CMS 155 Miniproject 1 Report

Lucien Werner
Spencer Gordon

February 10, 2018

1 Introduction

- **Group Members:** Lucien Werner and Spencer Gordon
- **Team name:** Numpty Dumpy (11^{th} place)
- **Division of labour**
  Although we mostly did pair coding, the table below displays a rough division of tasks.

Task	Person
Regression & ensemble methods coding	Lucien
Neural net coding	Spencer
Report writing	Shared

- **List of files in submission**
  - `regression.py`
  - `adaboost.py`
  - `gradientboosting.py`
  - `randomforest.py`
  - `naivebayes.py`
  - `deepnet.ipynb`
- **GitHub repository:** `www.internet.com`

2 Overview

- **Models and techniques tried**

  We tried a range of machine learning models and used a subset of the given training set to validate and compare their performance. Tuning the hyperparameters within the model classes was the crucial element in fitting a model that obtained good performance on the public leader board. Naive methods (regression) were tried first, followed by more complex ensemble approaches.

- **Logistic Regression:** Linear regression with logistic loss and regularization strength $C = 0.1$ was the first thing we tried, and ultimately it gave better performance than anything besides the neural net. Fitting this model was primarily about finding the best regularization strength. Using `sklearn`'s `GridSearchCV` method, we set a range of parameters for regularization strength, solver, and max iterations to train the model over. To asses the performance of the different parameter sets, we used 5-fold cross validation of the training set. Bag-of-words data is high dimensional and sparse, indicating that some kind of regression was a good place to start the modeling.
- **Random Forest:** Random forest classifiers are good all-purpose models and if the weak classifiers are shallow enough, they train quickly as well. We tested performance using `sklearn`'s `RandomizedSearchCV` method across a range of parameters including the number of weak classifiers, min_sample_split, and max_tree_depth.
- **Gradient Boosting and AdaBoost:** Gradient boosting and AdaBoost with shallow decision trees as weak classifiers did not give better results than the random forest. After tuning the hyperparameters with randomized grid search over a subset of the parameter space, we only achieved accuracy in the low 80% on our validation dataset.
- **Naive Bayesian Classifier:** Bayes classifiers generally train quickly on high dimensional data. We found this to be the case but the accuracy was poor.
- **Neural Network:** We attempted a 3-layer neural network with 1300 hidden units, ReLU activation, and Dropout layers between all of the hidden layers. We tuned the dropout probability of all the layers using the approach from the 4^{th} homework. This method ended up performing marginally better on our validation dataset and significantly better on the public leaderboard.
- **Data normalization:** We tried several methods to normalize the training data:
  - TF-IDF transformation (discounts words that appear frequently in other reviews)
  - Logarithmic normalization ($X_i = \log(1 + X_1)$) (discounts higher word counts)
  - Binary normalization (converts all non-zero entries to 1)
  - Point-wise normalization (divides every feature in a datapoint by the norm of the entire datapoint).

  None of these normalization procedures improved the accuracy of our classifier, although in some cases (TF-IDF, for example) there was a significant increase in the training speed. The trade-off between accuracy and training speed was not universal across the models we tested, but we ultimately decided against normalization for our neural network.

- **Work timeline**
  - **February 2-4:** Logistic regression
  - **February 5-7** Ensemble methods
  - **February 8:** Neural net

3 Approach

- **Data processing and manipulation**
  - **Bullet:** Bullet text.

- **Details of models and techniques**
  - **Bullet:** Bullet text.

4 Model Selection

- **Scoring** We used accuracy as our scoring metric in all tests. It is defined as

$$\text{accuracy} = \frac{\#\text{ of matches}}{\#\text{ of datapoints}}.$$

  When training with cross-validation, accuracy scores were averaged across folds. Our initial experiments with logistic regression set a baseline ~ 84% accuracy.

- **Validation and Test** We set aside 5% (1000 datapoints) of the training set to validate our models, separate from in-sample cross-validation during training. Scores from this validation step generally tracked closely to those subsequently attained on the public leaderboard.

5 Conclusion

- **Discoveries** We discovered that despite trying and optimizing numerous regression models, model ensembles, and hyperparameters, a neural network still gave the (marginally) best performance. It was also surprising that simple logistic regression performed nearly as well on this dataset.

- **Challenges** Avoiding overfitting to the public leaderboard was on our mind as we validated the models. By setting aside 5% of our training set to validate our model accuracy after training, we had a point of reference for the leaderboard score. Otherwise, establishing the neural net architecture was a process that felt like a lot of trial and error. Given the number of possible a structures and parameter values that are available to a neural net, it seemed like we might have landed upon something that worked well by luck. Doing this again, we would spend more time systematically testing different model structures (number of layers, activation functions, number of units per hidden layer, etc.), or alternatively considering ensembles of shallow neural nets.

- **Concluding Remarks** An interesting feature of this project was that there seemed to be a fundamental limit on the performance of machine-learned models on this dataset. The scoreboard results were all clustered within a couple percentage points around 85%. We found that most models we tried would approach this ceiling after tuning hyperparameters; in other words, the learning limit of the data seemed to be model independent.

References