# A Machine Learning Competition in Sentiment Analysis

Dryden Bouamalay
Jeffrey An
Rohan Batra

## 1 Overview

For this competition, we trained 5 base models: an adaboost classifier (via decision trees), a bagging classifier (via decision trees), a random forest classifier, a XGBoost classifier (via decision trees), and a SGD Classifier (via hinge loss). The base models were trained on 80% of the training data and then combined by training a model on a vector of their predictions for 16% of the training data. Finally, different blending models were compared on a 4% validation set. We compared decision tree and SGD classifiers for blending and found that there was no significant difference between the models.

It is important to note that we forgot to select any models for submission on the Kaggle competition site, and thus Kaggle picked our highest scoring models based on submission score only. These models were not our top choice, as they did not have the highest validation scores, and they did not generalize well. It is impossible to say specifically how well our models would have done on the leaderboard had we picked them, but it is possible that these would have performed better.

That said, the most significant challenge encountered during this project was overfitting. To overcome the dangers of overfitting, we did two main things: (1) Reasonable parameters for our estimators were chosen to regularize for model complexity. (2) Validation sets were used to estimate generalization using an appropriately blended final model.

As for the work distribution, Dryden Bouamalay is primarily responsible for the creation of this report as well as the basic initial implementation. The main implementation work was completed by Jeffrey An and Rohan Batra. In summary, all team members shared work fairly.

## 2 Data Manipulation

The primary tool used for data pre-processing was *tf-idf*, which stands for **term frequency-inverse document frequency**. The central idea underlying tf-idf is that certain words are inherently more useful for the purposes of classification than others, and so tf-idf is used as a way of normalizing for word frequencies for each training sample with respect to word frequencies across all training samples. Thus, the motivation for using tf-idf was to extract the most useful features from the dataset.

Tf-idf is calculated as the product of the **term frequency**, tf, and **inverse document frequency**, idf. There exist many methods for calculating the term and inverse document frequencies, but they all share the same key idea: $\text{tf}(t, d)$ is a measure of how many times a particular term $t$ appears in a given document $d$, and $\text{idf}(t, D)$ measures whether the term $t$ is common or rare across all documents $d \in D$.[1]

In our case, $t$ represents a particular component of our input vector and $d$ represents a particular vector. Furthermore, the collection $D$ represents the entire training dataset. Specifically for this project, we used the tf-idf implementation supplied by `sklearn`, which calculates tf-idf as:

$$\text{tf-idf} = \text{tf} \cdot (\text{idf} + 1) = \text{tf} + \text{tf} \cdot \text{idf}$$

which forces terms $t$ that occur in all $d \in D$ to still be considered despite their occurrence in all documents.

The process by which we used tf-idf was through the following code:

```
# ...
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(x_train)
np.savetxt('tf_idf.txt',
                tfidf.toarray(),
                delimiter='|',
                header="tf-idf weighting of 'training_data.txt'")
# ...
```

where `x_train` is the original training data. This was then written to a document `tf_idf.txt` in the same format as the training data so that it could easily be loaded into any of our models.

# 3 Learning Algorithms

Briefly introduce all the algorithms you tried to implement

Elaborate on the specifics of each algorithm you implemented, and the performance achieved.

We implemented the following algorithms:

### Adaboost Classifier (via Decision Trees)

Adaboost is a technique used to reduce bias in models, by combining many decision stumps. The one parameter we tuned was the number of stumps included in the model. We did so by increasing the number of stumps until the test score from 80/20 cross validation no longer increased. This model achieved on average a 0.90 training and 0.66 test score.

[1] https://en.wikipedia.org/wiki/Tf%E2%80%93idf

### Bagging (via Decision Trees)

Bagging is used to reduce variance in models that have a tendency to overfit, such as decision trees. The parameters tuned were the number of classifiers and the number of points to sample for each individual tree. For each, a randomized search was done to find the highest test score from training on 80% and testing on 20% of the data. This model achieved on average a 0.79 training and 0.66 test score.

### Random Forest

A random forest is an enhancement on bagging for decision trees where in addition to sampling data points randomly for each tree, the features included in that tree are also sampled randomly. Each tree used $\sqrt{1000} \approx 31$ random features, a value suggested by the SKLearn documentation. This model achieved on average a 0.91 training and 0.70 test score.

### XGBoost

Xgboost is short for extreme gradient boosting, and is a method to train a decision tree using gradient descent. The performance of the algorithm did not vary significantly with many paramters other than the number of iterations of the graident descent. The model acheived an average of a .80 training score and a .70 test score.

### SGD Classifier

The SGD classifier used stochastic gradient descent to train a weight vector to classify the points. We used hinge-loss, as this significantly improved the score over square-loss and other error fuctions. The regularization parameter was determined through cross validation. This model acheived an average of a .78 training score and .69 test score.

## 4 Model Selection

The model that Kagle chose used a simple majority vote of three models (adaboost, bagging, and random forest). However we tried another blending method on the 5 which involved training a final model on a vector of predictions created by each of our base models.

To do this, we partitioned our data into training and validation sets via `train_test_split()` in the `cross_validation` module. Specifically, we reserved 80% of the training data for training each of the 5 base models: adaboost (decision trees), bagging (decision tree), random forest, XGBoost (decision tree), SGD Classifier (hinge loss). After these models were trained, we further split the remaining 20% of the training data into a 16% training set and 4% validation set. In other words, the 16% partition is actually used to train a final model that blends the predictions of our 5 base models, and the last 4% is used to validate this final blended model.

*Details:* For every datapoint in the 16% partition, we had each of our 5 models make a prediction, gen-

erating a new dataset, which has the following form:

| $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $l$ |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| . | . | . | . | . | |

where $m_i$ for $i = 1, 2, ..., 5$ represent the predictions of our 5 base models and $l$ is the label for this sample.

We then trained two models using the above dataset: a decision tree classifier and an SGD classifier. We guided model selection by using the final 4% of the original training data as validation data for these two models. We found that their performance on the 4% validation data was comparable, and so neither model was more preferred over the other. The testing scores varied, as might be expected from a validation set with only 4% of the data. However, they averaged around .75, showing an improvement over any individual model.

# 5 Conclusion

We estimate that the performance of our overall project was good. We maintained a high position on the leaderboard until Kaggle automatically picked our final model for us, which likely overfit. However, it is unclear whether the models we would have chosen would have performed better. Thus, assuming most of our models overfit, an improvement would have been to more heavily regularize. While we tried to make good use of validation and blending to generate our final model, it was difficult to do so efficiently, as the final validation set only contained 4% of the data. Given more time, we could have experimented with other ways to allocate the data given this iterative training. We could have also experimented further with regularization parameters, as it is possible that we still overfit in the models not chosen by Kaggle.

In summary, this competition was a lesson in overfitting and data management. While data-processing such as tf-idf and blending techniques using validation are very useful and important, the final blended model will not come together if the individual models have not been appropriately regularized, and the final model has not been trained on sufficient data.