Project A
Shepard Rodgers, Alex Williams-Ferreira

Problem 1:

(a) For our BoW feature representation, we chose to break up the given passages into cleaned word tokens. Our definition of a cleaned token is a single word set to lowercase and cleaned of punctuation and white space. We tried using scikit-learn's list of English stop words as well as a more limited list to restrict our vocabulary, but we found that our model performed worse on held-out data. We thought this might be because common stop words, particularly their frequencies, were valuable to determining reading level. Thus, we decided to include all words from the training corpus in our vocabulary. Likewise, when encountering out-of-vocabulary words in the test set, we chose to ignore them because we did not see a clear correlation between out-of-vocabulary words and reading level, and we wanted to keep the model simple rather than introduce further complexity with unknown words. We used counts rather than binary values to retain the frequency information about words in the dataset. We concluded that higher-level passages would likely feature complex words at higher rates than lower-level passages. The total size of our vocabulary was 23971 words. We used scikit-learn's CountVectorizer and GridSearchCV to handle the main aspects of our BoW setup. We did not rely on other libraries for our implementation.

(b) The performance metric our search was optimized on was the roc_auc_score. We chose to run 5-fold cross validation. The total number of samples in the training set is 5557, so the size of each fold is around 1111 items. Our method does not split folds at random, we used sklearn's StratifiedKFold, which ensures that each fold has approximately the same proportion of class labels as the entire dataset. We chose to use this method because it can help improve the generalization of our model. In order to perform tests while limited to the x_train and y_train data we split the training datasets into testing and training subsets. This meant that we were not training on our entire dataset and therefore losing out on potential accuracy and the chance to reduce over-fitting. For our final model this split step can be removed and the model can be trained on all the available training data. To create our final model we trained on our best fixed hyperparameter configuration and used the entire provided train dataset.

For the most part we used off-the-shelf libraries within sklearn for data aggregation, data splitting, training, and metric parsing. This included sklearn's CountVectorizer, Logistic Regression, GridSearchCV, StratifiedKFold, classification report, etc..

(c) For our experimental design we tested on multiple hyperparameters and hyperparameter variations. Our parameter grid included testing over several different ngram ranges, minimum document frequencies, maximum document frequencies, stop word lists, and regularization strengths (C).

```
param_grid = {
    'vect__ngram_range': [(1,1), (1,2), (1, 3), (5,5)],
    'vect__min_df': [1, 2, 4],
    'vect__max_df': [0, .4, 0.6, 0.9, 1],
    'vect__stop_words': ['english', None, stop_words.txt],
    'clf__C': np.logspace(-4, 4, 9),
}
```

We focus on vectorization parameters and classifier regularization. Specifically, we vary `n-gram range` (`vect__ngram_range`), testing unigrams, bigrams, trigrams, and exclusive 5-grams to capture different contextual granularities. We explore `minimum document frequency` (`vect__min_df`) at 1, 2, and 4 to control the minimum occurrence of words included, and `maximum document frequency` (`vect__max_df`) from 0 to 1, filtering overly common words that may not help the determination process and increase run time. The impact of stop word removal (`vect__stop_words`) is tested using standard English stop words, a custom list, and no stop word removal. For the classifier, we tune `regularization strength` (`clf__C`) over a logarithmic scale from 10^{-4} to 10^{4}, allowing exploration of a wide range of penalty strengths for balancing model complexity and generalization. Often time these parameters were not searched all-together as to make search time more reasonable on our hardware. This grid search approach balances computational feasibility and comprehensive coverage, aiming to find an optimal configuration that generalizes well to unseen test data.

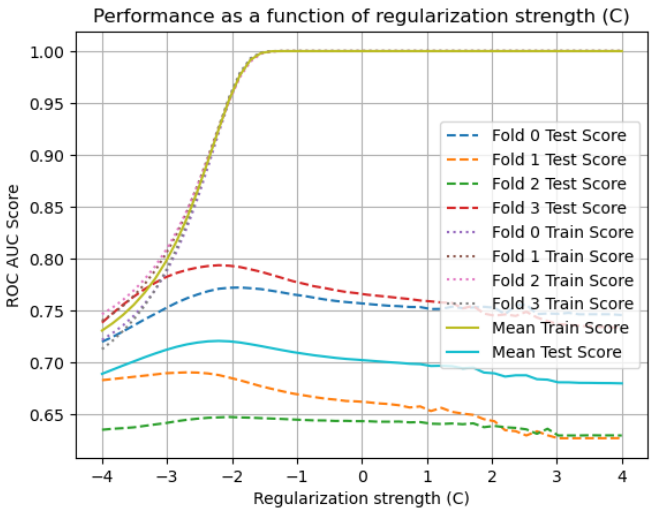
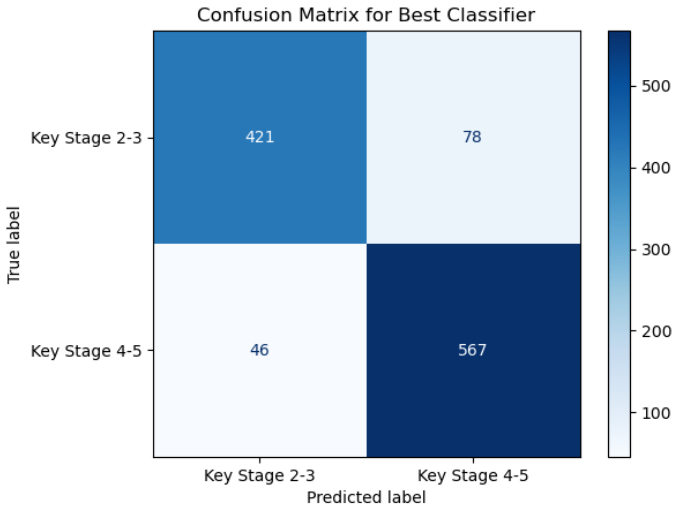


Figure 1: Performance as a function of regularization strength (C) in Logistic Regression.

The plot illustrates the effect of varying the regularization strength C on the ROC AUC score across different cross-validation folds. The training scores (dotted lines) increase as C grows, indicating that the model becomes more flexible and better fits the training data. However, test scores (dashed lines) initially improve, peak around $\log_{10} C \approx -2$, and then decline as the model starts overfitting at high values of C. The mean test score (solid blue line) suggests that moderate regularization provides the best generalization, while too much regularization (small C) results in underfitting, and too little regularization (large C) leads to overfitting.

We found the best parameters from our hyperparameter search to be: $\{clf\_\_C : 0.01, vect\_\_max\_df : 0.75, vect\_\_min\_df : 1, vect\_\_ngram\_range : (1,2), vect\_\_stop\_words : None\}$. In order to estimate the heldout error of this configuration, we used two different approaches. The first was to set aside a subset of the training data for testing our "best" model. This approach gave heldout error of approximately 0.11331 or 11.331%. Calculating the estimated accuracy of our classifier, we get 0.88849 or 88.849%. However, we also recorded the average validation error given by the cross-validation process, which ultimately decided which hyperparameter configuration we would choose. This CV error was approximately 0.29456 or 29.456%. Calculating the estimated accuracy of our classifier, we get 0.70544 or 70.544%. This is a notably lower, but perhaps more reliable, accuracy.

(d) Confusion matrix:



Our model appears to favor the Key Stage 4-5 positive class over the Key Stage 2-3 negative class, which we can observe in the relatively high False Positive Rate of 0.156 (when compared to the False Negative Rate of 0.075). This implies that we incorrectly identify more documents as belonging to the Key Stage 4-5 class than

the Key Stage 2-3 class. This might be explained by the fact that complicated documents share many of the "simpler" words seen in the 2-3 class. In this case, our model might be inclined to assign the positive class given test examples because it will come across many words found in both labels of the training documents, making it more sensitive to any "complicated" words, or those appearing in the 4-5 class for the training set. This is visualized as a set diagram in Figure 2.

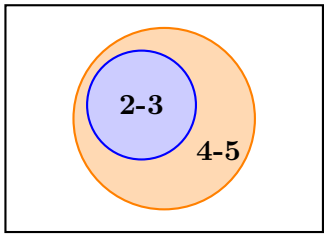


Figure 2: The blue circle represents words commonly found in the 2-3 class documents, while the orange circle represents words commonly found in the 4-5 class documents. Given the overlap, we can speculate that our model might be inclined to incorrectly identify new 2-3 class documents as 4-5 class documents.

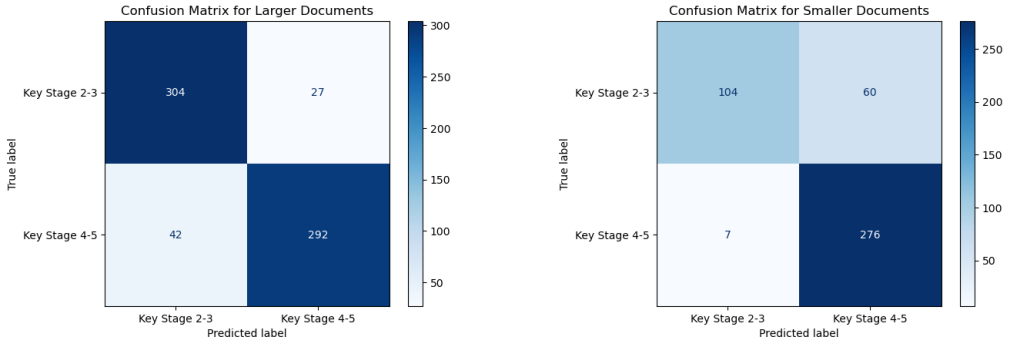


Figure 3: Confusion Matrices for Models isolated to Above and Below mean character counts

After creating our general confusion matrix, we chose to modify our input data by restricting the model to either the longer documents (those with character counts above the mean count) or shorter documents. We did this to see if we could isolate results for the two subsets of documents. From this isolation we found that the the classifier performs better on longer documents, as seen from the higher true negative rate (91.8% vs. 63.4%). It struggles with shorter texts, often misclassifying Key Stage 2-3 as Key Stage 4-5 due to a lack of distinguishing features. For longer documents, it correctly identifies Key Stage 23 more often but misclassifies more Key Stage 45 texts. This suggests the model relies on richer context for accurate classification. Future improvements could refine feature extraction for short texts and adjust thresholds to balance errors across both categories.

(e) Our final classifier for Part 1 achieved an AUROC score of 0.6983 on the leaderboard test set, with a balanced accuracy (BAL_ACC) of 0.63516. Compared to our estimated test set accuracy of 0.88849 from part 1C, there is a significant discrepancy. However, the cross validation accuracy we reported in 1C of 0.70544 is much closer to the test set performance. This makes sense because cross validation gives a better estimate of heldout error by averaging across folds. The subsets we set aside for testing and validation of our model may have represented the training data too closely, which could explain the difference between our estimated and actual accuracy on heldout data. Consider, for example, the fact that the train and test data have no overlap in authors or titles.

Problem 2:

(a) For part 2 of this project our chosen feature representation was a concatenation of the given BERT embeddings, our own Tfidf text vectorizations, and several of the provided non-text based features from the provided x_*.csv's. We included a majority of the given features. More specifically, we included every feature between column 7 and 23 (inclusive). We chose to disregard the first 6 columns because they focused on the length of the passages themselves, which were not clearly related to reading complexity. We chose to exclude the columns after the 23rd because we didn't consider them to be relevant to the classification problem. The middle section contained a lot of data that broke apart the complexity of the passages and several readability measures ran against them. To us, these data points all seemed reasonable to feed to our MLP model, as they all provided related variables. Unlike part 1, we did not do any text cleaning because all the BERT embeddings were preprocessed and we chose not to for the tfidf vectorization because we wanted to explore how the MLP classifier would handle the details of the full passages, including punctuation.

(b) (Very similiar to 1b) The performance metric our search was optimized on was the roc_auc_score. We chose to run 5-fold cross validation for each hyperparameter configuration (given in next section). The total number of samples in the training set is 5557, so the size of each fold is around 1111 items. Our method does not split folds at random, we used sklearn's StratifiedKFold, which ensures that each fold has approximately the same proportion of class labels as the entire dataset. We chose to use this method because it can help improve the generalization of our model. In order to gather metrics on our best cross-validated configurations for the x_train_BERT_embeddings and x_test_BERT_embeddings, we split the training datasets into training and heldout testing subsets. This meant that we were not training on our entire dataset and therefore losing out on potential accuracy and the chance to reduce over-fitting. To create our final model we trained on our best fixed hyperparameter configuration and used the entire provided train dataset.

For the most part we used off-the-shelf libraries within sklearn for text vectorization, data splitting, data scaling, training, and metric parsing. This included sklearn's TfidfVectorizer, MLPClassifier, GridSearchCV, StratifiedKFold, StandardScaler, and classification report.

(c) We chose to use scikit-learn's Multi-Layer Perceptron (MLP) classifier for this problem, motivated by the fact that we are using stacked feature vectors with many variables which could have complex, non-linear relationships and decision boundaries for classifying the text. The MLP classifier is able to learn these relationships with a large amount of flexibility. Using the Cross-Validation procedure above, we trained the model using sk-learn's "fit" method. To optimize our hyperparameters for the MLP, we experimented with both Randomized Search and Grid Search. We found that for experimentation with new hyperparameters, Randomized Search was faster. However, for our final model we decided to use Grid Search for a more comprehensive search. Grid Search also made more sense because many of our hyperparameters were discrete values and could not be sampled from a range. Our parameter grid included testing over

```
param_grid = {
    'hidden_layer_sizes': [(100,), (50, 50), (100, 50)],
    'activation': ['relu', 'tanh', 'logistic'],
    'solver': ['lbfgs', 'adam', 'sgd'],
    'alpha': [0.0001, 0.001, 0.01, 0.1],
    'max_iter': [1, 10, 50, 100, 1500, 2000, 2500],
}
```

We explored the max_iter hyperparameter, which controls the maximum number of optimization iterations during training. This is crucial because too few iterations can lead to underfitting (failure to learn meaningful patterns), while too many can result in excessive computation time and potential overfitting. We conducted a grid search over max_iter values: [0, 500, 1000, 1500, 2000, 2500], ensuring we covered cases of underfitting, optimal learning, and possible overfitting.

The training accuracy increases with max_iter, but validation accuracy slows around 200 and then plateaus at 1500, indicating diminishing returns. Beyond this point, additional iterations do not significantly improve performance, suggesting the model has reached a convergence point. The spread across cross-validation folds

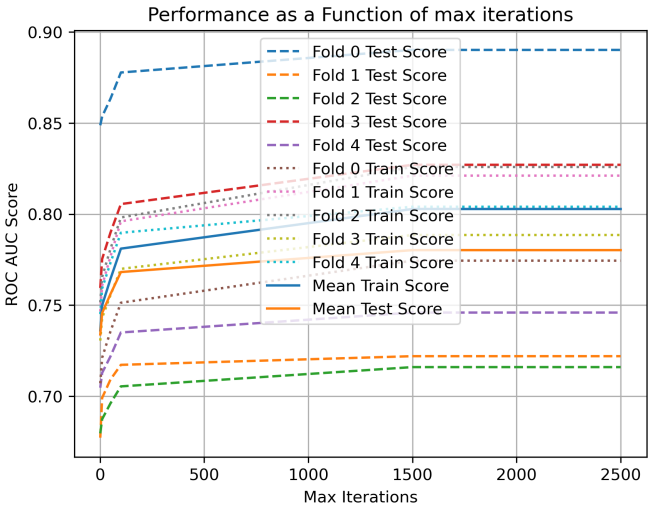
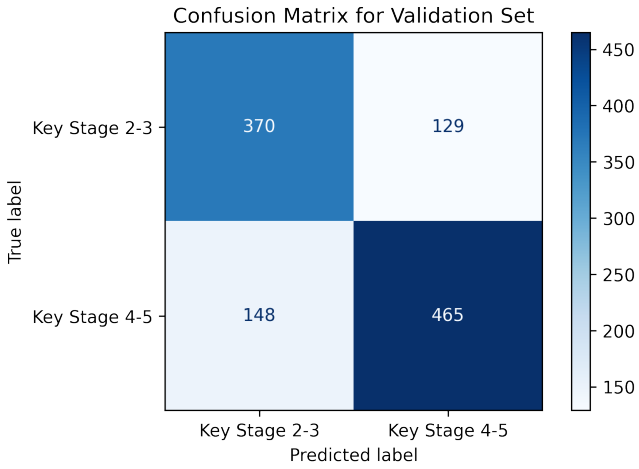


Figure 4: Performance as a function of max iterations during MLP.

remains small, indicating stable results. Based on this analysis, max_iter = 1500 is the preferred value, providing a balance between convergence and computational efficiency. The evidence is strong, as higher values show minimal additional gains. Confidingly, this is the value that the cross-validation chose to be optimal.

We found the best parameters from our hyperparameter search to be: $\{activation : logistic, alpha : 0.01, hidden\_layer\_sizes : (100,), solver : sgd\}$. The cross-validation approach given above estimated an average validation accuracy of 0.82415 or 82.415%, with an estimated validation error of 0.17585 or 17.585%.

(d) Confusion Matrix



This confusion matrix is not directly comparable to the one from Problem 1c:

i. Different feature representations: Problem 1 used a Bag-of-Words (BoW) approach, while Problem 2 used a combination of BERT embeddings, TF-IDF features, and selected readability metrics.

ii. Different model architectures: Problem 1 employed logistic regression, while Problem 2 used a Multi-Layer Perceptron (MLP) neural network classifier.

iii. Different feature preprocessing: Problem 2 used pre-processed BERT embeddings and additional meta-data features (columns 7-23) that weren't available/used to the Problem 1 model.

iv. Different modeling capabilities: The MLP can model non-linear relationships between features, whereas logistic regression is limited to linear decision boundaries.

While both confusion matrices evaluate reading level classification performance on the same dataset, the underlying approaches differ substantially in both feature representation and modeling technique. Therefore, differences in performance cannot be attributed solely to model quality but reflect the combined impact of different features and classification approaches.

The model correctly classified 370 Key Stage 2-3 texts and 465 Key Stage 4-5 texts, while misclassifying 129 Key Stage 2-3 texts as more advanced and 148 Key Stage 4-5 texts as simpler. Unlike the logistic regression model in Problem 1, which showed a bias toward the Key Stage 4-5 class with a higher false positive rate, this MLP classifier demonstrates a more balanced error distribution with similar misclassification rates in both directions. This suggests the richer feature representation and neural network's ability to model non-linear relationships between features has led to more consistent performance across reading levels. However, the model still struggles with approximately 25% of texts overall, likely representing borderline cases where the reading complexity metrics and linguistic features show characteristics of both reading level categories. The improved balance in errors compared to Problem 1 indicates that the additional features beyond word frequency patterns help the classifier make more nuanced distinctions.

(e) Our final classifier for Part 2 achieved an AUROC score of 0.72493 on the leaderboard test set, with a balanced accuracy (BAL_ACC) of 0.67433. Compared to our estimated test set accuracy of 0.82415 from part 2C, there is a discrepancy of about 0.15 between our estimated accuracy and actual test accuracy. Interestingly, this estimation overshot more than the estimation from part 1. This could be because BERT embeddings model similar relationships that exist in different folds. If structurally similar sentences appear in both a training and validation fold, the MLP might take advantage of these similarities, leading to overly optimistic estimates for the accuracy.

Across all metrics on the leaderboard test set, our MLP classifier for Part 2 outperformed our Logistic Regression classifier from Part 1. For some metrics, such as AUROC, BAL_ACC, and F1 Score, the improvement is significant, while for others such as True Positive Rate, they perform almost exactly the same. We can explain the overall better performance of the classifier from Part 2 using a few different ideas:

i. The largest change in these two models is the feature vector representations that they are fed to make their predictions. For the logistic regression classifier, the Bag of Words representation imposed limits on the model's contextual awareness because it only provided independent word counts. In comparison, the MLP classifier was able to leverage contextual relationships using the provided BERT embeddings, TF-IDF vectorizations, and readability metrics for each passage. This additional information provided a stronger starting point for the classifier.

ii. Of course, there is the notable difference in classifier structure. The logistic regression approach in Part 1 is limited in that it can only model linear relationships in the data. The MLP provides the advantage of being able to model flexible decision boundaries, which we would expect to arise with the use of the complex contextual data described above. This more flexible model required us to pay more attention to the risk of overfitting. We also had more hyperparameters to play with when building our MLP classifier.

iii. The stratified k-fold CV improved upon the version in Part 1 by balancing the proportions of the positive and negative class in each fold, allowing for a better relative estimate of accuracy.

This project, especially Part 2, demanded a lot of experimentation. We certainly did not achieve our best classifier on the first try, and while experimenting with new technologies and techniques such as BERT embeddings, random forest classifiers, randomized hyperparameter search (not to mention the multitude of hyperparameters), and stratified k-fold CV, we learned a lot about building a classifier from a large, unfamiliar toolbox. We also learned the importance of balancing optimization with availability of computing power, since we did not have an unlimited amount of time to reach our targets. It is interesting how seemingly vast structural changes in a model can often result in just marginal increases (or even unexpected decreases) in prediction accuracy. At times this uphill battle was frustrating but other times it was rewarding. In the end, the calculated decisions we made about which approaches to take for representing our features, CV, and hyperparameter search allowed us to build a model with competitive performance, which was a gratifying place to end up.