Classifying Reading-Level Difficulty from Text

Edward Lai

1A : Bag-of-Words Design Decision Description

To convert each passage into numerical features for classification, we used a Bag-of-Words (BoW) representation implemented via scikit-learn's TfidfVectorizer which uses TF-IDF features instead of counts or binary values to improve our AUROC value.

As a preprocessing step, we made all text lowercase using pandas.Series.str.lower() and also setting lowercase=True in the vectorizer. We did not manually remove punctuation, but relied on TfidfVectorizer's built-in tokenizer to handle this. To reduce noise and improve generalization, we excluded stopwords (e.g. "the," "a," "is," "and," and "in") by setting stop_words='english' in the vectorizer, and applied document frequency thresholds (min_df=3, max_df=0.8). This removed very rare terms (which might lead to overfitting) and overly frequent ones (since they won't give us reliable info in the prediction). We set the ngram range to use unigrams to consider each word individually. We also used regex in the token pattern setting to remove non-alphanumeric characters as they wouldn't affect the reading level as much. To ensure that words that appear more frequently don't become more important and have more influence than words that don't, we set sublinear_tf and binary to true.

Our final vocabulary size was **9414** terms. Since TfidfVectorizer builds vocabulary solely from training data, any out-of-vocabulary (OOV) words in the test set are ignored automatically at prediction time.

All components were implemented using existing tools from the **scikit-learn** library.

1B : Cross Validation Design Description

The performance metric we used to optimize on heldout data is AUROC on the train set. To select hyperparameters and estimate generalization performance, we used 5-fold cross-validation via GridSearchCV. The data was randomly shuffled into five folds, each containing approximately 20% of the training set for validation and 80% for training.

After identifying the best hyperparameter (best C value: 0.001) via mean AUROC across folds, we refit a final model on the entire training set with that configuration. This final model was used to generate predictions for the test set. All cross-validation and grid search procedures were implemented using existing tools from scikit-learn.

1C : Hyperparameter Selection for Logistic Regression Classifier

For the classifier we used Logistic Regression with the BoW pipeline from 1A and the cross-validation design from 1B. Logistic Regression performs well on high-dimensional and sparse data, such as Bag-of-Words representations. It provides probabilistic outputs, which can be useful for ranking and threshold tuning, especially when evaluating performance using AUROC. Compared to other classifiers,

Logistic Regression is simpler and easier to interpret, making it a good fit for understanding which words contribute to reading-level classification.

The hyperparameter we were searching for in our classifier was C, which controls the inverse strength of L2 regularization. Smaller C values can lead to stronger regularization and help prevent overfitting, while larger C values can lead to overfitting because the model will fit the training data more closely. We selected a logarithmically spaced grid of 9 values from 1e-4 to 1e4 to capture a wide spectrum from underfitting to overfitting and find the optimal balance.

We used scikit-learn's liblinear solver, which is optimal for small to medium-sized datasets and supports L2 regularization. There were no convergence issues during training. Early stopping was not needed since logistic regression training is convex.

Plot: Mean CV AUROC vs. C

Caption: The best performance was achieved with C=0.001, which produced a mean AUROC of approximately **0.7240** on the heldout set.

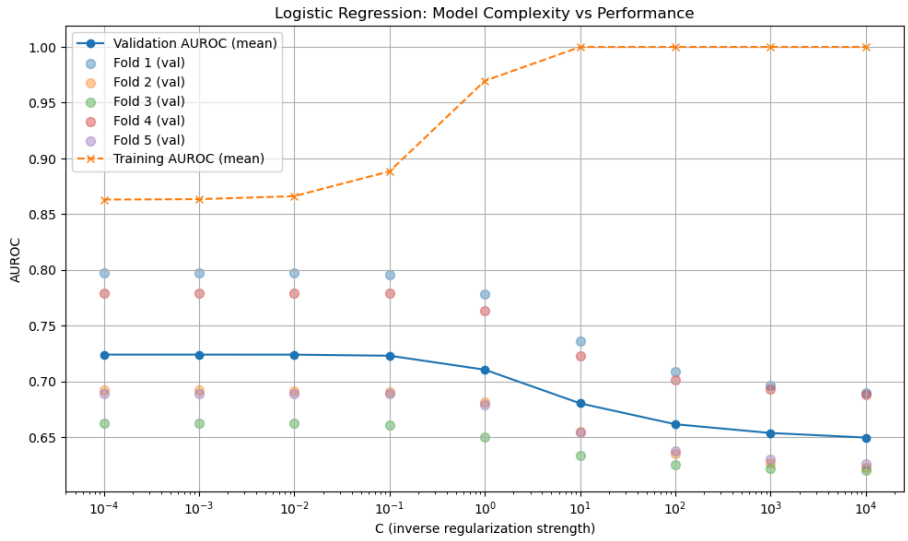


Figure: Held-out AUROC (y-axis) across different values of the regularization hyperparameter C (x-axis). Dots represent AUROC for each validation fold. Line shows average performance. Overfitting is visible at high C values; underfitting occurs at low C values.

1D : Analysis of Predictions for the Best Classifier

Below is a confusion matrix created using 5-fold cross-validation predictions from our best classifier (C=0.001):

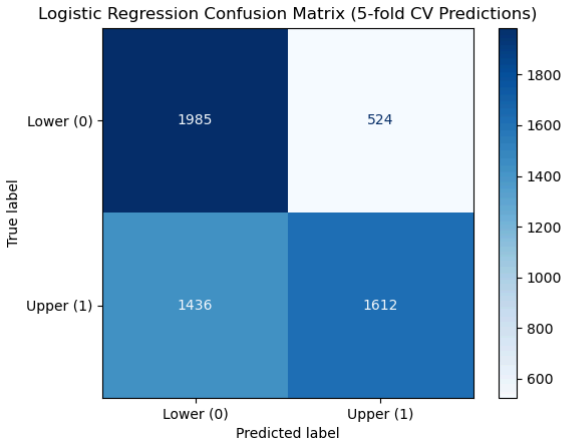


Figure: Confusion Matrix (CV Predictions)

Our classifier showed a strong bias toward predicting the lower-level class over the upper-level class. This can be seen from the confusion matrix, where TN is greater than TP. Our analysis revealed that classifying longer documents was marginally more accurate than classifying shorter documents (~0.02% more accurate) which means that the length of the document doesn't matter too much in how accurate the predictions are. This is because our classifier focuses more on what types of words are present in the text, not the frequency of the words.

When analyzing errors by the author, we observed that the classifier struggled most with literary or abstract writers such as Franz Kafka, Thomas Hardy, and Gustave Flaubert, who write more about complex and metaphorical topics which are supposed to make the reader think about a deeper meaning. Conversely, the classifier performed very well on authors like Lewis Carroll, Hugh Lofting, and Margaret Sidney, who write stories more for younger audiences like children, which makes sense since they often use simpler vocab and grammar, and have a clearer sentence structure that aligns with patterns our classifier can predict on. Overall, our model performed better on simpler material (e.g. children's literature or stories) rather than more metaphorical and complex pieces.

1E : Report Performance on Test Set via Leaderboard

Using the final pipeline with C = 0.001, we retrained the logistic regression model on the **entire training set** and applied it to the test passages from x_test.csv. The predicted probabilities were saved to a one-column plain-text file named yproba1_test.txt and submitted to the leaderboard.

Leaderboard Submission Result:
Our final test set AUROC was **0.67605** on the leaderboard, which is slightly lower than the cross-validation performance on the heldout data (mean AUROC ~0.7240). This slight drop suggests our cross-validation process provided a reasonably accurate estimate of generalization ability, though it may have slightly overestimated due to overlap in style or structure between CV folds. The difference could also stem from stylistic differences between the training and test sets, or from natural variation in passage complexity. Overall, the hyperparameter search and CV design were effective in guiding model selection.

Problem 2: Open-ended challenge

2A : Feature Representation description

For Problem 2, we used BERT-based document embeddings provided in the starter code as our feature representation. Specifically, we loaded from x_train_BERT_embeddings.npz, where each row corresponds to a passage in x_train.csv. These embeddings are generated by a pretrained BERT model. Unlike the BoW approach in Problem 1, which relies on simple word count frequencies, BERT embeddings are dense and use surrounding words to contextualize a word's meaning, so they can generalize better. The embeddings were used directly as input to our classifier without any additional preprocessing or feature engineering, allowing us to take advantage of BERT's expressive power out of the box.

2B : Cross Validation (or Equivalent) description

To evaluate generalization performance and tune hyperparameters, we used RandomizedSearchCV with 5-fold stratified cross-validation to ensure balanced label distribution in each fold. In contrast to problem 1B, we opted for randomized search over grid search to reduce training time. Grid search might have provided more exhaustive results, but it takes much longer. All model development and tuning were conducted exclusively on the training data (x_train.csv and y_train.csv). After selecting the best hyperparameters, we retrained the model on the full training split and evaluated it on a separate 20% held-out validation set that was not used during training or tuning. This setup allowed us to obtain a more reliable estimate of the model's performance on unseen data using AUROC.

2C : Classifier description with Hyperparameter search

We selected a random forest classifier (using the RandomForestClassifier from scikit-learn) due to its speed, effectiveness, and interpretability with dense feature vectors like BERT embeddings.

We tuned the max_depth hyperparameter, which controls tree complexity and influences overfitting. A smaller max_depth can reduce overfitting but may underfit the data, while a larger max_depth can have poor generalization and overfit the data. Our search used RandomizedSearchCV with 5-fold stratified CV, exploring the following grid:

```
param_grid = {
   'max_depth': [1, 2, 3, 4, 5, 10, 15, 20, 30, None],
   'n_estimators': [300],
   'max_features': ['sqrt'],
   'min_samples_split': [2],
   'min_samples_leaf': [1]
}
```

We used 10 iterations of randomized search and selected the configuration that maximized AUROC on the validation folds. The best model achieved a validation AUROC of **0.7262**. We visualized how max_depth affects performance below.

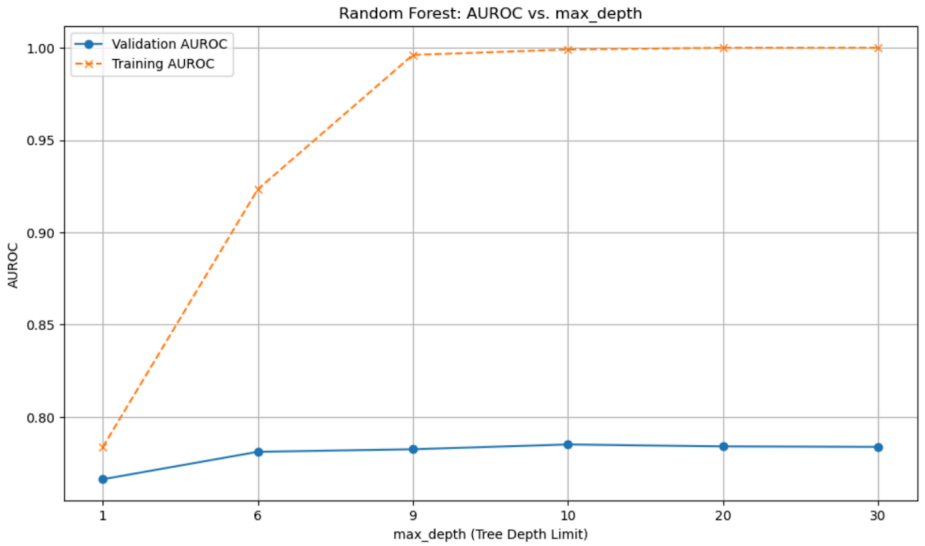
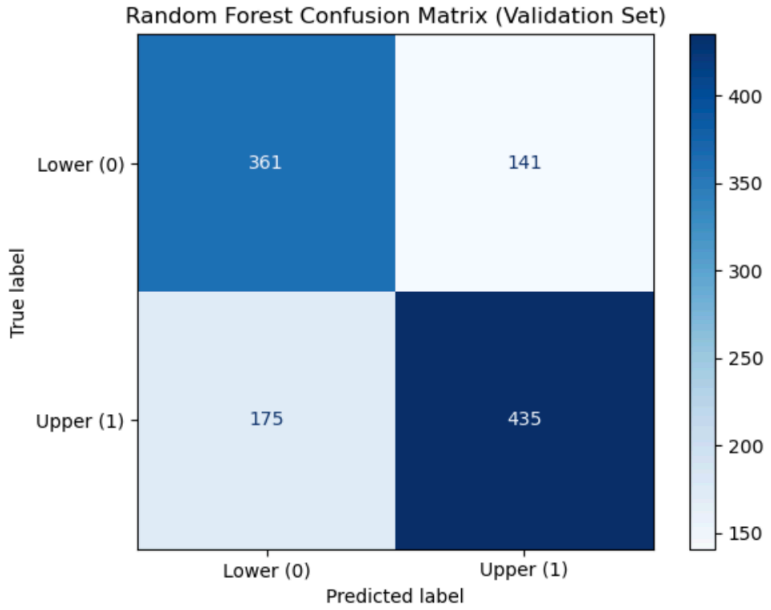


Figure: Training and validation AUROC across different max_depth values. Training AUROC rises with complexity, while validation peaks then declines, indicating overfitting.

The plot shows that validation AUROC peaks around max_depth=10 before slightly declining, indicating overfitting at deeper depths. This confirms that tuning max_depth is essential for balancing bias and variance in random forest models trained on BERT features.

2D : Error analysis



The confusion matrix shows that the classifier slightly favors the upper-level class, since TN is greater than TP. This differs from the Problem 1 model, which under-predicted the upper class. The change may reflect the improved representation power of BERT and contextualization, allowing the classifier to better identify complex passages. The validation set TPR (recall for upper-level) was **0.713**, while the TNR (recall for lower-level) was **0.674**, showing a bias toward identifying difficult texts. This confusion matrix is not directly comparable to Problem 1's, since it was generated from a held-out validation split rather than a cross-validation fold.

2E : Report Performance on Test Set via Leaderboard

Using the best configuration from our cross-validation procedure, we retrained the RandomForestClassifier on the full training set and generated probabilistic predictions for the test set using the BERT embeddings. The predictions were saved to yproba2_test.txt and submitted to the leaderboard.

Our final test set AUROC was **0.71989** on the leaderboard, which represents a strong improvement over our Problem 1 model (AUROC = 0.67605) which used TfidfVectorizer and LogisticRegression. The primary reason for this improvement is the shift from sparse BoWs to dense BERT embeddings, which encode deeper semantic and syntactic cues. Random Forest is also better suited to the high-dimensional, contextualized BERT vectors than linear models like LR.

From this project, we've learned about the many different ways to improve predictions, from using different classifiers, tweaking classifier settings, and using different features. Part 2 showed us specifically how better features can have a greater impact on model performance than changing the model, since our score improved more when using the BERT dataset over the regular x_train dataset than when we were tweaking the model settings and type.