CS 506 Spring 2021 - HW2

Classification and Dimensionality Reduction

Total: 32 points

Package Limitations: None.

1 Least Squares and Logistic Regression

In this section, we are going to compare two linear models for classification, especially when outliers are presented in the data. Please refer to Section 4.1.3 at Page 184 of the book Pattern Recognition and Machine Learning for the context of this problem as well as the explanation of Figure1

Figure 1: The comparison of least square and logistic regression in classification of two classes with/without outliers

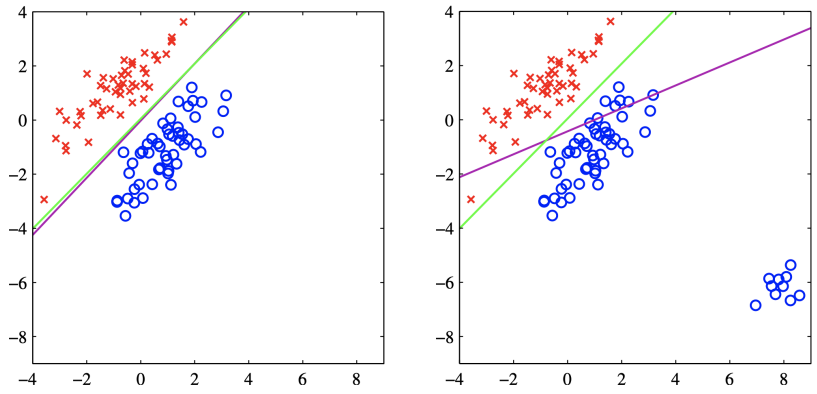

Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

a) [1 pt.] Generate labeled random 2D points like the ones shown in the left subfigure of Figure1. Note that the red crosses and blue circles are points of different classes, so you may need to have a third column storing the label info of the 2D points. Let's call this data "data_without_outlier".

Now on top of this data, add a few outliers to the blue circles just like the right subfigure of Figure1 and save the data as "data_with_outlier". You may either use code or even manually choose some random 2D points. **Your data need not to be exactly the same as the ones shown on the plots.**

b) [4 pts.] Use both the least squares method and the logistic regression method to classify the "data_without_outlier" and "data_with_outlier".

c) [2 pts.] Plot the classification results into two figures side by side just like Figure1. Have you got similar results like Figure1? Explain briefly why the logistic regression is not sensitive to outliers.

2 Logistic Regression and kNN Classification

The goal of this problem is to perform classification on the famous **MNIST** dataset.
We have already preprocessed a sample of this dataset (30% of the original dataset), that you can find here: Download from Google Drive in the format of NumPy arrays.
File *mnist_data.npy* contains an array of the data -each row corresponds to a 28×28 digit picture vectorized to create 28*28=784 features, while the file *mnist_labels.npy* contains the respective labels of the images.

a) [0 pts.] Randomly **split the dataset**, using 20% of the samples as your test set and the remaining 80% as the train set that you will use to fit your models.

b) [2 pts.] Try to classify the images using **Logistic Regression**. Have in mind that the dataset contains more than 2 labels, hence is a **multinomial classification** problem. What is your **train accuracy and test accuracy**?

c) [3 pts.] Now, try to classify the dataset using a **k-Nearest Neighbor classifier**. **Plot the train and test accuracy** as you **vary** k from 1 to 25 with a step size of 2.

d) [1 pt.] Explain your results.

e) [5 pts.] Now use kNN to explore how a different sized train set affects your results. Plot the accuracy of your model when only using 3,000 of the images in the train set (repeat this experiment using 6,000, 9,000, and so on until you are using the full train set). Use whatever value of k you found that worked best in part (c). You will be doing something similar in Problem 3(d) so it makes sense to run both experiments at the same time.

f) [2 pts.] Give a few bullet points explaining the pros and cons of these algorithms and when and why we use logistic regression over linear regression.

3 PCA - Dimensionality Reduction

The original dataset contains $28 * 28 = 784$ features. Therefore, we will try to reduce the dimension by using PCA.

a) [1 pt.] Perform PCA decomposition, initially using **all principal components**. Before performing PCA you usually need to **mean-center** the data, see **here** why, which means you have to calculate the mean of each variable (column) and subtract it from the respective column. However, many libraries perform this step implicitly, so consult the documentation of the library you are going to use (e.g. PCA of Sklearn).

b) [2 pts.] Plot the **CDF of the explained variance** as a function of the number of principal components.

c) [1 pt.] **Choose a number of principal components** to use by arguing why your choice is reasonable as a trade-off between the number of components used and classification performance. Afterwards, **train a kNN classifier** (choose a k that gave you the best results in Problem 2c.) and report **train and test accuracy**.

d) [5 pts.] For this part we will perform the following experiment: **First**, randomly **sample a part of your dataset** (using a fixed k and all features), of size ranging from 3,000 to 21,000 (the whole dataset) in increments of 3,000. **Fit a kNN classifier and plot the running time.**
Now, use a fixed k and all samples of your dataset, but **fit a kNN classifier using a varying number of Principal Components**, ranging from 50 to 750 in increments of 100. **Plot the running time on the same plot as above**.
Describe the plot. What seems to affect -as a trend- the fitting time more? Number of samples used for trying, or the dimensions of the data?

-Python's time package may be useful for this problem.

e) [3 pts.] Using your results from the previous question, produce the most accurate model you can on the condition that it is faster than 50% of the models you tested. List the values chosen for: k, number of samples, and number of principal components. How does this model compare to your most accurate model (the best model you created when time wasn't a factor)?

f) [1 pt.] **Bonus point**: Plot the images of the 10 first Principal Components. That is, plot the image approximation using each principle component.