# Project A: Classifying Reading-Level Difficulty from Text

Cooper Golemme

April 1, 2025

## 1 Bag of Words Model

### 1.1 1A: Bag-of-Words Design Discussion

The specific BoW representation that I used was term frequency inverse document frequency (Tf-idf). This is similar to a simple BoW representation that keeps track of the counts of each word in the vocabulary, but instead of creating a simple count, Tf-idf stores floating point values, seeking to capture the relative importance of words that appear in a given text over some corpus (collection of documents). For a given term, $t$ and document,$d$, the Tf-idf is computed as

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

where, $t$ is the specific term, $d$ is the document that this term is in and $D$ is the whole document corpus. The term frequency $\text{tf}(t, d)$ in the equation above, indicated how many times a given term appears in a document (selection of text). The inverse document frequency relates how frequently that term appears across all documents, $D$.

Before performing TF-IDF, I ran a few steps of pre-processing of that data.

1. Lowercase the words and strip punctuation marks. **Reasoning:** I want to consider words that start a sentence and thus are capitalized, the same as words that appear in the middle of the sentence

2. Stemming. **Reasoning**: stemming converts different forms of verbs to their root forms. For example, "running" becomes "run." I want the BoW representation to treat running and run the same way becuase they are the same word, but with different endings.

3. Stop words. **Reasoning**: Stopwords are common words like "the", "and", "a" that provide little information in the sentence but are included to make it syntactically correct. I removed them to not clutter the BoW model and provide little information about the difficulty of text.

The final vocabulary size was tuned during cross validation. I want a vocabulary size that is big enough to capture the many different words in the training set that can contribute to a text's difficulty, but not so big that it captures the noise of the training set and does not generalize well to new data. I tested ranges form 5000 - 40,000 words. Ultimately, during cross validation, I determined a vocabulary size of 10,000 to be the best suited to this task.

I also experimented in cross validation with leaving out words that appear too frequently across the train set and words that appear too infrequently. I did this because I thought that it would be hard for the model to assess the difficulty of new text if are very rare or very common to appear in a given text. The ranges I tested were excluding words if they appeared in 50-90% of texts, and excluding words if they only appear in 1 - 10 texts. For testing, when the model encountered words that were out of its vocabulary (OOV words), I chose to ignore them. I did this because I thought it was hard for the model to reason about how those words contribute to the difficultly of the text if it had not previously seen them. the model only makes predictions based on words that appear in the vocabulary generated from the training. To implement the TF-IDF, I used scikit-learn's implementation of TF-IDF Vectorized which I can tune with vocabulary size, rare and frequent words, and more. For pre-processing, I used the NLTK Python library for stemming and stopwords.

## 1.2 Cross Validation Design

I employed a randomized, 3-fold cross validation on 50 different parameter configurations. Each fold had around 1750 training instances. I first randomized the dataset, because there are ordered by author name. Randomizing the dataset helps ensure that one author is not overrepresented in a given CV split. Because the dataset is roughly balanced between the numbers of examples of each class (Key Stage 2-3 and Key Stage 4-5), I didn't find it necessary to enforce that the splits have similar numbers of positive and negative examples as they are likely to have similar numbers anyways. After running the cross validation to select the best hyperparameters, I trained the model on the entire dataset to generate a final model that I will apply the test set on. For all cross validation, I used the Scikit-learn CV libraries. The performance metric that I choose to optimize for was as this is what the model will be tested on when I go to test.

## 1.3 Hyper Parameter Selection for LR Classifier

Logistic regression is potentially well suited to this task because of its simplicity and compatibility with classification tasks. Logistic regression is a relatively simple method of classification that can easily be applied to a 2 class classification problem. It is also fast to train and highly tuneable. For the hyperparameter selection for the logistic regression classifier, there are two main choices of hyper parameters to control model complexity: the $C$ penalty value and a penalty function (either L1 or L2). L1 penalty forces some of the weights to be close to 0, effectively performing feature selection, which I thought would potentially be

well suited to the BoW feature representation because not all words (features) in the BoW likely contribute to the difficulty of the text. L2 penalizes large magnitude weights which helps to limit model complexity and reduce the risk of overfitting. The C factor determines the strength of both of these penalties.

During testing, there were some convergence issues with L1 penalties, specifically when trying to use an L1 penalty with the same values of C used for the L2 penalty. L1 preferred higher penalties (smaller values for C in scikit-learn LR). When preforming hyper parameter selection, I had to account for this.

Here are the specific values I tried

| | |
|---|---|
| Penalty | L1, L2 |
| C | 0.1, 1, 10, 100 |

Table 1: Logistic regression hyper parameter CV search values

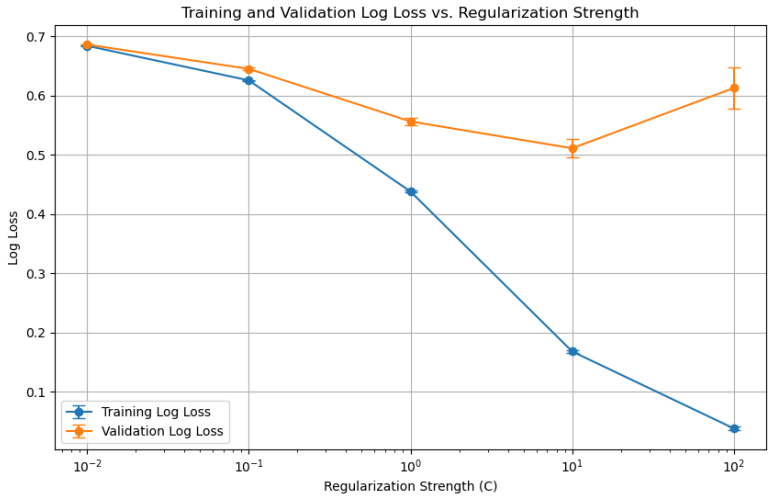


Figure 1: Binary cross entropy loss or log loss across different folds of cross validation. The bars represent the standard deviation in the data across the fold. Larger bars represent more variance in the error across the folds, and smaller bars represent more similarity across the folds. From this graph, I can see that a good value for the regularization strength, $C$, is about 10 as after 10, the validation error goes up, indicating I begin overfitting to the data. The evidence is pretty conclusive as the log loss point for $C = 100$ falls outside of the standard deviation range of the $C = 10$ point.

### 1.4 Analysis of Predictions for the Best Classifier

Below is the confusion matrix of the classifier on validation data. To produce this matrix, I split the dataset into 80% for training and 20% for validation randomly.

| | Actual Positive | Actual Negative |
|---|---|---|
| Predicted Positive | 338 | 161 |
| Predicted Negative | 171 | 442 |

Table 2: The classifier generally does better at assessing if a lower reading level text is in fact a lower level reading text. From further analysis, it typically does worse on shorter texts as opposed to longer texts. This might be because there are less words in shorter text and less context to gain insights about the difficultly of the text. There is no assessable difference of model performance on different author's texts

### 1.5 Performance on Test Set

Ultimately, on the test set, I achieved a performance of 0.67696 on the auroc benchmark. Consistently in cross validation, I observed auroc scores of .80-.85. My hyper parameter search procedure did not give me an accurate estimate of the performance on the unseen test data. This could be due to the fact that the training set featured multiple texts by the same authors, which are likely to appear in both the training and the validation set for each CV fold. Authors tend to use similar words and have similar writing styles across their various texts, so it may be the case that I was experiencing higher scores because my model became familiar with the author's in the training set's style of writing. Then, when my model was applied to new authors, it preformed worse because it was trained on multiple texts of the same author.

## 2 Open-ended Challenge

### 2.1 Feature Representation

I processed the text field similar to how I did with the BoW model, using TF-IDF with stopwords and 10,000 max features. For the numerical features, I preformed a simple scalar transform to fit all the numerical feature values in the range $(0, 1)$ for numerical stability. Additionally, I used the BERT embeddings to provide the model with extra context about the text. BERT stands for bi-directional encoder representations from transformers. Unlike BoW models that assign words to entries of a vector regardless of their position in the text, for example "feet" in "Cooper has two feet" has the same BoW representation as "Cooper is two feet tall", BERT embeddings can capture the meaning of words based on their usage in the sentence. Theoretically, BERT embeddings should be able to capture not only words that are challenging, but also the challenge

associated with arranging those words in a certain way, or using them to mean certain things.

## 2.2 Cross-validation

I performed cross validation similar to how I did for the BoW model. I first shuffled the datasets and then performed randomized search cross validation on the parameters. I used 3 fold cross validation on the dataset, leading to about 1700 samples in each fold. For each fold, 50 parameters are sampled randomly according to the distributions in Table 3 and their performance is compared.

## 2.3 Hyperparameter search

I used an XGBoost classifier for this problem. I initially experimented with support vector machines after reading some research about text classification and how SVMs seem to work well in this setting. However, for my purposes, I found that SVMs took too long to train and had similar results to XGBoost classifiers. In the end I choose XGBoost because of its non-linear decision boundaries, training speed, data efficiency, and ability to handle the large feature space that I had.

To perform hyper parameter search, I followed a similar strategy to the first BoW model, but instead of picking from a fixed list of parameters, I sampled parameters randomly from distributions shown below. I performed 3 fold cross validation with 50 iterations on each fold, each time randomly sampling a parameter from the provided ranges below in Table 3.

To control the complexity of the model, I used cross validation to select several parameters

- **max_depth** - Maximum depth of a tree. Increasing the tree depth makes the model more complex and as a result, more prone to overfitting.
- **learning_rate** - Step size shrinkage used to prevent overfitting. The learning rate determines the contribution of each tree to the final outcome by scaling the weights of the features.
- **colsample_bytree** - Fraction of features to be randomly sampled for each tree during the model's training process. Too high a value can risk overfitting as the model may start to remember the particular noise of the training data and thus not generalize well to new data
- **reg_lamda** - L2 regularization strength parameter. Similar to other L2 regularization punishes high weight values to decrease model complexity.

During the training process, I scored cross validation based on area under the roc curve of the various models because it overall is a good measure of the model's performance and is how the model will be judged on the leader board. There were some issues with convergence and training times when the learning

rate was too low. A smaller learning rate is necessary to prevent overfitting, but too small and the training process takes too long.

| Parameter | range |
|---|---|
| n_estimators | 200-1000 |
| max_depth | 2-10 |
| learning_rate | 0.01-0.29 |
| colsample_bytree | 0.4-0.6 |
| reg_lambda | 0-1 |

Table 3: XGBoost parameters tuned during cross validation.

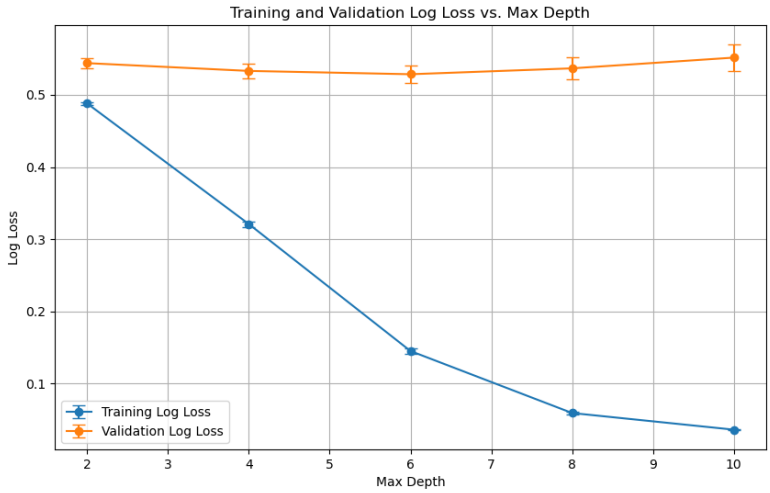


Figure 2: Log loss across different folds of cross validation. From the figure, you can see that increasing the max_depth of the XGBoost trees significantly reduces the training error, bringing it close to 0. At the same time, the validation error is quite steady, perhaps with a best validation error around 6 depth. The results are rather inconclusive to this parameter's effect on the validation error. Because of this, when choosing the best parameters for the model, I chose to go with 3 as it reduced the training time and seems to have a negligible effect on the validation loss.

## 2.4 Confusion Matrix

Below is the confusion matrix of the XGBoost classifier on validation data. To produce this matrix, I followed the same procedure as 1.4; first splitting the data into train and validation (80/20) and then reporting confusion on the 20% validation data.

| | Actual Positive | Actual Negative |
|---|---|---|
| Predicted Positive | 350 | 152 |
| Predicted Negative | 139 | 471 |

Table 4: Confusion matrix for the XGBoost Classifier using BERT embeddings. This confusion matrix looks similar to the confusion matrix in 1.4, but with overall better performance. One key difference in the types of errors that this classifier makes is that it preforms much better on short texts than the BoW logistic regression one. I believe this can be attributed to using the BERT embeddings which capture more context of the text, and the feature selection characteristics of the XGBoost classifier.

## 2.5 Performance on Test Set

Ultimately, I achieved 0.70447 AUROC score on the test set. Unlike the previous classifier, this score is more in line with what I was experiencing during cross validation. During cross validation, I was achieving scores more in the range of 0.70-0.75 for the AUROC. With the last classifier, I attributed the difference in validation AUROC and test AUROC to author's having similar writing styles and using the same words. Perhaps using the BERT embeddings helped add more context which could help prevent a situation like the one described in 1.5.