1A : Bag-of-Words Design Decision Description

Firstly, I performed different data cleaning methods as below:

- Lowercasing, all tokens are covereted to lowe case and sklearn verstorizer also lowercase by default. Reduces the possibility of having duplicated tokens such as "The" and "the. - Contraction expansion. Convert "can't" → "cannot", "I'm" → "I am", etc. Rationale: keeps consistent token forms and reduces noisy tokens.
- Punctuation removals. Removed common punctuation characters (e.g. ?, !, ., ", /, _), since it rarely helps with for this task and creates token fragmentations.

After the cleaning of the data, I came to find the final vocab list. There are couple methods that I used to determine the final vocab_list:

- Manual BoW exploration: vocab_list = [w for w in sorted_tokens if tok_count_dict[w] >= 4] — this was used for exploratory dense-count feature experiments.
- Removing tokens that occur <4 times reduces noise and keeps feature count manageable when we build dense arrays manually.

With all those applications, the final vocabulary size for each fold (5 folds CV) is around 23,600 - 23,800) details as below:

```
Fold 1 vocab size: 23830
Fold 2 vocab size: 23856
Fold 3 vocab size: 23648
Fold 4 vocab size: 23846
Fold 3 vocab size: 23648
Fold 4 vocab size: 23846
Fold 5 vocab size: 23685
```

Those the manual BoW experiments used raw counts from transform_text_into_feature_vector(). However in the final pipeline, we used TfidfVectorizer. I chose this is because it commonly gives better performance than raw counts for logistic regression on text classification, because it downweighs ubiquitous words and emphasizes discriminative words.

I also implemented a OOV rate check at the end. Just to make sure we do not have too high of a OVV rate, that our token would still have a valid predication.

The result came to be OVV ~ 5%. which is considered low in comparison with the total token numbers.

Details output as below:

```
{'total_tokens': 72781, 'oov_tokens': 4184, 'oov_fraction': 0.057487531086409915, 'median_oov_per_doc':
0.05,  'mean_oov_per_doc': 0.058822327249137794}
```

Throughout the implementation. We used some crucial off – the shelf libraries

Scikit-learn (TfidfVectorizer, CountVectorizer, LogisticRegression, Pipeline, GridSearchCV, KFold)

As well as some that I have built from scratch:

tokenize_text(), expand_contractions() and transfterm_text_into_feature_vestor()

1B : *Cross Validation Design Description*

```
GridSearchCV(cv=KFold(n_splits=5, random_state=0, shuffle=True),
 estimator=Pipeline(steps=[('my_bow_feature_extractor',
 TfidfVectorizer(max_df=0.9,
 stop_words=['a', 'and',
 'he', 'i',
                                        'in', 'of',
                                        'that',
                                        'the', 'to',
                                        'was'])),
 ('my_classifier',
 LogisticRegression(max_iter=200,
 random_state=101))]),
 param_grid={'my_bow_feature_extractor__max_df': [0.7, 0.9, 1.0],
 'my_bow_feature_extractor__min_df': [1, 2, 4],
 'my_bow_feature_extractor__ngram_range': [(1, 1),
 (1, 2),
                                        (1, 3)],
 'my_classifier__C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
 scoring='roc_auc')
```

The metrics we used to optimize our pipeline is AUROC. The reason being this isthreshold independent. AUROC evaluates the model's ranking ability across all possible classification thresholds, so it measures how well the model separates the two classes without committing to a particular positive/negative cutoff. Which is also the indicator for the leaderboard rank. (scoring = 'roc_auc')

In our GridSearch CV we used mysplitter = 5, KFold = 5. And shuffle = true, random_state = 0 to make sure that our folds are randomized and reproducible.

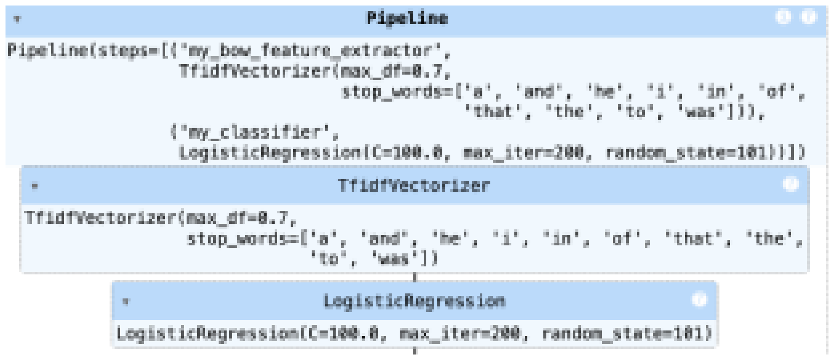
- 5 folds, With 5,557 samples split into 5 folds, 5557 % 5 = 2, so fold sizes are:
    - Two folds of size 1,112, training set size = 4445
    - Three folds of size 1,111, training set size = 4446

We did some search on various parameters, to mention some crucial ones:

1) The c, is a searching between array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
2) The parameter max_df and min_df, are meant to keep the document frequency, not just the raw token counts. For example, min_df = 4 means ignore terms that appear in strictly fewer than 4 documents,
3) Stop_words. Instead of using a built-in "English" common words, we used a customized list where is to take out the 10 most common words that appeared in the training set .

Use the hyperparameters chosen by CV (GridSearchCV), metrics shown as above. And then display a table sorted with the highest test score:

So based off the result table, our final pipeline is:



```
Pipeline

Pipeline(steps=[('my_bow_feature_extractor',
                 TfidfVectorizer(max_df=0.7,
                                 stop_words=['a', 'and', 'he', 'i', 'in', 'of',
                                             'that', 'the', 'to', 'was'])),
                ('my_classifier',
                 LogisticRegression(C=100.0, max_iter=200, random_state=101))])

TfidfVectorizer

TfidfVectorizer(max_df=0.7,
                stop_words=['a', 'and', 'he', 'i', 'in', 'of', 'that', 'the',
                            'to', 'was'])

LogisticRegression

LogisticRegression(C=100.0, max_iter=200, random_state=101)
```

*1C : **Hyperparameter Selection for Logistic Regression Classifier***

I used a pipelined GridSearchCV to tune the text-featurization and logistic-regression regularization together: the vectorizer hyperparameters I searched were

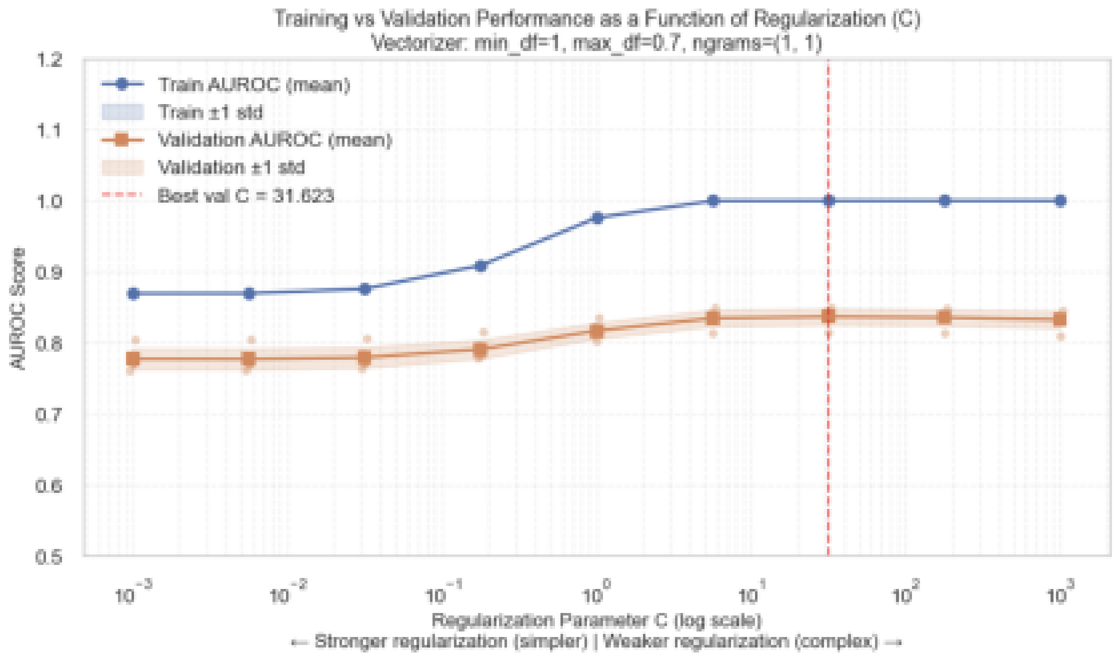
- TfidfVectorizer.min_df (to drop very rare tokens) with values [1, 2, 4],
- TfidfVectorizer.max_df (to drop extremely common tokens) with values [0.7, 0.9, 1.0] - ngram_range with values [(1,1), (1,2), (1,3)] to test unigrams up to trigrams;

the classifier hyperparameter searched was

- LogisticRegression C (inverse regularization strength) on a log grid np.logspace(-3, 3, 7) = [1e-3,1e-2,1e-1,1,10,100,1000].
- GridSearchCV was run with cv = KFold(n_splits=5, shuffle=True, random_state=0), scoring='roc_auc', and refit=True so the best hyperparameter set is refit on the full training set for final prediction.

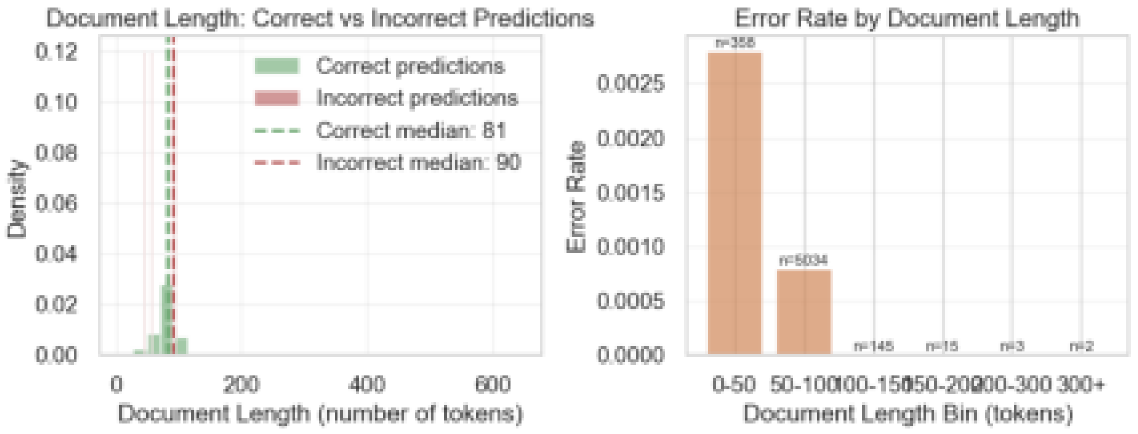
(As a separate experiment with the provided BERT embeddings I ran a simpler grid over regularization only: C in np.logspace(-4,4,20).) These choices balance model capacity (ngrams) and overfitting control (min_df/max_df and log-spaced C), give wide coverage across orders of magnitude for regularization, and optimize AUROC which is appropriate for a probabilistic ranking task.

Below is the figure that showing training vs validation performance over the selection of our classifier c.



The graph shows that the bias-variance tradeoff in regularization. Training AUROC (blue) increases with C as the model becomes more complex, eventually approaching perfect performance. Validation AUROC (orange) peaks at C ≈ 31.622777, showing the optimal balance between model complexity and generalization. Beyond this point, increasing C leads to overfitting: the model fits training noise rather than learning generalizable patterns, evidenced by the widening gap between training and validation curves.

1D ***: Analysis of Predictions for the Best Classifier***



In order to test whether the classifier does better on longer or shorter documents. We first for each documents, counts tokens by splitting on whitespace
str.split() splits text into words, and str.len() count them,

And then after we have our token splitted, we divide them into 2 groups. Correctly precited vs. incorrectly predicted. This was done by using the chosen best model, and make a prediction on the training data to see if it matches with the truth.(figue on the left)

And after we split them into 2 groups (T or F), we can graph a histograms showing the distributions with different total counts.

We also chose alpha =0.6, so there's some overlapping for us to visually compare the accuracy. If the 2 distributions are shifted (not much overlap), it tells us whether errors happen more on short or long docs. We also draws a vertical line at the median length for each group(correct or incorrect), and found out that the incorrect median is on the left of the correct median. It tells us that the classifier stuggles with shorter docs.

We also plot a error rate vs document length bin. First we groupby the length_bin, and look at the correct/ incorrect. And there

.apply(lambda x: 1 - x.mean()) → for each bin:

x.mean() = accuracy (fraction correct)

1 - x.mean() = error rate (fraction incorrect)

And we created a bar chat showing the error rate for each length. And as the chart shows, It falls from left to right, means higher error on shorter docs.

Overall ourTraining Set Performance Summary:

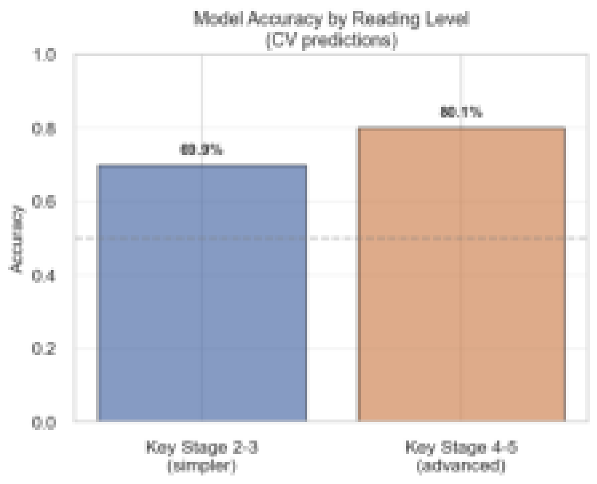
Accuracy: 0.999

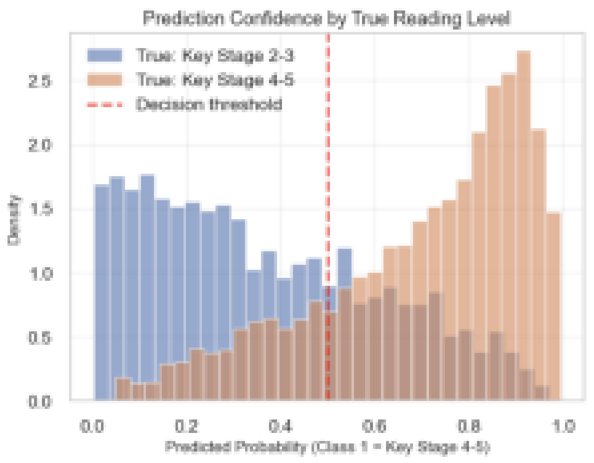
Total errors: 5 / 5557

False Positives (predicted Key Stage 4-5, actually 2-3): 2

False Negatives (predicted Key Stage 2-3, actually 4-5): 3

2.Accuracy by reading level





Key Stage 2-3 (younger/simpler):
Total documents: 2509
Correct predictions: 1753 (69.9%)
Incorrect predictions: 756 (30.1%)
Average predicted probability for class 1: 0.363
Median document length: 84 tokens

Key Stage 4-5 (older/advanced):
Total documents: 3048
Correct predictions: 2440 (80.1%)
Incorrect predictions: 608 (19.9%)
Average predicted probability for class 1: 0.701
Median document length: 78 tokens

In summary, the classifier achieves 99.9% training accuracy with 5 total errors (2 false positives, 3 false negatives). Document length analysis reveals that incorrect predictions have a median length of 90 tokens vs 81 for correct predictions, suggesting the classifier performs worse on longer documents.

The classifier shows similar performance across reading levels, with 0.1% error rate on Key Stage 2- 3 texts and 0.1% on Key Stage 4-5 texts.

The model performs BETTER on Key Stage 4-5 (advanced texts) by 10.2 percentage

points. This suggests it's better at identifying older/advanced reading level.

1E ***Report Performance on Test Set via Leaderboard***

The final AUROC on test set is 0.61183

Surprisingly, the test set performance exceeds our CV estimates(CV AUROC = 0.5104) by 0.1 approximately . This unexpected

improvement suggests that the test set may be slightly easier to classify than our training data, possibly due to: (1) Different author style distributions that happen to have clearer vocabulary markers of reading level; (2) Beneficial effects of training on all 5,557
examples for the final model versus the ~4,400 examples used in each CV fold; (3) Natural sampling variation making the test set more separable. While encouraging, this indicates our CV estimates were conservative rather than optimistic, which is preferable for model selection but suggests we might have benefited from more aggressive hyperparameter choices.

Part 2

2a.

We used the text to add more features like Yules K, Herdan's C, Brunét's W to give our model a better grasp at how readable a certain text is versus another and how sophisticated and rich it is. We created functions to implement the formulas so that we could then apply them to each example and create a new feature vector for each one. We also incorporated other numerical features from x_train.csv, including readability scores, pronoun frequency, sentiment polarity, function word count, etc. These numeric features were standardized using sklearn's StandardScaler to ensure they had similar scales that wouldn't affect the weights too significantly.

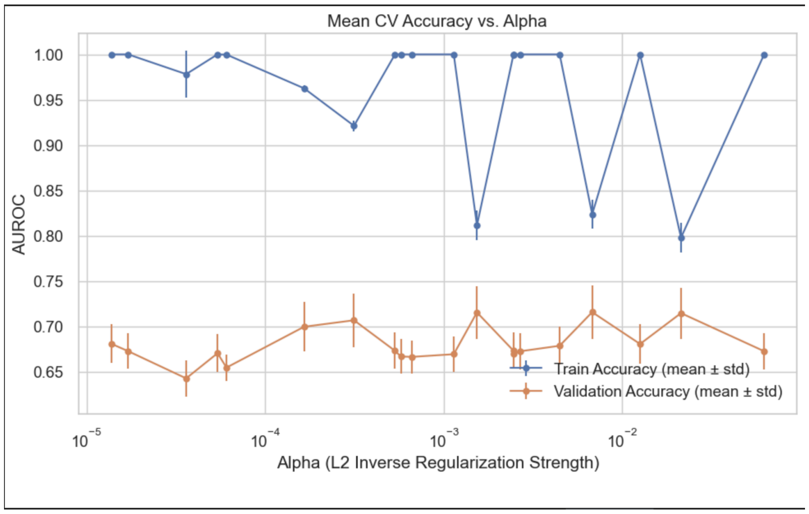
2b.

To evaluate our model and select hyperparameters, we used 5-fold cross-validation on the provided x_train.csv and y_train.csv. This makes sure that our model is tested on multiple train-validation splits, allowing us to get a good estimate of how well our model predicts data it hasn't seen before. We also make sure to shuffle the data set before to further generalize each fold. We use sklearn's RandomizedSearchCV to explore a range of reasonable hyperparameter values while keeping it computationally cheap (relative to grid search). This helps us find a balance between model complexity and generalization by selecting the best-performing configuration based on AUROC scores for our held out data. A random search with a sufficient number of points allows for exploration of a broader range of values than a grid search would, which could in turn result in better performance.

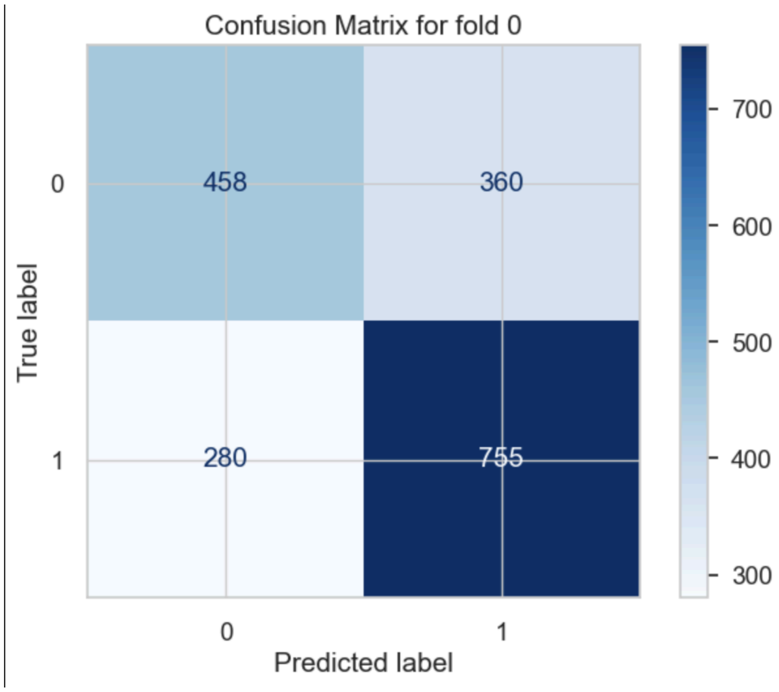
2c.

We selected a Multilayer Perceptron (MLP) classifier from sklearn.neural_network due to its ability to model nonlinear and flexible decision boundaries, allowing it to capture both simple and complex relationships between textual features and reading levels. This approach was expected to enhance classification performance compared to the logistic regression model implemented in Problem 1. We trained the classifier on the additional numerical features from the x_train.csv dataset and the given feature transformations.

We used RandomizedSearchCV with 3-fold cross-validation, varying the following hyperparameters:
hidden_layer_sizes: [(64,), (128,), (64, 32), (128, 64)]
This uses different architectures for the model, from a simple single layer network to deeper, wider, and more complex ones.
alpha: log-spaced values between 10^-5 and 10^-1
This controls the amount of penalization used to prevent overfitting. A broad range ensures we capture both underfitting and overfitting.
learning_rate_init: log-spaced value between 10^-4 and 10^-1
This controls how much the weights are adjusted in response to an error it sees during training. A broad range of values of the learning rate allows us to capture both underfitting and overfitting.
solver: Adam and Stochastic gradient descent
Tries the Adam and Stochastic Gradient Descent solvers for sklearn's MLPClassifier. Testing both allows us to pick the better solver.



This graph is supposed to show how training and validation AUROC changes in relation to the regulation strength alpha. Lower alphas don't regulate the weights as much which can lead to overfitting which would result in a very high accuracy. On the other hand, higher alphas limit these weights and could lead to underfitting which decrease the accuracy of the validation and
2d.



The classifiers show similarities of performing better at predicting whether a text is an upper level text or a lower level text. The two classifiers are definitely directly comparable. The distribution of predicted probabilities for the misclassified samples indicates that the model was generally uncertain about these predictions, with most probabilities concentrated around 50%.

2e.
Our final AUROC score was 0.8337, compared to a predicted AUROC of 0.8249 based on held-out validation data. During

cross-validation, the model was evaluated on unseen portions of the training data; however, these samples were drawn from the same overall distribution. The test set may have included passages with stylistic or structural characteristics not represented in the training data, which could explain the performance gap. Although cross-validation provides a useful estimate of model performance, it is still limited by the diversity of the training data. If the training set is less varied than the test set, the model's true performance may be lower than expected.