Comp135-ProjB

Emily Holt, Dylan Phelan

November 2020

1 Bag-of-Words

1.A Bag-of-Words Design Decision Description

For our to Bag-of-Words feature representation experiments, we explored sklearn's $CountVectorizer$ and $TfidfVectorizer$. By using these built-in feature transformation modules, we were able to remove common punctuation marks, leverage their builtin tokenizers, strip reviews of stopwords in sklearn's default stopwords list, and lowercase all words in the dataset.

To configure the hyperparameters for these vectorizers, we built pipelines using these two vectorizers and a baseline L1 Logistic classifier Regression (with solver='saga', C=10.0, max_iter=20) We then used *GridSearchCV* to find optimal hyperparameter values, using 5-fold validation and varying minimum-document frequency (1-9), maximum-document frequency (0.01, 0.05, 1.0), and use of unigrams vs. bigrams. We selected the vectorizer that achieved the best average balanced accuracy across holdout datasets, the measure of success used on the project leaderboard. In our case, this was the TfidfVectorizer using both unigrams and bigrams, minimum-document frequency of 1 and maximum-document frequency of 1.0.

Accordingly, we use the $TfidfVectorizer$ transform for the rest of our Bag-of-Words experiments. This final vectorizer had a vocabulary size of 14311 elements and translates each review into a vector of 14311-features, where each feature corresponds to the frequency of that term in this document, multiplied by a measure of how common or rare that term is across other documents. If a review contains only new unigrams or bigrams, the feature vector would contain exclusively 0s.

1.B Cross Validation and Hyperparameter Selection Design Description

In each of our Bag-of-Words experiments, we leveraged the sklearn *Pipeline* and *GridSearchCV* to perform a grid search over different possible hyperparameter values for each of our models, running 5-fold cross validation for each combination of hyperparameter values. We leveraged the sklearn $StratifiedKFold$ splitter in order to split our data into 5 folds while preserving the ratio of positive to negative samples across each fold. We chose to optimize for balanced accuracy on the holdout dataset, since we aim to create a classifier that can successfully classify both positive and negative samples. Additionally, we chose this performance metric since it is how we will be scored on the community leaderboard.

Finally, we use the hyperparameter values that led to the best average balanced accuracy on our heldout sets to train a new model on the entire train set, which we use as our "final" model to generate test set predictions. Specifically, we leverage the *GridSearchCv.best_estimator* property, which takes care of this final training behind the scenes.

1.C Hyperparameter Selection Figure for Classifier 1

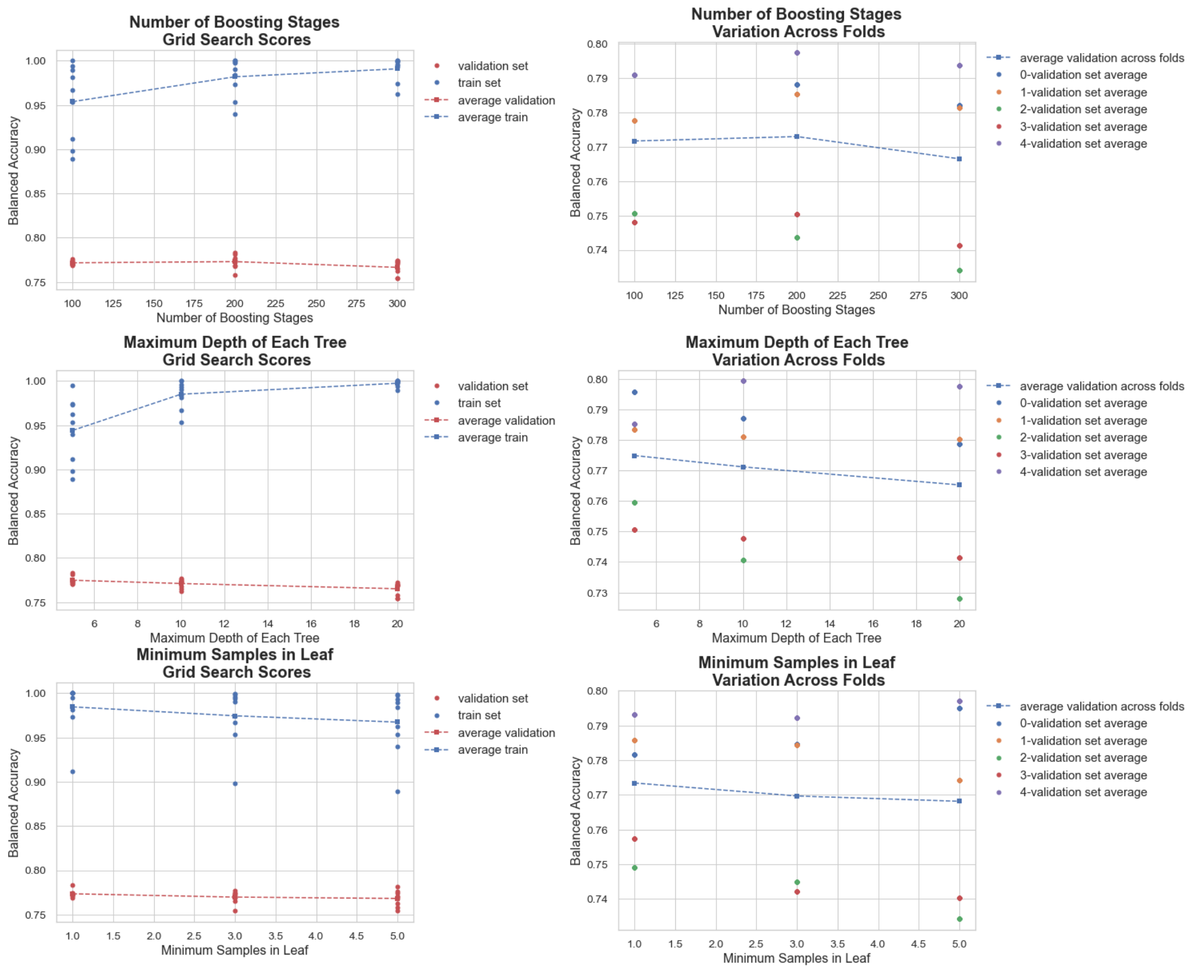

Figure 1: Visualizing our Bag-of-Words-trained MLP GB-Tree Gridsearch over three hyperparameters: *n_estimators*, *max_depth* and *min_samples_leaf*. Each row visualizes the results for one of these three hyperparameters across their possible values. Panels in the left column compare average train and average test performance across all folds. Panels in the right column explore variation across each fold's performance as compared to the overall average heldout set performance.

We chose to explore Gradient Boosted Tree classifiers because they are invariant to scaling and do not expose too many hyperparameters, thus reducing model complexity. Since gradient boosting is fairly robust against overfitting, we should not have to worry too much about early stopping.

We explored tuning three hyperparameters: numbers of boosting stages with $n\_estimators$ 100, 200, and 300; depths of each classification tree with max_depth 5, 10, and 20; and numbers of minimum samples in every leaf node with $min\_samples\_leaf$ 1, 3, and 5. The ranges explored are in the neighborhood of the default values recommended by sklearn, making these values our preferred candidates. Based on our Gridsearch results, we selected $n\_estimators = 300$, $max\_depth = 5$ and $min\_samples\_leaf = 1$. That said, it should be noted that several combinations of hyperparameters achieved similar balanced accuracy scores, demonstrating their promise for future experiments.

1.D Hyperparameter Selection Figure for Classifier 2

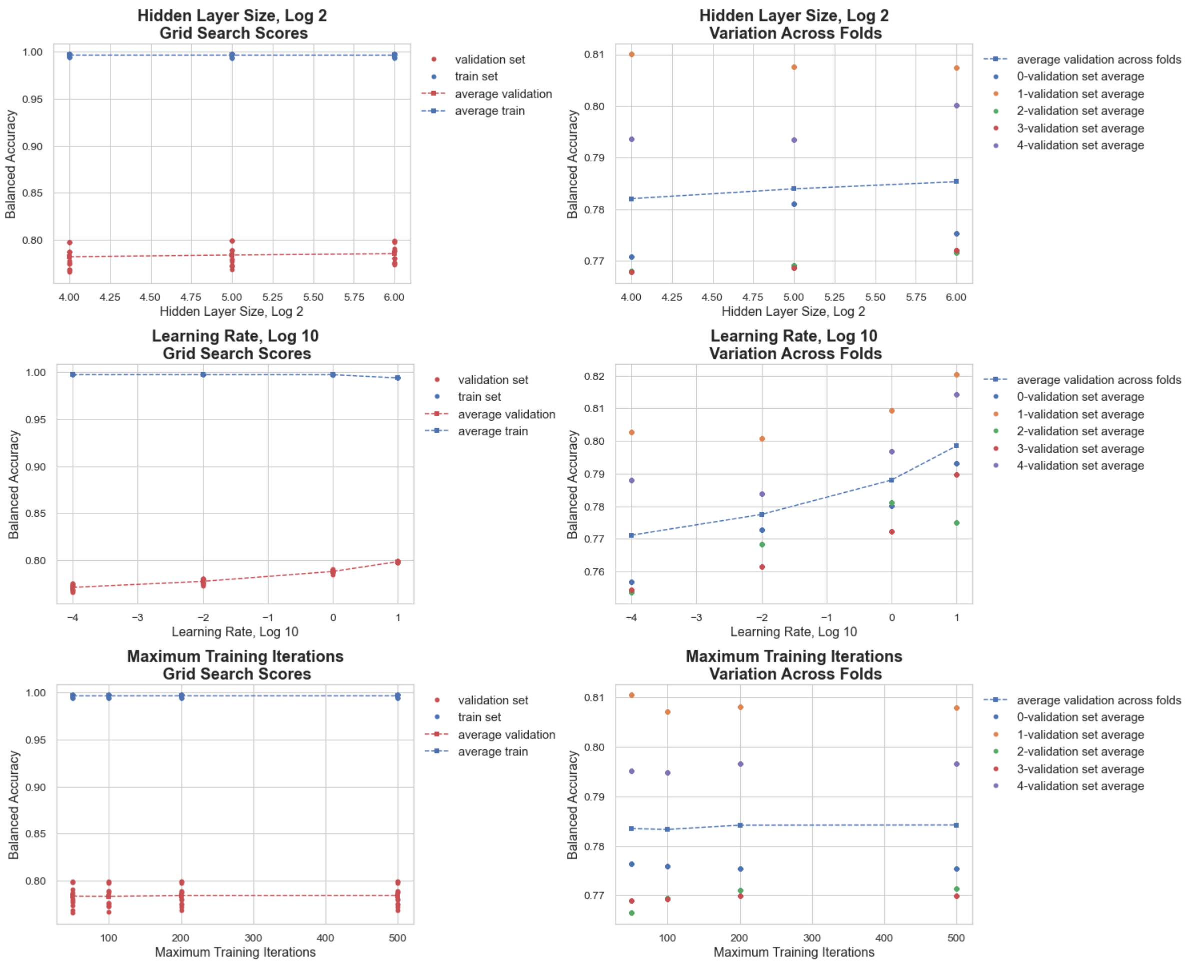

Figure 2: Visualizing our Bag-of-Words-trained MLP Gridsearch over three hyperparameters: $hidden\_layer\_sizes$, and $learning\_rate$, and max_iter. Each row visualizes the results for one of these three hyperparameters across their possible values. Panels in the left column compare average train and average test performance across all folds. Panels in the right column explore variation across each fold's performance as compared to the overall average heldout set performance.

We trained a Multi-Layer Perceptron Neural Net classifier for our second Bag-of-Words classifier. We hypothesized that the flexibility of MLP decision boundaries would make the classifier more successful and obtain a better balanced accuracy compared to the tree-based classifiers. During these experiments, we explored tuning three hyperparameters: size of the hidden layer with *hidden_layer_sizes* 16, 32, and 64; different penalty values with *alpha* 0.0001, 0.01, 1, and 10; and finally, different maximum number of training iterations by exploring *max_iter* 50, 100, 200, and 500. We explored smaller hidden layer sizes and higher alpha values than the default sklearn values because our previous labs demonstrated the benefits of reducing model size and complexity when generalizing to heldout data. We explored max iterations smaller than the sklearn default for similar reasons, as we had previously noted the benefits of early stopping.

In our experiments, we noticed that hidden layer size of 32 led to the optimal model, though models with hidden layer sizes with 16 and 64 both showed similar performance. Similarly, while the best number of maximum iterations is 50, other values of 100, 200, and 500 all showed similar performance. However, the best *alpha* value is decisively 10, and showed remarkably better performance than models trained with smaller *alpha* values.

1.E Determination of the Best Classifier Pipeline

Based on our cross-validation experiments, the MLP classifier performed better on heldout data than our Gradient Boosted Tree. Our MLP classifier with 32 hidden layers, an alpha value of 10, trained for 50 iterations, was able to obtain an average balanced accuracy score on the train set of 0.993 and an average balanced accuracy score on the test set of 0.800 across our 5 cross validation folds. Our Gradient Boosted Tree classifier with 300 estimators, a max depth of 5 nodes, and a minimum of 1 samples per leaf node, was only able to obtain an average balanced accuracy score on the train set of 0.944, and an average balanced accuracy score on the test set of 0.767 across our 5 cross validation folds. We hypothesize that the MLP classifier was able to outperform our tree-based classifier due to its flexible decision boundaries and, with 32 hidden layers, better suited to fit this data.

Though the MLP classifier obtained better balanced accuracy, the Gradient Boosted Tree was slightly faster to train. The optimal combination of Gradient Boosted Tree parameters had a mean fit time of 5.464 seconds, while the optimal combination of MLP parameters had a mean fit time of 6.351 seconds. While the Gradient Boosted Tree was faster to fit, the improvement in balanced accuracy on the heldout data lead us to choose the MLP classifier as our Best Classifier.

1.F Analysis of Predictions for the Best Classifier

False Positive Examples	False Negative Examples
Not good enough for the price.	Would recommend this item.
Graphics is far from the best part of the game.	GO AND SEE IT!
It is not good.	Predictable, but not a bad watch.
Why was this film made?	The last 15 minutes of movie are also not bad as well.
Nothing at all to recommend.	Waste your money on this game.
It was not good.	The soundtrack wasn't terrible, either.
Would not recommend to others.	I don't think you will be disappointed.

Table 1: Bag-of-Words MLP Classifier Mistakes

During our analysis of our MLP Classifier's mistakes, we discovered that the average length of the False Positive samples was 5.571 words. Of the samples that were categorized as False Positive, 57.143% were from IMDB, 14.286% were from Amazon, and 28.571% were from Yelp. We also discovered that 71.429% of samples categorized as False Positive contained negation words. However we recognize that our analysis of negation words is fundamentally flawed, due to the fact that sklearn's list of stopwords contains several of our identified words like 'not'. We also discovered that the average length of the False Negative samples was 5.75 words. Of the samples that were categorized as False Negative, 75% were from IMDB, 12.5% were from Amazon, and 12.5% were from Yelp. We also discovered that 50% of samples categorized as False Negative contained negation words. Again, we recognize that our analysis of negation words is fundamentally flawed, due to the fact that sklearn's list of stopwords contains several of our identified negations like 'not'.

1.G Report Performance on Test Set via Leaderboard

Our MLP classifier obtained a test set balanced accuracy score of 0.802, as reported by the leaderboard. This score was very similar to, though slightly higher than, the MLP's mean balanced accuracy score of 0.800 across the 5 folds of our heldout set.

2 Word Embedding

2.A Word Embedding Preprocessing Description

For our Word Embedding experiments, we cleaned and standardized our data by converting all words to lowercase and stripping all punctuation from each sample. We then stripped all words found in sklearn's list of English stopwords from the dataset; additionally, we stripped all words not found in the GloVe embedding from then dataset. When iterating through the cleaned dataset to convert each sample to a vector, we kept a set of *unique_vocabulary_words*: every word in our dataset that was not a stopword but that was in the GloVe embedding was added to the set. At the end of our data processing step, we calculated the length of the *unique_vocabulary_words* set to obtain the final size of our vocabulary set, which was 3902.

Each vocabulary word was represented by its 50-feature vector found in the GloVe embedding. The vectors for each word in a sample were then averaged together to obtain the resulting vector for a given sample in the dataset. Thus the resulting sample's vector also has 50 features. If a sample was comprised exclusively of stopwords and words not found in the GloVe embedding, we represented that sample as a vector with 50 features where each feature had the value 0. We followed the same data processing steps when processing the test set.

2.B Cross Validation and Hyperparameter Selection Design Description

We used the same strategy as described in section 1B: $GridSearch$ over a 5-fold cross validation. We leveraged the sklearn $StratifiedKFold$ splitter in order to split our data into 5 folds while preserving the ratio of positive to negative samples across each fold. We chose to optimize for balanced accuracy on the holdout dataset, since we aim to create a classifier that can successfully classify both positive and negative samples (specifically on the leaderboard). Ultimately, we evaluated the classifiers based on the average balanced accuracy score across the 5 folds. We then used the hyperparameter values that led to the best average balanced accuracy to train a new model on the entire train set to create our "final" model that we then used to generate test set predictions. Specifically, we leverage the $GridSearchCv.best\_estimator$ property, which takes care of this final training behind the scenes.

2.C Hyperparameter Selection for Classifier 1

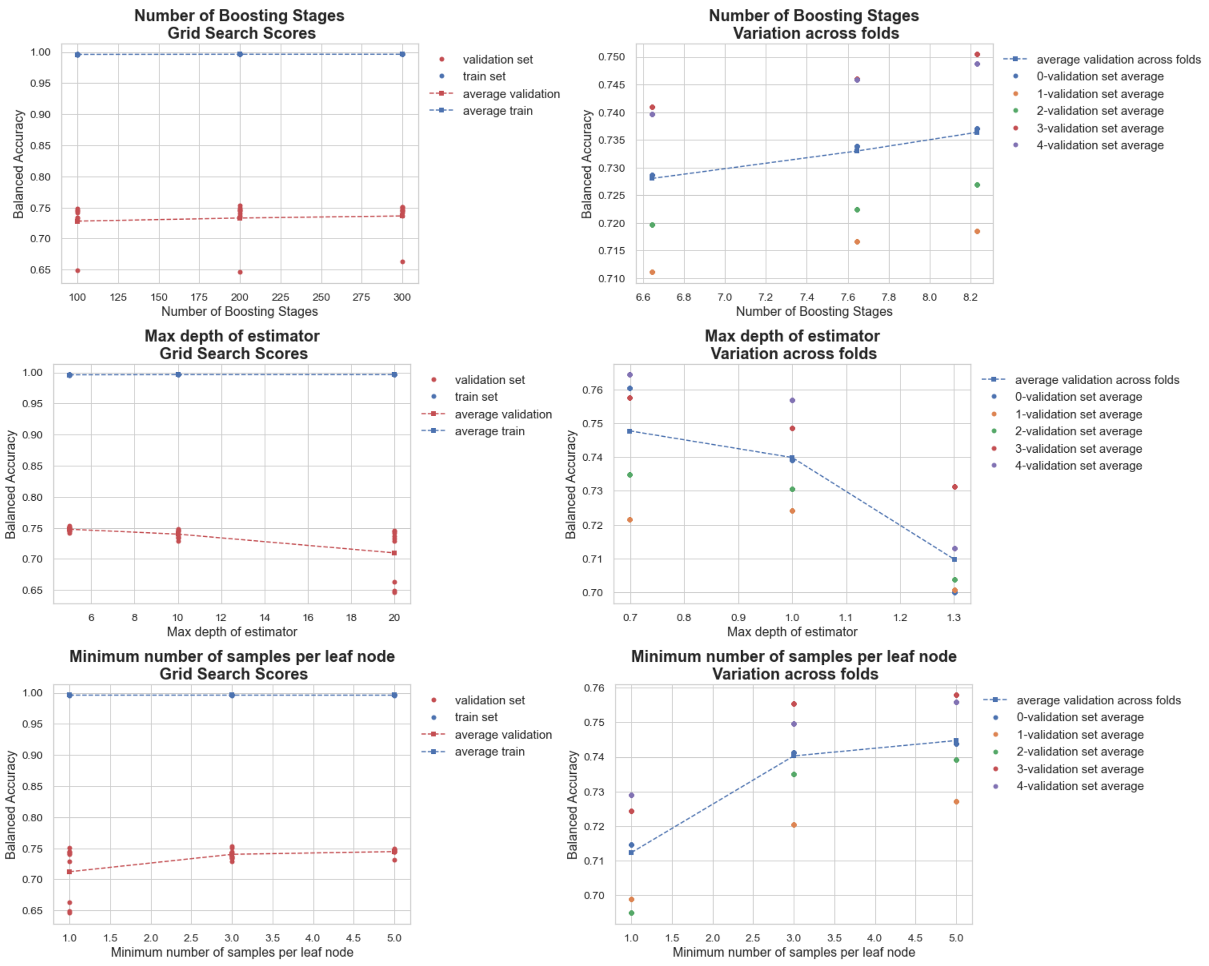

Figure 3: Visualizing our Word-Embedding-trained Gradient Boosted Tree Gridsearch over three hyperparameters: *n_estimators*, *max_depth* and *min_samples_leaf*. Each row visualizes the results for one of these three hyperparameters across their possible values. Panels in the left column compare average train and average test performance across all folds. Panels in the right column explore variation across each fold's performance as compared to the overall average heldout set performance.

We also chose to explore Gradient Boosted Tree classifiers for our Word Embedding experiments because they are invariant to scaling and do not expose too many hyperparameters, thus reducing model complexity. Since gradient boosting is fairly robust against overfitting, we should not have to worry too much about early stopping.

We explored tuning three hyperparameters: numbers of boosting stages with *n_estimators* 100, 200, or 300; depths of each classification tree with *max_depth* 5, 10, or 20; and numbers of minimum samples in every leaf node with *min_samples_leaf* 1, 3, or 5, as we did in Section 1C. The ranges explored are in the neighborhood of the default values recommended by sklearn, making these values our preferred candidates. Based on our Gridsearch results, we selected $n\_estimators = 200$, $max\_depth = 5$ and $min\_samples\_leaf = 3$, which were nearly identical hyperparameters to our hyperparameters from Section 1C (save for the change to *min_samples_leaf*). In our Word Embedding experiments, however, we noticed much larger changes in balanced accuracy scores across different hyperparameter combinations, and that the $max\_depth = 5$ and $min\_samples\_leaf = 3$ were decisively the optimal values.

2.D Hyperparameter Selection for Classifier 2

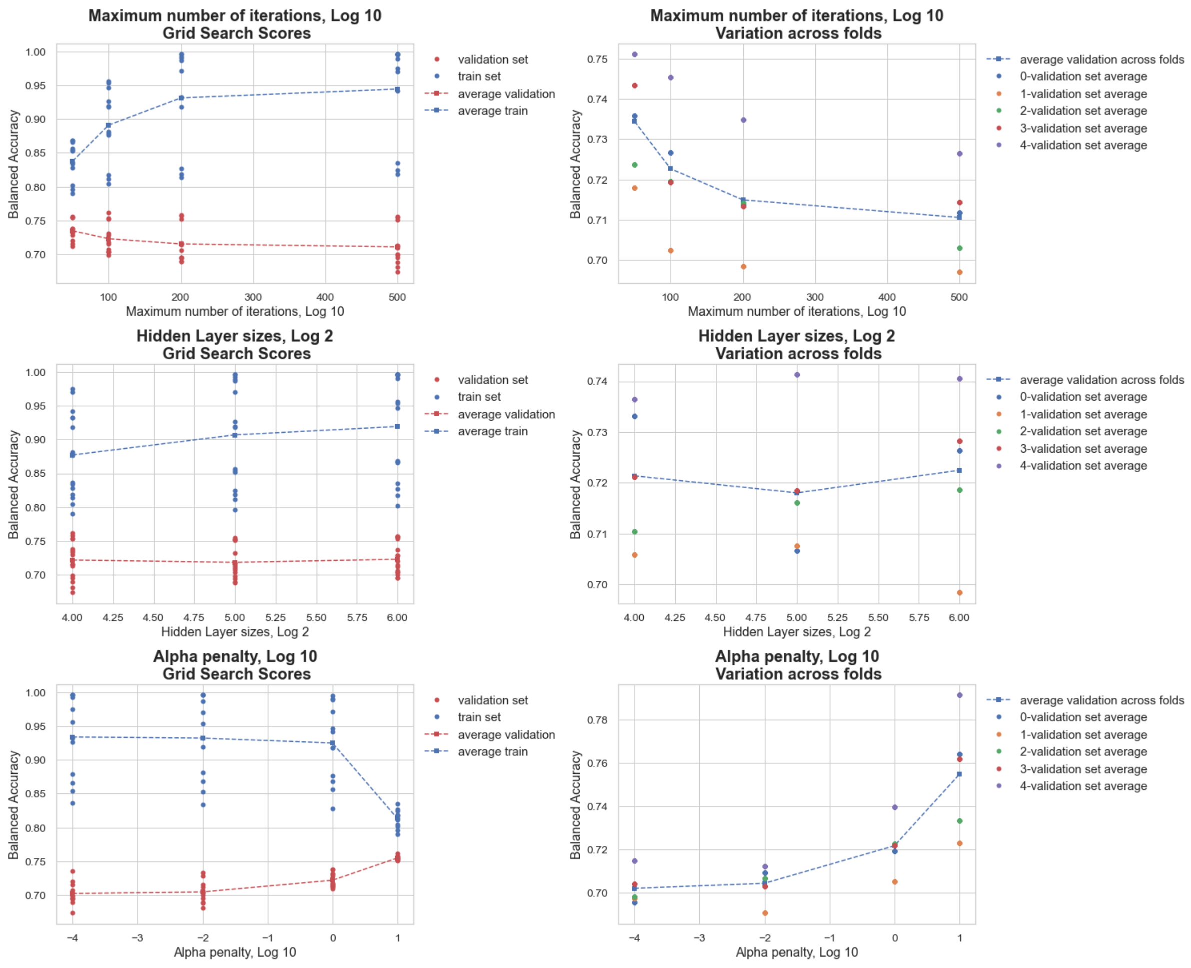

Figure 4: Visualizing our Word-Embedding-trained MLP Gridsearch over three hyperparameters: *hidden_layer_sizes*, and *learning_rate*, and *max_iter*. Each row visualizes the results for one of these three hyperparameters across their possible values. Panels in the left column compare average train and average test performance across all folds. Panels in the right column explore variation across each fold's performance as compared to the overall average heldout set performance.

We trained a Multi-Layer Perceptron Neural Net classifier for our second Word Embedding classifier, using the same experiment design as in Section 1D. We again hypothesized that the flexibility of MLP decision boundaries would make the classifier more successful and obtain a better balanced accuracy compared to the tree-based classifiers. During these experiments, we explored tuning three hyperparameters: size of the hidden layer with *hidden_layer_sizes* 16, 32, and 64; different penalty values with *alpha* 0.0001, 0.01, 1, and 10; and finally, different maximum number of training iterations by exploring *max_iter* 50, 100, 200, and 500. We explored smaller hidden layer sizes and higher alpha values than the default sklearn values because our previous labs demonstrated the benefits of reducing model size and complexity when generalizing to heldout data. We explored max iterations smaller than the sklearn default for similar reasons, as we had previously noted the benefits of early stopping.

We noticed that while the best number of maximum iterations is decisively 100, and the best *alpha* value is decisively set to 10, the hidden layer sizes did little to change the balanced accuracy scores, though we noticed that hidden layer size set to 16 reduced complexity and resulted in the best performance on the heldout set.

2.E Determination of the Best Classifier Pipeline

Based on our cross-validation experiments, the MLP classifier performed better on heldout data than our Gradient Boosted Tree. Our MLP classifier with 16 hidden layers, an alpha value of 10, trained for 100 iterations, was able to obtain an average balanced accuracy score on the train set of 0.804 and an average balanced accuracy score on the test set of 0.761 across our 5 cross validation folds. Our Gradient Boosted Tree classifier, with 200 estimators, a max depth of 5 nodes, and a minimum of 3 samples per leaf node, was only able to obtain an average balanced accuracy score on the train set of 0.996, and an average balanced accuracy score on the test set of 0.753 across our 5 cross validation folds. We hypothesize that the MLP classifier was able to outperform our tree-based classifier due to its flexible decision boundaries and, with 16 hidden layers, better suited to fit this data.

Though the MLP classifier obtained only slightly better balanced accuracy, the MLP classifier was significantly faster to train. The optimal combination of MLP parameters had a mean fit time of 0.239 seconds, while the optimal combination of Gradient Boosted Tree parameters had a mean fit time of 28.033 seconds. In addition to being faster to train, the MLP classifier showed fewer signs of overfitting, considering the smaller margin between train set and heldout set balanced accuracy performance of the MLP classifier compared to the Gradient Boosted Tree classifier. For these reasons, we chose the MLP classifier as our Best Classifier for the Word Embedding dataset.

2.F Analysis of Predictions for the Best Classifier

False Positive Examples	False Negative Examples
Oh and I forgot to also mention the weird color effect it has on your phone.	The phone loads super!
I bought these hoping I could make my Bluetooth headset fit better but these things made it impossible to wear.	Car charger as well as AC charger are included to make sure you never run out of juice.Highy recommended
Not impressed.	The sound is clear and the people I talk to on it are amazed at the quality too
Unfortunately the ability to actually know you are receiving a call is a rather important feature and this phone is pitiful in that respect.	Plan on ordering from them again and again.
If you are looking for a good quality Motorola Headset keep looking, this isn't it.	Gets a signal when other Verizon phones won't.
My father has the V265, and the battery is dying.	This is cool because most cases are just open there allowing the screen to get all scratched up.
It dit not work most of the time with my Nokia 5320.	Setup went very smoothly.

Table 2: Word Embedding MLP Classifier Mistakes

When analyzing our Word Embedding MLP Classifier's mistakes, we took the predictions our model made on our transformed word embedding feature vectors, then ran our analysis on the original sample after preprocessing (converted to lowercase and stripped of stop words and punctuation). Since our preprocessing involved stripping stopwords from our samples before converting to embeddings, and because sklearn's list of stopwords contains words like 'not' and 'should', we did not analyze performance of samples with negation words.

After observing the inadequacy of our negation-word analysis in part 1.F, we decided to forgo an analysis of negation words in our word embeddings, since our word-embedding, stop-word preprocessing suffers from the same fatal flaws. During our analysis, we discovered that the average length of the False Positive samples was 4.991 words. Of the samples that were categorized as False Positive, 33.043% were from IMDB, 29.565% were from Amazon, and 37.391% were from Yelp. We also discovered that the average length of the False Negative samples was 5.483 words. Of the samples that were categorized as False Negative, 38.362% were from IMDB, 32.758% were from Amazon, and 28.879% were from Yelp. These results demonstrate a more balanced performance across the three sources compared to the Bag-of-Words MLP classifier, where the False Positives and False Negatives were overwhelmingly from IMDB, as seen in Section 1F.

2.G Report Performance on Test Set via Leaderboard

Our MLP classifier described in Section 2D was able to obtain a balanced accuracy score of 0.765 on the test set, as reported by the leaderboard. Our test set performance (0.765) was nearly identical to, though slightly higher than, our heldout set performance reported in section 2E (0.761). Our MLP's balanced accuracy score on the Word Embedding test set was lower than our MLP's balanced accuracy score on the Bag-of-Words test set, though the Word Embedding MLP classifier also obtained a lower heldout set balanced accuracy score compared to the Bag-of-Words MLP classifier. We hypothesize that our Word Embedding classifier from 2D performed worse than our Bag-of-Words classifier from 1D because the Bag-of-Words feature transform lead to a much larger feature vector with 14311 elements, leading to a richer and more comprehensive representation of the data, compared to the Word Embedding features, which only had 50-element feature vectors.

3 Open Ended Approach

3.A Methods Description

For our Open Ended experiments, we first split the training data into three different subsets based on the website from where the sample review was collected (*amazon*, *imdb*, and *yelp*). We hypothesized that each website would have specific context, which one classifier may not be able to effectively capture. We then used sklearn's $TfidfVectorizer$ with both unigrams and bigrams, with minimum-document-frequency of 1, and maximum-document-frequency of 1.0, on each of the three subsets to transform the reviews into feature vectors. Our experiments using Bag-of-Words feature transforms consistently achieved higher balanced accuracy scores, and the $TfidfVectorizer$ out-performed the $CountVectorizer$ on our baseline experiments. By using these built-in feature transformation modules, we were able to remove punctuation, seamlessly tokenize, strip all words found in sklearn's stopwords list, and convert all letters to lowercase in all of the reviews in the dataset, as we did in Section 1A. The $TfidfVectorizer$ transformations resulted in a vocabulary length of 4233 elements for the *amazon* dataset, 6749 elements for the *imdb* dataset, and 4613 elements for the *yelp* dataset.

We trained a three separate L1-Logistic Regression models, one on each of our three feature-transformed training subsets. We used sklearn's $StratifiedKFold$ validation with five folds in combination with Grid Search to determine the effect of different penalty values by exploring C values set between 0.01 and 1000; and different number of maximum iterations by exploring max_iter set to 20, 40, and 60. The best L1-Logistic Regression classifier on the *amazon* training subset had optimal parameters of $C = 100$ and $max\_iter = 60$, and was able to achieve a mean balanced accuracy of 0.810 on five folds of the *amazon* heldout subset. The best L1-Logistic Regression classifier on the *imdb* training subset had optimal parameters of $C = 1000$ and $max\_iter = 60$, and was able to achieve a mean balanced accuracy of 0.756 on five folds of the *imdb* heldout subset. Finally, the best L1-Logistic Regression classifier on the *yelp* training subset had optimal parameters of $C = 100$ and $max\_iter = 40$, and was able to achieve a mean balanced accuracy of 0.788 on five folds of the *yelp* heldout subset.

Given our results from Sections 1 and 2, we trained a MLP classifier using the same design as described in 1D and 2D, hypothesizing that the MLP architecture would continue to result in the best-performing classifier. However, the combination of 3 MLPs each trained on one of the 3 training subsets resulted in lower balanced accuracy score on the test set than the combination of 3 L1-Logistic Regression models described above, which we hypothesize is due to the smaller training sets which were likely not sufficient to train successful MLPs.

3.B Report Performance on Test Set via Leaderboard

The combination of our 3 website-specific L1-Logistic Regression models achieved a balanced accuracy score of 0.788 on the test set, as reported by the leaderboard. Ultimately, this combination of models performed better than our Word Embedding-base MLP classifier from Section 2 (which obtained a balanced accuracy score of 0.765 on the test set), though this combination did perform as well as our Bag-of-Words MLP classifier from Section 1 (which obtained a balanced accuracy score of 0.802 on the test set). We hypothesize that although the models were able to better determine the context of the reviews, there ultimately was not enough data for the models to train on once we split the trainset into three subsets for the model to successfully generalize to the test set.