CS7CS4/CSU44061 Machine Learning: Week 2 Assignment

Cian Higgins 22 October 2020

 # id:1--2-1

(a)(i) See data set ID above. The pandas code given with the assignment was used to read in the data. The X1 and X2 values were saved to a NumPy array, the y (target) values were saved in a corresponding vector. Matplotlib's scatter() function was used to plot the data. Points which have an actual target value of +1 were visualised as green '+' markers. Target values of -1 were displayed as red 'o' markers. The data seems to be in the shape of a quadratic curve.

```
plt.scatter(X1[y==1], X2[y==1], color='green', marker='+')
plt.scatter(X1[y==-1], X2[y==-1], color='red', marker='o')
```

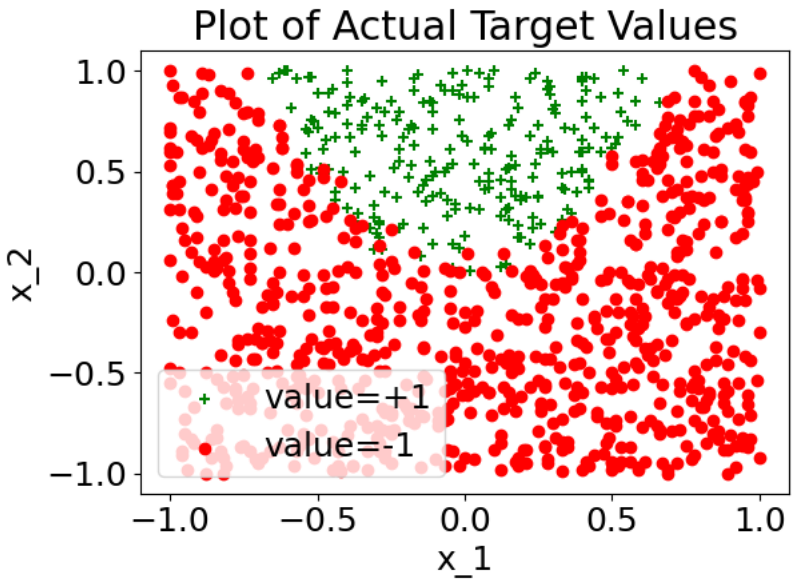


Figure 1

(a)(ii) A logistic regression classifier was trained on the data. The default solver (lbfgs) was used and there was no penalty applied. The input of this classifier was the X1 and X2 values contained in the array X. The actual target values passed to it were in the vector y.

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

The intercept and the coefficients (altogether: the parameters) of the model were printed to the console. They were as follows:

Theta 0: -2.20724516

Theta 1: -0.1239963

Theta 2: 3.84480387

Values correspond to the parameter values the classifier is training based on the model $sign(\theta^T x)$. The values can be used to calculate the decision boundary which in this case is a linear two-dimensional line. This is discussed further at (a)(iii).

(a)(iii) After using the fit() method to train the logistic regression model, it was then used to make predicted based off the training data passed to it. These predicted target values were then added to the same plot, using the same method as seen in the code for (a)(i). Different colours and markers were used to distinguish the data points from the training data.

```
pred_vals = log_reg.predict(X)
plt.scatter(X1[pred_vals==1],X2[pred_vals==1],color='c',marker='+')
plt.scatter(X1[pred_vals==-1],X2[pred_vals==-1],color='orange',marker='x')
```

The decision boundary of the classifier is given by setting the logistic regression model, $sign(\theta^T x)$, equal to zero. This equation defines a line in two dimensions, because we have two input features in our model, X1 and X2, and three parameters $\theta_0, \theta_1, \theta_2$.

We are then left with the equation for the decision boundary of a two-dimensional problem: $0 = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

This can be thought of as the equation for a line: $y = mx + c$

Rewriting it to suit the variables: $x_2 = mx_1 + c$

One can set $x_1 = 0$ which would leave $x_2 = c$ (the intercept). Substituting it into the original equation:

$$0 = \theta_0 + \theta_1(0) + \theta_2 x_2 \quad \rightarrow \quad 0 = \theta_0 + \theta_2 c \quad \rightarrow \quad c = -{}^{\theta_0}/_{\theta_2}$$

To obtain the slope, m, two points on the decision boundary are chosen, (x_1^a, x_2^a) and (x_1^b, x_2^b), such that $m = (x_2^b - x_2^a)/(x_1^b - x_1^a)$, 'rise over run'. Once again, using the model's equation:

$$0 = \theta_0 + \theta_1 x_1^b + \theta_2 x_2^b - (\theta_0 + \theta_1 x_1^a + \theta_2 x_2^a) \quad \rightarrow \quad -\theta_2(x_2^b - x_2^a) = \theta_1(x_1^b - x_1^a)$$

$$\therefore m = -{}^{\theta_1}/_{\theta_2}$$

With the intercept and slope of the line of the decision boundary calculated the final prediction equation is:

$$\hat{y} = -\frac{\theta_0 + \theta_1 x}{\theta_2}$$ *Note that x represents the array of the two features X1 and X2*

The decision boundary was calculated and plotted in Python as follows:

```
parameters = np.array(log_reg.coef_).flatten()
pred = -(log_reg.intercept_ + np.dot(parameters[0], X)) / parameters[1]
plt.plot(X, pred, color='black', linewidth=2)
```

Figure 2 shows the actual target values as they were plotted in figure 1 and the predicted target values. **The black line on the plot is the decision boundary** that was calculated with the equation above. It is clear from the plot that any point above the decision boundary was predicted to be +1 and any point below it predicted as -1.

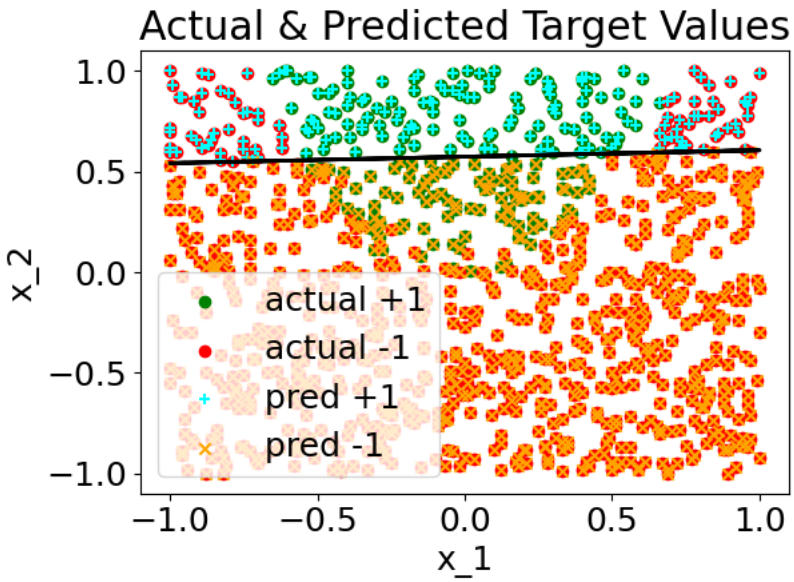


Figure 2

(a)(iv) The above plot shows that the actual training data and the predictions differ substantially. In the top left and right of the figure a number of false positives can be seen and in the centre under the decision boundary there is false negatives. This model has an accuracy of 81.58%. The accuracy is calculated like so:

$$accuracy = \frac{true\ positives + true\ negatives}{all\ samples}$$

True positive means correctly predicted as +1; true negative means correctly predicted as -1.

The sklearn method score() can also be used to quickly obtain the accuracy of a model:

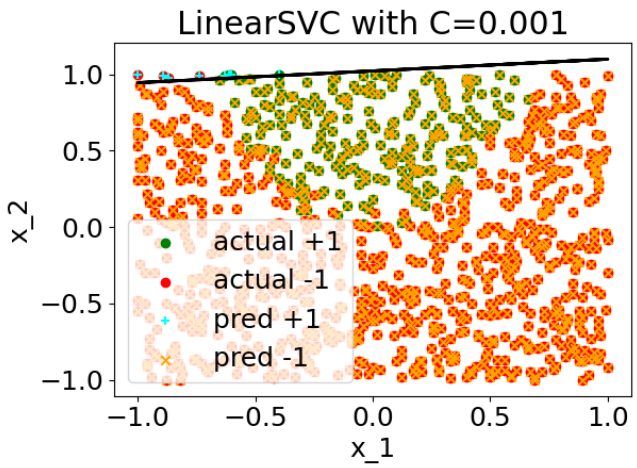
```
log_reg.score(X, y)
```

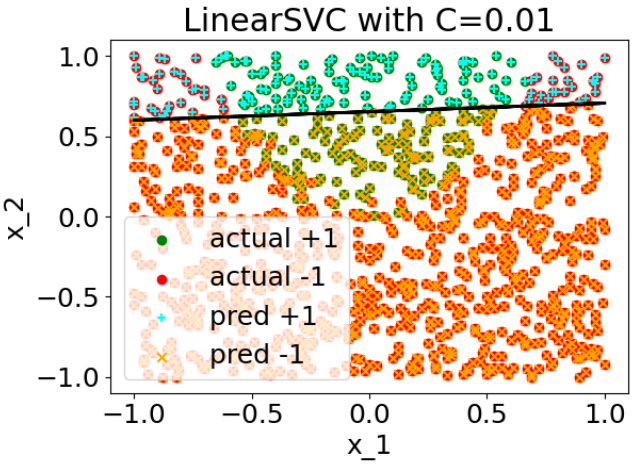
(b)(i) Code used to train linear SVM classifiers with a range of C penalty values:

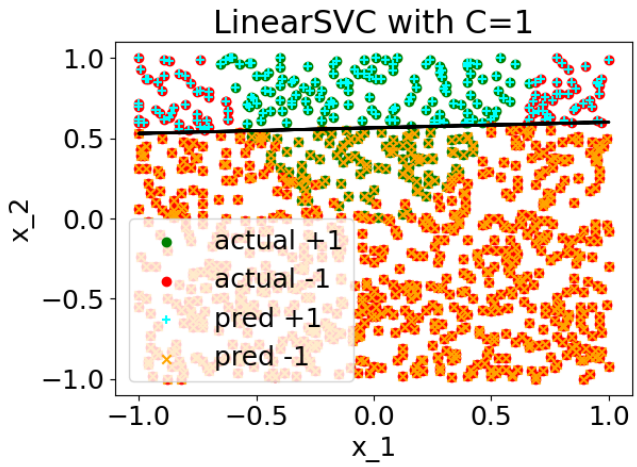
```
c_param = [0.001, 0.01, 0.1, 1, 100, 1000]
for c_val in c_param:
  lin_svc = LinearSVC(C=c_val)
  lin_svc.fit(X, y)
  lin_svc_pred = lin_svc.predict(X)
  parameters = np.array(lin_svc.coef_).flatten()
  pred = - (lin_svc.intercept_ + np.dot(parameters[0], X)) / parameters[1]
  score = lin_svc.score(X, y)
```

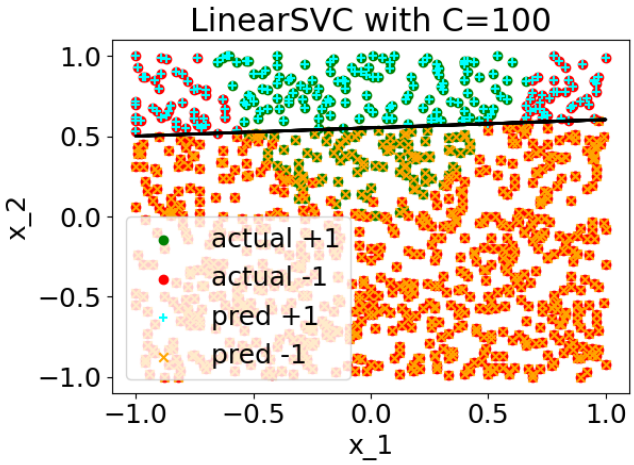
(b)(ii) On each iteration the target and predicted values were plotted, along with a decision boundary. See figures 3 – 7 below.

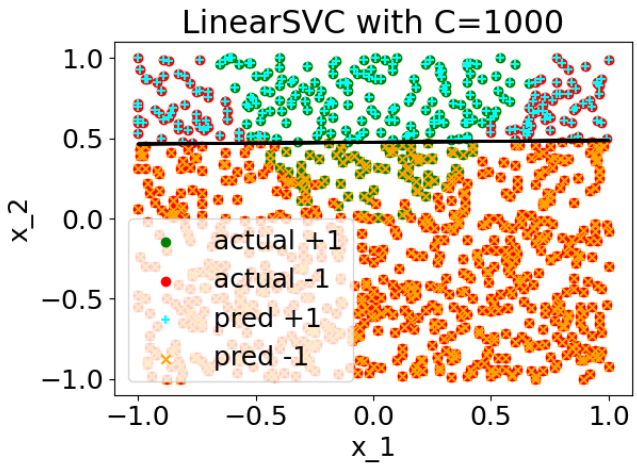
Figures 3 – 7











C=0.001
Accuracy: 76.577%
Theta0: -0.35036989
Theta1: -0.02647117
Theta2: 0.34255941

C=0.01
Accuracy: 81.381%
Theta0: -0.53400574
Theta1: -0.04345591
Theta2: 0.8168468

C=1
Accuracy: 81.181%
Theta0: -0.73315212
Theta1: -0.04566779
Theta2: 1.29605441

C=100
Accuracy: 81.582%
Theta0: -0.35036989
Theta1: -0.02647117
Theta2: 0.34255941
failed to converge

C=1000
Accuracy: 81.481%
Theta0: -0.87657791
Theta1: -0.02064025
Theta2: 1.8431962
failed to converge

(b)(iii) The C penalty appears in the cost function with the regularisation - $\theta^T\theta/C$.
A larger C reduces the effect of the regularisation because it is being divided. A small C makes more of an effect.
A small C causes a big margin (distance between support vectors or the decision hyperplane). This means training instances appearing on the wrong side of the margin are tolerated more. It includes a lot (or maybe all) of the data points and means the margin is calculated using all of the points which lie in this large area.
A large C causes the model to have a small hyperplane and will not tolerate training instances being on the wrong side of the margin as much. Only points close to the decision boundary will be considered (fewer support vectors).

(c)(i) Two new features are created by squaring X1 and X2. All four features put into a new vector, new_X:

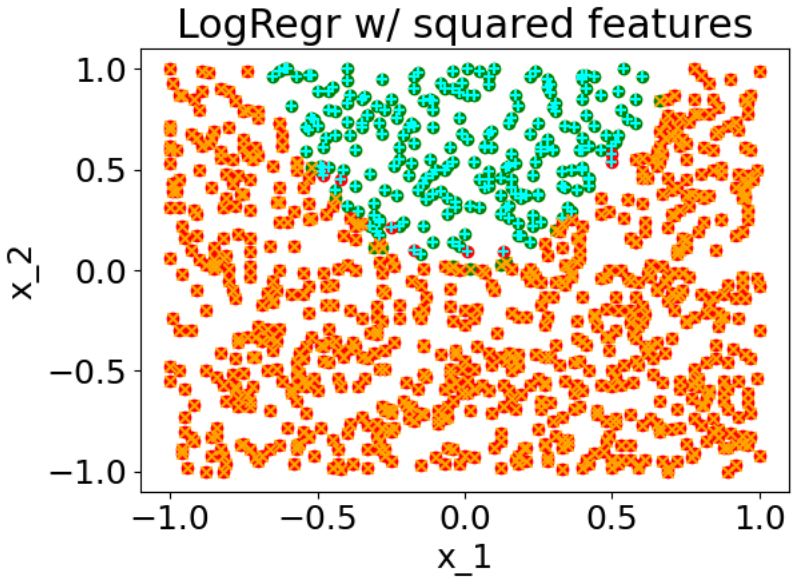
```
X3 = np.square(X1)
X4 = np.square(X2)
new_X = np.column_stack((X1, X2, X3, X4))

new_log_reg = LogisticRegression(penalty='none', solver='lbfgs')
new_log_reg.fit(new_X, y)
new_pred = new_log_reg.predict(new_X)
accuracy = new_log_reg.score(new_X, y)
```

The logistic regression classifier has an accuracy of 97.998%. Accuracy was calculated according the formula shown at (a)(iv).

Theta0: -0.62194964
Theta1: -0.6417546
Theta2: 28.08701565
Theta3: -50.58763097
Theta4: -2.53920756

The model has the intercept value (theta0) and four parameter values. Two for the original features and two for the new square features.
The values correspond to the parameter values that are trained by the model and are also related to the quadratic formula $y = ax^2 + bx + c$.
The model in the case of this logistic classifier is: $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

(c)(ii) The plot shows that this model has only a few incorrectly predicted data points. This model is far more accurate than the one with only two features. By using feature engineering to add two new features of their squares, the model is a much better fit. This is because the data looks to be quadratic, with a small amount of noise at the bottom of the parabola.



(c)(iii) The sklearn dummy classifier was used to compare the new model to a baseline which consists of only the most frequent prediction. The code used:

```
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(new_X, y)
dummy_clf.predict(new_X)
```

The accuracy of the dummy classifier was 76.577% compared to the new classifier's accuracy of 97.998%. It is clear that by engineering two new features which are the squares of the original X1 and X2 the model has improved significantly.

Appendix

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

df = pd.read_csv("week2.csv", comment='#', header=None)
print(df.head())
X1 = df.iloc[:, 0]
X2 = df.iloc[:, 1]
X = np.column_stack((X1, X2))
y = np.array(df.iloc[:, 2])

# a(i) visualise:
plt.rc('font', size=18)
plt.scatter(X1[y==1], X2[y==1], color='green', marker='o')
plt.scatter(X1[y==-1], X2[y==-1], color='red', marker='o')
plt.title("Actual & Predicted Target Values")
plt.xlabel("x_1")
plt.ylabel("x_2")
# plt.show()

# a(ii) sklearn training:
log_reg = LogisticRegression(penalty='none', solver='lbfgs')
log_reg.fit(X, y)
pred_values = log_reg.predict(X)

plt.scatter(X1[pred_values==1], X2[pred_values==1], color='cyan', marker='+')
plt.scatter(X1[pred_values==-1], X2[pred_values==-1], color='orange', marker='x')
plt.legend(["actual +1", "actual -1", "pred +1", "pred -1"], loc='lower left')
print(f"Intercept={log_reg.intercept_},Coefs={log_reg.coef_}")
print("LogReg Score:", log_reg.score(X, y))

# a(iii) plotting predictions and decision boundary
parameters = np.array(log_reg.coef_).flatten()
print(parameters)
pred = -(log_reg.intercept_ + np.dot(parameters[0], X)) / parameters[1]

plt.plot(X, pred, color='black', linewidth=2)
plt.show()

# b(i) and b(ii) varying C param and plotting each iter
print('\nLinearSVC:')
c_param = [0.001, 0.01, 1, 100, 1000]

for c_val in c_param:
  lin_svc = LinearSVC(C=c_val)
  lin_svc.fit(X, y)
  lin_svc_pred = lin_svc.predict(X)
  parameters = np.array(lin_svc.coef_).flatten()
  pred = - (lin_svc.intercept_ + np.dot(parameters[0], X)) / parameters[1]
  score = lin_svc.score(X, y)
  print(f"C={c_val},Accu={score},Intercept={lin_svc.intercept_},Coefs={lin_svc.coe
f_}")

  plt.scatter(X1[y==1], X2[y==1], color='green', marker='o')
```

```
  plt.scatter(X1[y==-1], X2[y==-1], color='red', marker='o')
  plt.scatter(X1[lin_svc_pred == 1], X2[lin_svc_pred == 1], color='cyan',
marker='+')
  plt.scatter(X1[lin_svc_pred == -1], X2[lin_svc_pred == -1], color='orange',
marker='x')
  plt.legend(["actual +1", "actual -1", "pred +1", "pred -1"], loc='lower left')
  plt.plot(X, pred, color='black', linewidth=2)
  plt.title(f"LinearSVC with C={c_val}")
  plt.xlabel("x_1")
  plt.ylabel("x_2")
  plt.show()

# c(i) feature engineering
X3 = np.square(X1)
X4 = np.square(X2)
new_X = np.column_stack((X1, X2, X3, X4))

# c(ii)
new_log_reg = LogisticRegression(penalty='none', solver='lbfgs')
new_log_reg.fit(new_X, y)
new_pred = new_log_reg.predict(new_X)
accuracy = new_log_reg.score(new_X, y)
print("Score:", accuracy)

# plotting actual values
plt.scatter(X1[y==1], X2[y==1], color='green', marker='o')
plt.scatter(X1[y==-1], X2[y==-1], color='red', marker='o')
# plotting predicted values
plt.scatter(X1[new_pred==1], X2[new_pred==1], color='cyan', marker='+')
plt.scatter(X1[new_pred==-1], X2[new_pred==-1], color='orange', marker='x')
plt.title("LogRegr w/ squared features")
plt.xlabel("x_1")
plt.ylabel("x_2")
plt.show()
print(new_log_reg.intercept_, new_log_reg.coef_)



# c(iii)
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(new_X, y)
dummy_clf.predict(new_X)
print("Score dummy:", dummy_clf.score(new_X, y))

# c(iv)
coefs = np.array(new_log_reg.coef_).flatten()
# y = ax^2 + bx + c
```