2 Logistic Regression

Task 2.1

For each target data set we used a different design matrix, which consists of two already given features and one or more additional features we created based on the given features and the target data set.

Data set A:

The matrix for A consists of the two features in X-1-data.npy as well as two additional binary features. When x_1 is greater or equal 10 the corresponding value in the additional x1-feature is 1, else it is 0. And when x_2 is smaller or equal 20, the corresponding value in the additional x2-feature is 1, else it is 0.

```
x1_feature = np.select([X_data[:,0] >= 10], [1], default=0)
x2_feature = np.select([X_data[:,1] <= 20], [1], default=0)
```

Data set B:

The matrix for B, again, consists of the two features in X-1-data.npy and one additional feature. Due to the decision boundary having the shape of a quarter circle (with its origin at 0,0) we chose for our additional feature to represent the distance from each point to the origin. The points inside the circle have a distance to the origin less than a certain value while the points outside the circle/ on the other side of the decision boundary will have a distance greater than this value - so with this added feature the classifier is able to separate points into classes based on their position relative to the decision boundary.

```
final_feature = np.square(X_data[:,0]) + np.square(X_data[:,1])
```

Data set C:

The matrix for C consists of the two features in X-2-data.npy as well as one additional feature. The decision boundary between the classes in the target data set resembles a 5th order polynomial. We did some curve fitting with a number of coordinates of the points on/near the decision boundary and came up with this function: $y = 0.16x^5 + 0.23x^4 - 0.5x^3 - 0.68x^2 + 0.25x - 0.52$

Based on whether a point is located above or below this curve we fill our additional feature with 0 or 1 respectively.

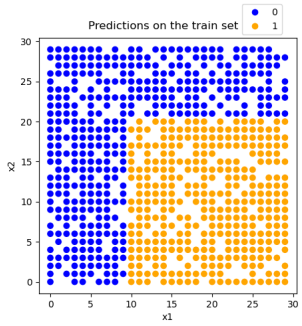
```
final_feature = np.where((X_data[:,1] < 0.16 * X_data[:,0]**5 +
0.23 * X_data[:,0]**4 - 0.5 * X_data[:,0]**3 - 0.68 * X_data[:,0]**2 +
0.25 * X_data[:,0] - 0.52), 1, 0)
```

Task 2.3

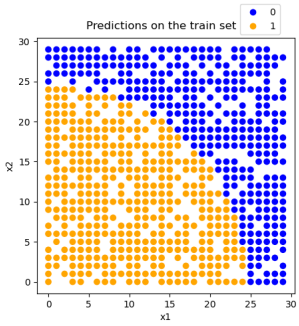
```
---- Logistic regression task 1 ----
Train accuracy: 100.00. Test accuracy: 100.00.
Train loss: 0.0000. Test loss: 0.0000.
---- Logistic regression task 2 ----
Train accuracy: 100.00. Test accuracy: 100.00.
Train loss: 0.0000. Test loss: 0.0000.
---- Logistic regression task 3 ----
Train accuracy: 92.50. Test accuracy: 93.75.
Train loss: 0.2339. Test loss: 0.1941.
```

Here you can see accuracy and loss on the train and test set using logistical regression with the lbfgs solver and the 'none' penalty. The lbfgs solver we used only supports the penalties 'l2' (L2 regularization) and 'none' (standard logistic regression without any regularization). For our use case the 'none' gave us slightly better results (0.5% better train accuracy for the data set C)

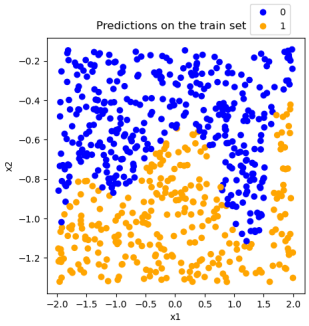
Task 2.4



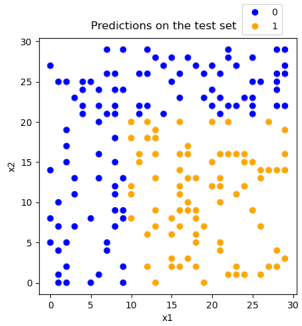
(a) data set A



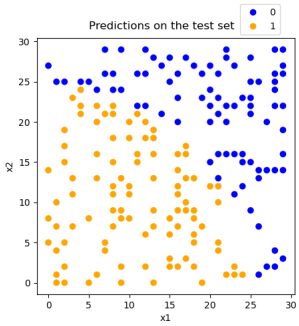
(b) data set B



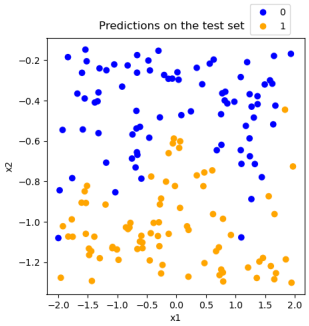
(c) data set C



(a) data set A



(b) data set B



(c) data set C

Task 2.5

θ^* vector and bias term

```
---- Logistic regression task 1 ----
Theta vector:  [[2.91237699e+01 5.74419965e-04 7.09532519e+02]]
Bias term:  [-986.56500081]
---- Logistic regression task 2 ----
Theta vector:  [[ 3.3346542   3.23125485 -3.9818712 ]]
Bias term:  [2280.14367503]
---- Logistic regression task 3 ----
Theta vector:  [[ 0.45785328 -5.42397815  3.69086177]]
Bias term:  [-6.6207794]
```

Task 2.6

When do we use logistic regression

Logistic regression is an example of supervised learning. It can be used for binary classification problems, such as is the case four our three data sets. We are using it when we want to model the relationship between the features in a dataset and the probability of a data point belonging to one of two classes.

```
yhat_train = clf.predict(X_train)
yhat_test = clf.predict(X_test)
```

Here the predict() method of the classifier/ logistic regression model is used to predict the class labels of the test set based on the previously calculated probabilities.

Task 2.7

A classifier could predict everything correctly and achieve 100% accuracy, but it can happen that the loss is not zero - Why?

The loss function measures the difference between the predicted probabilities and the true labels. It is designed to penalize incorrect predictions as well as uncertainty in the predictions. If the predicted probabilities aren't exactly equal to 0 or 1, this will result in a loss regardless of all the sample data being classified correctly. If the predicted probabilities are very close to 0 or 1 (like 0.001 and 0.999), the loss will be quite small but not non zero.