CS 725 : Machine Learning Project
Accelerometer Biometric Competition on Kaggle.com

Aman Madaan 133050004
Ashish Mittal 133050005
Harshit Pande 133050040
Lekha 133050002

November 12, 2013

Contents

Chapter 1

Introduction

1.0.1 Plots

To begin the project we plotted the X, Y, Z components of acceleration of randomly picked 6 class labels using and Octave script to understand if we can observe some pattern visually. The plots are embedded following this section in the report. Although the plots are only for 6 class labels but it gave us an idea that data points from a particular class tends to form clusters. This forms the basis for the distance based KNN approach that gave us the best accuracy.

The octave script is attached and can be run by reading README.

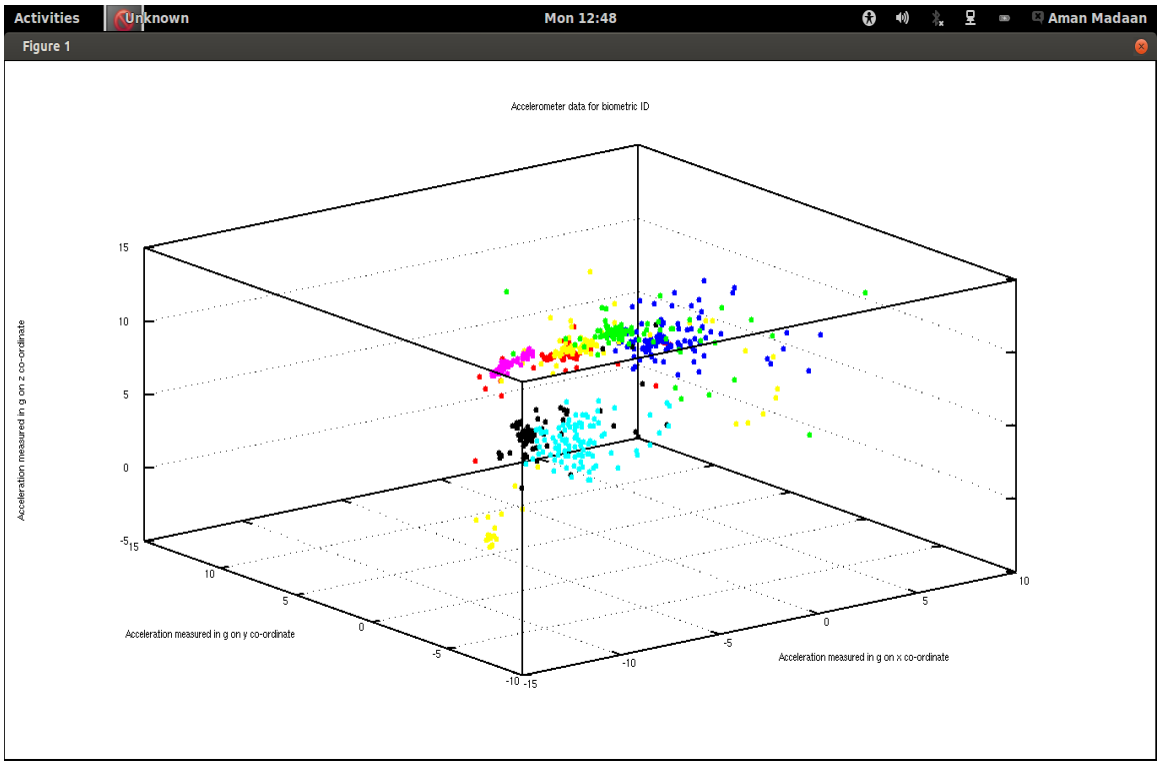

Figure 1.1: Sampled accelerometer data for 6 devices.

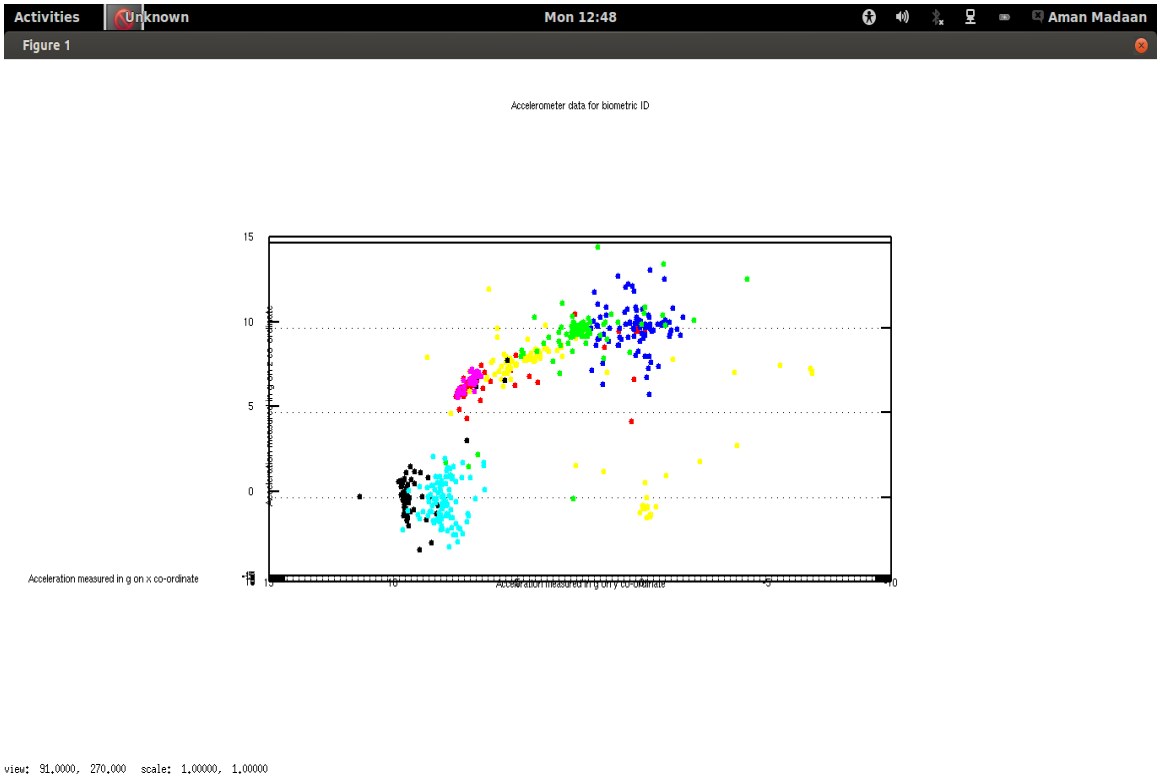

Figure 1.2: Sampled accelerometer data for 6 devices.

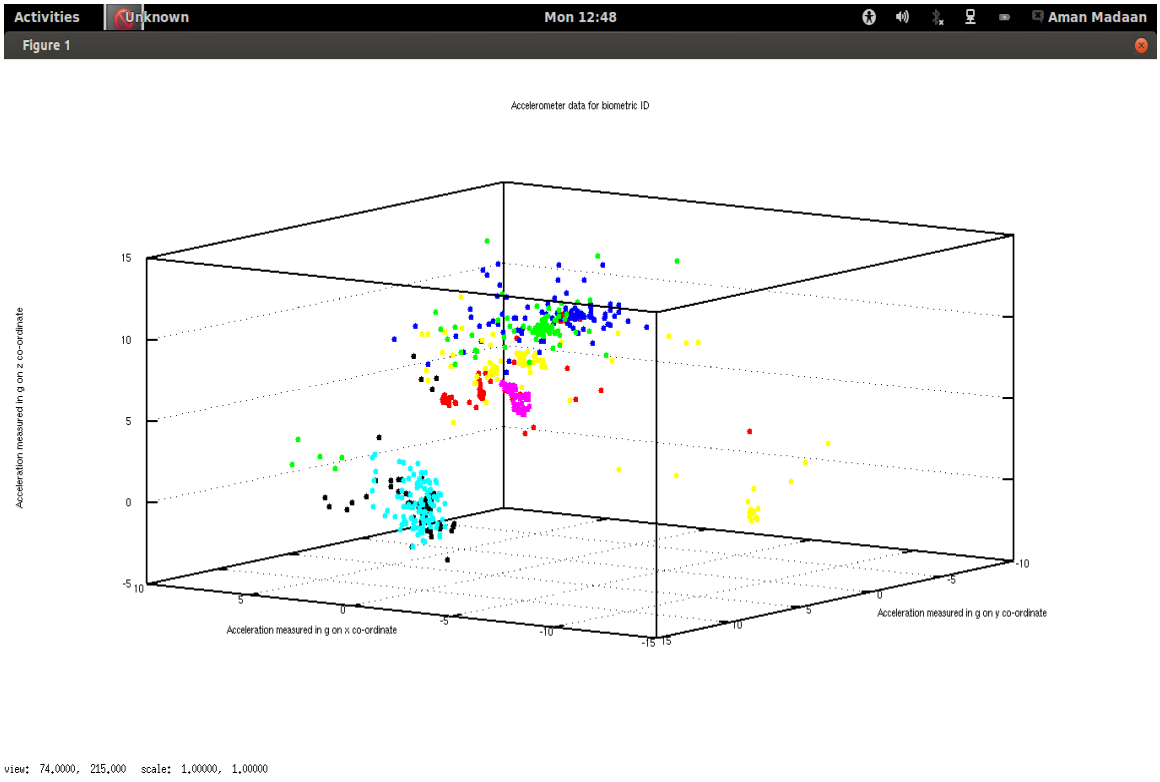

Figure 1.3: Sampled accelerometer data for 6 devices.

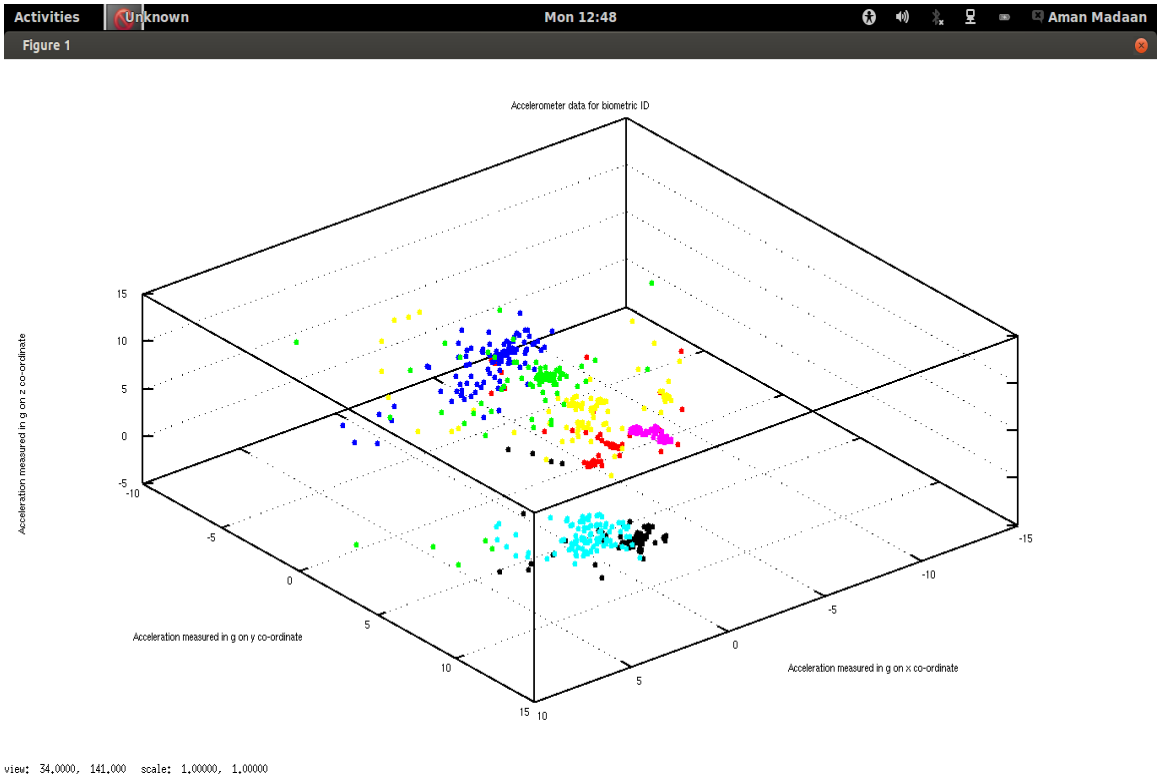

Figure 1.4: Sampled accelerometer data for 6 devices.

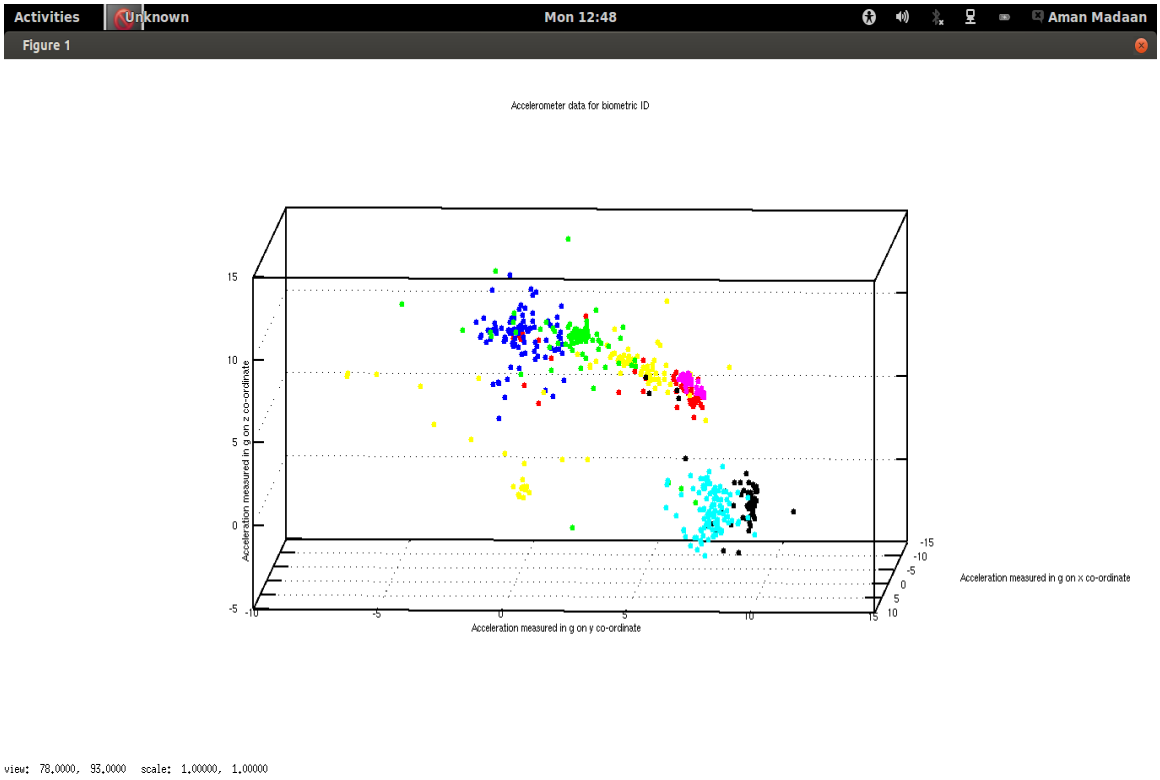

Figure 1.5: Sampled accelerometer data for 6 devices.

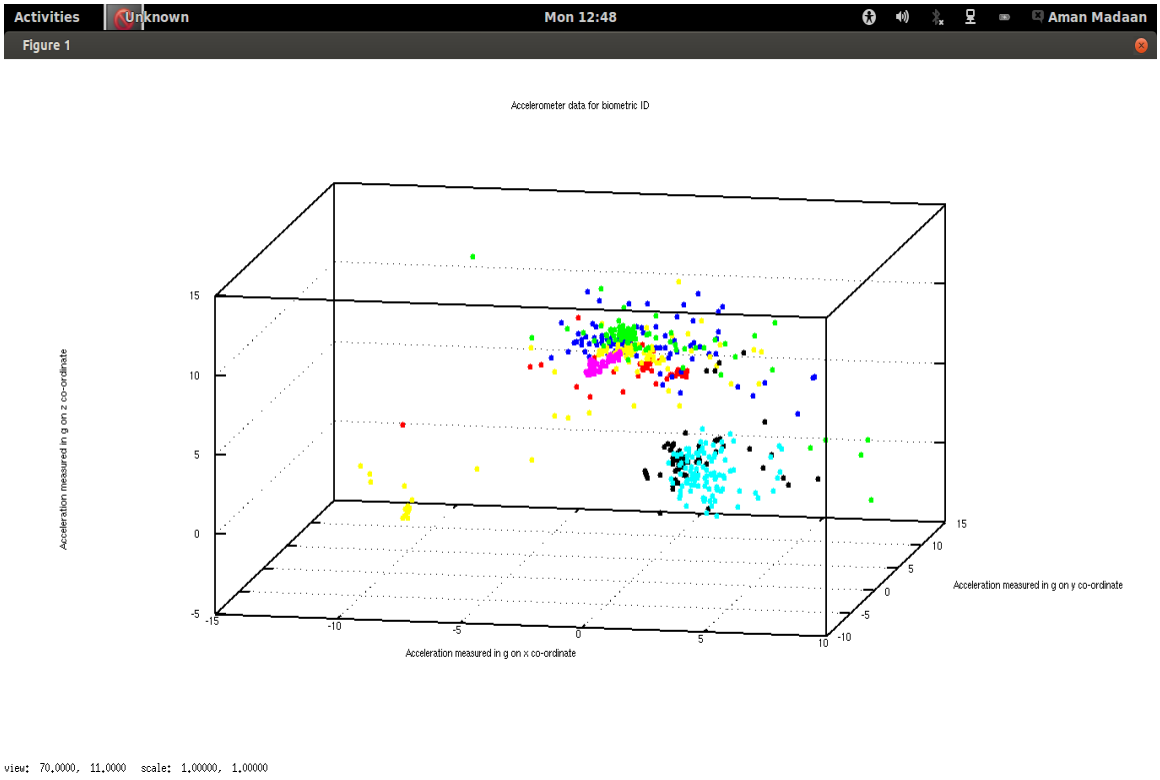

Figure 1.6: Sampled accelerometer data for 6 devices.

Chapter 2

Results

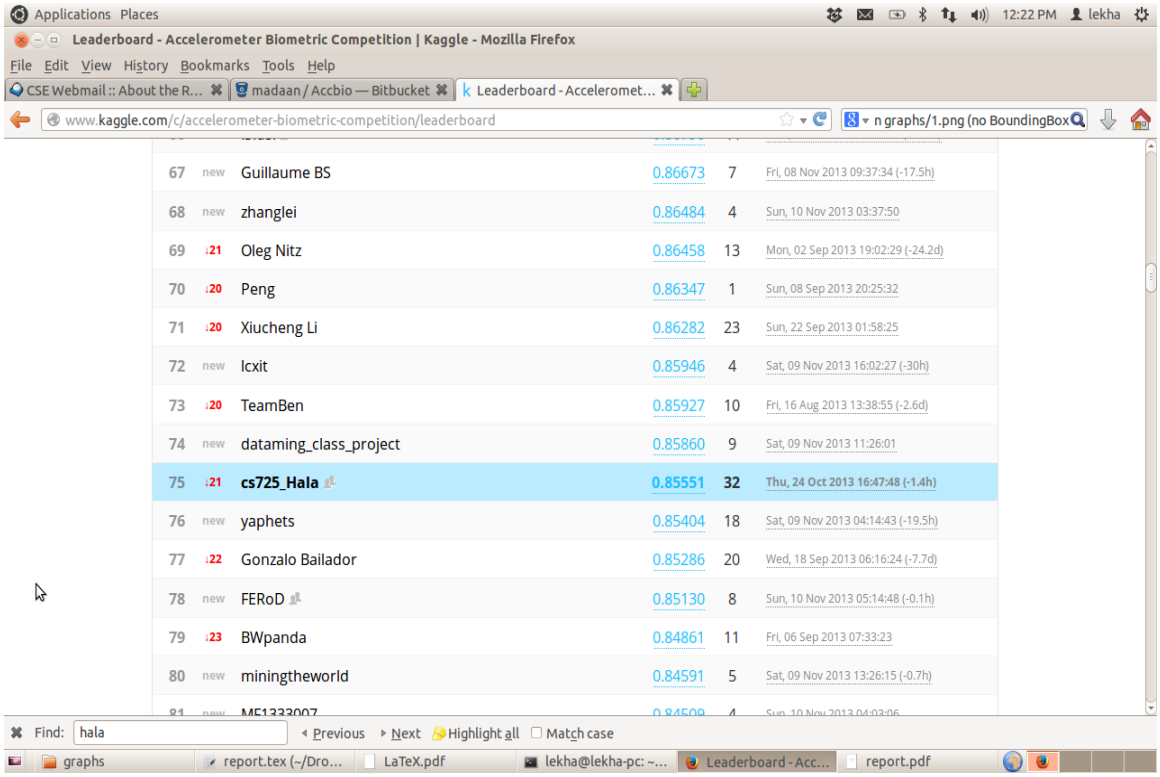

Figure 2.1: Leaderboard position as of November 10, 2013

Chapter 3

First attempt : Randomly picking

3.1 Random classification

3.1.1 Approach

The random classificatin method was the most obvious start. Flip a coin and depending on where it lands, predict true or false.To our surprise, this approach was enough to beat the benchmark.

3.1.2 Accuracy

50.317

3.1.3 Code

The code can be found in solutionsRepo/random

Chapter 4

Naive Bayes

4.1 Naive Bayes

4.1.1 Motivation

Once we resolve the accelaration due to gravity along each axis, the independence assumption became quite valid. Thus, we reached to the conclusion that naive bayes might help here. As the accuracy figures show, our assumption was quite valid.

Since using naive bayes implicitly means that we are considering standard deviation of the classes, a separate feature was not designed for this.

4.1.2 Approach 1

Using sklearn's implementation of naive bayes

1. Implementation : sklearn.naive_bayes
2. Training data : Sampled training data
3. Test data : We took mean of the test data and made predictions using the same i.e. the 300 odd rows for each of the 90024 test classes was compressed into 1, and this was fed to the classifier to get the prediction.
4. Accuracy : 0.7866
5. Code : solutionsRepo/nb/ (Details of running in readme)

4.1.3 Approach 2

Hand written implementation of Naive bayes. The sklearn's implementation was failing for large amounts of training data. We wrote our own naive bayes from scratch and it yielded the second best accuracy we were able to achieve to date.

1. Implementation : standard python
2. Training data : The complete Kaggle training data
3. Test data : The complete Kaggle test data
4. Accuracy : 0.82280
5. Code : solutionsRepo/naiveBayes (Details of running in readme)

Chapter 5

Distance Based

5.1 Motivation

It is quite imperative to use a distance based approach for classification not only because of the simplicity of the approach, but also because we observed in our initial plots of data using Octave suggested that data different users.

5.1.1 Approach

We used K-nearest neighbour (KNN) approach for a distance based approach. The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. www.scikit-learn.org/stable/modules/neighbors.html

We took mean of the original test data for each sequence to get the test data that we actually used.

1. **Implementation:** sklearn.neighbors.KNeighborsClassifier
2. **Training data:**
   (a) 1. For K = 1000 and K = 2000: sampled data
   (b) 2. For K = 4000, 6000, and 10000: complete data
3. **Test data:** sequence id wise mean data of original test data
4. **Accuracy:** We tried the KNN approach for incremental values of K with weights turned on
   (a) **K = 1000:** 77.97%
   (b) **K = 2000:** 79.02%
   (c) **K = 4000:** 85.11%
   (d) **K = 6000:** 85.33%
   (e) **K = 10000:** 85.55%

5.1.2 Code

The details to run this code has been given in the README file.

5.1.3 Results

The distance based approach using KNN with K=10000 on complete data gave us the best accuracy of 85.55%.

Chapter 6

Stochastic Gradient Descent

6.1 Motivation

Motivation behind the stochastic gradient descent was the amount of large training data. So it was very intuitive to use stochastic gradient descent on this data to obtain the results in reasonable time with reasonable accurracy. Though the data was not linearly separable, it was worth trying it to test the validty of scalibility argument of stochastic gradient descent. We used hinge loss as the loss function to further exploit the accuracy if any achievable through linear decision boundary.

6.2 Implementation

1. Implementation : sklearn
2. Training data : Sampled training data
3. Test data : Sampled test data
4. Accuracy : 0.80
5. Code : solutionsRepo/stochastic_gradient_descent (Details of running in readme)

Chapter 7

Stochastic Gradient Descent - Logloss

7.1 Motivation

We already trained the model using stochastic gradient descent with hinge loss as a loss function. So we wanted to try out some other approximate loss functions that are guaranteed to give good results with stochastic gradient descent. We also used L2 regularizer in both the cases of stochastic gradient descent (hinge loss and log loss) to prevent overfitting.

7.2 Implementation

1. Implementation : sklearn
2. Training data : Sampled training data
3. Test data : Sampled test data
4. Accuracy : 0.76105
5. Code : solutionsRepo/stochasticDescentLogloss (Details of running in readme)

Chapter 8

Logistic Regression

8.1 Motivation

The stochastic gradient descent with log loss gave reasonable accuracy. So we wanted to try out that whether logistic regression gives the similar accuracy or more. In thoery we expected it to give slightly better accuracy at the cost of high running time. Here also we used L2 regulaizer to prevent overfitting. And we used less accurate one vs all scheme rather than true multinomial Logistic regression.

8.2 Implementation

1. Implementation : sklearn
2. Training data : Sampled training data
3. Test data : Sampled test data
4. Accuracy : 0.76105
5. Code : solutionsRepo/logistic_regression (Details of running in readme)

Chapter 9

Support Vector Machines

9.0.1 Motivation

The various plots indicated that the decision boundaries involved were non linear. This, combined with the fact that SVM is the best off the shelf learning algorithm available lead us towards SVMs.

9.0.2 Approach

Using sklearn (python wrapper over libsvm)

1. Implementation :from sklearn import svm, grid_search
2. Training data : Sampled Sampled training data (We sampled the sample data)
3. Test data : We took mean of the test data and made predictions using the same i.e. the 300 odd rows for each of the 90024 test classes was compressed into 1, and this was fed to the classifier to get the prediction.
4. Accuracy : N/A. The training of svm never stopped.
5. Code : solutionsRepo/svm/

Chapter 10

Other approaches

In options to the approaches already tried, we also experimented with the following :

1. **Decision Tree** (.51)
2. **Naive bayes** with time as one of the parameters (.60)
3. Predicting by counting the number of training data samples and test data samples. (0.803) We realize this counts as data leak
4. **Random forests** (The training never stopped)