ML Contest Report

Team mOtu_Lambu_patlu

November 2015

1 Team Details

- Team Name : mOtu_Lambu_patlu
- Team ID: 2
- Team Members:
  - Sanchit Agrawal (CS13B061)
  - Ujjawal Soni (CS13B053)
  - Ishu Dharmendra Garg (CS13B060)

2 Libraries Used

Most of the code was written in Python.
The following libraries were used:

- scikit-learn
- numpy
- matplotlib
- scipy

3 Feature Reduction Techniques

We tried and tested the following feature reduction techniques:

- Feature importance taken from Random Forests: When Random Forests are trained, they can be used to get an estimate of which features are most useful. The top N features were taken and the rest were dropped.

- Feature Agglomeration: This is basically hierarchical clustering, but on the features instead of the observations. The clusters themselves can then be treated as features. We chose to go with the ward linkage as it gives compact and dense clusters. Ward linkage was also experimentally determined to be better than average and complete linkage.
- Principal Component Analysis: This is the vanilla feature reduction technique. However, we suspected that the data set might be noisy, and PCA might return the noisy directions as they could give high variance.
- Restricted Boltzmann Machines: The computation for this took too long (several hours) and we decided it wasn't feasible to try.

We found that reducing the number of features did decrease the performance of our classifiers. The time taken to train the classifiers was only marginally decreased due to feature reduction. We chose not to compromise our predictions to gain on training time, and hence did not do any feature reduction at all.

4 Data Visualization

Data visualization is a tough task, especially when the data has 2048 features. We tried a naive method for getting a sense of what the data looks like. We projected the train set on to the first 3 principal component directions. This 3-D data was then plotted with different colors used to represent different classes. As expected, the results were not so good. The figure was extremely dense and it was nearly impossible to make any sense of the data. However, we did see a few long trailing tails for some classes.

5 Class Imbalance

In the very beginning we realized that there was heavy class imbalance present in the train set. We had classes that were more than 1000 strong and classes that had less than 100 points.
The judgement for this contest is based on the average f_1 score, which means that each class needs to be given the same importance, irrespective of how many members it has.
So, we set the "weights" for each class to be $\frac{1}{N_c}$, where N_c is the number of observations belonging to class c. This was done for all the classifiers that we trained.

6 Oversampling

Another alternative to weighting the classes was to oversample the underrepresented classes. We tried this, but there was no difference in the results.

7 Undersampling

Next, we tried to reduce the number of samples of the overrepresented classes. We brought down the number to around 200 for each class. This had very bad effects on our results as the train data was cut down. Undersampling was rejected.

8 Bad Classes

We noticed that certain classes, particularly $4, 33, 34, 40, 46, 77, 86, 96, 98$, had a poor f_1 score for all the classifiers we tried. Hence we tried to give a boost to these classes by increasing their weights in our classifiers. This did not go well, and their f_1 score decreased further. So we tried to capture these classes by clustering.

9 Clustering

We were inclined towards clustering the data and fitting separate classifiers for each cluster that we found.
Since the data was high dimensional, clustering was infeasible. To counter this, we did feature reduction prior to clustering. Feature Agglomeration and PCA were used for this purpose. The number of features were also varied to achieve a good clustering.
We tried the following clustering algorithms:

- DBSCAN: Performs horribly due to the high dimensionality of the data set. This was expected beforehand. Even after increasing ϵ and decreasing minPts to ridiculous orders, most of the train points were classified as outliers.
- KMeans: Does the clustering in a reasonable amount of time. However, we knew from our data visualization that there were long chains present in the data, so KMeans need not perform very well. We varied K, but kept it around 100 in the hopes of clustering the classes.
- Mean Shift: This is a non-parametric clustering approach that is supposed to decide the number of clusters as well as the outliers. It almost always gave 10 clusters and listed approximately 2000 points as noise, which mostly defeated the purpose of clustering, since we had hoped to get approximately 100 clusters.
- Hierarchical Clustering: We tried ward, average and complete linkage with hierarchical clustering.

After clustering, we fit separate classifiers on each cluster. However, contrary to our intuition, the bad classes did not do any better. In fact they got worse f_1 scores.

10 Noise Removal

We had the suspicion that the bad classes were plagued with lots of noisy data. This could cause all the classifiers to perform badly. So we tried to remove the noisy points. We tried the following methods:

- Outlier detection via clustering: We removed the unclassified points from DBSCAN and Mean Shift clustering. This did not help.
- One Class SVM: This is a method for outlier detection for high dimensional data. However, it assumes that the train data is noiseless. This wasn't the case as we weren't sure which points were noisy. We went ahead with this method, but as expected, it didn't really help.
- Elliptic Envelope: This method fits an elliptic boundary around the data and whatever lies beyond is considered to be an outlier. But there were 2 issues with this method: it is for novelty detection, not for outlier detection and we didn't know if our data was really elliptical! As expected, this method did not help.
- Ad-Hoc: We noticed that the classifiers we built always performed badly on all of the first 100 points of the train data. So we simply removed these points and trained on the rest of the points. This decreased our performance. Because of the failures with noise removal, we discarded our assumption about the bad classes being very noisy. We went ahead with normal classification techniques.

11 Parameter Estimation Method

For all the classifiers we tried, the parameter estimation was done using grid search on a 5-fold stratified cross validation on the train set. This ensured that the class distribution was the same, which couldn't be guaranteed when doing a random split. 5-folds also ensure that the positive bias in estimating classification errors is reduced. After each grid search, the grid was made finer to fine tune the parameters.

12 Classifiers

We tried and tuned the following classifiers for the contest:

- K Nearest Neighbours
- Logistic Regression
- Linear Regression with indicator matrix
- Linear Discriminant Analysis

- Quadratic Discriminant Analysis
- Ridge Regression
- Lasso Regression
- Linear Classification with Stochastic Gradient Descent
- Support Vector Classifiers with various kernels
- Passive Aggressive Classifier
- Softmax Neural Networks (reused the code written for assignment 2)
- Decision Trees (CART)
- Random Forests
- Extra Random Trees
- AdaBoost
- Gradient Trees
- Naive Bayes Classifier
- Gaussian Mixture Models

Passive Aggressive Classifier was the best performer among all classifiers. Surprisingly, linear methods, in general, performed very well.

13 Stacking

We chose our top few classifiers and decided to combine them. We took their predictions and treated them as a new set of features. This was the input for training our ANN. However, the time taken to reach a reasonable number of iterations was too high despite code vectorization. This may be due to the large number of classes. Hence we had to discard ANNs. The next thing we tried was to feed the new set of features to Extra Random Trees, which is something like Random Forests. This also did not give good results. Then we tried a simple majority voting classifier, which predicted the class label that had the highest number of votes among the chosen classifiers. This did the trick and we got good results, landing us on number 2 on the leaderboard.

14 Weighting the classifiers

We decided to include weights for the classifiers for our majority voting algorithm. Basically, the vote of a classifier would be multiplied by its weight. The weights were first set to the f_1 scores of the classifiers. Later we set the weights to the accuracy given by each of the classifiers using cross validation. Both these methods did not outperform the unweighted majority voting algorithm. We realized that the test set was drastically different from the train set, and it would be unwise to set weights for the classifiers based on our experience with the train set.

15 Voting using probabilities

Some of our classifiers returned the probability of each class for a given observation. We extracted these probabilities and found the sum of probabilities for each class over all the classifiers. For each observation, we predicted the class that had the maximum probability sum over all the classifiers. This gave us a huge performance jump, and we decided to set this as our final stacked classifier. However, for this to work, we had to discard some our best classifiers (like PAC) as they did not compute the probabilities. Our final list of classifiers was:

- Stochastic Gradient Descent Linear Classifier with Modified Huber loss
- Linear Discriminant Classifier
- Extra Random Trees with Gini Criterion
- Logistic Classifier
- Support Vector Classifier with Linear Kernel

With this ensemble, we finally ended up on the 1^{st} position on the leaderboard.