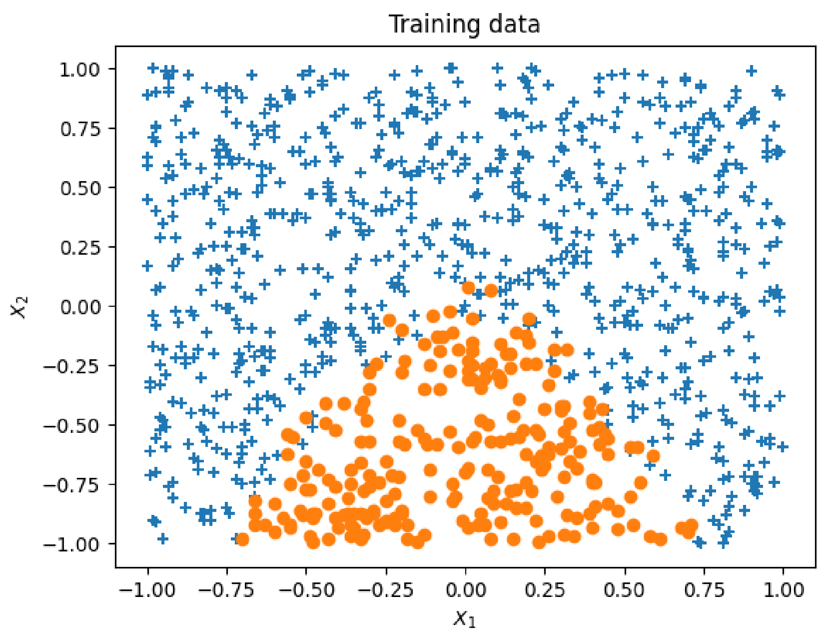
Assingment 2: Logistic Regression and SVMs

Ciarán Donegan

16 October 2020

Dataset # id:12-24-12

(a) (i)



Plot of training data with (+1) examples indicated with a '+' marker and (-1) examples indicated with a dot.

The above plot of the training data shows that x_1 and x_2 occupy the range of values from -1 to 1. Therefore, it won't be necessary to normalise the data.

(ii) Recall that a logisitc model is described as a linear combination of features x and parameters θ.

$$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

where

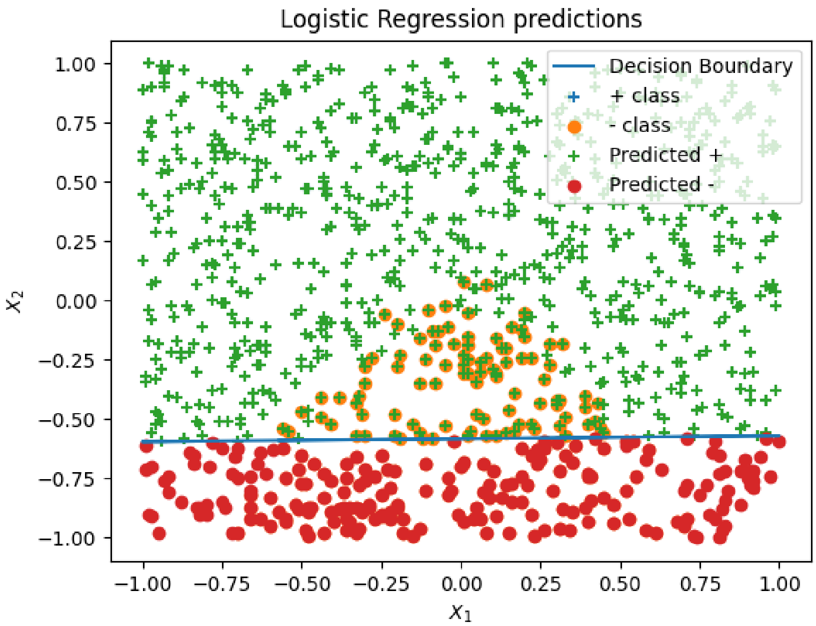
$$\theta_0 = 2.18988341$$

$$\theta_1 = -0.04710467$$

$$\theta_2 = 3.75961182$$

θ_2 (the parameter applied to the feature x_2) is much greater than θ_1. This tells us that the feature x_2 is more important is classification than x_1. This makes sense from the plot above, because you can draw a horizontal line through $x_2 = 0$ and already you can classify a lot of the data.

(iii)



Since this logistic regression model is using linear features, the decision boundary will be linear of the form $y = mx + c$.

The decision boundary is defined as

$$\theta^T x = 0$$

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$$

To get the equation of a line, we simply rearrange in terms of x_2:

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

Rewriting this in the form $y = mx + c$, the values m and c are found as follows:

$$m = -\frac{\theta_1}{\theta_2}$$

$$c = -\frac{\theta_0}{\theta_2}$$

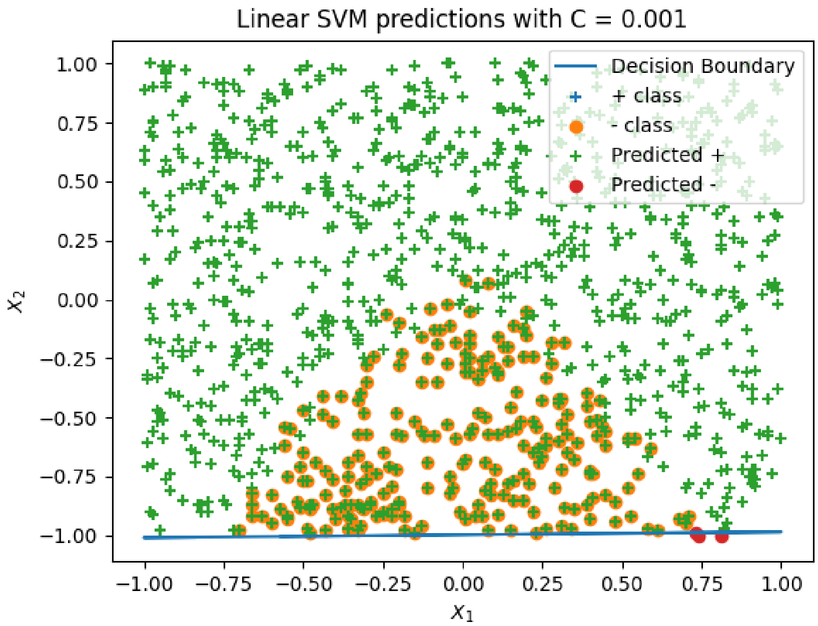
(iv) It is clear from the plot above that the logistic model is not fitting the data properly, and therefore many predictions are wrong. The data looks to be non-linear so using linear features (x_1 and x_2) will not be adequate to fit this data. Higher order features are needed.

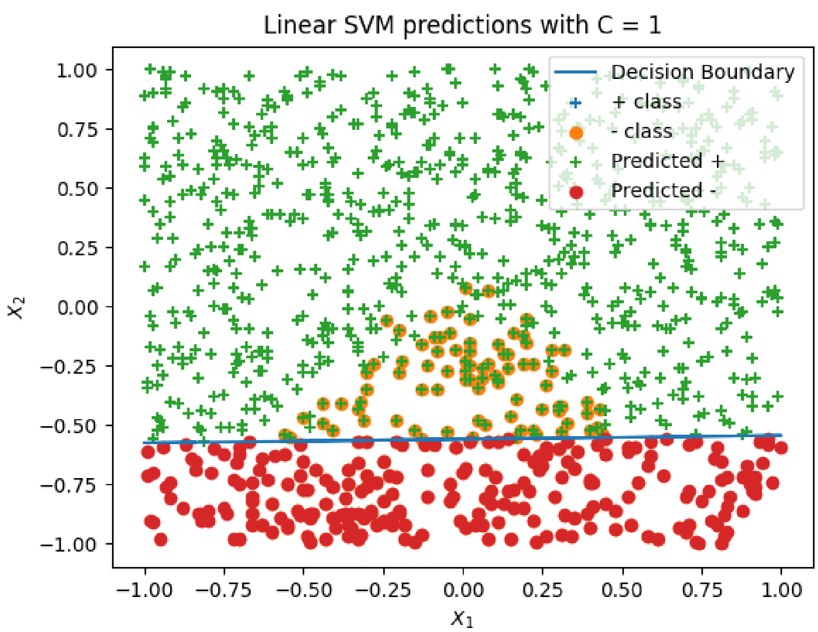
(b) (i) Parameters of learned linear SVM models.

C	θ_0	θ_1	θ_2
0.001	0.35117338	-0.0044659	0.35186694
1	0.77452955	0.77452955	1.38581541
10	0.7804777	-0.02229775	1.3980272
100	0.78108504	-0.02233454	1.39927435
1000	0.78114781	-0.02233518	1.3993989

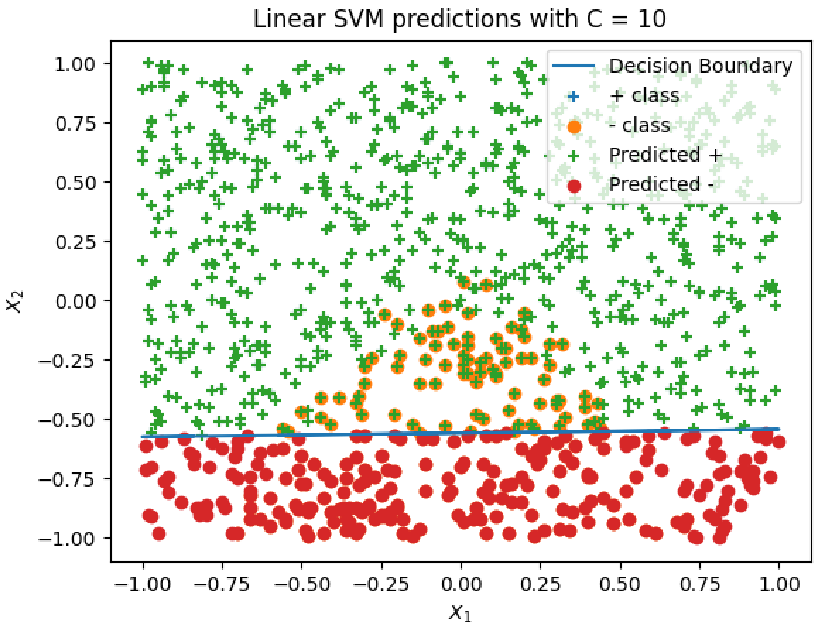
The above table shows that as C increases, the model parameters converge to the same values.

(ii)

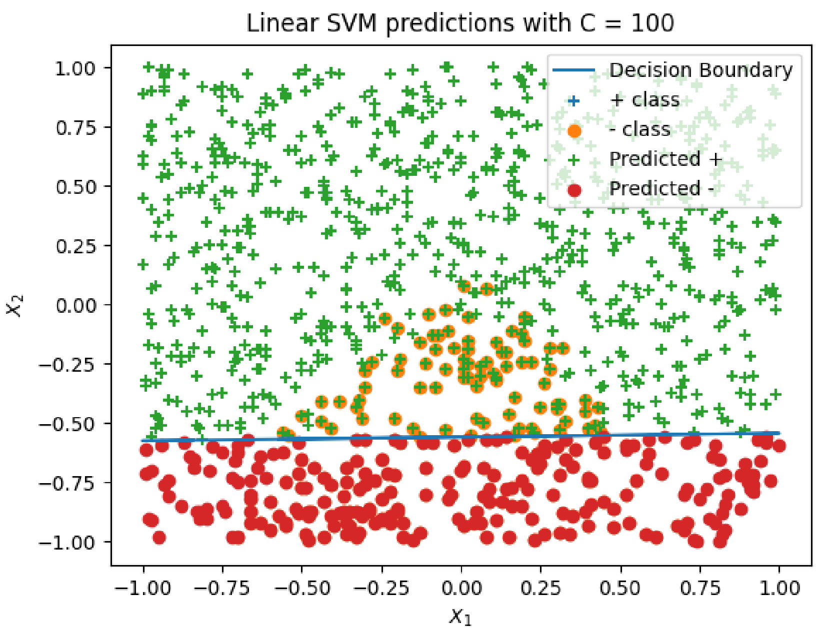




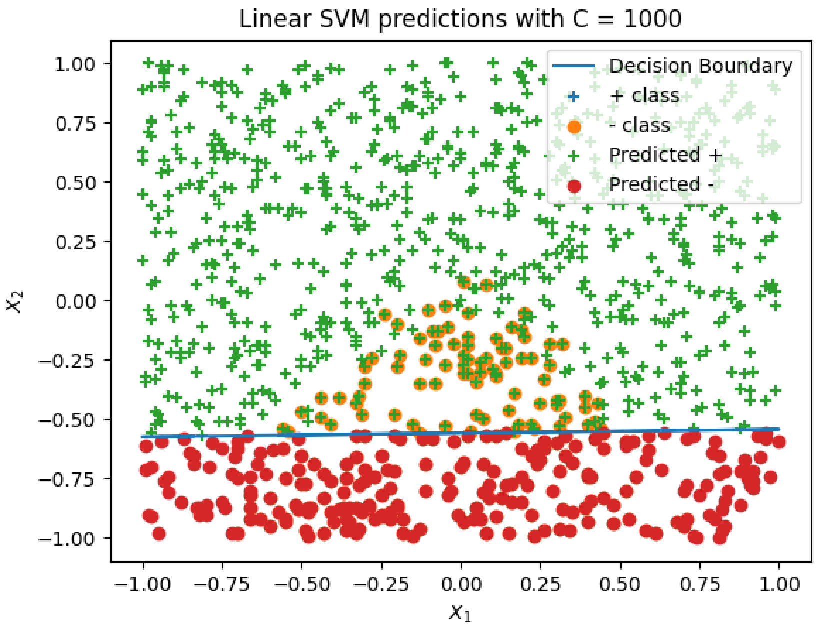
The SVM model with $C = 0.001$ is clearly a terrible fit for the data. As C increases, the fit is less terrible, but it is still not capturing the true distribution of the data. i.e. it is trying to fit a linear model to a higher order distribution.



Linear SVM predictions with C = 10
Decision Boundary
+ class
- class
Predicted +
Predicted -
x_1
x_2



Linear SVM predictions with C = 100
Decision Boundary
+ class
- class
Predicted +
Predicted -
x_1
x_2



(iii) Recall that the SVM cost function is

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

C is a regularisation parameter in which the regularisation strength is inversely proportional to C. This means that a large C value will apply low regularisation and may lead to overfitting of the data and vice versa.

From the figures above, we see that when C is too small ($C = 0.001$), the regularisation is very strong and leads to a large underfitting of the data.

As the value of C is increased, leading to less regularisation, the paramaters of the learned models remain steady as can be seen from the table in (b)(i).

Using C values of $\{1, 10, 100, 1000\}$ leads to almost no difference in the learned model paramters. Furthermore, the number of positive predictions and negative predictions are the same for these C values. This indicates that a C values of 1 would be the best in this case, because larger C values lead to longer convergence time.

(c) (i) Learned model :

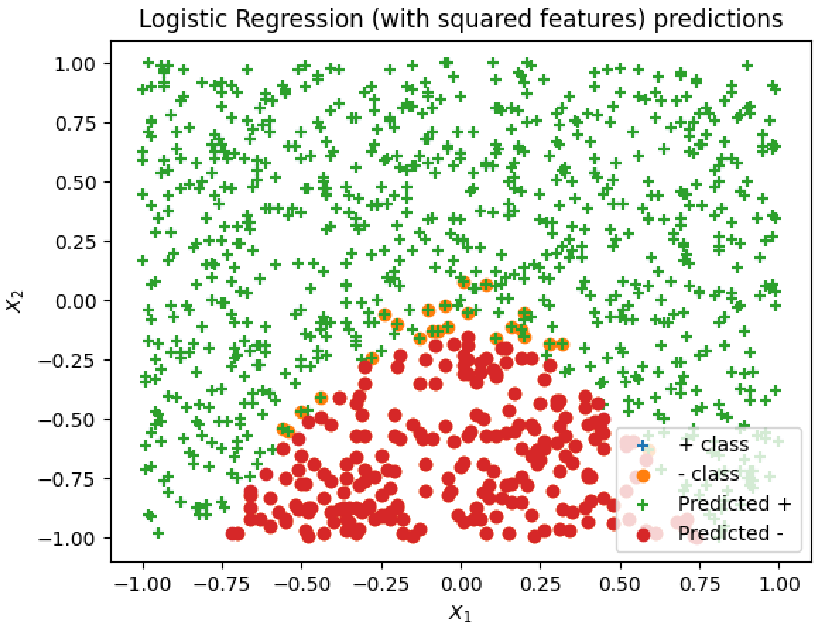
$$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2$$

where

$$\theta_0 = 0.78465272$$
$$\theta_1 = 0.014936467$$
$$\theta_2 = 5.38724334$$
$$\theta_3 = 8.04907867$$
$$\theta_4 = -0.16079957$$

These parameters make sense. The θ_2 and θ_2 values are high which tells us that the features x_2 and x_1^2 are very important in fitting a model to our data. Looking at the plot of the data in (a)(i) we see how you could very accurately determine the class given just the x_2 and x_1^2 features. It seems as though x_1 and x_2^2 add little information to our model and theses features could just as well be ignored.

(ii)



(iii) The most common class is the +ve class (763 vs 236).
I will use accuracy to compare my logistic model against a baseline that always predicts positive.

$$\texttt{Accuracy} = \frac{\texttt{Correct predictions}}{\texttt{total}}$$

$$\texttt{Accuracy}_{baseline} = \frac{763}{763 + 236} \approx 76.4\%$$

$$\texttt{Accuracy}_{logistic} = \frac{965}{999} \approx 96.9\%$$

The fact that my logisitc model is far outperforming the baseline predictor tells me that I haven't just wasted my time and that this model actually works.

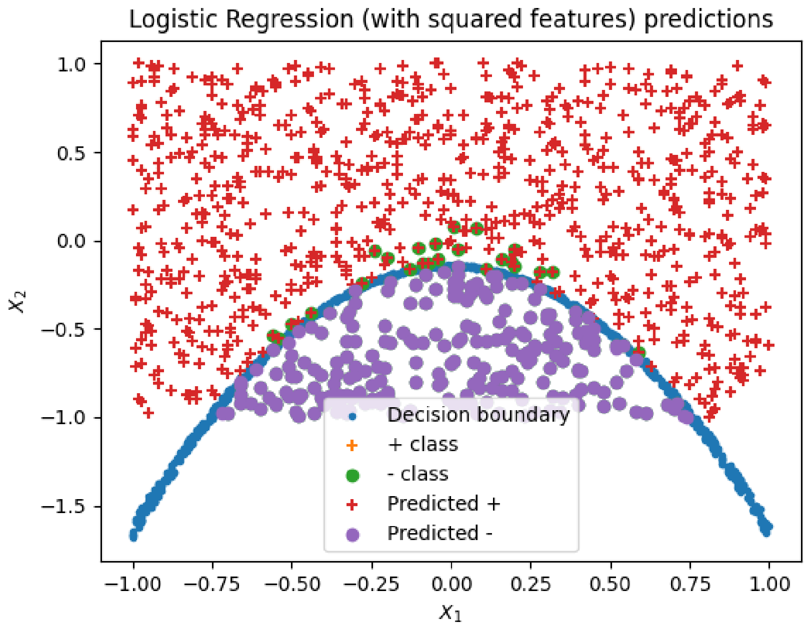
(iv) The logistic regression model is defined by:

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2$$

Rearranging this in terms of x_2 gives:

$$x_2 = \frac{-(\theta_0 + \theta_1 x_1 + \theta_3 x_1^2)}{\theta_2 + \theta_4 x_2}$$

Plotting this equation will give the decision boundary as shown below.



Logistic Regression (with squared features) predictions
Decision boundary
+ class
- class
Predicted +
Predicted -
x_1
x_2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.0
0.5
0.0
−0.5
−1.0
−1.5

A Source code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC

def read_csv(filename):
    df = pd.read_csv(filename, comment="#", header=None)
    X1 = np.array(df.iloc[:,0])
    X2 = np.array(df.iloc[:,1])
    X = np.column_stack((X1, X2))
    print(X)
    y = np.array(df.iloc[:,2])
    return X, y


def visualise_data(X, y):
    pos_classes = X[y == 1]
    neg_classes = X[y == -1]
    print("# true_pos ", len(pos_classes))
    print("# true_neg ", len(neg_classes))
    plt.figure(1)
    plt.scatter(pos_classes[:,0], pos_classes[:,1], marker="+")
    plt.scatter(neg_classes[:,0], neg_classes[:,1], marker="o")
    plt.xlabel("$X_1$")
    plt.ylabel("$X_2$")
    plt.title("Training data")
    plt.savefig("training_data.png")

def logistic_classifier(X, y):
    # Train a logistic classifier on data
    clf = LogisticRegression(random_state=0)
    clf.fit(X, y)
    print("classes = ", clf.classes_)
    print("intercept = ", clf.intercept_)
    print("coefficients = ", clf.coef_)

    # Use classifier to predict targets on training data
    preds = clf.predict(X)
    pred_pos = X[preds==1]
    pred_neg = X[preds==-1]
    print("# pred_pos ", len(pred_pos))
    print("# pred_neg ", len(pred_neg))

    # Retrieve the model parameters.
    b = clf.intercept_[0]
    w1, w2 = clf.coef_.T
    # Calculate the intercept and slope of the decision boundary.
    c = -b/w2
    m = -w1/w2

    # Plot predictions along with decision boundary
```

```
    decision_bnd = m * X[:,0] + c
    plt.figure(1)
    plt.plot(X[:,0], decision_bnd)
    plt.scatter(pred_pos[:,0], pred_pos[:,1], marker="+")
    plt.scatter(pred_neg[:,0], pred_neg[:,1], marker="o")
    plt.xlabel("$X_1$")
    plt.ylabel("$X_2$")
    plt.title("Logistic Regression predictions")
    plt.legend(["Decision Boundary", "+ class", "- class", "Predicted +", "Predicted -"])
    plt.savefig("logistic_train_and_predicted.png")


def linear_svm(X, y):

    for i, C in enumerate([0.001, 1, 10, 100, 1000]):
        print("C =" , C)
        clf = LinearSVC(random_state=0, C=C, max_iter=90000)
        clf.fit(X, y)

        print("intercept = ", clf.intercept_)
        print("coefficients = ", clf.coef_)

        # Retrieve the model parameters.
        b = clf.intercept_[0]
        w1, w2 = clf.coef_.T
        # Calculate the intercept and slope of the decision boundary.
        c = -b/w2
        m = -w1/w2

        # Use classifier to predict targets on training data
        preds = clf.predict(X)
        pred_pos = X[preds==1]
        pred_neg = X[preds==-1]
        print("# pred_pos ", len(pred_pos))
        print("# pred_neg ", len(pred_neg))

        # Plot training data
        pos_classes = X[y == 1]
        neg_classes = X[y == -1]
        plt.figure()
        plt.scatter(pos_classes[:,0], pos_classes[:,1], marker="+")
        plt.scatter(neg_classes[:,0], neg_classes[:,1], marker="o")
        # plot predictions
        plt.scatter(pred_pos[:,0], pred_pos[:,1], marker="+")
        plt.scatter(pred_neg[:,0], pred_neg[:,1], marker="o")
        # plot decision boundary
        decision_bnd = m * X[:,0] + c
        plt.plot(X[:,0], decision_bnd)
        plt.xlabel("$X_1$")
        plt.ylabel("$X_2$")
        plt.title("Linear SVM predictions with C = " + str(C))
        plt.legend(["Decision Boundary", "+ class", "- class", "Predicted +", "Predicted -
        plt.savefig("svm_train_and_predicted_c_" + str(C) + ".png")
```

```
def logistic_classifier_with_squares(X, y):
    # Train a logistic classifier on data
    clf = LogisticRegression(random_state=0)
    clf.fit(X, y)
    print("classes = ", clf.classes_)
    print("intercept = ", clf.intercept_)
    print("coefficients = ", clf.coef_)

    # Use classifier to predict targets on training data
    preds = clf.predict(X)
    pred_pos = X[preds==1]
    pred_neg = X[preds==-1]
    num_correct = np.sum(preds == y)
    print("num_correct", num_correct)
    print("# pred_pos ", len(pred_pos))
    print("# pred_neg ", len(pred_neg))

    # Retrieve the model parameters.
    bias = clf.intercept_[0]
    w1, w2, w3, w4 = clf.coef_.T

    # Plot predictions along with decision boundary
    decision_bnd = - (bias + w1*X[:,0] + w3 * (X[:,0] **2)) / (w2 + w4 * X[:,1] )
    plt.figure()
    plt.scatter(X[:,0], decision_bnd, marker=".")
    # Plot training data
    pos_classes = X[y == 1]
    neg_classes = X[y == -1]
    # plt.figure()
    plt.scatter(pos_classes[:,0], pos_classes[:,1], marker="+")
    plt.scatter(neg_classes[:,0], neg_classes[:,1], marker="o")
    # plot predictions
    plt.scatter(pred_pos[:,0], pred_pos[:,1], marker="+")
    plt.scatter(pred_neg[:,0], pred_neg[:,1], marker="o")
    plt.xlabel("$X_1$")
    plt.ylabel("$X_2$")
    plt.title("Logistic Regression (with squared features) predictions")
    plt.legend(["Decision boundary", "+ class", "- class", "Predicted +", "Predicted -"])
    plt.savefig("logistic_squares_train_predicted.png")


if __name__ == "__main__":

    X, y = read_csv("week2.csv")

    visualise_data(X, y)

    logistic_classifier(X,y)

    linear_svm(X, y)

    # add square features
    squares = X * X
```

```
X_new = np.hstack((X, squares))
logistic_classifier_with_squares(X_new, y)
```