Problem 1

1a Bag-of-Words Decision Description

We randomly split the data from `x_train.csv` 80/20 for training and final testing. Our pipeline uses CountVectorizer for feature extraction and LogisticRegression to train the model.

We used word counts for a more informative representation than binary values and omitted stop words, which do not contribute to passage complexity. L2 regularization, unlike L1, does not remove features but shrinks less important weights toward zero, helping prevent overfitting. Therefore, to reduce outliers, we set a minimum word frequency of 4 to filter rare words. We stripped accents to improve consistency across word variations. We ignore out-of-vocabulary words, as they do not have associated weights in our model. This prevents arbitrary importance, reducing noise and improving generalization. Due to the random 80/20 split of the data, the final vocabulary size ranges from around 6340 to 6400. We chose L-BFGS as our solver because it is computationally efficient and well-suited for large feature spaces. We selected L2 regularization because it ensures smooth, gradual weight adjustments, performs well with high-dimensional data, and is fully supported by L-BFGS, unlike L1 regularization.

1b Cross Validation Design Description

Within classifier training using sklearn's LogisticRegression, we used cross validation with 10 folds, creating shuffling before we create a fold by using StratifiedKFold with a pseudorandom number generator. This random folding is necessary, as in the provided training data, texts are primarily organized alphabetically, placing all texts from a particular author or book next to one another. Therefore, if we simply took a contiguous group of examples, we might make the model bias towards one particular text or author. Each fold is approximately 444 examples long. We evaluated the best model best on its AUROC on its associated validation set. At the end of cross validation, the model was refit using the selected hyperparameters (of best AUROC) and the whole dataset.

1c Hyperparameter Selection for Logistic Regression Classifier

The chosen classifier for this text classification is a logistic regression model, which is good as it can very simply classify data into binary labels. For this model, we performed a GridSearch on C, the inverse L2 regularization strength in `LogisticRegression`. We defined the search space as $\{0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000\}$. This explores the transition between underfitting (decreasing test error, decreasing train error) and overfitting (increasing test error, decreasing train error) well because the lowest element is a very slight penalty, and the highest element is a very strong penalty. So, we can assert that slight penalties do, indeed, cause underfitting, and that strong penalties cause a great amount of overfitting. We also logarithmically spaced the parameter values to ensure that we observe error across a large range of magnitudes, while saving time by not searching for parameters that would hardly change error (like in a linear range of parameters). We are using L-BFGS to maximize AUROC and minimize log loss, a necessarily convex function. So, we do not have to worry about reaching a relative, but not absolute, minimum.

We can see that at $C = 0.1$ (-1 on the x-axis), we have the lowest mean test error, and a fairly low mean train error. So, we prefer this hyperparameter. The evidence is decisive since there is low variability

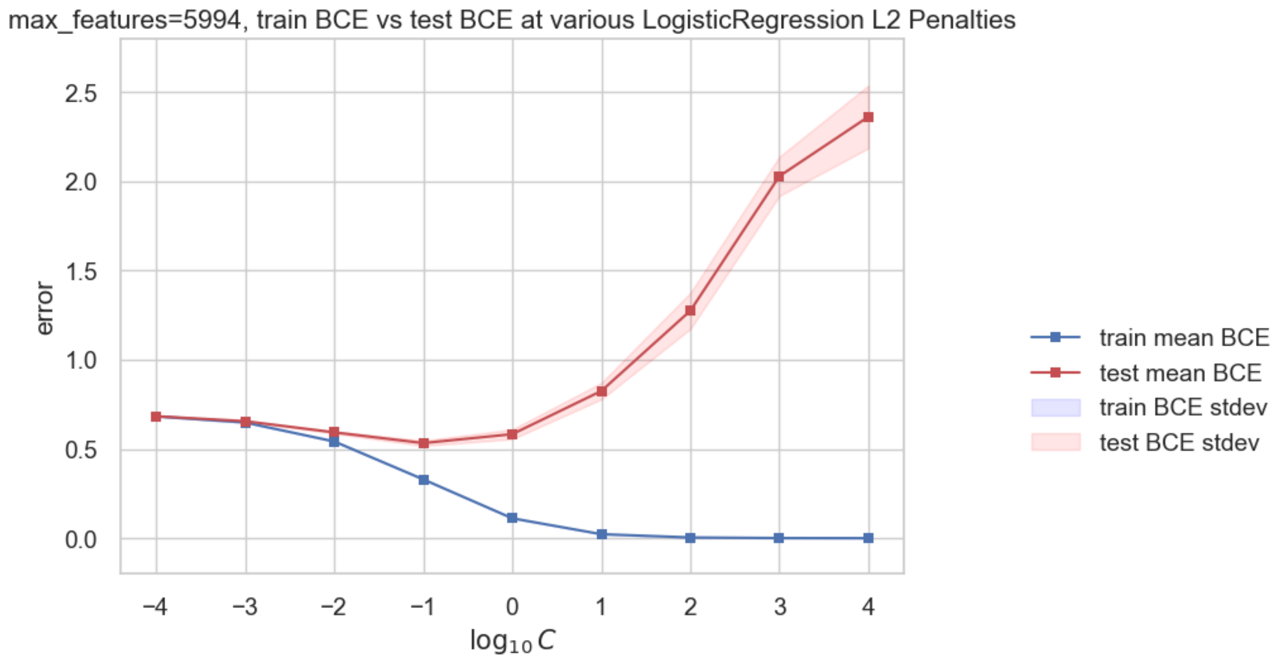

Figure 1: Train vs. test error performance averaged across 10 folds with residuals included

(i.e. not much area shaded around the mean curve). So, across all folds this hyperparameter value yields a low test error and a low train error, and boasts good performance.

1d Analysis of Predictions for the Best Classifier

Predicted	0	1
True		
0	178	36
1	17	214

Figure 2: Confusion matrix for heldout error during fold 4 for best estimator

During this fold, the authors with the most amount of false negative predictions were those like Charles Dickens and H.G. Wells. Upon inspecting the data written by Charles Dickens, we found that his sentences were very long, and therefore would be more complicated to understand (i.e. have a higher reading level; see Figure 3). BoW representation does not take into account sentence length, however, which could be improved by some processing like BERT that takes into account context surrounding words. By introducing this ability, the model would perform better in long sentences since it would be able to track dependencies between words, which are key to understanding longer sentences. By understanding context between words, the model might also take into account complex sentence structure, such as the structure of a sentence that contains several dependent clauses. Thus, by using BERT, the model might see improved performance in documents in common, simple words in complex sentence structures.

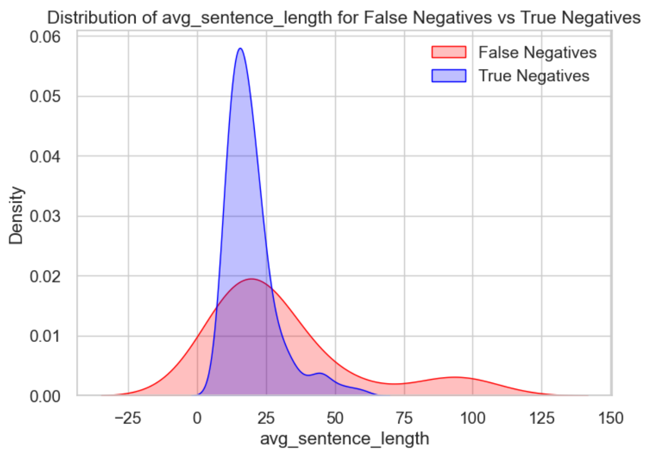

Figure 3: In this distribution, we see that many false negatives (i.e. in reality, high level passages) have a high mean sentence length, and likely complex.

1e Report Performance on Test Set via Leaderboard

Our final test set AUROC was 0.658, while our predicted performance was 0.790, resulting in a difference of 0.132. This discrepancy may stem from the way our test and training sets were split from the same initial dataset, potentially reducing generalization. Specifically, if passages from the same source appeared in both sets, the model may have learned patterns specific to those works rather than the broader reading-level distinctions present in the ultimate test set.

Problem 2

2a Feature Representation description

Text from `x_train.csv` was encoded using the provided BERT embeddings, which represent each document as a fixed 768-dimensional vector of float64s. BERT can understand word relationships, complex sentence structure, and alternate senses of words. There is no need to discount words with lower frequencies or omit stop words, as in Problem 1, as BERT embeddings reflect which terms contribute more to their sentence's meaning. We performed an 80/20 split of the training data, just as we did in Problem 1, to later assess heldout error more effectively.

2b Cross Validation description

After partitioning BERT embeddings into train and test data, we scaled each column in each example to mean 0, and variance 1. This was done to ensure that each feature was given fair input to the next step in the training process, and there was not a bias to train based on features with a higher overall magnitude than others.

Then, we passed these scaled vectors to sklearn's `MLPClassifier` with two hidden layers of sizes 512 and 256, aiming for a balance between model complexity and performance. We selected ReLU as the activation function due to its efficiency and ability to mitigate vanishing gradient issues. We used the Adam optimizer instead of stochastic gradient descent (SGD) or LBFGS, as Adam combines momentum and adaptive learning rates, which can lead to faster convergence compared to standard SGD. However, while Adam often accelerates training, it may not always generalize better than SGD. To further optimize training, we enabled early stopping, an adaptive learning rate, and increased the tolerance from the default (0.0001 → 0.01) to prevent overfitting and reduced unnecessary computations. We also shuffled the examples to remove patterns in the features (since similar documents might live near each other).

We used a similar strategy to our CV for our Bag-Of-Words descriptor, as described in 1B. We used a GridSearchCV with 10 random folds, evaluating heldout performance using AUROC. Each fold was about 444 examples long. At the end, we refit our model to fit the hyperparameters with the best heldout AUROC.

2c Classifier description with Hyperparameter search

We chose an MLP classifier due to its ability to learn nonlinear relationships in BERT embeddings, making it well-suited for text classification. Unlike linear models, an MLP can capture complex interactions between features, which may improve generalization to unseen texts. According to the Universal Approximation theorem, with enough hidden layers and nodes, an MLP can approximate any function, no matter how complex.

The key hyperparameter we searched over was α, which controls the strength of L2 regularization. Lower α values allow for more complex models that may fit training data well but risk overfitting, while higher α values enforce stronger regularization, reducing variance at the risk of underfitting. To systematically explore this trade-off, we searched over seven α values: 0.0001, 0.002154, 0.04642, 1.0, 21.544, 464.158, 1000. For results, see Figure 4.

During training, we used early stopping to prevent overfitting and improve efficiency. We also selected the Adam optimizer, which adapts learning rates and incorporates momentum, helping to speed up convergence compared to standard SGD.

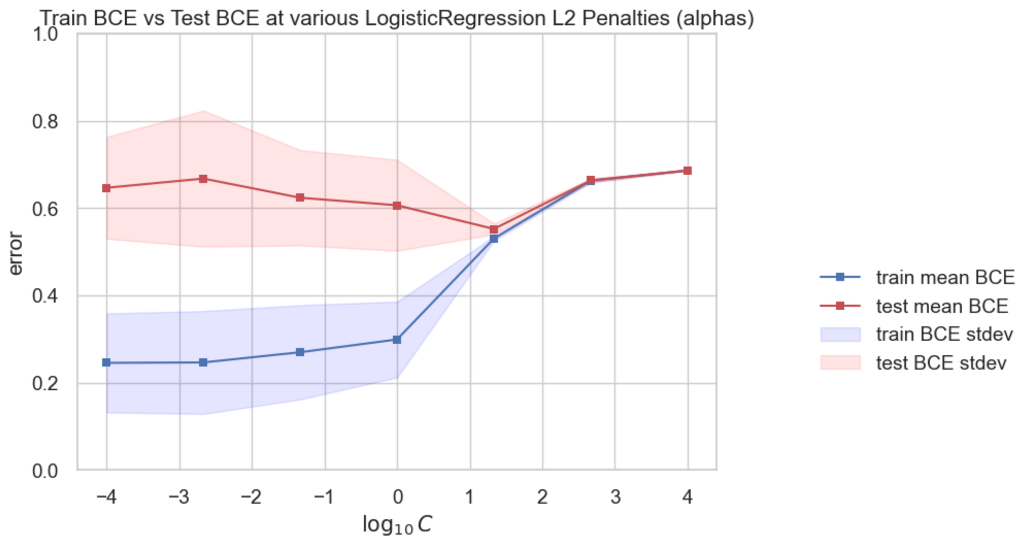

Figure 4: In our figure, we observe signs of overfitting until $\alpha = 21.544$, where test and train BCE converge. Beyond this point, the test and train BCE values align closely, with minimal variance in both cases. Thus, $\alpha = 21.544$ provides the lowest error without overfitting, making it the optimal choice for our model.

2d Error analysis

Predicted True	0	1
0	142	59
1	66	178

Figure 5: Confusion matrix for heldout error during fold 4 for best estimator. Note that this is the same data inspected in Problem 1d, as the random state initializer has remained the same.

We found that in Problem 1d (see Figure 3) that for documents with longer sentences, the classifier was incorrectly reporting them as low level. This was a limitation of the context-blind BoW representation, which is no longer the case with the BERT representation. Indeed, we see by Figure 6 that this model does not fall victim to incorrectly classifying documents comprised of simple words, but organized in complex sentence structures (i.e. works of Charles Dickens).

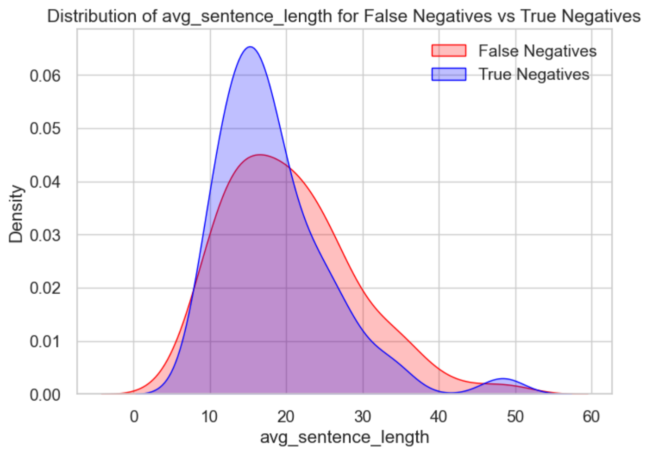

Figure 6: In this distribution, we see that not that many false negatives have high mean sentence length, relative to the amount of true negatives that do.

Upon inspecting which authors the model (incorrectly) identified as positive, we found that M.G. Lewis was the most common author where this happened. After observing his text in the database, we found that his text often relies on polysyllabic, but not complex words. So, we found that the Simple Measure of Gobbledygook (SMOG) Index, which is higher for documents with more polysyllabic words, had a difference in false positives vs. true positives (see Figure 7). In the figure, documents with a higher SMOG indexes are more likely to be incorrectly predicted as positive. Otherwise, we did not find any other columns from `x_train` with such a noticeable difference in correctly identified and incorrectly identified classes.

2e Report Performance on Test Set via Leaderboard

Our predicted AUROC for our model was 0.794, as compared to our BoW predicted AUROC of 0.790, and our actual performance on the test set was 0.737, as compared to 0.658 for our BoW representation 0.132. This difference may result from similar generalization errors as discussed in Problem 1e.

Our model in Problem 2 out performed that of Problem 1 due to our shift from a BoW representation

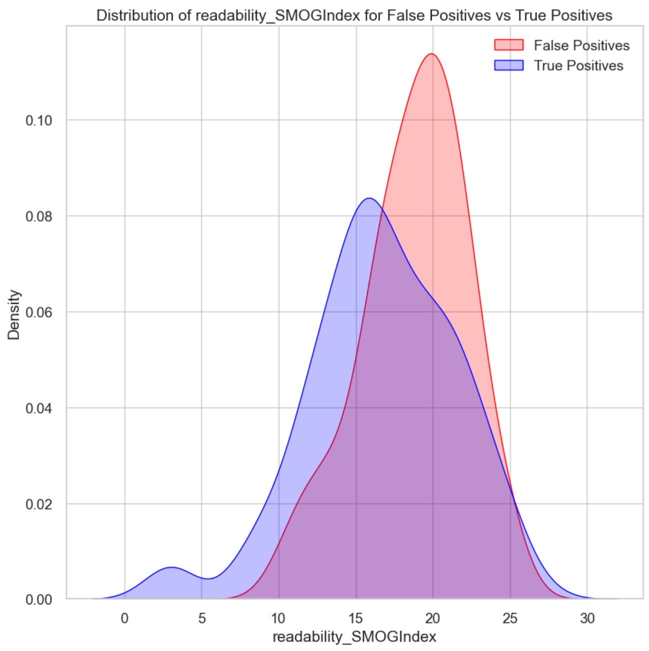

Figure 7: In this distribution, we see that many false positives (i.e. in reality, low level passages) have a high SMOG index, but are not complex.

to BERT embeddings, and proper usage of an `MLPClassifier` in place of `LogisticRegression` to perform classification. We improve the model's ability to classify the reading level by leveraging BERT embeddings' proven ability to encode sentence complexity, and capturing context between words. Our use of `MLPClassifer` utilizes the result of the universal approximation theorem. That is, we are able to learn nonlinear decision boundaries, unlike `LogisticRegression`. So, the model can learn nonlinear relationships between the linguistic features in BERT, providing better performance.