Bag of Words

1. Formation of Bag of Words

All the text went through some cleaning and preprocessing.

1. Run the PorterStemmer through all text
2. Remove all punctuations and numbers
3. Change all case of text to lowercase

The PorterStemmer algorithm was used to lower the different forms of the same base word. For example, if we have run, running, ran, and runs, they would all be stemmed down to its original form, run. This was done because we are trying to analyze the sentiment of a sentence which means the Bag of Words will benefit from having all forms of a word reduced to its base form. Additionally, punctuations were also removed because they do not carry any sentiment meaning. This was also the case for numeral texts. Finally, to standardize all text, they were all converted to lowercase.
One immediate shortcoming I witnessed was the existence of ill-spelled text in the data. These words are not properly stemmed by the PorterStemmer and thus stays as a weird form of misspelled word. Due to this 2 of the same words may get a different representation from the Bag of Words. To clarify this problem, consider this example:

Word 1: initialize -> initializ
Word 2: initialization -> initializ
Word 3: initializing -> initializ
Word 4: iniatlize -> iniatlize

Word 4 should also be stemmed down to "initializ" but due to the spelling error it is still iniatlize.

After the preprocessing, a CountVectorizer was initialized to create the Bag of Words. Stop words (common words) were removed and any words that occur less than 2 times were ignored. Also instead of having a Bag of Words consisting of single word features it consists of both single word and also the bigram. This was important because in sentiment analysis it is important to consider groups of words together instead of individual ones. Furthermore, stop words were removed because they carry little to no sentiment information and words that occur less than 2 times were dropped because of their rarity, they too carry little to no sentiment information.

Important Details that will Matter from this point on
The later sections will mention data regarding average error (log loss in this case), average ROC, and average score. These data were acquired with a personally written cross-validation function. Here is how the values were acquired:

1. Set up a KFold object with 12 splits.

2. Split up the given training data into training and validation portions. Unless otherwise specified when “training” data is mentioned it is referring to the training portion from the train-test-split.
3. Fit model with the training data
4. Get score, ROC, and log loss values with predictions made (with the training data) from the model and the given true values.
5. Get score, ROC, and log loss values with predictions made (with the validation data) from the model and the given true values.
6. Accumulate these values (score, ROC, log loss) for each fold and return values for all 12 folds at the end.

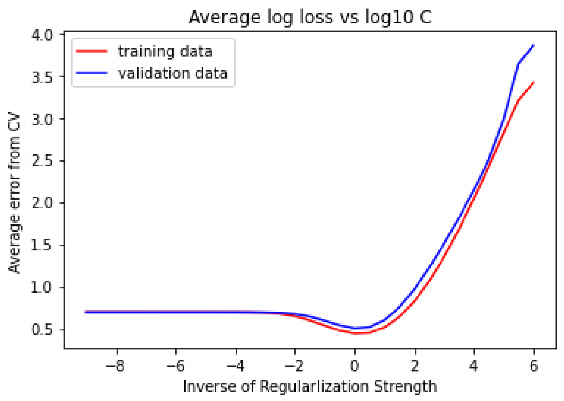
2. Logistic Regression Model on the BoW representation

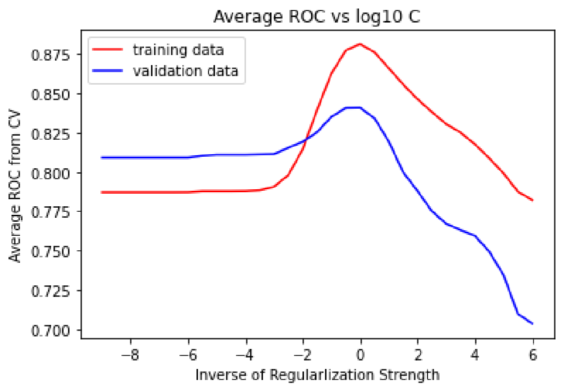
Chosen model: LogisticRegression(C=0.31622776601683794,
penalty=’l2’,
solver=’liblinear’)

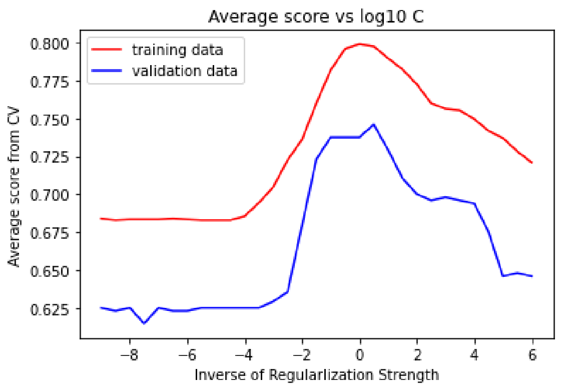
The penalty and solver parameters were chosen through a simple grid search on the entire data. The grid search then performed a 3-fold cross validation test for each of the possible models.
The values tried for the two hyperparameters are:

Penalty = [L1, L2, elasticnet, none]
Solver = [newton-cg, lbfgs, liblinear, sag, saga]

After the two hyperparameters were chosen an appropriate C value had to be found.



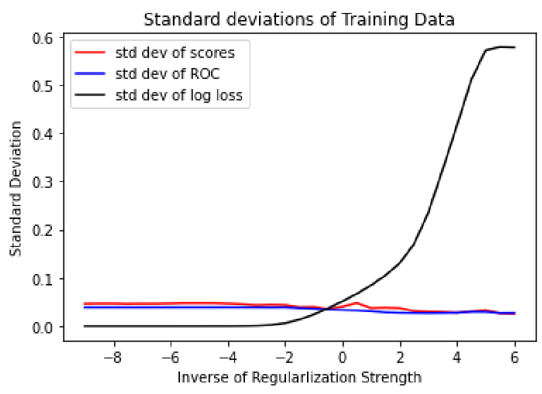


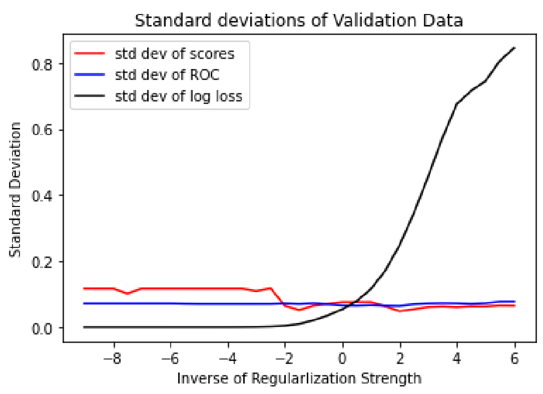


C-value Testing process:

1. Split training data into training and validation sets with ratios 80:20.
2. Create a log space of 31 values between -9 and 6.
3. For each of the 31 values run the 12-fold cross validation test on the training set.
4. For each of the 31 values run the 12-fold cross validation test on the validation set.

The range of C was set to the logspace(-9, 6, 31) because the optimal C value should be in the range since if it is outside there is a good chance of over/underfitting.





Through testing I decided that the best C value is 0.31622776601683794. In all 3 graphs showing the average error, score, and roc the training and validation dataset tends to perform in similar fashion. In other words, when the training performance goes down, the validation performance also goes down. We see the lowest average log loss at around the chosen C value. Furthermore, we also see that the maximum score and ROC are both at around 0.31622776601683794 as well. Since the performance of the validation dataset seems to also follow the training dataset, I can say that the model is not overfit.
Additionally, in the graphs depicting the standard deviations of the log loss, ROC, and scores over different C values we see that log loss' standard deviation skyrockets after the chosen C value. This further expresses that increasing the C value past 0.31622776601683794 will make the model worse.

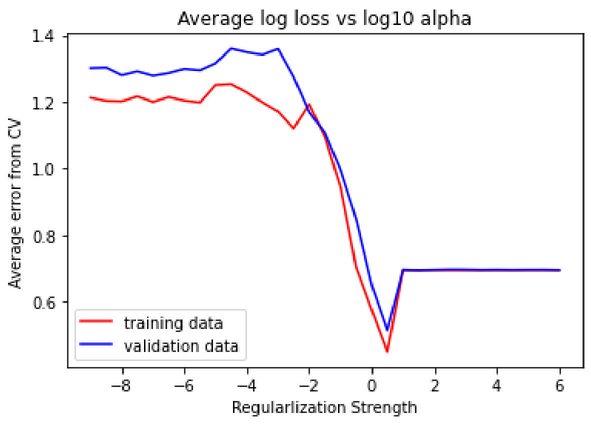
3. MLP Model on the BoW representation

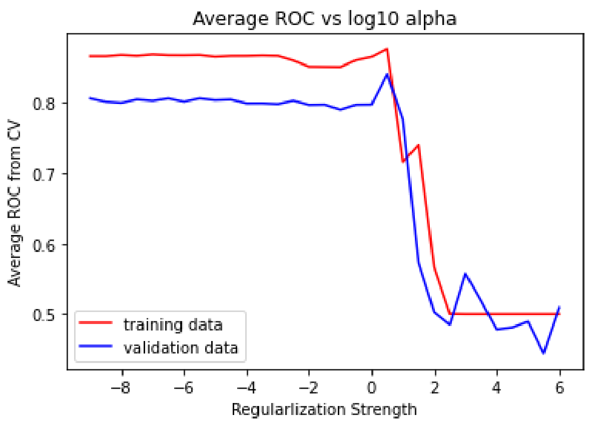
Chosen Model: MLPClassifier(hidden_layer_sizes=(800, 200),
activation='relu',
learning_rate='constant'
momentum=0.1,
nesterovs_momentum = True,
solver = 'adam',
alpha=0.03162277660168379)

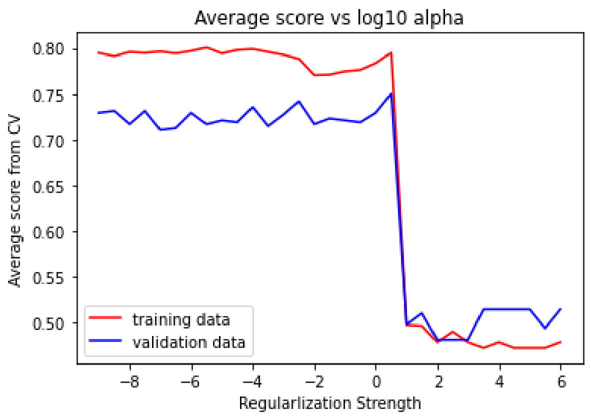
The hyperparameters were chosen through a grid search of the entire training data set with 3-fold cross validation. The hyperparameter values tested were:

Hidden_layer_sizes: [(300, 150), (200,250), (800, 200), (300, 500), (850, 800)]
Activation: [identity, logistic, tanh, relu]
Solver: [lbfgs, sgd, adam]
Learning_rate: [constant, invscaling, adaptive]

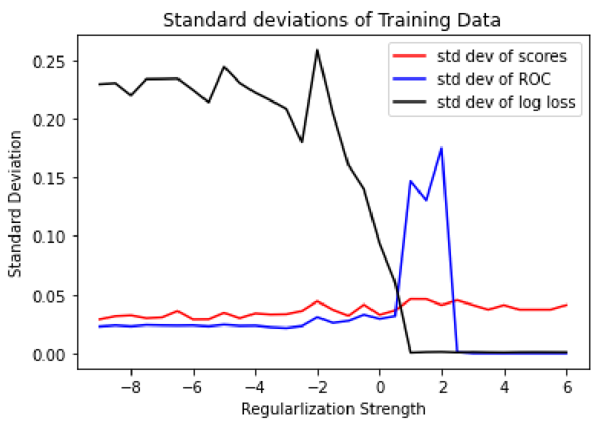
After these hyperparameters were tested, an appropriate alpha value had to be found. Using the same process as testing for C-values mentioned above the alpha value was found in the ranges of logspace(-9, 6, 31). Again this range was used because it covered a large portion of adequate regularization strengths (both severely low and high and everything in between).

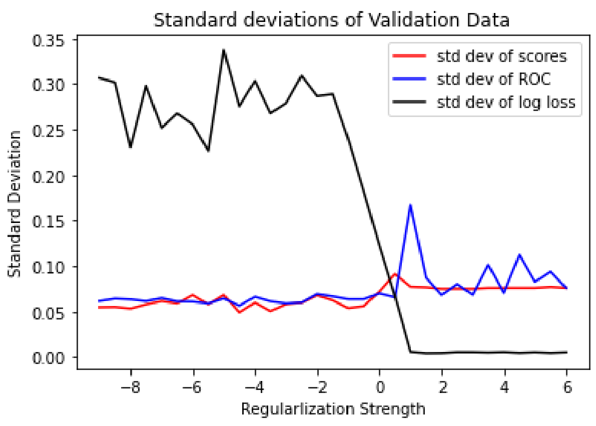






And the standard deviation for each fold on the 3 metrics:





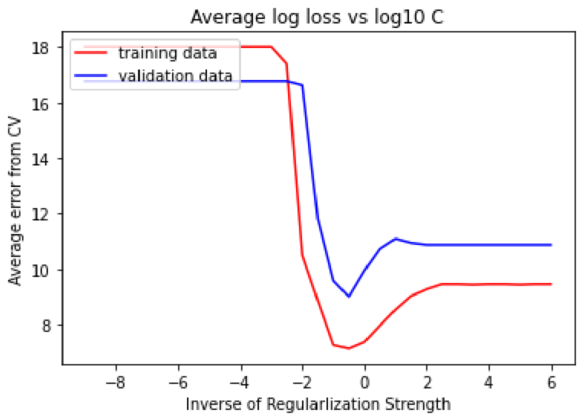
Through testing for different alpha values in the same way I tested for C values in section 2., I arrived at alpha=0.03162277660168379. At this point average error is minimized for both training and validation datasets while average score and ROC is at maximum. Furthermore, although the standard deviations for both datasets start at different points, at the chosen alpha point all 3 metrics' standard deviation converges to approximately 0.04 for training dataset and 0.06 for validation dataset. Overall, this model has relatively low uncertainty.

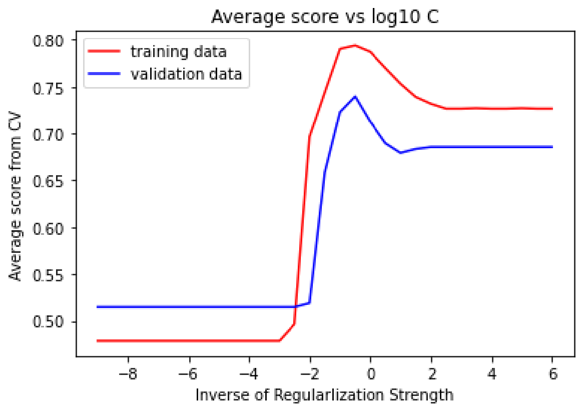
4. SVM Classifier on BoW representation

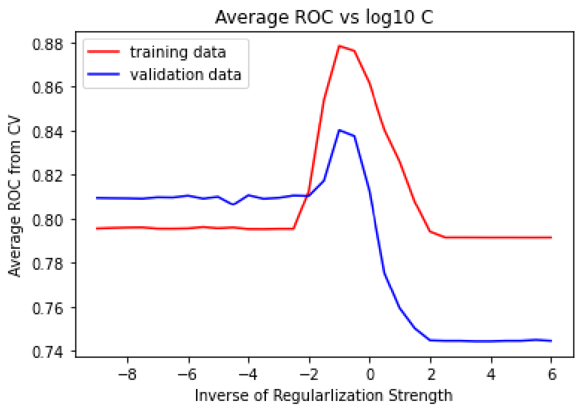
Chosen Model: SVC(decision_function_shape='ovo',
gamma='scale', kernel='linear', shrinking=True,
C=0.31622776601683794)

Again, these hyperparameters were chosen through a grid search. Hyperparameter values:
Kernel: [linear, poly, rbf, sigmoid]
Shrinking: [True, False]
Decision_function_shape: [ovo, ovr]
Gamma: [scale, auto]

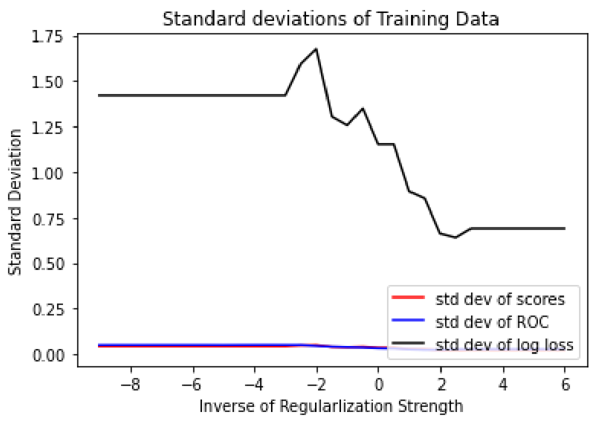
After these parameters were found, the appropriate regularization strength had to be found. This was done in a similar fashion as the Logistic Regression, through 12-fold cross validation.
Again, logspace(-9, 6, 31) was used because it covered the appropriate range of regularization strengths.

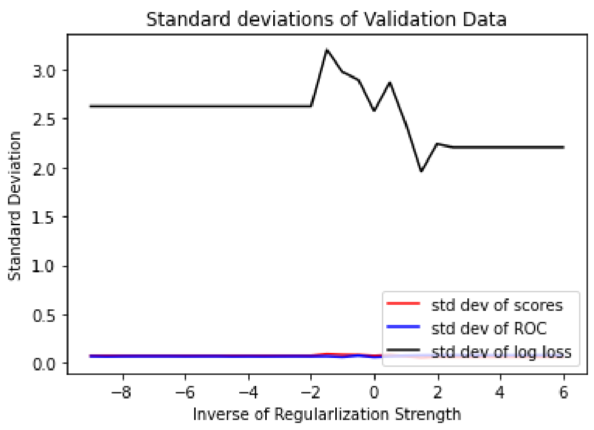






Standard deviation for each fold on the 3 metrics:





The average error, score, and ROC curves follow the same trend for both training and validation data. Furthermore we see the lowest error, score, and ROC values at C=0.316 (which is a negative value on the graphs due to all C values being scaled by log_10)… and any further on the performance suffers. One concerning fact is the uncertainty for the error at the chosen C value is rather high, which is shown by a relatively high standard deviation.

5. Best classifier

	Avg score (testing, validation)	Avg ROC (testing, validation)	Avg error (testing, validation)
Logistic Regression	0.7953125, 0.4770833	0.877664525371, 0.454893063888	0.47600826, 0.73695301
MLPClassifier	0.7578125, 0.4687499	0.8432703286904, 0.4680810919332	1.1331617126756413, 2.3464367803516795
Support Vector Classifier	0.79427083, 0.44791666	0.8762253293657722, 0.4582238156642907	7.105702817272466, 19.06849769316630

With a 80:20 train-test-split, the 12-fold cross-validation test was run again on the best models. Above are the results.
All 3 metrics of all models data for validation dataset is rather concerning. None of the models go above 0.5 average score and their average errors are all high (lowest one being 0.7369...). However this is due to how the data is preprocessed and split up. Since the bag of words is formed based on the input data, if the input data is very small then the bag of words will also be small. This could be a potential cause on why the validation dataset metrics are so low.

The best model by average scores, ROCs, and errors is the Logistic Regression model. It has the highest average score for both training and validation datasets. The Support Vector Classifier also has a rather high average score but this model is not suitable due to its high average error.

6. Performance

The Logistic Regression model achieved an AUROC of 0.85692 and an error rate of 0.22167. Furthermore, MLPClassifier model was also run and it achieved an AUROC of 0.85053 and an error rate of 0.225. This result was expected as my cross validation test on the validation data achieved the best results on all 3 metrics.

Word Embeddings

1. Formation of vectors

Initially, the text is preprocessed.

1. All punctuation is removed and text is converted to lowercase.
2. Text is parsed and all stop words are removed.

For this representation text was not stemmed because it seemed that the given word embeddings are dependent on spelling. Since the PorterStemmer often stems words to incorrect spelling bases this step was not applied. Furthermore, if the misspelled text does not exist in the given word embeddings then some constant value's word embedding is returned (in this case the vector for word "arbitrary").
Punctuations were removed to avoid getting different vectors for the same words. For instance, "negative" and "negative???" can have different vectors. To standardize this all punctuations were removed. The same is applied for the reason why all text was changed to its lowercase form.
Lastly, stop words were removed to avoid combining word vectors with little sentiment meaning. Words such as "and", "but", and "the" are grammatically important but they carry little to no sentiment information.

After the preprocessing, each of the sentences had to be converted to a single feature vector that comprised all the individual word vectors in the sentence.

Steps in words vectors to a single sentence vector:

1. For each word in the sentence look up the corresponding vector in the given pre trained word embedding dictionary.
2. If the queried string is not found in the word embedding, return the vector of text 'arbitrary'[1].
3. After getting all the individual word vectors, these are all combined by averaging the i-th entry in the vector.

   For instance,
   Combining <1, 0, 4, 5>, <2, 1, 2, 0>, <0, 0, 0, -1> would yield <1, 1/3, 2, 4/3>
4. Aggregate these combined vectors to produce the input training/testing set of features.

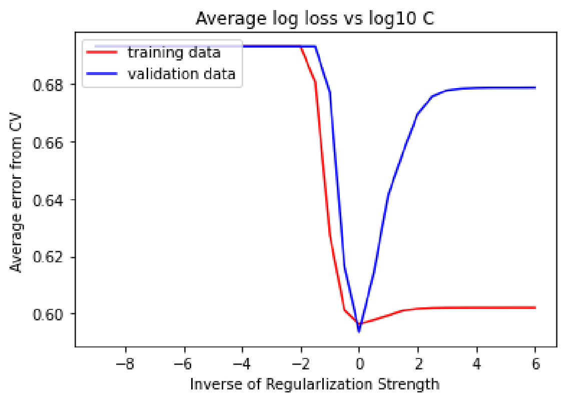
[1] This step had to be done for handling value not found exceptions. Information-wise since all unfound cases are handled this way there should be little effect in sentiment analysis (i.e. due to its consistency this step is justified).

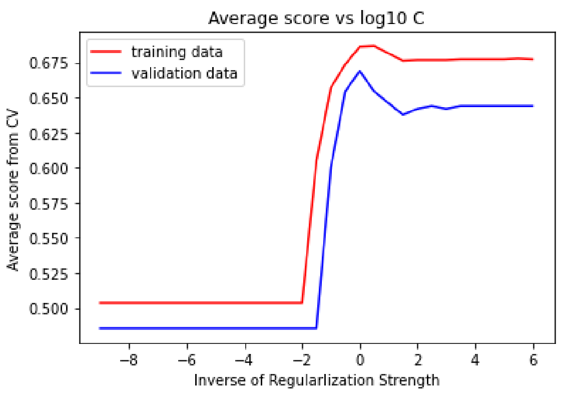
2. Logistic Regression on Word Embeddings

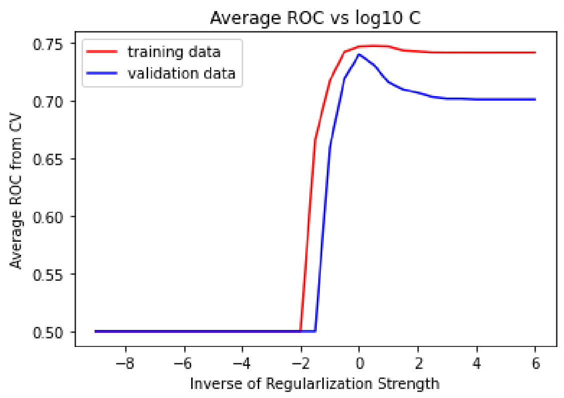
Chosen Model: LogisticRegression(C=1.0 , penalty='l2',
solver='liblinear', warm_start=True)

Hyperparameters were chosen through a gridsearch with 3-fold cross validation. The grid search was done on the entire given training dataset (no test-train-split).
Hyperparameter values:
Penalty: [L1, L2, elasticnet, none]
Solver: [newton-cg, lbfgs, liblinear, sag, saga]
Warm_start: [True, False]

After the grid search the appropriate regularization strength is searched for. Here, we once again try the ranges of logspace(-9, 6, 31) on the training and validation datasets.

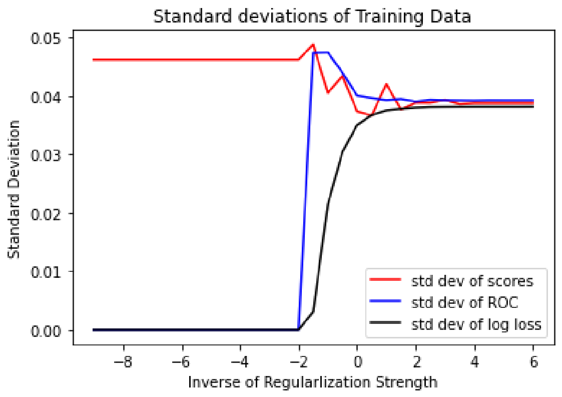


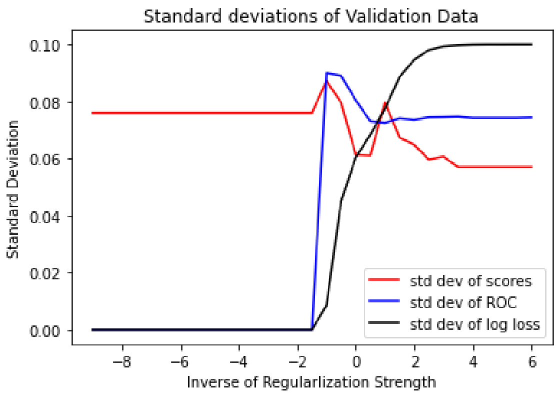




Here, we see a classic example of a model being overfit. The log loss graph shows the log loss for the training data improving then coasting. On the other hand, the log loss graph for the validation data shoots up after the C value. The model is not learning but memorizing the data set at this point and this is evinced by the performance on log loss for validation data plummeting.

Standard deviations of 3 metrics:





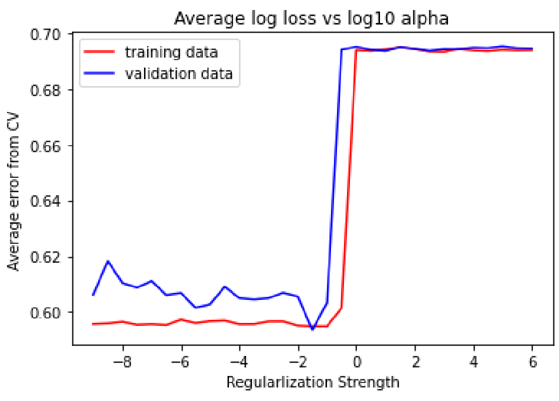
Standard deviations for the training and validation data are both relatively low at approximately 0.04 at C=1.0 (which is 0 after being scaled by log_10) for training data and 0.07 for validation data for all 3 metrics. Interestingly, while the standard deviations for the 3 metrics start out as very different, they all seem to converge at an approximate area given the appropriate regularization strength. Overall, the uncertainty for this model is rather low as the standard deviation never goes above 0.1.

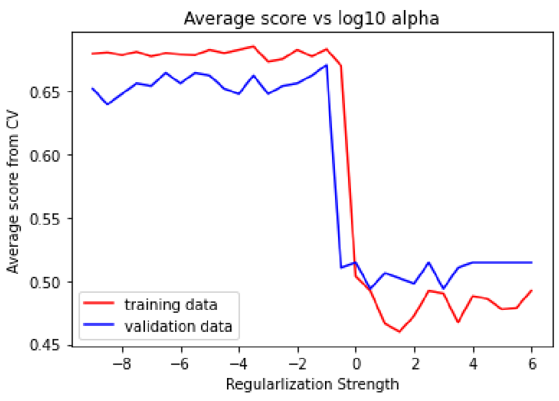
3. MLP on Word Embeddings

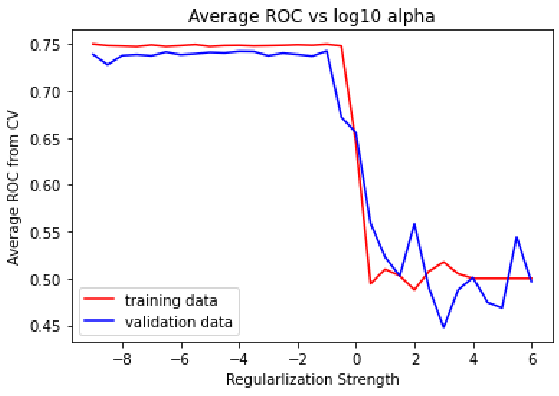
Chosen Model: MLPClassifier(hidden_layer_sizes=(50, 100),
activation='logistic', learning_rate='adaptive',
momentum=0.1, nesterovs_momentum=False,
solver=adam, warm_start=False,
alpha=0.00031622776601683794)

These hyperparameters were chosen after a grid search. Following values were searched for in the grid search with the entire training dataset:
Hidden_layer_sizes: [(50, 100), (80), (50, 50), (50, 30), (100, 90)]
Activation: [identity, logistic, tanh, relu]
Learning_rate: [constant, invscaling, adaptive]
Momentum: [0.1, 0.5, 0.9]
Nesterovs_momentum: [True, False]
Warm_start: [True, False]

After finding the best hyperparameters for the dataset, an appropriate alpha value (regularization strength) was looked for with the training and validation dataset.

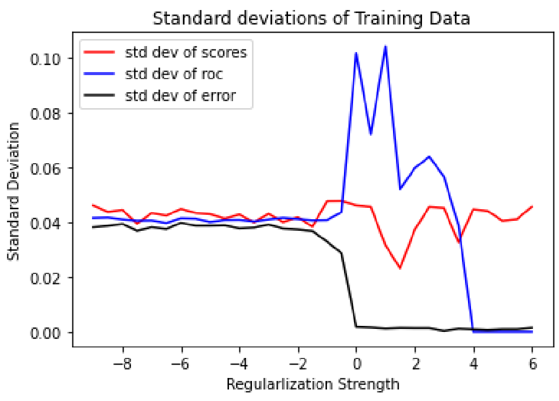


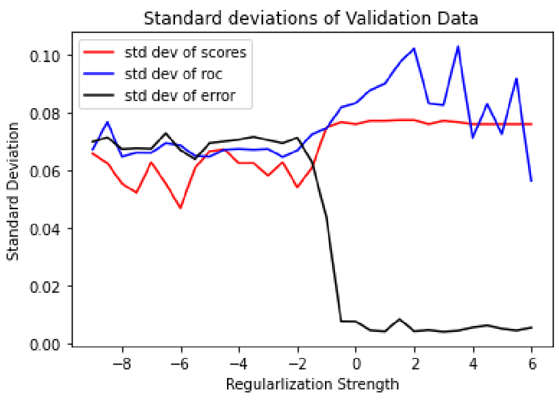




Both training and validation datasets seem to trend similarly on all 3 metrics. The average error is minimized at alpha=0.00031… and average score, ROC are also maximized at that point. Therefore the chosen alpha value is the best value to use as there is no evidence of overfitting.

Standard deviations of the 3 metrics:





The standard deviations for the training and validation datasets seem to be stable and relatively low as long as alpha <= 0.00031622776601683794. After that point the model gets incredibly uncertain as the graphs show the standard deviation of the average error reaching 0.0 and the standard deviation of average scores staying high for both training and validation datasets. This implies that for alpha values larger than 0.00031622776601683794 the model is most likely overfitting. Overall, this model's uncertainty is relatively low.

4. SVM Classifier on Word Embeddings

Chosen Model: SVC(kernel='rbf', gamma='scale',
shrinking=True, C=1.0)

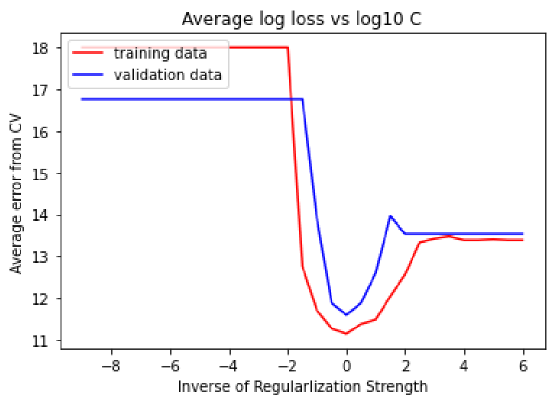
Aside from the C-value all other parameters were chosen through a grid search over the entire training dataset (no test-train-split). The hyperparameter space searched through:

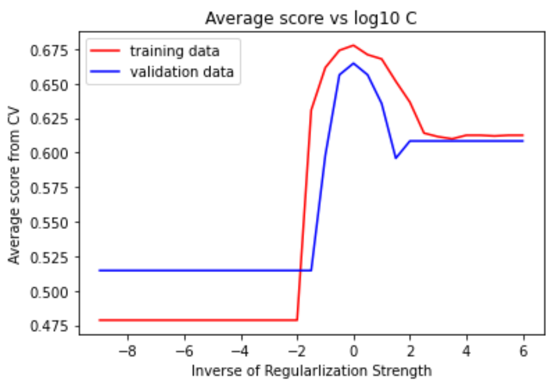
Kernel: [linear, poly, rbf, sigmoid]
Gamma: [scale, auto]
Shrinking: [True, False]
Decisition_function_shape: [ovo, ovr]

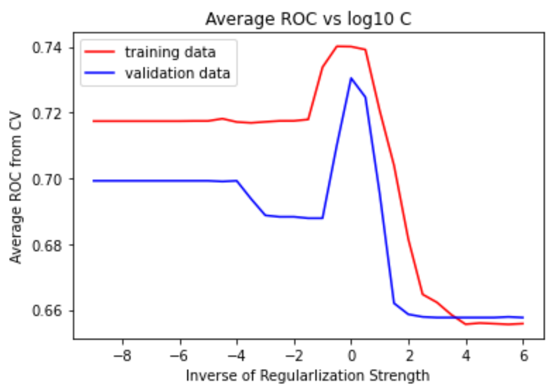
After finding the best hyperparameters for the hyperparameter space above another space was tried with the 'kernel' parameter set to 'poly'.

Degree: [2, 3, 4, 5, 6, 7, 8, 9, 10]
Gamma: [scale, auto]
Shrinking: [True, False]
Decision_function_shape: [ovo, ovr]

This hyperparameter space was searched due to the degree parameter being applied only to the poly kernel. However, the performance of these models were inferior to the grid search from before so the 'poly' kernel was not used.

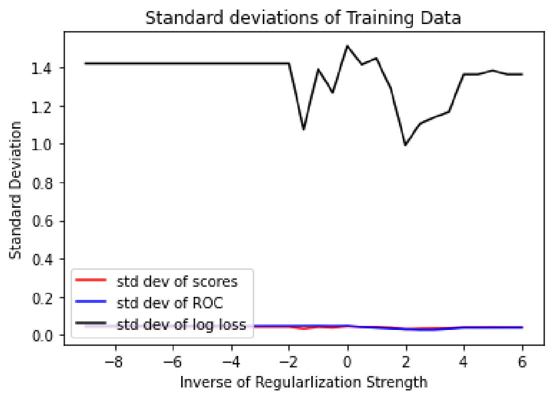


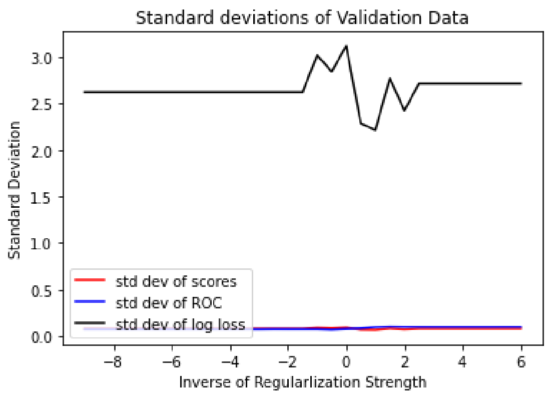




In the graphs, we see the lowest average error for both training and validation data at C=1.0. Furthermore, both average score and ROC are at maximum when C=1.0.

Standard deviations of the 3 metrics:





This model's standard deviation of the average error metric seems to be very high for both training and validation datasets. As a result this model has a relatively high uncertainty.

5. Best Classifier

	Avg score (testing, validation)	Avg ROC (testing, validation)	Avg error (testing, validation)
Logistic Regression	0.683854166, 0.660416666	0.747547001641, 0.73515977448	0.596064112795, 0.595815959173
MLPClassifier	0.678645833 0.645833333	0.74767270450, 0.74196367439	0.595459168102, 0.607931279108
Support Vector Classifier	0.67760416, 0.66458333	0.74005980759, 0.73046293363	11.13525338325, 11.58504616609

With a 80:20 train-test-split, the 12-fold cross-validation test was run again on the best models. Above are the results.
For embedded wording representation, the results of training and validation data are very consistent unlike the bag of words representation. Highest average score on the validation data is achieved by the Support Vector Classifier but since other models also have a similar average score and SVC's average error being 11.5… this model is dismissed. Next best model is the Logistic Regression model which has the second highest average score on and lowest average error on the validation data. Thus, this is the chosen best model.

6. Performance

The logistic regression model was able to achieve an error rate of 0.33667 and an AUROC of 0.72726. Additionally the MLPClassifier achieved an error rate of 0.335 and an AUROC of 0.72702. These two models produced the same score, 2.3166 out of 3. This result was not unexpected as the cross validation test data on the validation dataset above shows MLPClassifier and Logistic Regression producing similar results.