Machine Learning & Data Mining
Miniproject 1

Caltech CS/CNS/EE 155
Released January 28^{th}, 2017

---

1 Introduction

- **Group members**
  Vaibhav Anand
  Nikhil Gupta
  Michael Hashe

- **Team name**
  The Breakfast Club

- **GitHub link**
  https://github.com/nkgupta1/CS155-Project1-Voter

- **Division of labour**

2 Overview

- **Models and techniques tried**
  - **Neural Network**
  - **Random Forest** We experiemented with various types of random forest-based classifiers. In addition to the standard random forest model contained in sklearn, we examined boosted and bagged trees, as well as a feed-forward stack of random forest classifiers.
  - **Linear Models**
  - **Mixed Ensembles**

- **Work timeline**
  - **Bullet:** Bullet text.

3 Approach

- **Data processing and manipulation**
  - **Original Data:** At first, we trained the models on the raw data. These models ended up performing relatively poorly on the leaderboard. In looking closer at the data, we realized this was because of the way the data was labeled: questions with categorical answers had integers as labels for the different categories. As such, the models were trying to put an ordering to the dimensions where none existed, which hurt the accuracy of the models.
  - **All Categories:** The first thing we tried in order to resolve this issue was to flatten the whole data set and make it all categorical. In particular, this choice was motived by a cursory look through the handbook, from which we noticed that a majority of the dimensions were categorical. Before we invested time in splitting the data into ordered and unordered dimensions, we tried training

models when the whole data was made into categories. Models trained on this data had memory issues because of the size of the data set and there a very little chance of generalization because of the number of categories present.

- **Categories and Ordered:** After taking a closer look through the handbook, we realized that many of the issues we were having with categorizing the data was due to the specific characteristics of the dimensions. While most of the dimensions were unordered, many of them still had an ordering. As such, this forced us to manually sift through the data in order to bin the dimensions into ordered and unordered. In order to ease this process, we wrote a parser so that we could later modify which dimension were kept and how they were treated.

  The range of each of the ordered dimensions was mapped to [0,1] so that the models could better deal with them. This choice was made after trying to normalize the data and having some of the values still being very large. The unordered data was made into categories with one category for each of the values present. The parser kept track of the way each dimension was processed so that the test data could be processed in the same way. This raised an issue of how to treat values for dimensions that were in the test set but not in the training set for the unordered dimensions (while this occurred, it was not particularly frequent). We ended up putting no value for this because we thought it would be better to omit data than to lie about the data.

  After putting the data through this processing, the number of resulting dimension was around 50000, which is too many for the number of points that we had. In looking through how many dimensions the processor made each of the original dimensions into, there was one dimension that accounted for all but about 4000 of the dimensions, a unique ID for each family. As such, we added another option to the parser to completely delete dimensions. Models trained on this new, processed data set performed much better than the models performed on the original data set. We used NumPy for much of the data processing because of the number of methods present in the NumPy library which meant that we did not have to reimplement many of the functions. Furthermore, NumPy methods run very quickly on the NumPy arrays.

- **Trimming Dimensions:** After using the last data set for most of the models, we were not able to improve on accuracy much more so we thought this might be due to noise in the dimensions of the data set. As such, we made the decision to go through the handbook more carefully and make the decision to keep or delete each of the dimensions. This is because there were a significant number of dimensions that added noise to the dataset. For example, there were many dimensions for line number which was not something that the people actually answered, so were something that we decided were insignificant. After this final trimming process, the number of dimensions present were reduced from about 4000 to 3000. This reduced dimensionality improved the performance of some of the models.

- **Details of models and techniques**

  - **Bullet:** Bullet text.

    We did lasso regression.

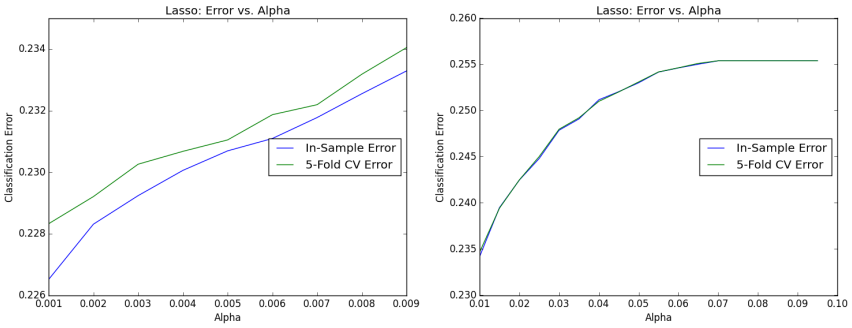

Figure 1: Insert caption here.

We did ridge regression.

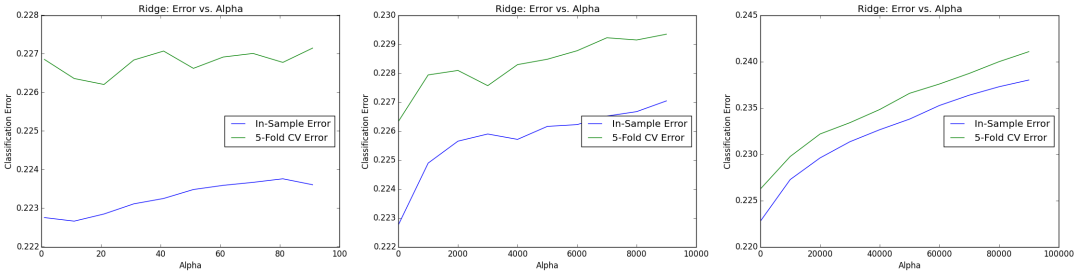

Figure 2: Insert caption here.

We did stochastic gradient descent.

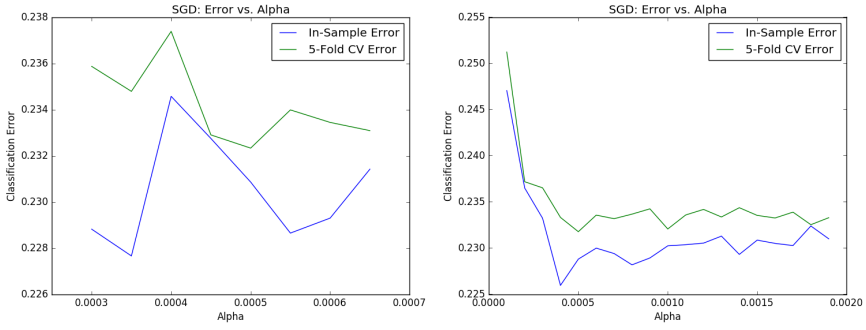

Figure 3: Insert caption here.

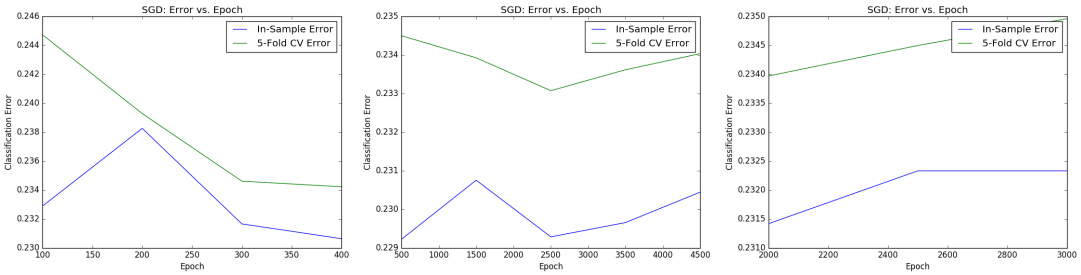

Figure 4: Insert caption here.

4 Model Selection

- **Scoring**
  Due to the binary output classification, we chose to do scoring by classification accuracy (as opposed to loss measure). As shown in the cross validation results in Table 1, amongst the linear classifiers, Logistic Regression performed best on the pre-processed categorized data, although all had very similar results, except SVM, which performed significantly worse. Additionally, the neural network (MLP Classifier), even after efforts to optimize its parameters, could not perform near the level of the linear regressors. After the de-categorization of the data into around 5000 labels, many of the linear regressors performed worse, whereas the SVM and the stochastic gradient descent that performed worse before, performed better. However, the conversion of the data into labels with real-valued meaning significantly benefited the training of the neural networks, which we surmise is due to the fact that the categorical labels posses greater correlation to the output when considered together, which are the types of patterns that the neural network naturally performs better with.

- **Validation and Test**
  We used 5-fold cross validation for all model selection and choosing semi-finalists of the neural-network models, except for choosing the finalists in the neural network models, where we used cross-10 validation. By generating these cross validations, we were able to select the optimal learning parameters, such as regularization terms, early stopping (max iterations), and loss functions. This was done by manually fixing all parameters except for one, optimizing that parameter, then moving on to optimizing another parameter, before circling back to the same parameter. In a way, we performed a manual step-wise gradient-like descent on parameters. The results of cross validation for various models can be seen in Table 1 in the Appendix. The parameter variables refer to the API SKlearn's definition of parameters.

5 Conclusion

- **Discoveries**
  The main discovery from this competition was the difficulty of fitting a "hard" and imbalanaced dataset, and in particular the challenges accompanying creating a model both sufficiently able to fit a training dataset (2008 data) and sufficiently general to adapt to an unknown and distinct new dataset

(2012 data). These challenges will be analyzed in more depth in the section below.

In terms of models used, we experimented with variants of common models examined in class. In particular, we utilized variations of trees aside from the default CART tree and various linear models. While our best performing models remained standard neural networks (and, on the test dataset, random forests), we certainly learned more about the variants of these models currently in use.

In examining the occasionally massive changes in the leaderboard between the first and second parts of the competition, and in examining the differing performances of our own models on these two datasets, we also discovered the dangers of overfitting and the propensities of various models for doing so. In particular, we found (after submission) that our more "general" models (i.e., collections of networks and forests) tended to perform reasonably well on the new dataset, whereas models more prone to overfitting (smaller forests and networks, linear models) tended to suffer more. Considering the large drops experienced by many teams initially above us on the leaderboard, it is our belief that this "overfitting the leaderboard" was in fact a fairly general problem.

- **Challenges**
  As mentioned above, the main problem experienced in the competition was the dataset. In particular, we found that the dataset was lopsided (75% of those polled voted), rather different from the test set (in which 65% of those polled voted), and very noisy (i.e., even the best/most overfitted models were unable to break 80% classification accuracy). In dealing with such a dataset, we found that most models performed within a percent or two of each other, which made distinguishing the best models somewhat difficult. Further, it meant that it was very easy to fall into a trap of overfitting (i.e., gridsearching parameters, submitting an excessive number of models, and falsely concluding that the models that best fit the data did so because they were the best models). We managed to avoid this reasonably well, although we could have improved on our part 2 score had we selected better. Were we to do this competition again, we would have put more thought into how to best select models.

  Within the dataset, we also found that many of the parameters were either repeated, arbitrary (i.e., identification numbers), or answered for only a very small number of responders (i.e., several questions focused very specifically on certain subsets of the population, and therefore had response rates in the hundreds). We curated the dataset and removed many of these parameters from consideration, although determining whether or not a parameter should be included was not always a clear decision. Were we to redo this competition, we would have curated the dataset in a more principled (and, ideally, automated) manner, through some manner of cross-validation on model performance with various parameters removed. We would also likely have removed even more parameters; it is our belief that this might help with the generalization accuracy of the model.

  TENSORFLOW

  We additionally encountered problems with resource limitations in training large models. While most

of our models trained in reasonable time, we found that some (in particular, certain variants of random forests and SVMs) were slow enough to be infeasible.

- **Concluding Remarks**

6 Appendix

Table 1: With data normalization, without de-categorization of data (does not include all tests)

Type	K-Folds	Parameters	Classification Accuracy
SVM	5	4-degree polynomial kernel	0.7568
SVM	5	RBF kernel	0.76797
Logistic Regression	5	SAG solver, 25 iterations (converged), 10^{-5} regularization strength	0.774
Logistic Regression	5	SAG solver, 25 iterations (no convergence), $C = 10^0$ regularization strength	0.773
Logistic Regression	5	SAG solver, 100 iterations (no convergence), $C = 10^0$ regularization strength	0.773
Logistic Regression	5	SAG solver, 100 iterations (no convergence), $C = 10^5$ regularization strength	0.773
Logistic Regression	5	SAG solver, 400 iterations (no convergence), $C = 10^0$ regularization strength	0.773
Logistic Regression	5	Liblinear solver, 100 iterations (no convergence), $C = 10^0$ regularization strength	0.774
Ridge Regression	5	$\alpha = 20$ regularization strength, (optimal alpha found by plotting CV vs. alpha)	0.7738
Lasso Regression	5	$\alpha = 10^{-3}$ regularization strength, (optimal alpha found by plotting CV vs. alpha)	0.7718
MLP* Classifier	5	Hidden layers=(200,100), iterations=5 (optimal iterations found by plotting CV vs. iter.)	0.7711
MLP* Classifier	5	Hidden layers=(100), iterations=3 (optimal iterations found by plotting CV vs. iter.)	0.7726
MLP* Classifier	5	Hidden layers=(100, 50, 10), iterations=4 (optimal iterations found by plotting CV vs. iter.)	0.7725

MLP* Classifier	5	Hidden layers=(20), iterations=10 (optimal iterations found by plotting CV vs. iter.)	0.7734
MLP* Classifier	5	Hidden layers=(20, 20), iterations=15 (optimal iterations found by plotting CV vs. iter.)	0.7733
MLP* Classifier	5	Hidden layers=(20, 20,20), iterations=10 (optimal iterations found by plotting CV vs. iter.)	0.7730
MLP* Classifier	5	Hidden layers=(200,100), iterations=5 (optimal iterations found by plotting CV vs. iter.)	0.7711

Table 1: Without data normalization, with de-categorization of data

SVM	5	RBF kernel	0.7707
Logistic Regression	5	Liblinear solver, 50 iterations (converged) $C = 1$ regularization strength	0.7725
Logistic Regression	5	Liblinear solver, 100 iterations (converged) $C = 1$ regularization strength	0.7725
SGD	5	Hinge loss, 1000 iterations, $\alpha = 0.001$ regularization strength	0.7748
SGD	5	Hinge loss, 500 iterations, $\alpha = 0.001$ regularization strength	0.7737
SGD	5	Hinge loss, 100 iterations, $\alpha = 0.001$ regularization strength	0.7727

Table 1: Finalists from 43 cross validations of various layer dimensions and iterations (bounded cross validation by iterations on both sides, such that an increase or decrease in iterations increased validation error significantly):

MLP* Classifier	10	Hidden layers=(50, 50), iterations=4 (optimal iterations found by plotting CV vs. iter.)	0.7775
MLP* Classifier	10	Hidden layers=(100, 50, 10), iterations=9 (optimal iterations found by plotting CV vs. iter.)	0.7782
MLP* Classifier	10	Hidden layers=(150, 50), iterations=5 (optimal iterations found by plotting CV vs. iter.)	0.7776
MLP* Classifier	10	Hidden layers=(150, 50, 10), iterations=6 (optimal iterations found by plotting CV vs. iter.)	0.7781

Table 1: *MLP = Multilayered Perceptron. *SGD = Stochastic Gradient Descent. We saw de-categorization improve performance significantly in neural network models, whereas linear models performed slightly worse.