Kellen O’Connor

Assignment #1 CS-7641

Supervised Learning

Summary:

Two datasets are studied and five supervised machine learning algorithms (Decision trees, k-nearest neighbors, Support Vector Machines, Boosting and Neural Networks) are used and optimized appropriately for the given dataset. The changes observed while optimizing are documented and analyzed. All work is completed in Python 3 using Scikit-learn, numpy, pandas and matplotlib.

Datasets:

Company Bankruptcy prediction. Gathered by the Taiwan Economic Journal from 1999 to 2009 this dataset includes ninety-four metrics of company value, financial health, business performance and includes a bankruptcy flag label for companies that have filed for bankruptcy. The goal is to be able to correctly identify behaviors and patterns that lead to bankruptcy using supervised machine learning techniques. The many variables available to us makes this an interesting task and provides a nuanced and potentially complex view of an organizations financial state. Variables include financial metrics like cash flow per share, total asset turnover, operating gross margin, and net value growth rates. This dataset is particularly interesting because of the number of observations provided to us (6,500+), the information available to us from the number of variables, and the potential uses of a classifier able to correctly identify financial states or behaviors associated with bankruptcy. Advanced knowledge of a particular company’s potential risk of bankruptcy could be valuable to many different people or organizations. All features within this dataset are numeric and positive which will make normalization easier, but the dataset is imbalanced.

HR Job Placement for Data Scientists. Data put together by an organization seeking to better invest the efforts of its human resources department. This dataset was gathered to help understand what aspects about an applicant will make them more likely to accept a job offer if given one. The goal is to be able to correctly classify that types of job applicants most likely to accept and begin employment based on a series of variables that include information about education background, previous employment history, and information about the applicants existing city or location. This dataset includes variables that are non-numeric categorical, and the set is imbalanced so steps must be taken to accommodate this.

<u>Pre-processing data:</u>

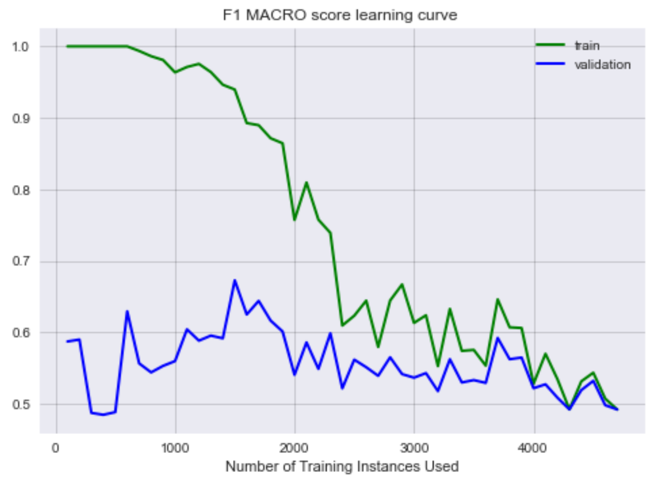
The bankruptcy dataset is luckily quite clean with all positive values that are either Boolean 0/1's or scaled from 0 to 1. Unfortunately the HR placement data contains string labels from some of it's variables that must be encoded to numeric values.

Scikit-learn's StratifiedKFold module is used heavily throughout this analysis to provide train / test sets for cross validation and to make the best use of data provided.

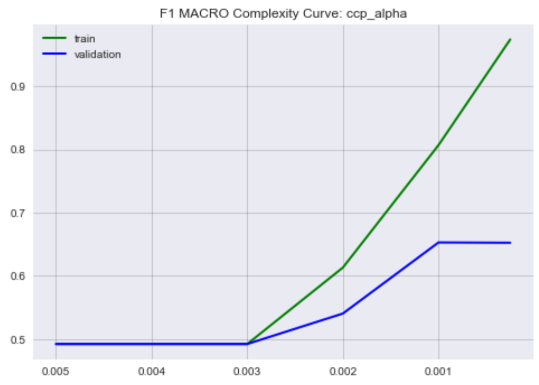
<u>Decision Trees:</u>

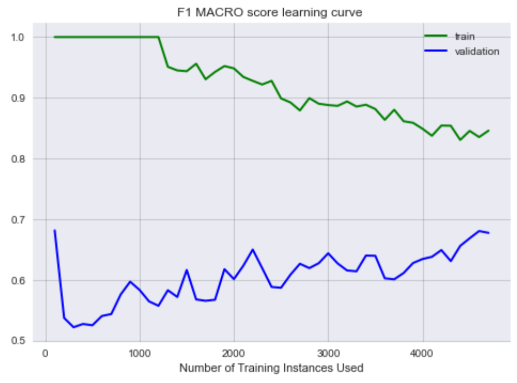
Bankruptcy Dataset:

Starting with the company bankruptcy data I implement the decision tree algorithm with the starting settings ccp_alpha=0.005 and criterion=entropy. Because of the imbalanced nature of the bankruptcy dataset, we use the f1_macro scoring method. Using f1_macro we provide additional importance to the instances flagged for bankruptcy since they occur far less frequently **(Lipton, et all).**



Using the starting point settings of ccp_alpha=0.005 and criterion entropy the model achieves a macro F1 score of 0.4918, accuracy of 0.9677, and runtime of 20.43 seconds. In order to better tune the algorithm I used scikit-learn's GridSearch. Using 0.001 incremented ccp_alpha values from 0.01 to 0.001 and both the entropy and gini criterion values GridSearch reports that a ccp_alpha value of 0.001 and gini criterion provide a better result in terms of macro F1 score.

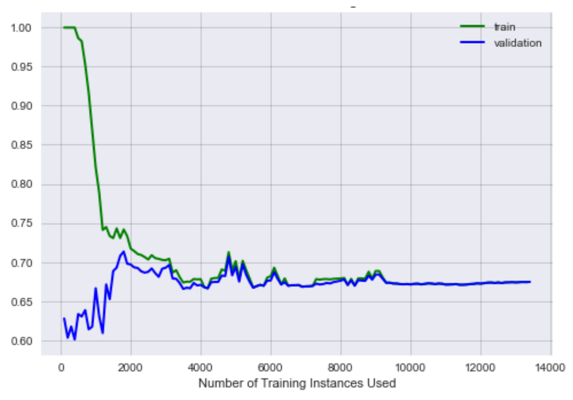




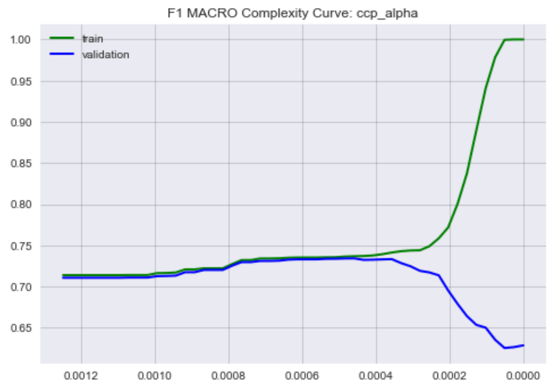
The new settings reported by GridSearch achieve a macro F1 score of 0.6738, accuracy of 0.9629, and runtime of 34.96 seconds.

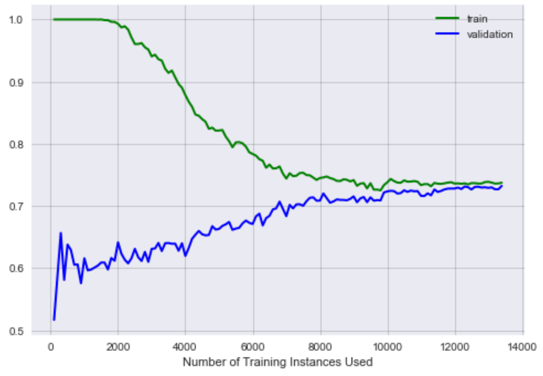
HR Dataset:

Starting again with ccp_alpha=0.005 and criterion=entropy I evaluated the decision tree classifier against the HR placement data.



With these initial settings the decision tree classifier achieves an F1 macro average score of 0.6816, accuracy of 0.7855, and completes in 31 seconds. I use GridSearch just like I had with the bankruptcy data to search for ideal values of ccp_alpha and criterion methods, below is a complexity curve to display the effects of changing alpha values on F1 score.



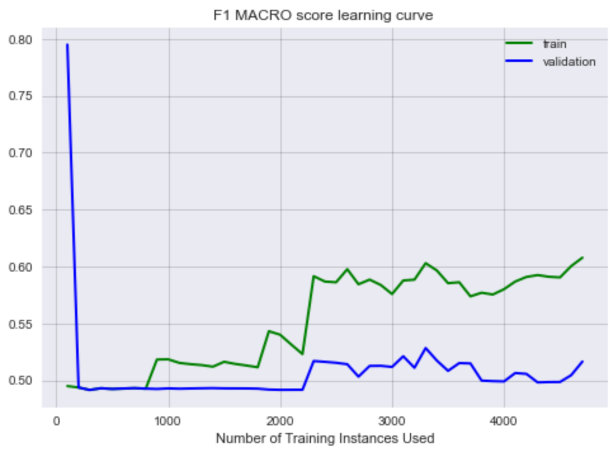


Our use of GridSearch shows that a small ccp_alpha value of about 0.0005 should improve model performance. When we run out classifier against the HR placement dataset with a ccp_alpha value of 0.0005 and gini criterion we achieve an F1 macro average score of 0.7386, accuracy of 0.7985, and a runtime of 19 seconds.

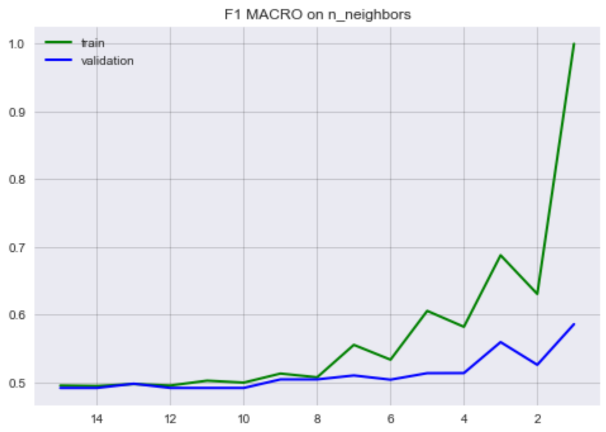
K-nearest neighbors:

Bankruptcy dataset:

Initially starting with default parameters of n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30, p=2, and with the minkowski distance metric we train the classifier to gauge it's initial performance.



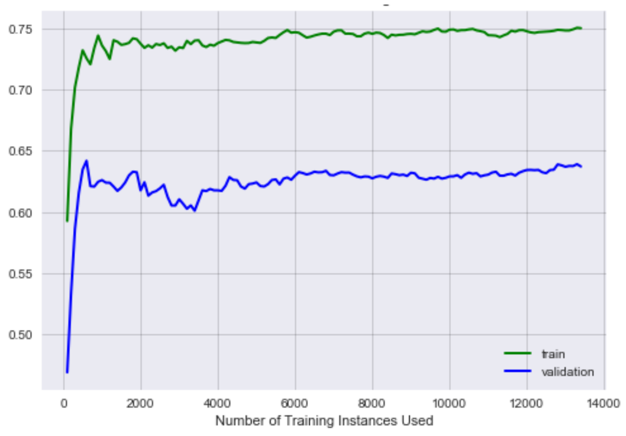
Displayed above the starting settings for the KNN model achieves a macro F1 score of 0.5349, accuracy of 0.9687 and completes in 170 seconds. In order to optimize our parameters GridSearch is used to evaluate different numbers of neighbors K and tells us that n_neighbors = 2 is optimal. With how imbalanced this dataset is this intuitively makes sense to me, if an observation flagged for bankruptcy is compared to dozens of neighbors it is significantly more likely for most of them to not belong to the bankrupt class.



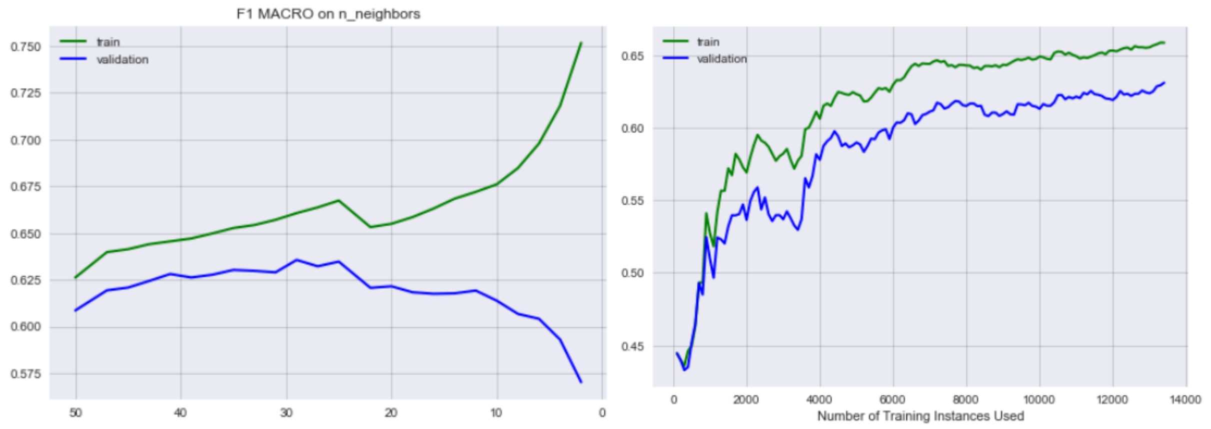
GridSearch is also used to evaluate the leaf_size parameter, but altering the leaf size appears to have virtually no effect on model performance. By lowering n_neighbors from 5 to 2 this implementation of KNN is able to achieve an F1 Macro score of 0.5972, accuracy of 0.9575 and completes in 134 seconds. The reduced runtime intuitively makes sense as the algorithm only needs to fetch information about 2 local neighbors and not 5 as is default in the algorithm.

HR Dataset:

Starting again with the default settings for our KNN model we run it initially against the HR placement data using n_neighbors = 5, weights=uniform, algorithm=auto, leaf_size=30, p=2, and with the minkowski distance metric.



On its initial run the model is able to achieve an F1 macro average score of 0.6376, an accuracy of 0.7498 and with a runtime of 326 seconds. We use GridSearch to better optimize our use of the n_neighbors parameter to better improve our implementation, below is a complexity curve of n_neighbor's impact on F1 macro score.



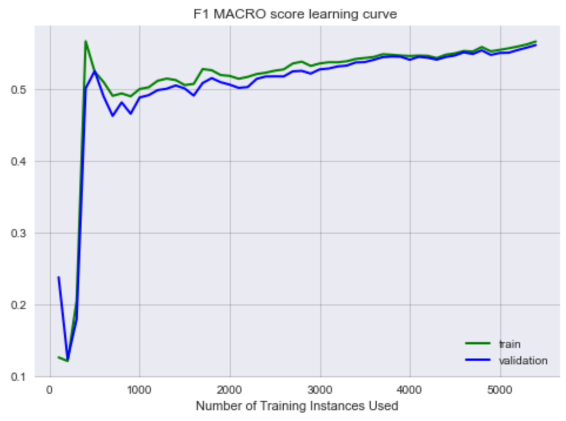
Seeing that n_neighbors is optimally set to somewhere around 29, we change the default n_neighbors = 5 to 29. Our slightly more optimized model achieves an F1 average score of

0.6355, accuracy of 0.7641 and completes in 454 seconds. The increased run time makes sense considering we are requiring the model consider more neighbors than before while classifying.

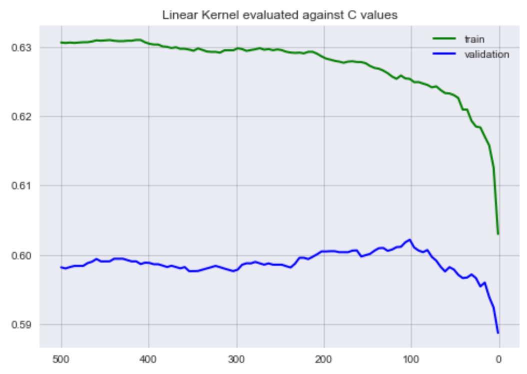
<u>Support-vector machines:</u>

Bankruptcy dataset:

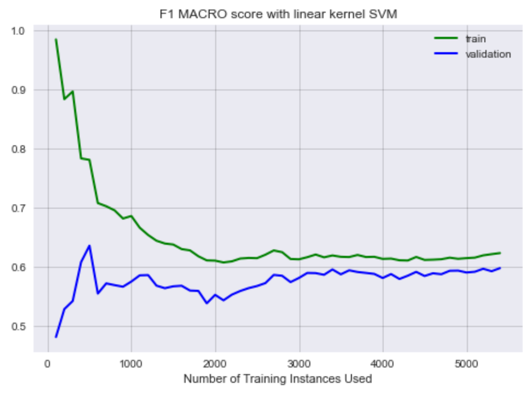
Starting at first with scikit-learns default settings for a support-vector machine model we implement the model to see how it initially preforms.



The initial use of the model achieves a macro F1 score of 0.5766, accuracy of 0.8284 and a runtime of 282 seconds. In order to optimize our implementation of the SVM algorithm we will be using GridSearch to evaluate several parameters, notably it's regularization parameter, kernel type, and gamma values where applicable based on kernel type. First we use gridsearch to evaluate different values of regularization parameter C across 3 different kernel types linear, sigmoid, and RBF.



The best results from these evaluations return the linear kernel with a regularization parameter C at 101. We then evaluate the linear kernel with C set to 101 across varying levels of gamma but find minimal impact on model performance with varied gamma levels.



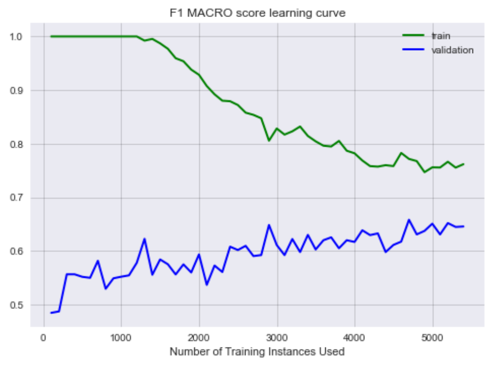
The final implementation of the support-vector machine algorithm achieves a F1 macro average score of 0.6001, accuracy of 0.8592 and a runtime of 166 seconds. This makes sense as the default implementation of the SVM algorithm used the radial basis function kernel which has a complexity that grows with the size of the training set whereas the linear SVM kernel is parametric. The use of a simpler model in place of RBF could explain the significantly shorter runtime.

HR Dataset:

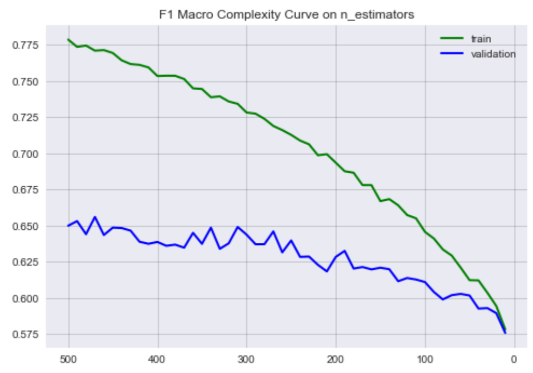
<u>Boosting:</u>

Bankruptcy dataset:

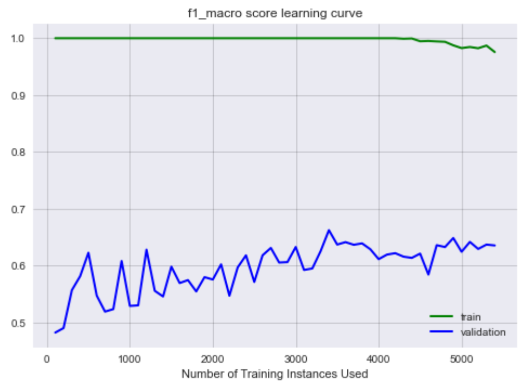
Initially I implement scikit-learn's AdaBoost algorithm with the default settings of n_estimators=50, learning_rate=1, algorithm=SAMME.R and a random state.



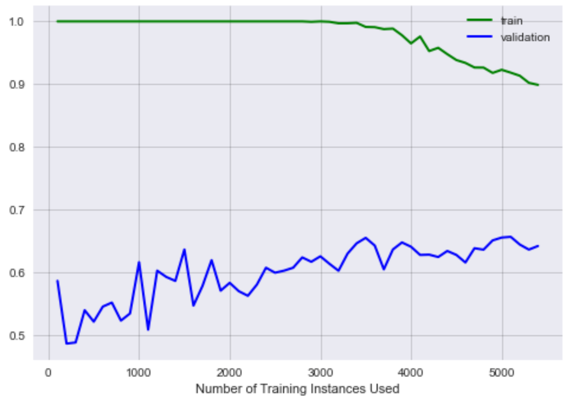
The default implementation of AdaBoost achieves an F1 macro average score of 0.6663, accuracy of 0.9692, and has a run time of 225 seconds. As with the other algorithms I attempted to use GridSearch to better optimize beyond the default values but was unable to drastically improve model performance. In fact, deviating from the default values often led to drastically diminished performance or took an unrealistic amount of compute time. (come back). This complexity graph provides an overview of the effect on F1 macro score across many values of n_estimators.



We can see that performance generally increases with n_estimators, but starts to provide diminishing improvements after 300 estimators. Ultimately my tuned algorithm wound up underperforming against the default settings of scikit-learn's AdaBoosting algorithm.



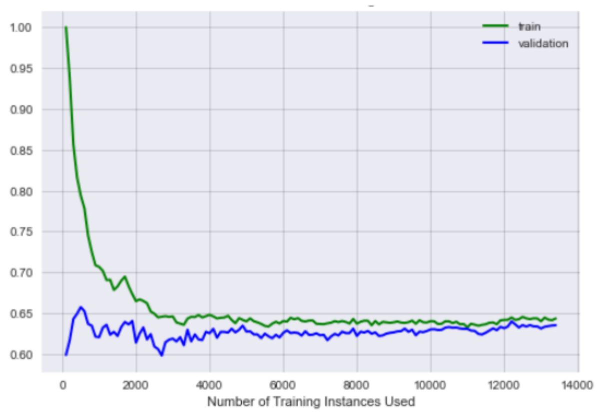
Using 60 decision tree estimators utilizing gini criterion, a max depth of 2 and with balanced class weights my implemented AdaBoosting model achieved a macro average F1 score of 0.66508.



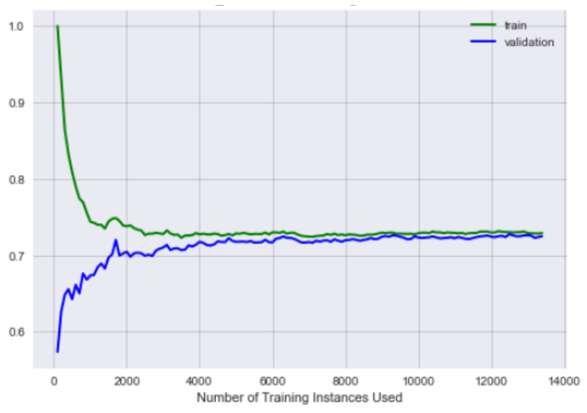
Another interesting observation noted while trying to optimize the default AdaBoost algorithm was that in addition to diminished model performance, my attempted optimization of the algorithm with 60 estimators took significantly more computing time, at 581 seconds versus the 225 seconds of AdaBoost utilized with default settings.

HR Data:

For the HR placement data I implement the AdaBoost algorithm with the default settings of n_estimators=50, learning_rate=1, algorithm=SAMME.R and a random state.



In this default state the model achieves an F1 macro average score of 0.6441, an accuracy of 0.7813 and completes in 138 seconds. As done elsewhere I utilize GridSearch to hone in on ideal values for n_estimators, and max_depth. Using the values returned from GridSearch we recreate the model with n_estimators = 25, max_depth=1 and weight's balanced.

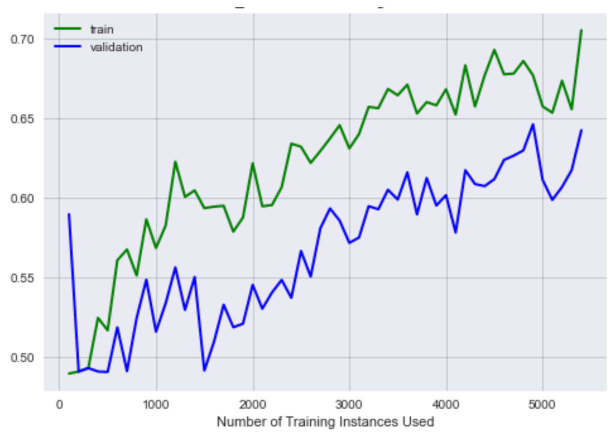


The changes to the model help marginally with our slightly optimized SVM model achieving a F1 macro weighted average score of 0.72527 and completing in 118 seconds. Reducing the number of estimators to 25 from 50 resulted in the slight improvements we see in runtime.

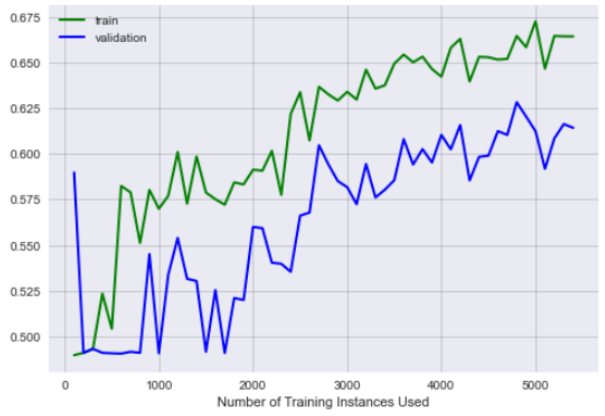
<u>Neural Networks:</u>

Bankruptcy dataset:

For our use of a nueral network I used scikit learn's multilayer perceptron classifier and implemented it with package default settings. The model's hidden layer size is set to 100, and alpha is set to 0.0001, it's performance can be seen below.



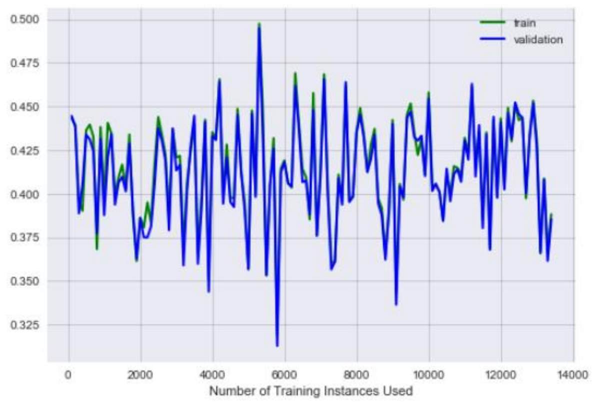
The default settings of the MLP classifier achieves an F1 macro average score of 0.6394, an accuracy of 0.97 and uses 620 seconds of computation time. I use GridSearch to try and find optimal hidden layer sizes and alpha values.



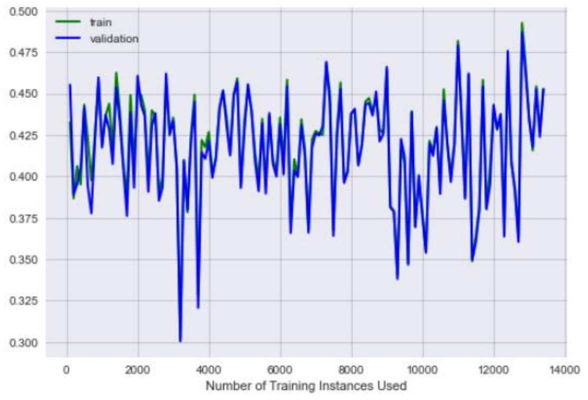
Based on the output from GridSearch we set our alpha value to 0.001 and hidden_layer_sizes to 75. This slightly optimizes our MLP model which achieves an F1 macro average score of 0.6541, accuracy of 0.97 and runs a little faster in 529 seconds. Reducing the number of hidden layers to 75 from it's default of 100 is what reduced computation time, which makes sense intuitively.

HR dataset:

For our implementation of the MLP model against the HR placement data we once again use scikit-learn's default settings. The performance of this model are captured below.



With default settings the model achieves an F1 macro average score of 0.45, an accuracy of 0.75 and runs in 414 seconds. As before we use GridSearch to try and find better values for the hidden_layer_sizes and alpha values. We test 50, 75, and 100 for our hidden layer size and 0.01, 0.001, and 0.0001 for our alpha values.



Using the results of our GridSearch we run a model with hidden_layer_size set to 50, and an alpha value of 0.001. The attempts at optimization don't provide too much improvement though as the new model achieves an F1 macro average score 0.4611 and takes 390 seconds to complete.

References:

Lipton, Elkan, and Naryanaswamy, "Optimal Thresholding of Classifiers to Maximize F1 Measure."

"How to Select Support Vector Machine Kernels," n.d (https://www.kdnuggets.com/2016/06/select-support-vector-machine-kernels.html)

"Decision Trees," 2020 (https://quantdare.com/decision-trees-gini-vs-entropy/)