CS 4641: Machine Learning

Final Project — Experimental Design for Supervised Learning

Submitted to Dr. Hrolenok

12/6/2019

Introduction

I really enjoy listening to music, and I listen to it in a way that is common to many people. I save the songs I like, and when I like a couple songs from the same album, I listen to the whole thing. Fortunately, I listen to music on Spotify, and they provide access to audio features for each of their songs. The goal of the following project was to create a classification model that will use these features to predict whether I would save a song or not. The metric used to measure performance is precision. I wanted to reduce the likelihood of false positives (I would rather that most of the songs that the model recommends be good, even if I miss out on other potentially good songs).

The dataset consists of 3797 examples, with about 40% having the positive class label (saved) and about 60% having the negative class label (unsaved). Spotify provides 12 audio features, described further in Table 1 below.

Table 1: Audio features and their descriptions from Spotify[1]

Name	**Description**	**Data Type**	**Range / Possible Values**
Danceability	How suitable is a track from dancing (1.0 is most danceable)	Continuous	0.0 to 1.0
Energy	Perceptual measure of intensity and activity (1.0 is most energetic)	Continuous	0.0 to 1.0
Key	The estimated overall key of the track in standard Pitch Class notation	Ordinal	Integers from 0 to 11 -1 if no key was detected
Loudness	Overall loudness in decibels	Continuous	-60.0 to 0.0
Mode	Modality (major or minor) of a track	Binary	1 for Major 0 for Minor
Speechiness	Exclusiveness of spoken words in a track (1.0 is exclusively spoken words – poetry for example)	Continuous	0.0 to 1.0
Acousticness	Confidence that the track is acoustic (1.0 is high confidence)	Continuous	0.0 to 1.0
Instrumentalness	Likelihood that a track contains vocals (1.0 is very likely)	Continuous	0.0 to 1.0
Liveness	Likelihood that an audience was present (1.0 is very likely)	Continuous	0.0 to 1.0
Valence	Musical positiveness of the track (1.0 is very positive)	Continuous	0.0 to 1.0
Tempo	Estimated beats per minute	Continuous	Positive real numbers
Time signal	The number of beats in a measure	Discrete	Positive integers

[1] https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/

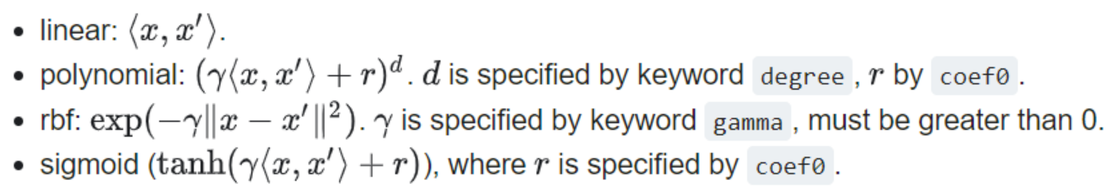
The *Mode* feature is a binary variable and was encoded in two columns: Mode Major and Mode Minor, such that a 1indicates the presence of the category and a 0 indicates the absence of a category. This results in a total of 13 features for the model. The dataset is included in dataset.csv. For more information on how the dataset was constructed, please refer to the Dataset Construction.ipynb or Dataset Construction.html files that are included.

Algorithms

Three algorithms will be tested: Random Forests with Boosting, Support Vector Machines, and Neural Networks. A brief description of each algorithm and its hyperparameters is as follows:

Random Forests with Boosting – This algorithm trains many decision trees in sequence, training each tree to prioritize the points that the previous tree classified incorrectly. The implementation used will be sklearn.ensemble.AdaBoostClassifier from the scikit-learn package. The hyperparameters to be tuned are the number of trees and the learning rate. The learning rate determines how much weighting to give the incorrectly classified points for the following tree. A larger learning rate will likely require few trees before the performance stabilizes, as the classifier will quickly fit. A smaller learning rate will likely require many trees before the performance stabilizes, as the classifier will slowly learn the decision boundary. The number of trees controls the complexity of the hypothesis class, as more trees will allow for a more complex model.

Support Vector Machines – This algorithm finds the hyperplane that creates the maximum margin between two classes. For data that is not linearly separable, a nonlinear kernel can transform the inputs into a higher dimensional space where they are more likely to be linearly separable. The implementation used will be sklearn.svm.SVC from the scikit-learn package. The hyperparameters to be tuned are the kernel, penalty parameter C, gamma, and the degree for the polynomial kernel. The kernels impact the complexity of the hypothesis class, with the rbf, sigmoid, and polynomial kernels being more complex than the linear kernel, and higher order polynomials corresponding to increased complexity. The kernels that will be explored are shown in Figure 1 below. The penalty parameter C penalizes softer margins, and lower C values allow more outliers. Higher C values will encourage overfitting as they will drive the model to have fewer mistakes.



- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma\langle x, x' \rangle + r)^d$. d is specified by keyword `degree`, r by `coef0`.
- rbf: $\exp(-\gamma\|x - x'\|^2)$. γ is specified by keyword `gamma`, must be greater than 0.
- sigmoid $(\tanh(\gamma\langle x, x' \rangle + r))$, where r is specified by `coef0`.

Figure 1: Kernel hyperparameter options for sklearn.svm.SVC[2]

[2] https://scikit-learn.org/stable/modules/svm.html#svm-kernels

<u>Neural Networks</u> – This algorithm uses a multilayer perceptron network and learns the weights between each perceptron using backpropagation to minimize error. The implementation used will be `sklearn.neural_network.MLPClassifier` from the `scikit-learn` package. The hyperparameters to be tuned are the activation function, number of hidden layers, width of hidden layers, and regularization parameter alpha. The number and width of hidden layers correspond to the model's complexity, with a deeper or wider network corresponding to a higher complexity hypothesis class. Alpha is a regularization parameter, and higher values reduce overfitting.

Hyperparameter Tuning

To tune the hyperparameters, the dataset will be split up such that 40% of the data is withheld for comparing model performance later, and 60% of the data is used for training and hyperparameter tuning. This corresponds to 2278 examples for training and hyperparameter tuning. The hyperparameters for each algorithm will be tuned using a grid search across a subset of the hyperparameter space. To calculate the precision of each hyperparameter combination, 5-fold cross-validation using the training data will be used. Each fold will contain approximately 456 examples, so for each hyperparameter setting, there will be about 1822 examples for training and 456 examples for validation. This will lead to 5 measurements of precision for each hyperparameter setting. A Student's t distribution will be used to calculate the 95% confidence interval of the precision metric. All hyperparameter tuning experiments will follow this sampling procedure.

Random Forests with Boosting

<u>Experiment 1</u>
The purpose of this experiment is to find the hyperparameter settings that maximize precision for the Random Forests with Boosting algorithm. All combinations of the following hyperparameter settings will be tested:

- Learning Rate: 1, 0.1, 0.01, 0.001, 0.0001
- Number of Trees: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350

<u>Results</u>
The total computation time for this experiment was approximately 357 seconds. Based on Figure 2 below, for all learning rates less than or equal to 0.01, the precision was 0.0. This indicates that the model was either not predicting any songs as 'saved' or that all the songs it predicted as 'saved' were incorrect. A learning rate less than or equal to 0.01 would require more than 350 estimators to produce a useful model. For a learning rate of 0.1, as the number of estimators increase from 10 to 100, the model learned more about the data and both training and validation precisions increased. Beyond 100 estimators, the model performance plateaued with a precision of approximately 0.45. For a learning rate of 1, the model had a roughly constant validation precision of about 0.45 for 10 or more estimators, though the training precision continually increased, indicating overfitting. Based on these results, another experiment was designed with finer intervals for the learning rate.

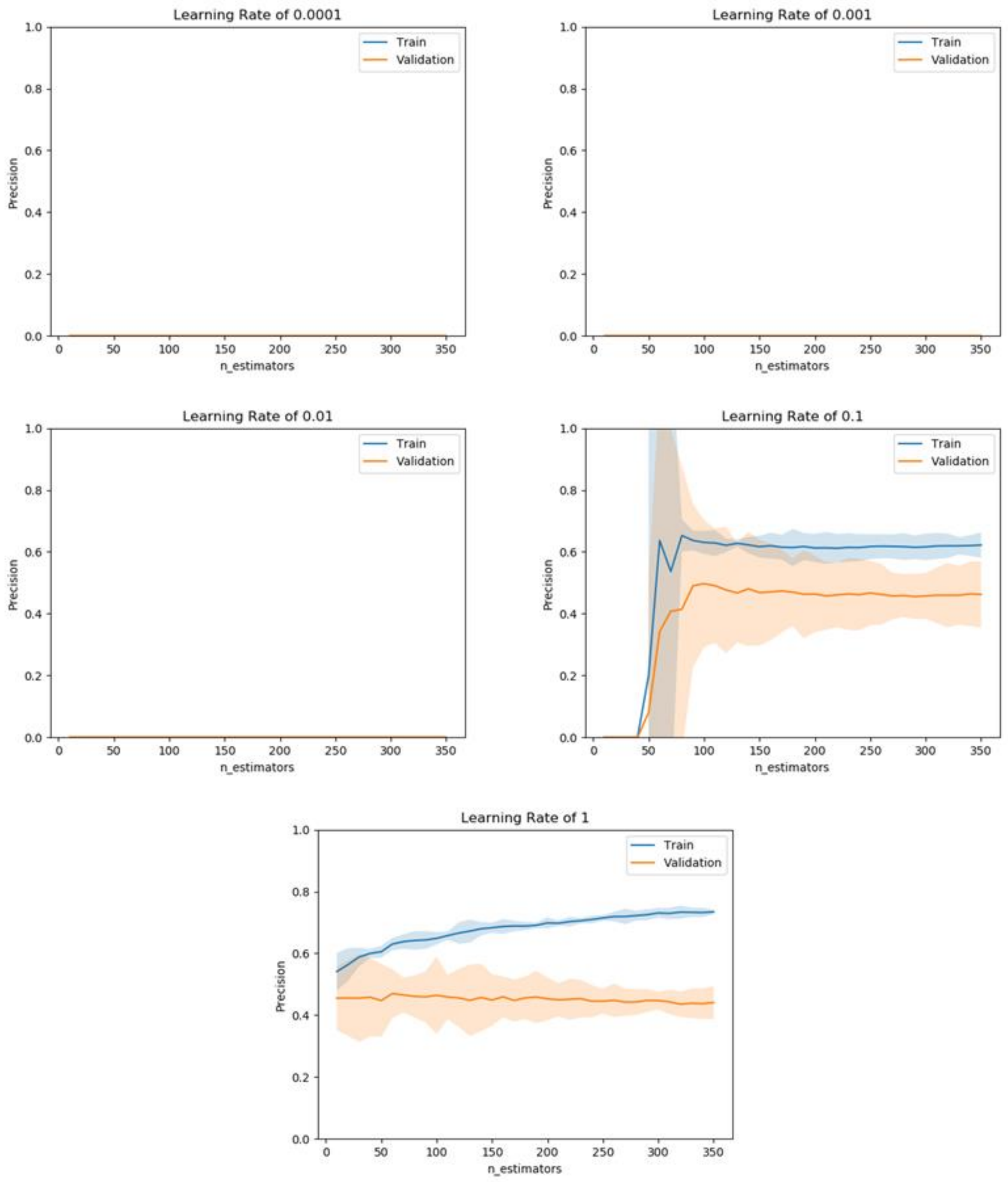


Figure 2: Experiment 1 model complexity curves for Random Forests with Boosting.

<u>Experiment 2</u>
The purpose of this experiment is to find the hyperparameter settings that maximize precision for the Random Forests with Boosting algorithm using finer intervals for learning rate. All combinations of the following hyperparameter settings will be tested:

- Learning Rate: 0.03, 0.05, 0.07, 0.09, 0.3, 0.5, 0.7, 0.9
- Number of Trees: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350

Results

The total computation time for this experiment was approximately 538 seconds. As shown in Figure 3 below, for a learning rate of 0.03, the model did not improve until it had 160 estimators, and it required about 240 estimators to achieve a precision comparable to other models. With a learning rate of 0.05, fewer estimators were required to start improving (only 80). The rest of Figure 3 and Figure 4 makes the trend clear: as learning rate increases, fewer estimators are required for the model to start improving. For all learning rates, the validation precision eventually rose to about 0.45 and plateaued, and the peak performance for each learning rate is not statistically significantly different. The learning rate that achieves this performance with the fewest number of estimators and thus the simplest model is 0.7, as the validation precision for 10 estimators is about 0.5.

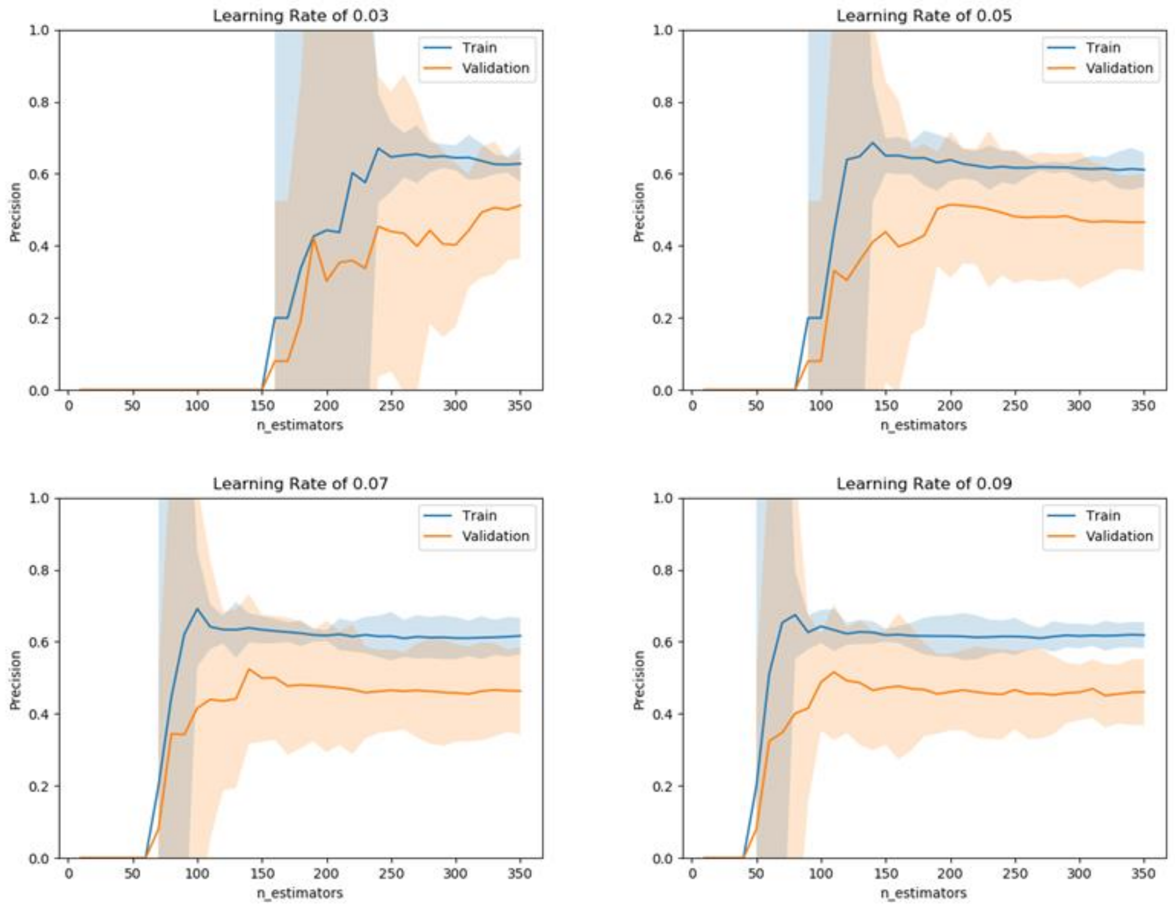


Figure 3: Experiment 2 model complexity curves for Random Forests with Boosting.

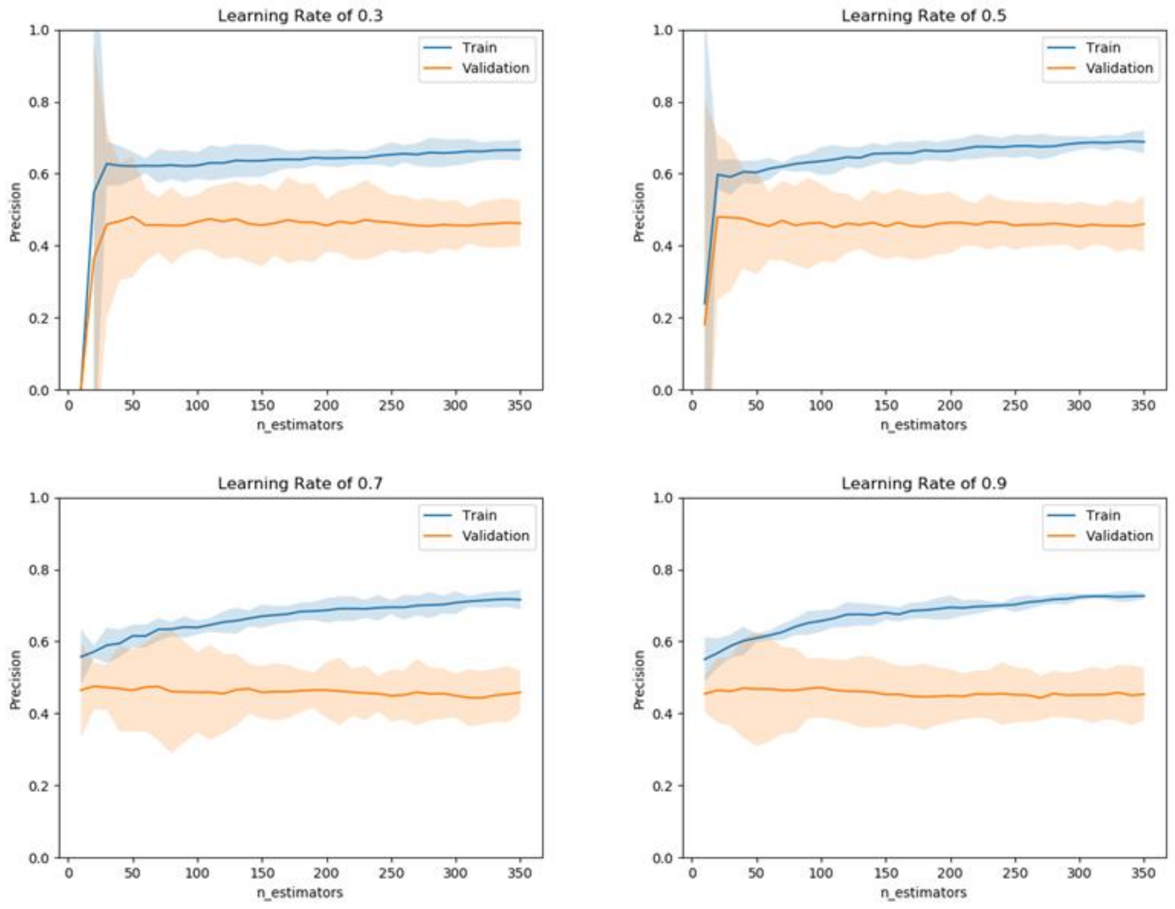


Figure 4: Experiment 2 model complexity curves for Random Forests with Boosting.

Hyperparameter Choice
Based on these results, the final hyperparameters chosen for the Random Forests with Boosting algorithm are a learning rate of 0.7 and 10 estimators. The validation precision for these hyperparameters was 0.46 ± 0.13.

Support Vector Machines

Experiment 3
The purpose of this experiment is to find the hyperparameter settings that maximize precision for the Support Vector Machines algorithm. The following hyperparameter settings will be tested:

- Linear kernel
    - C: 100, 10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001
- Sigmoid and rbf kernels
    - C: 100, 10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001
    - Gamma: 1, 0.1, 0.01, 0.001
- Polynomial kernel
    - C: 1, 0.1, 0.01, 0.001, 0.0001, 0.00001

- Gamma: 1, 0.1, 0.01, 0.001
- Degree (d): 3, 5, 7

<u>Results</u>

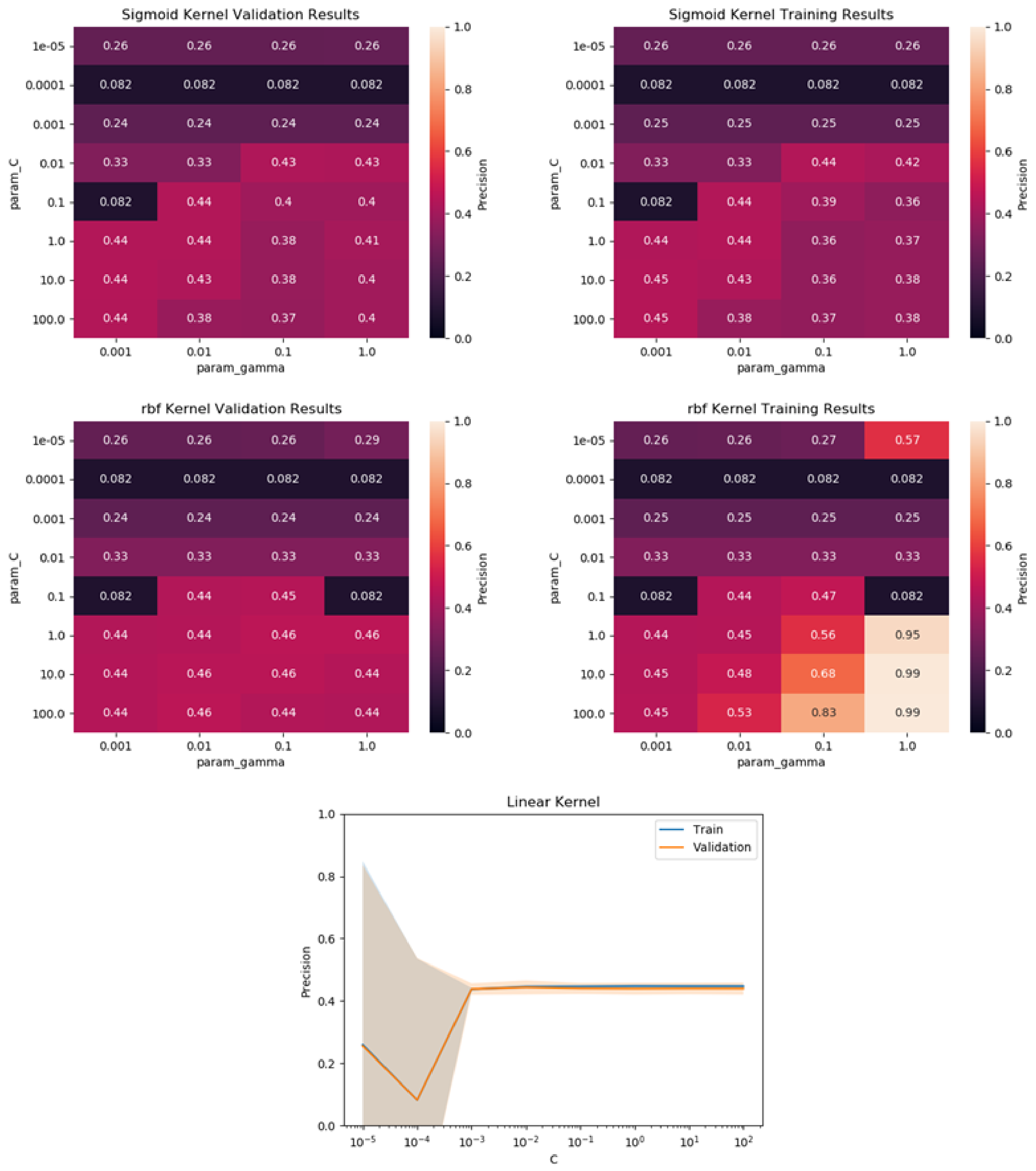


Figure 5: Experiment 3 hyperparameter heatmaps for Sigmoid and rbf kernel Support Vector Machine. Also a model complexity curve for the Linear kernel Support Vector Machine.

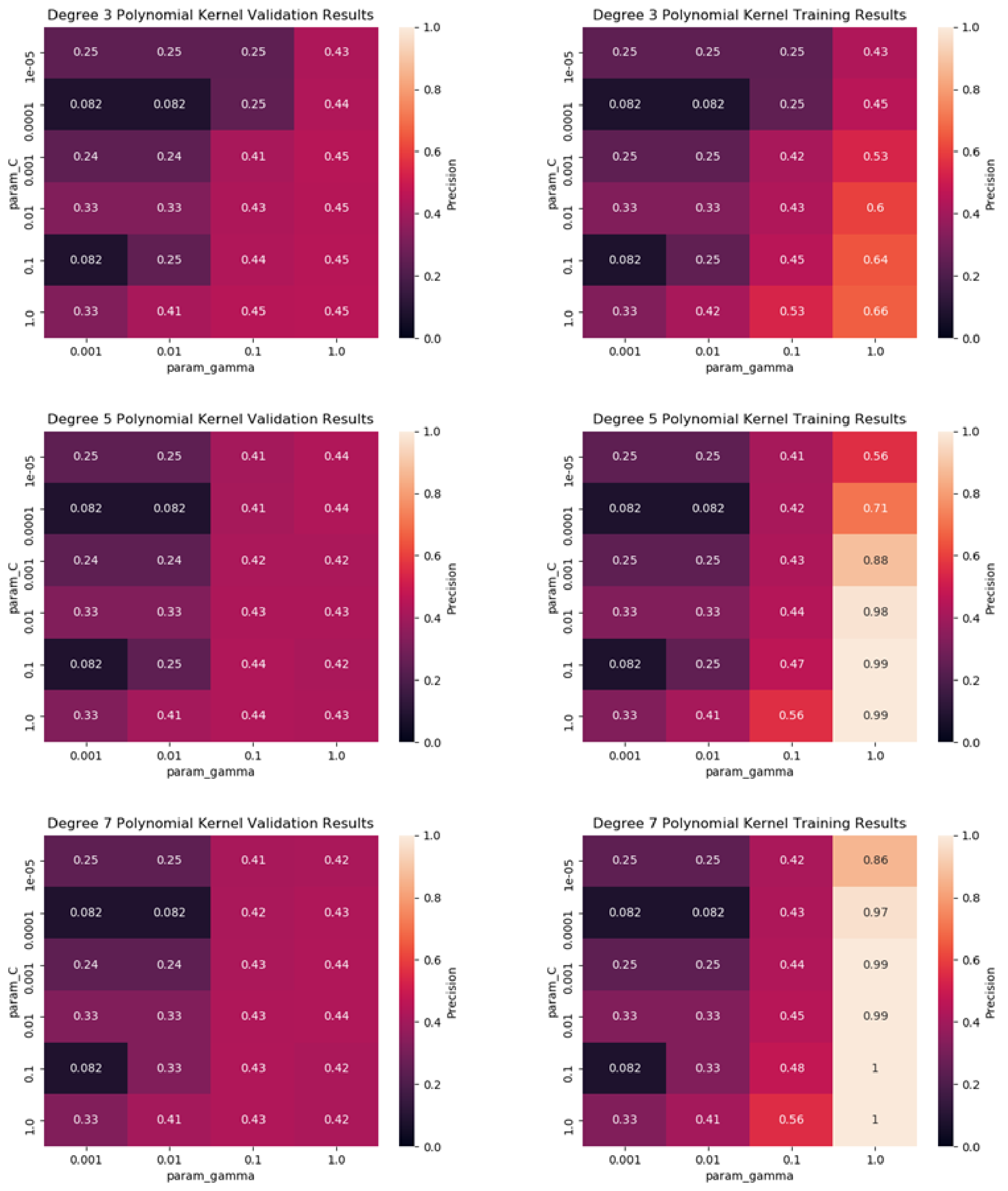


Figure 6: Experiment 3 hyperparameter heatmaps for polynomial kernel Support Vector Machine.

The total computation time for this experiment was about 545 seconds. Based on Figure 5, the peak validation precision for the sigmoid, rbf, and linear kernels were about 0.45. For the sigmoid and linear kernels, the model does not seem to be overfitting for any values of gamma and C, as the training and validation results are very similar. However, for the rbf kernel with C less than 0.1, the model seemed to be overfitting as gamma increased as the training precision increased beyond the validation precision. For the sigmoid and rbf kernels, as C increases, validation precision increases and then plateaus after C=0.1. For the sigmoid kernel, as gamma decreases, the validation precision increases. For the rbf kernel, gamma does not seem to affect validation precision. As shown in Figure 6, the peak validation precision for all the polynomial kernel models was also about 0.45. All the polynomial kernel models exhibited the trend that as C increased and as gamma increased, validation precision increased. Additionally they all showed signs of overfitting, because as C and gamma increased, training precision grew beyond validation precision.

Hyperparameter Choice

Based on these results, the final hyperparameters chosen for the Support Vector Machine is the linear kernel with C = 10^{-3}. These hyperparameters were chosen because the model had a validation precision of 0.44 ± 0.02 which is not statistically significantly different from the validation precision of any of the other hyperparameter settings, and is the simplest of the models.

Neural Network

Experiment 4

The purpose of this experiment is to find the hyperparameter settings that maximize precision for the Neural Network algorithm. All combinations of the following hyperparameter settings will be tested:

- Activation: logistic, relu, tanh
- Number of hidden layers: 2, 3
- Layer width: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25
- Alpha: 1, .1, .01, .001, 0.0001, .00001, .000001, .0000001

Results

The total computation time for this experiment was about 949 seconds. Based on Figures 7 and 8, with both 2 and 3 hidden layers, all three activation functions produced similar results for peak validation precision. For all three activation functions, the training results were generally better than the validation results, suggesting overfitting. However, there seemed to be more overfitting for the tanh and relu activation functions than for the logistic activation function, as their heatmaps show a lighter color overall, indicating higher training precision. For 2 hidden layers and all three activation functions, as the nodes per layer increased beyond about 7, there was no apparent improvement in validation performance. The same trend was seen for 3 hidden layers, except with the logistic activation function, which had poorer performance for certain layer widths. Training performance increased beyond validation performance as the nodes per layer increased for all combinations of activation functions and hidden layer depths, suggesting that the models with more than 7 nodes per layer are overfitting. The alpha parameter does not have a discernible effect on validation results for all three activation functions. However, for the training results, decreasing the alpha parameter appears to decrease the precision.

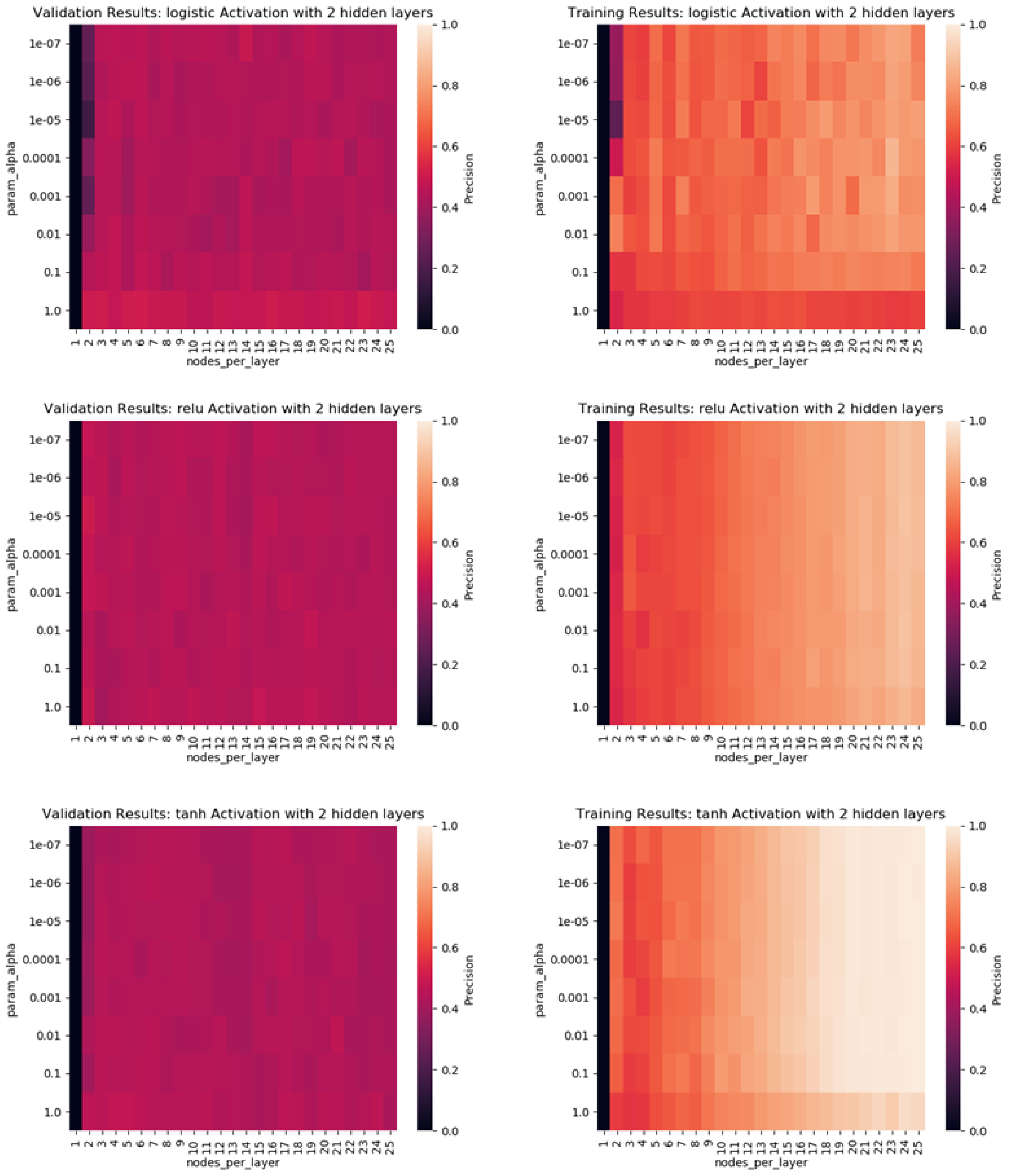


Figure 7: Experiment 4 hyperparameter heatmaps for Neural Network with 2 hidden layers.

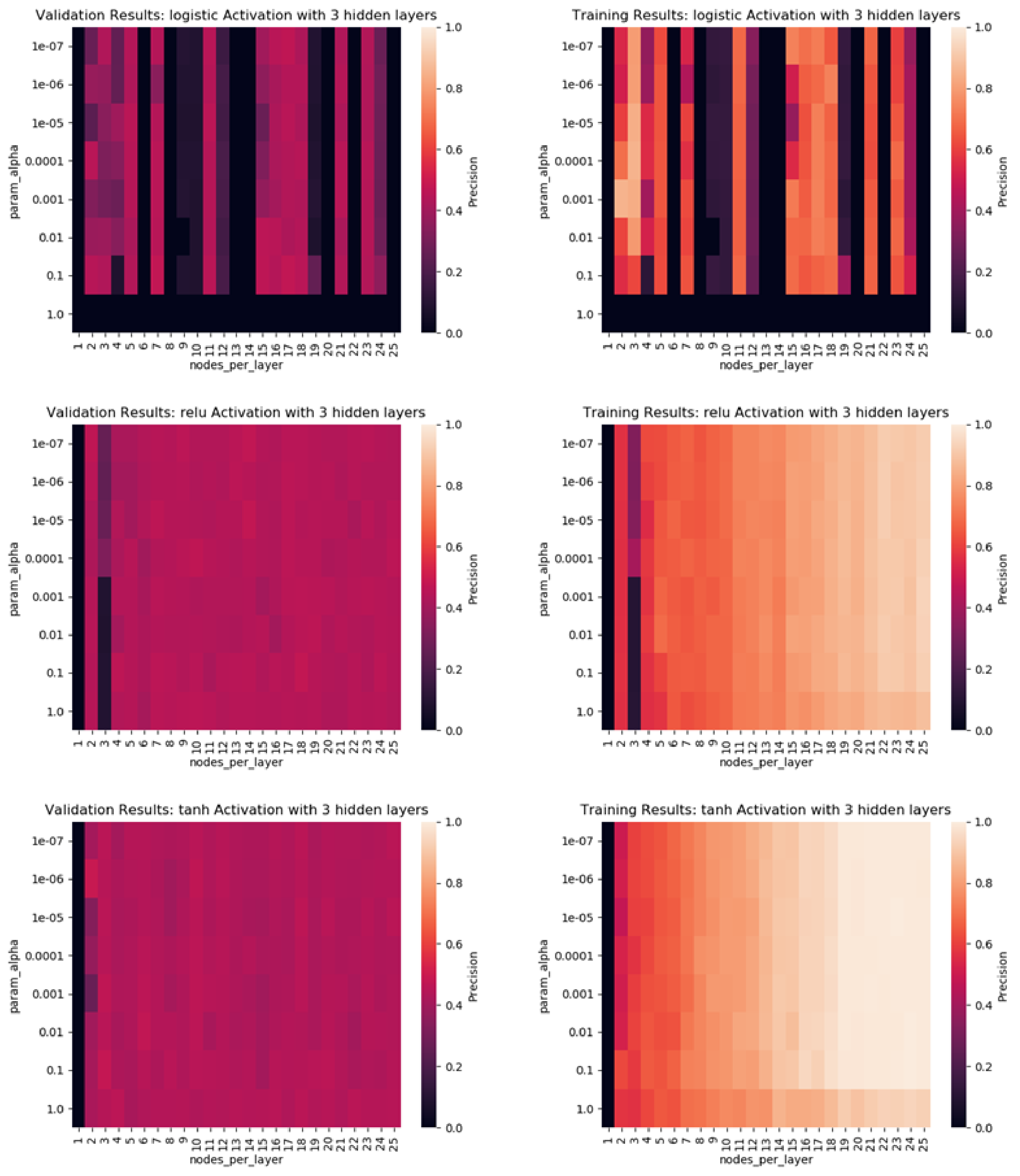


Figure 8: Experiment 4 hyperparameter heatmaps for Neural Network with 3 hidden layers.

The plot in Figure 9 below shows the performance of the best model for each activation function and number of hidden layers combination. The validation results of all the models are not statistically significantly different from one another.

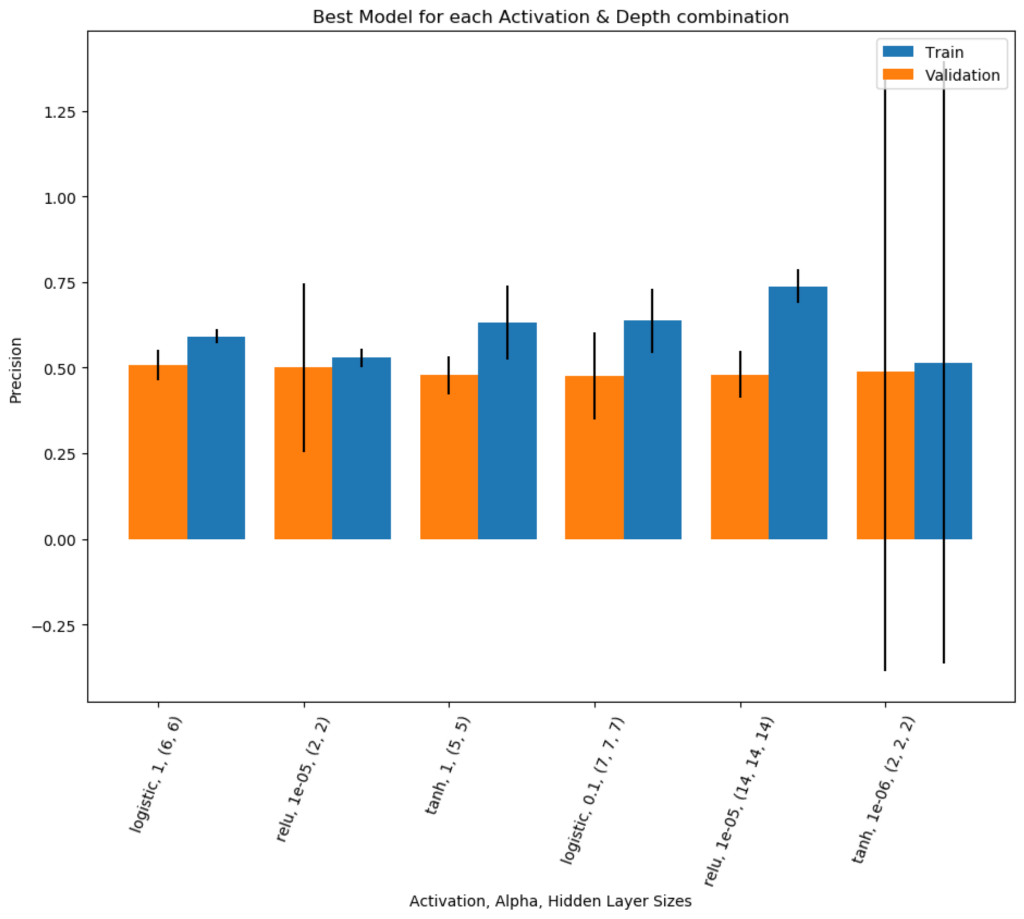


Figure 9: Best Neural Network hyperparameter settings for each activation function and depth combination.

Hyperparameter Choice

The hyperparameters chosen for the neural network are an activation function of tanh, 2 hidden layers, 5 nodes per layer, and an alpha of 1. Since none of the six best models were statistically significantly different from one another, a model with 2 hidden layers was chosen because it is simpler than a model with 3 hidden layers. Of the remaining models, the logistic activation function had a mean fit time of 0.63 seconds, which was 3 times longer than the relu and tanh functions which had mean fit times of 0.25 and 0.29 seconds respectively. Between the tanh and relu activation functions, the tanh function had a much smaller standard deviation. The final model chosen had a precision of approximately 0.48 ± 0.02.

Algorithm Performance

As stated previously, the entire dataset was split up such that 40% of the data was withheld for comparing model performance, and 60% of the data was used for hyperparameter tuning. This corresponds to 2278 examples for training and 1519 examples for testing. To calculate the precision of each of the three algorithms, the testing data will be split into 3-folds and the algorithms will be trained on the training data and 2 folds of the testing data, and will be tested one the remaining fold of the testing data. This corresponds to about 3290 examples for training and 506 examples for testing. This will lead to 3 measurements of precision for each algorithm. A Student's t distribution will be used to calculate the 95% confidence interval of the precision metric.

Experiment 5

The purpose of this experiment is to find the tuned algorithm with the best precision. Three algorithms will be compared:

- Random Forests with Boosting: 10 estimators, learning rate = 0.7
- Support Vector Machines: linear kernel, C = 10^{-3}
- Neural Network: tanh activation function, 2 hidden layers, 5 nodes per layer, alpha = 1

Results

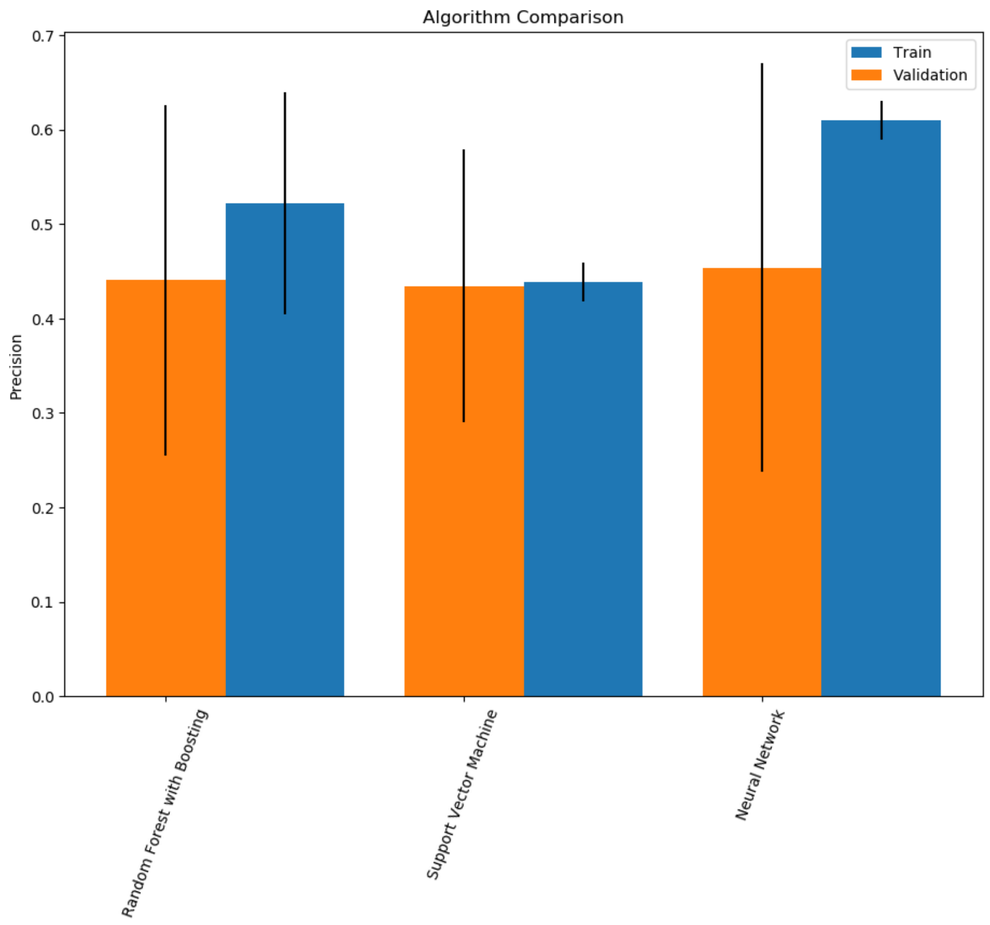


Figure 10: Algorithm Performance Comparison

Based on the validation results in Figure 10, the precision was not statistically significantly different for the three algorithms. The Random Forest with Boosting had a precision of 0.44 ± 0.19. The Support Vector Machine had a precision of 0.43 ± 0.14. The Neural Network had a precision of 0.45 ± 0.22. Based on these results, all three algorithms are an equally good choice.

The training times for the three algorithms were as follows:

- Random Forests with Boosting: 0.12 seconds
- Support Vector Machines: 0.53 seconds
- Neural Network: 0.52 seconds

The confusion matrices are shown in Figure 11 below. In each cell is the mean value of the corresponding measurement.

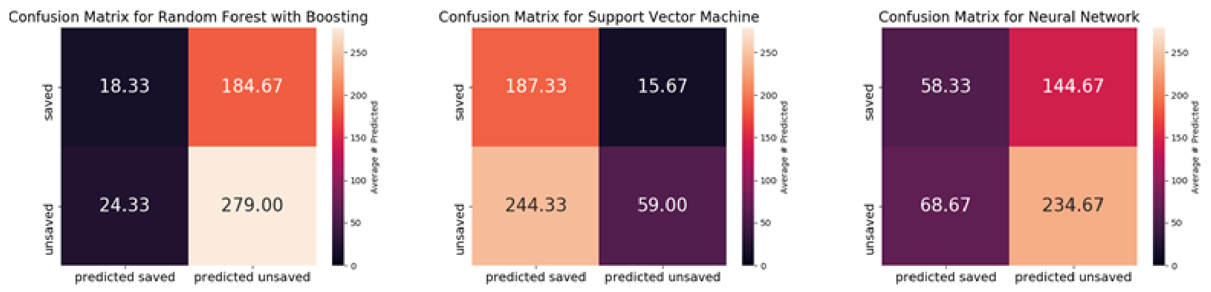


Figure 11: Confusion Matrices for final algorithms.

Conclusions

Since performance of all three algorithms are similar, other metrics will be used to determine which algorithm to use in practice. Looking at the training times, the Support Vector Machine and Neural Network took 4 times as long as Random Forests with Boosting to train. From the confusion matrices in Figure 11, it appears that Random Forests with Boosting almost always predicts that the song should be unsaved and Support Vector Machines almost always predicts that the song should be saved. The Neural Network however, predicts that about 25% of the songs should be saved. If one of these algorithms was to be used in practice, the Neural Network seems to be the best because although its performance was similar to the other algorithms and its training time was longer, it filtered out some songs rather than Random Forests with Boosting which filtered out all songs and Support Vector Machines which filtered out no songs. Also of note is that since the distribution of classes is about 40% saved and 60% unsaved, so the Neural Network only slightly outperformed a classifier that always predicts that the song should be saved. A potential cause of poor performance could be that the lyrics of the songs influence whether people save them much more than how the song sounds. Two songs could have the exact same features but the words that the artists say could be a major factor. Or maybe the sentimental meaning of a song plays a big role in whether people save it or not. Or maybe the person’s mood at the time that they saved it. In conclusion, of the algorithms tested the Neural Network was the best.

Acknowledgements

- https://scikit-learn.org/stable/modules/classes.html
- https://towardsdatascience.com/preprocessing-with-sklearn-a-complete-and-comprehensive-guide-670cb98fcfb9
- https://towardsdatascience.com/preprocessing-with-sklearn-a-complete-and-comprehensive-guide-670cb98fcfb9
- https://towardsdatascience.com/basic-ensemble-learning-random-forest-adaboost-gradient-boosting-step-by-step-explained-95d49d1e2725
- https://machinelearningmastery.com/tune-learning-rate-for-gradient-boosting-with-xgboost-in-python/
- https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.t.html
- https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.rv_continuous.interval.html
- https://stattrek.com/probability-distributions/t-distribution.aspx