Andrew Dodd
2/9/21
Machine Learning (CS 7641)

Assignment 1: Supervised Learning

1. Dataset 1 Analysis:

Dataset Overview - this dataset is the Pima Indians Diabetes Dataset [1]. The dataset has factors such as pregnancies, glucose level, blood pressure, skin thickness, insulin, BMI, etc. In total, it has 768 observations and 8 numeric columns that are normalized with SciKitLearn's (SKLearn) StandardScaler which subtracts the mean and divides by the standard deviation for each feature individually. The classification label is whether or not the individual has diabetes, either 1 for True or a 0 for False. It has approximately a 1:2 True:False ratio. It is mildly unbalanced but this imbalance should not be a serious detriment to most learning algorithms used here. This dataset seems interesting due to the relatively small size compared to Dataset 2, which will allow the researcher to perform quicker and perhaps deeper analysis with quicker ML model training. Furthermore, the class imbalance and lack of scaled variables could also have interesting effects.
Choice of metric - due to a class imbalance in our data, we will proceed with Receiver Operating Characteristic Area Under the Curve (henceforth just AUC). Other options we could have used are F1 score, PR-AUC, and even accuracy may have worked (with the caveat that ~⅔ or 67% would be our baseline accuracy performance).
Train/Val/Test Split: The train/test split is 80/20. However, the training set was split up in different ways depending on the choice of ML algorithm. For hyperparameter tuning, cross validation is performed.

1.1 Decision Trees (with some form of pruning)

We can create some interesting trees using SKLearn [2]. Specifically, we will use the GINI index to greedily decide on the splits. SKLearn has implemented many different pruning methods in their decision trees, but focusing on max leaf nodes, we can compare the effects of a small number vs. a much larger number in Figure 1. It can be seen that with the pruning effect with a small number of leaf nodes leads to a much smaller tree. Also of note is that if one wanted to understand the effect of the variables better we could look at the splits displayed. We find that the most important split is glucose level being less than 127.5 while the second most important split in predicting diabetes is BMI (this particular analysis was performed before normalization to improve understandability).

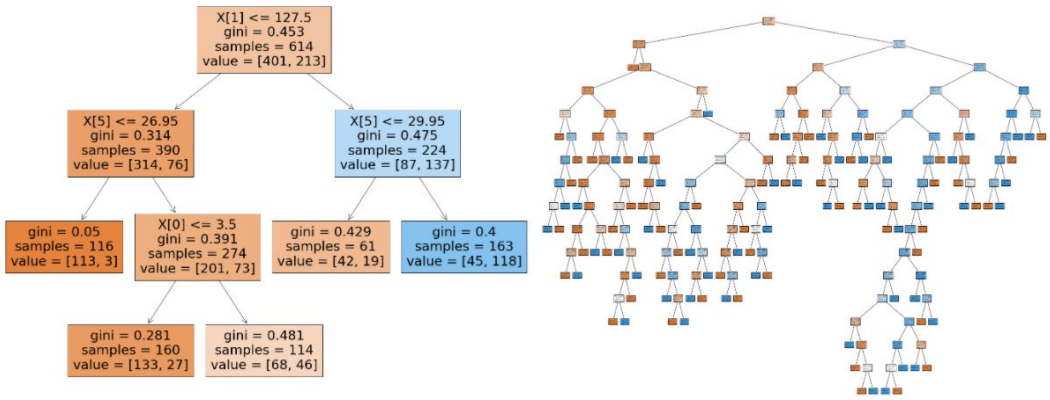

Figure 1: Some decision trees with different numbers of terminal nodes, 5 (left) and 100 (right).

Looking specifically at a few different methods for pruning, we can expect that with absolutely no pruning validation AUC may be poor (due to overfitting, this is leading to high variance of the model). And on the other

side of the spectrum we can also expect validation performance to be poor when we prune aggressively (few leaf nodes, high cost complexity). This is because the model will be forced to be overly simplistic, leading to high bias. Figure 1's left tree is a great demonstration of what may be an overly simplistic tree while Figure 1's right tree could be an overfit model (though it is hard to tell without checking validation AUC). In Figure 2, we explore the effects of varying cost complexity pruning (ccp_alpha in SKLearn) and the depth of the tree. Interestingly, the curves are not nearly as smooth as one might expect. There is a massive amount of variability in both. It does make sense that with a large amount of pruning we converge to 0.5 AUC on the Figure 2's left graph. What is most likely happening is that the tree becomes a single node with zero differentiation - and so always predicts the same output class. The relatively small size of the dataset is another factor in the jagged look to both curves. Without many training samples, small fluctuations in hyperparameters could have large effects on the data.

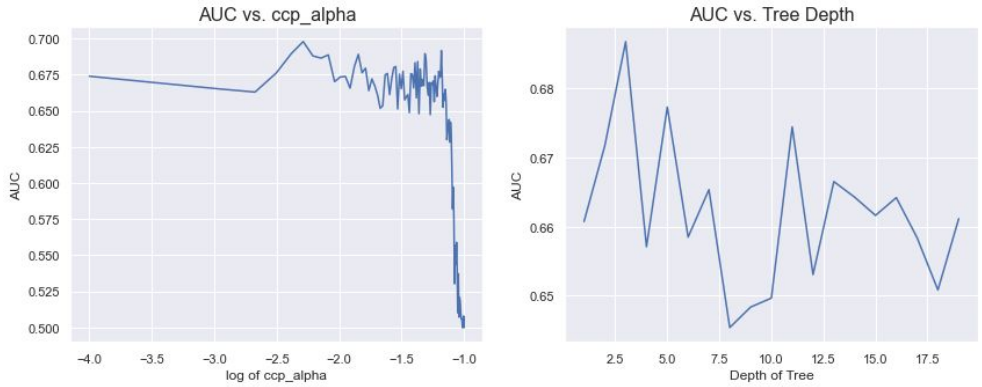

Figure 2: Validation AUC as a function of different pruning parameters - cost complexity pruning and tree depth.

Lastly for decision trees, we can visualize performance as it scales with train set size (Figure 3). Unsurprisingly, AUC increases very fast at first with small amounts of data, but the benefit slows down as more data is added. If we transform the X-axis to the log base 2 of the dataset size, then it looks slightly closer to linear, but still not quite so. Train AUC starts out high with small dataset size because a tree with a max depth of 5 can perfectly fit a small number of training samples.

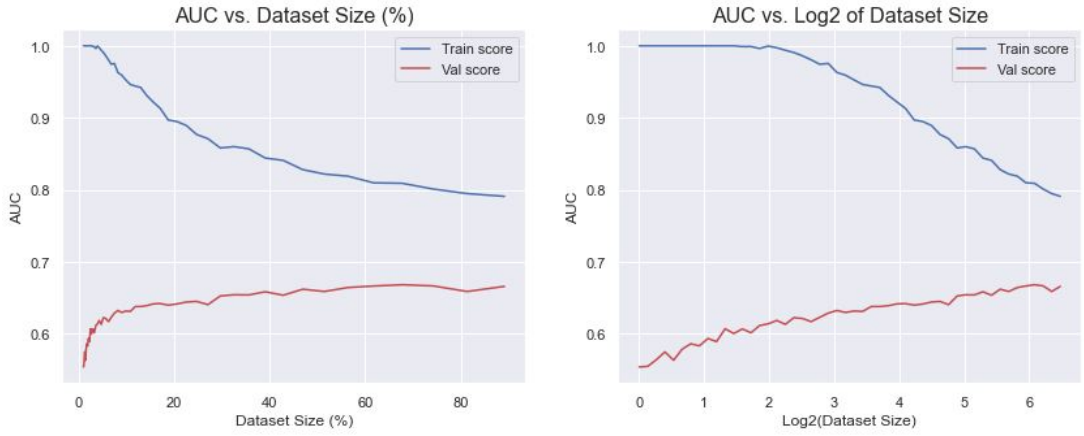

Figure 3: Effect of dataset size on AUC for a decision tree. Note that to create smoother curves, a 5 sample moving average was used. Note - dataset size is bounded to ~85% because we need to keep enough data to test on.

1.2 Neural Networks

The neural network (NN) tuned here was kept extremely simple and consists of only one hidden layer and one output layer with a ReLU nonlinearity and a dropout layer (probability of 0.2 for dropout) after the hidden layer. The framework used to train was PyTorch, on the CPU. The loss function used was binary cross entropy, and the

Adam optimizer was used. Batch size was 64 and this was run for up to 30 epochs and early stopping after 3 epochs (for faster results and to prevent overfitting/high variance). In addition, due to the mild imbalance of the data, the positive examples were overfitting to get close to a 1:1 ratio during training, but not testing. In cases of high data imbalance, oversampling the minority classes can improve the decision boundary learned. Some classic examples might be finding cancer or anomaly detection - these classes occur infrequently, and without rebalancing classes the loss function can perform relatively well by always predicting 0. Here, perhaps rebalancing is unnecessary but still this cannot hurt. Another approach to this potential problem is increasing the weights of the positive examples in the loss function.

Three hyperparameters were tuned for this NN - the number of hidden layers, the hidden layer size, and the learning rate. It was relatively quickly found that any more than 1 hidden layers would not train quickly and did not increase performance, so 1 hidden layer was chosen for further experiments. In Figure 4, the optimal hidden layer output size was found to be in the range of 48 and 2000, though the exact number seems to matter only a little (with 1280 being the best). It is likely that hidden layer output sizes of 10 and 20 lacked sufficient model complexity to fully model the function while some larger hidden layer sizes were overly complex and began to slightly overfit training data. Furthermore, an optimal learning rate was found to be between 1e-4 and 1e-3, with slower learning rates converging too slowly and higher ones taking too large steps - leading to poor convergence.

To better understand the effect of overfitting, AUC vs. training epoch was visualized in Figure 5 for a few different hidden layer sizes. This supports the efficacy of early stopping. Finally, we analyze the effect of dataset size on NN performance in Figure 6, finding that very small amounts of training data lead to models that are significantly lower performing (in the 50s for AUC).

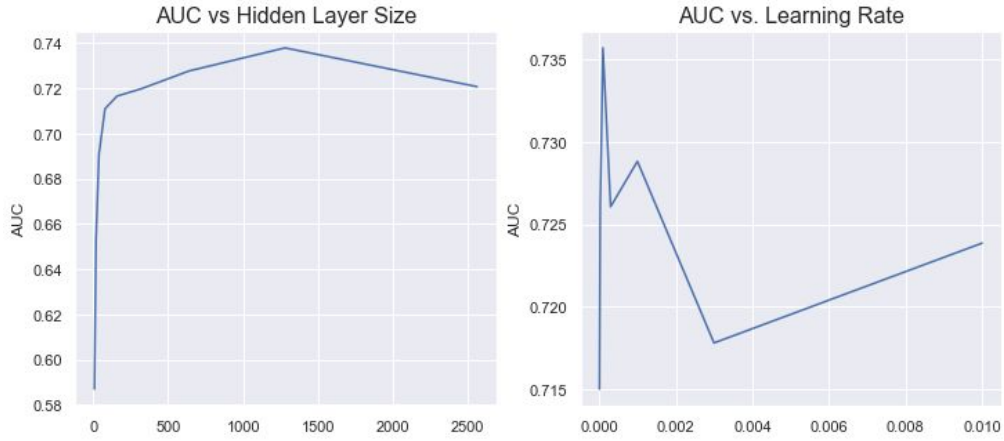

Figure 4: Tuning hyperparameters for a small NN, varying hidden layer size (left), and learning rate (right). Validation AUC is used.

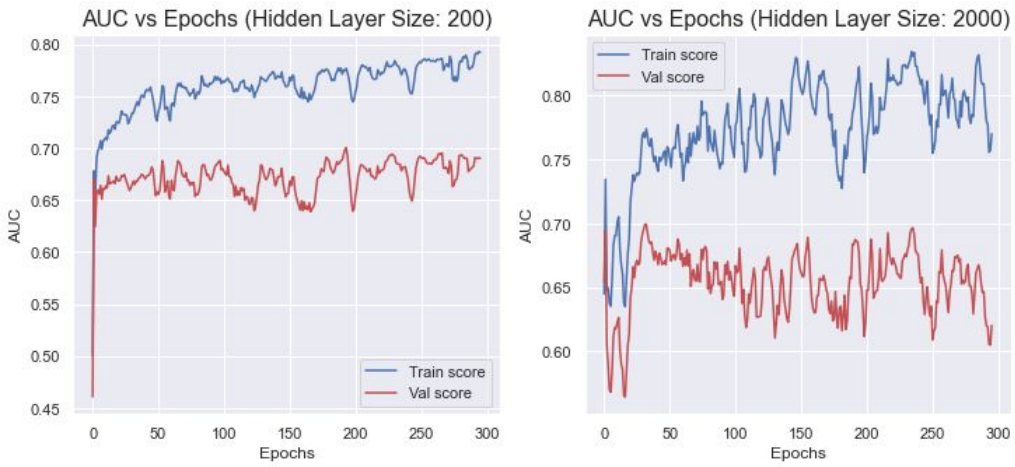

Figure 5: Effect of training iterations on for a neural network with hidden layer size of 200 (left) and 2000 (right). We see that particularly there is some overfitting in the right graph where validation performance begins to drop after the first 50 epochs.

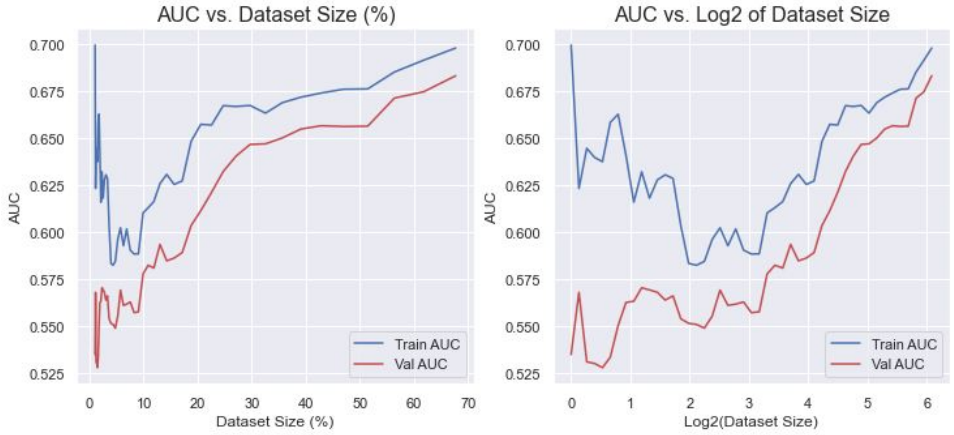

Figure 6: Analyzing neural network performance - smaller dataset leads to much worse performance on validation data - with large gap between training and validation AUC. This could be related to overfitting of the training data and such a small dataset compounds this problem. (Note - these tables were created with moving averages of 5 samples and 10 simulations were run for each dataset size.)

1.3 Boosting

For boosting, SKLearn's Gradient Boosting Classifier is used. Using the Adaboost algorithm, we can create a graph of performance vs. number of boosting iterations to develop a sense of what the best number of iterations is. From this result, we can see that a max depth of 2 with approximately 100 boosting iterations does best on the validation performance. Of course, both training curves in Figure 7 continue to improve (overfitting eventually), while the validation curves stop improving or even show signs of overfitting. We can also visualize the effect of dataset size on performance (Figure 8), which shows that performance begins to level out in the low 70s AUCs for this specific set of hyperparameters (100 boosting iterations, max depth of 2).

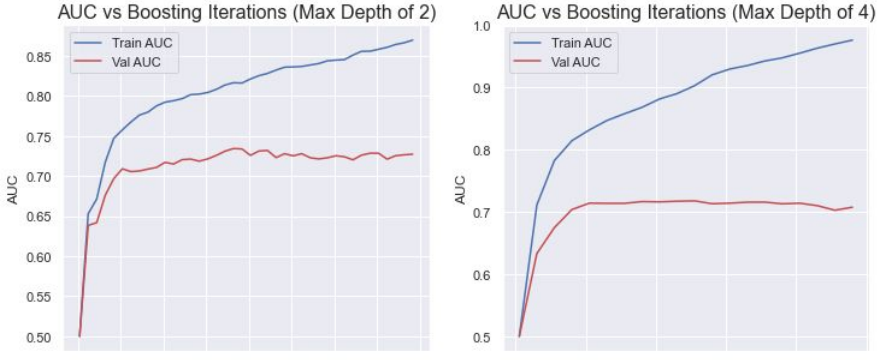

Figure 7: Effect of number of boosting iterations on AUC for different max depths - 2 (left) and 4 (right).

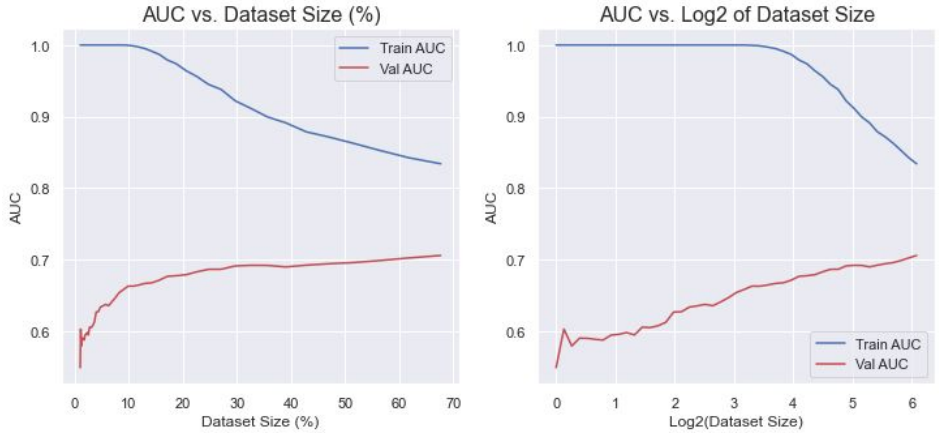

Figure 8: Effect of dataset size on AUC. Training AUC is perfect at first because the model is able to perfectly fit the small amount of data. We have seen this earlier with other models.

1.4 Support Vector Machines (SVM)

For SVMs we compare a few different kernels on our dataset, RBF, polynomial and sigmoid. Cross validation leads to the choice of the RBF kernel with a slightly higher validation performance prior to hyperparameter tuning. Now, grid searching over C and gamma - our hyperparameters for SVM we find that the optimal C is approximately 1e-1 and gamma is best around 1e-4 (Figure 9). We can also compare AUC as a function of dataset size using the grid search optimal hyperparameters (Figure 10).

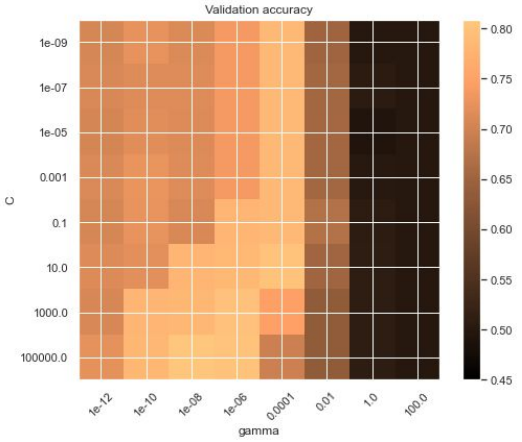

Figure 9: Result of grid search over gamma and C for SVM with RBF kernel (Note - grid search visualization code based on that from SKLearn).

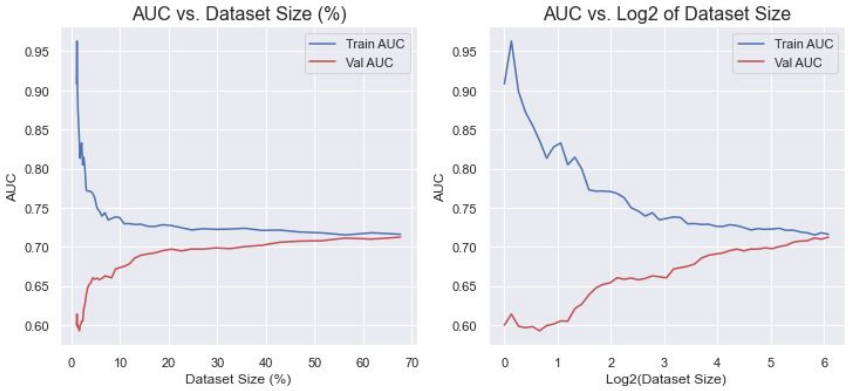

Figure 10: Effect of dataset size on performance. We see that dataset size of 20-30% is close to the full dataset performance.

1.5 K-Nearest Neighbors (KNN)

Using KNN we find using k-folds cross validation that the optimal values for the number of neighbors is approximately 40 and that the best distance metric is Manhattan distance (though they don't differ greatly), in Figure 11. We also look at the effect of dataset size on performance (Figure 12).

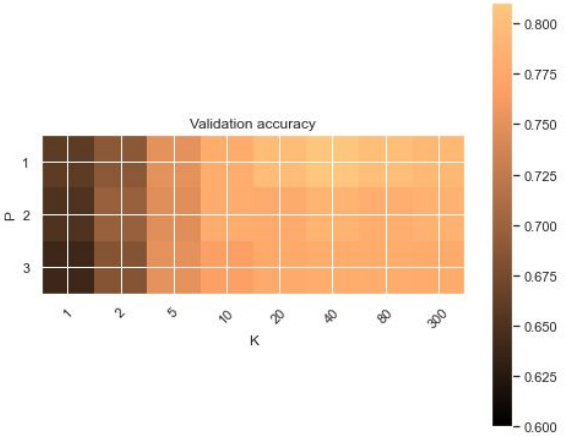

Figure 11: Validation accuracy as a function of power parameter for the Minkowski distance metric (P), and the number of neighbors (K).

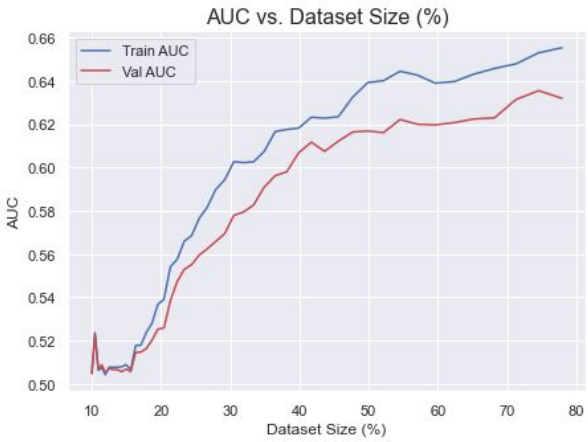

Figure 12: AUC vs Dataset size.

1. Dataset 1 - Summary

In Table 1 we find a measure of test performance using tuned hyperparameters for each of the 5 models. Note that the test data was not touched during hyperparameter tuning. Based on these results, it looks like the NN is the best model based on looking at test AUC. At first glance it looks like the Decision Tree may have overfit on the data but this seems unlikely given that the max allowable depth was set to 5 and this seemed to work relatively well on the entire train dataset - despite having a small model complexity. Therefore it seems more likely that the test set could have been simply harder to predict than the training set for the Decision Tree rather than an overfitting problem. However, the Boosting model may have slightly overfit. Based on the slopes of dataset size vs model AUC (Figure 6), it seems that with a larger dataset, the neural network might eventually become the best model, whereas boosting performance seems to have leveled out already (Figure 8). Train and inference times are all relatively short due to the relatively small dataset size and are therefore not considered strongly in this comparison.

Table 1: Train and Test AUC for different ML models. Note that the NN outperformed others.

Model	Train AUC (%)	Test AUC (%
Decision Tree	79.4	71.8
NN	81.6	85.7
Boosting	80.3	73.8
SVM	71.5	74.2
KNN	69.5	66.4

2. Dataset 2 Analysis:

Dataset Overview - this dataset is the Telco Customer Churn dataset from Kaggle [3], The dataset has features such as Gender, Senior Citizen, Partner, Partner, Phone service, amongst others. The target variable is whether or not the customer left the service (AKA - churning). Data preprocessing involved dummifying some of the categorical data, filling in a few NaNs, and converting columns to numeric, as well as using SKLearn's StandardScaler to normalize the data. The class imbalance here is approximately 1:5 (True:False). This dataset is interesting partly for its high imbalance which may require some creative solutions. After creating dummy variables, it also has many more features than dataset 1 which could lead to some interesting comparisons (in addition to having 5x more observations).

Choice of metric - We will use the ROC AUC again here because it tends to perform well for data with mild class imbalance.

Train/Test Split: The train/test split is 80/20. The training data will be further split for cross validation.

2.1 Decision Trees (with some form of pruning)

Just as before we can run SKLearn's DecisionTreeClassifier with the GINI impurity for splits and choose best splits. We can run the same types of hyperparameter searches - looking at the effects of cost complexity pruning and tree depth on AUC (Figure 13), and explore AUC as a function of dataset size (Figure 14). Based on these parameters we settle for a tree depth of 5 because it has the highest cross validation accuracy. We could have used cost complexity pruning instead as well - these are both pruning methods to avoid overfitting on the training data.

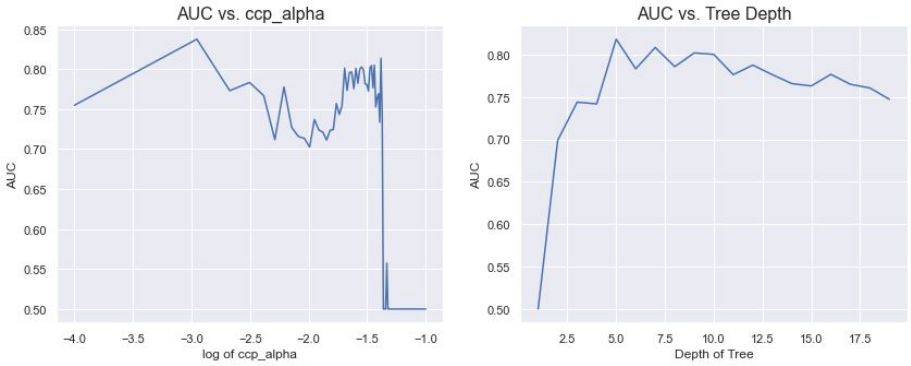

Figure 13: Validation AUC as a function of different pruning parameters - cost complexity pruning (Left) and tree depth (Right).

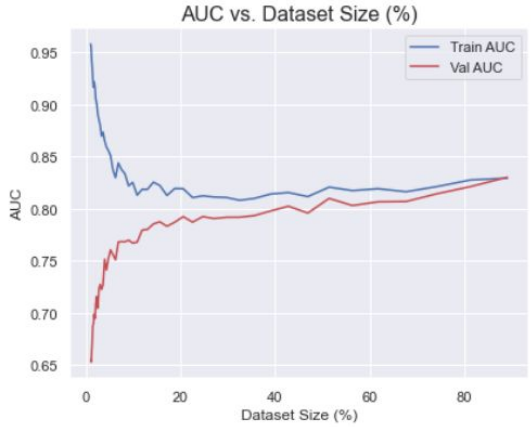

Figure 14: (Left) Effect of dataset size on AUC for a decision tree. Note that to create smoother curves, a 5 sample moving average was used.

2.2 Neural Networks (NN)

As before, we investigate a neural network for this dataset. Given that it is a considerably larger dataset with ~5,000 training observations, we can potentially expect relatively better performance than on Dataset 1. NNs tend to improve substantially with more data. Using the same general architecture as in Section 1.2 (NN for Dataset 1), we proceed with tuning our single hidden layer size and finding a good learning rate to achieve optimal

performance. Due to the amount of time taken to run training, only a small grid search was run with different hidden layer sizes and learning rates (Figure 15). Surprisingly, all models perform very consistently with k-folds cross validation, when the experiment was repeated and averaged a few times. Based on these results, a hidden layer size of 50 with a training rate of 0.003 was selected for further evaluation. Next we see how long training should proceed before subsequent training proves ineffective. To achieve this, we can train the models for a large number of epochs and see when validation AUC either flatlines or becomes worse (Figure 16). This seems to occur around epoch 100. Lastly, we analyze the effect on validation AUC vs dataset size used for training (Figure 17).

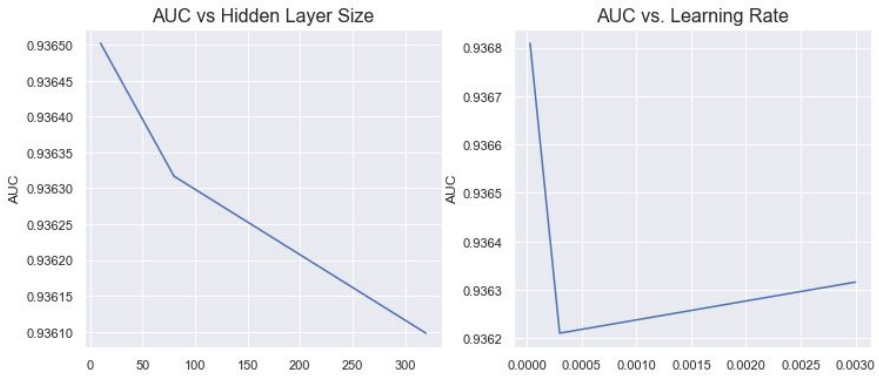

Figure 15: Tuning hyperparameters for a small NN, varying hidden layer size (left), and learning rate (right). Validation AUC is used. Note the lack of AUC fluctuation across hyperparameters.

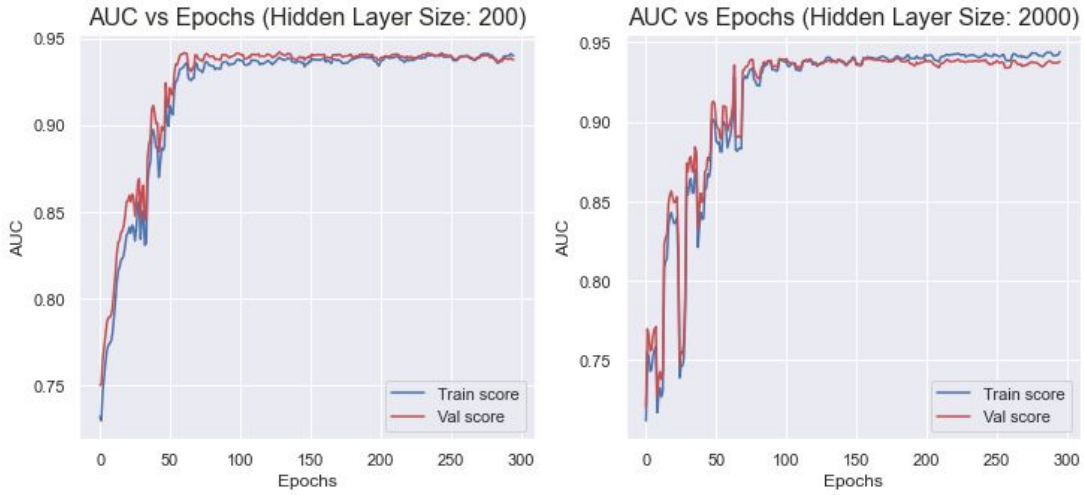

Figure 16: Effect of training iterations on for a neural network with hidden layer size of 200 (left) and 2000 (right). Interestingly in the case of the size 2000 hidden layer, we see minimal issues with overfitting, whereas in Dataset 1 we saw a noticeable dropoff in performance with too much training.

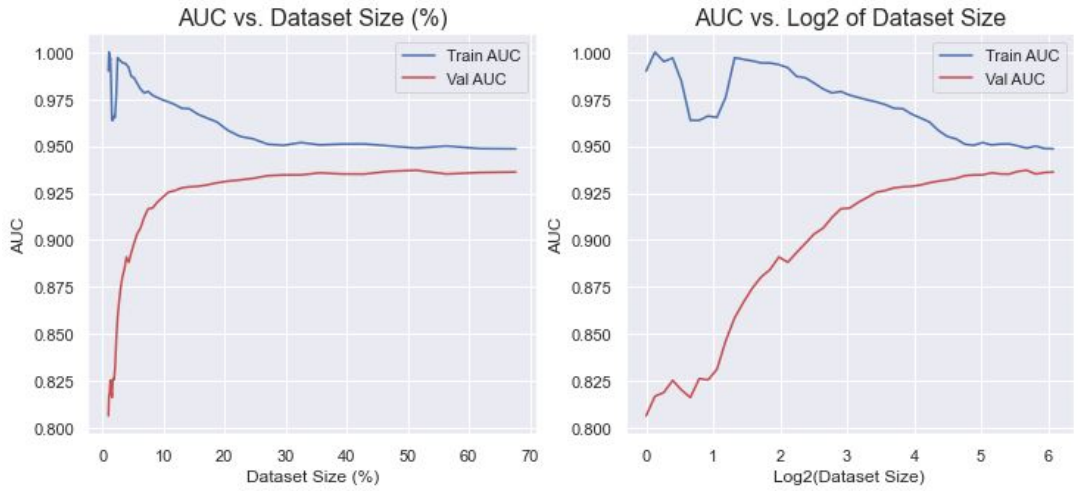

Figure 17: AUC vs. dataset size, here we see that validation AUC seems to asymptote to 93.5%, while training accuracy is markedly higher. This is likely related to some overfitting on the training dataset and lack of generalizability on unseen data. We also don't see the large improvement gains from more data that we saw on Dataset 1; this could be due to the 5x amount of data - so our model is no longer as "data hungry".

2.3 Boosting

For boosting, we use the same approach model as Section 1.3 and compare AUC across boosting iterations with different max depths as seen in Figure 18. We also look at the effect of dataset size on AUC in Figure 19, finding that the validation performance peaks around 80% while the training AUC slightly outperforms it at 81%, which is to be expected.

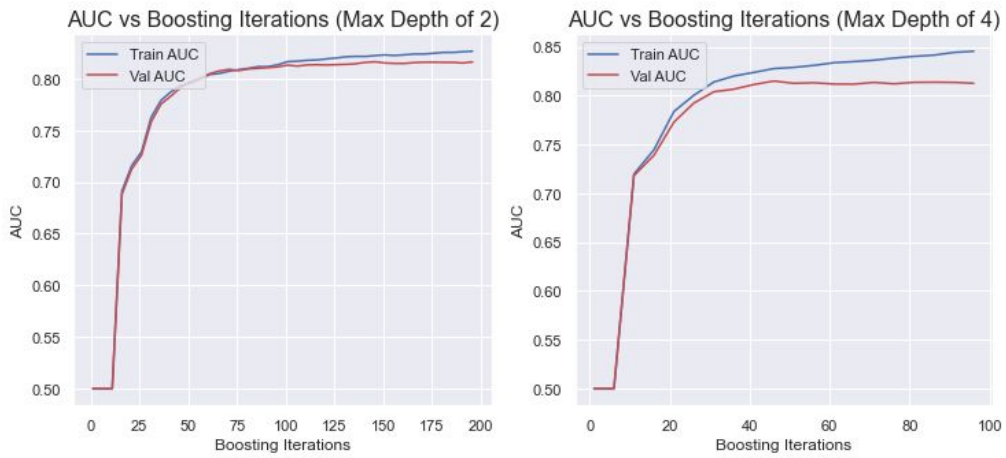

Figure 18: AUC vs. Boosting iterations - max depth of 2 and 4 (left and right respectively). It seems like the max depth of 4 has a slightly higher tendency to overfit after many boosting iterations, which makes sense.

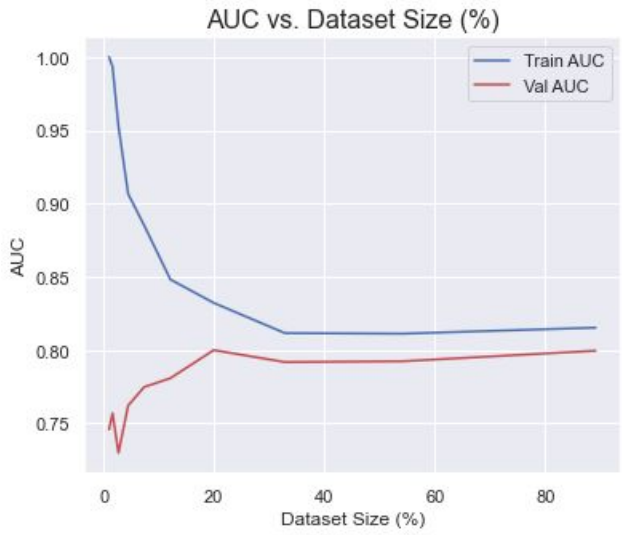

Figure 19: (Left) AUC vs. dataset size for a boosted decision tree.

2.4 Support Vector Machines (SVM)

As in 1.4 we follow a similar framework for analyzing the SVM for Dataset 2. There is a key difference here that seemed to lead to significant improvement. When the class weights of labels were both set to 1, the AUC was nearly 0.5 - as bad as random guessing. However, when the class weight for the positive label was changed to 5 (the same as the class ratio), the AUC rose to 0.6. Though this is still far from optimal, it is an improvement. Next, we grid search over gamma and C finding good hyperparameters at C of 1e5 and gamma of 1e-6 (Figure 20). Next we can visualize AUC vs dataset size, finding that validation AUC asymptotes around 84.5% while train AUC settles slightly higher (Figure 21).

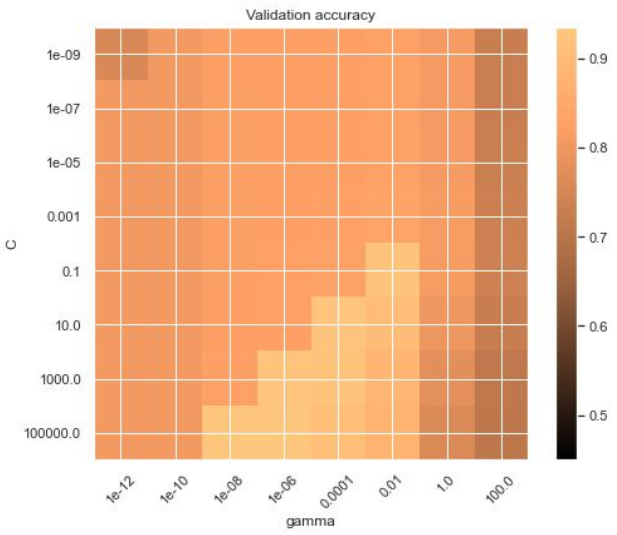

Figure 20: Grid search over C and gamma.

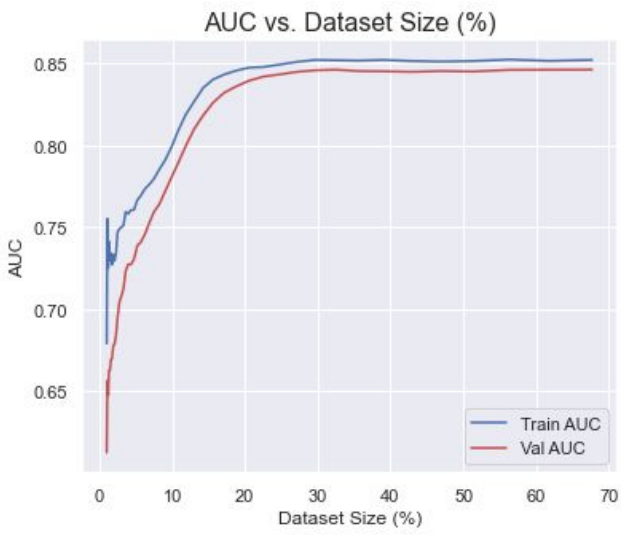

Figure 21: AUC versus dataset size.

2.5 K-Nearest-Neighbors (KNN)

With SKLearn, we can assess choice of distance metric and the number of neighbors to use with a grid search and can analyze the effect of dataset size as shown in Figures 23 and 24 respectively.

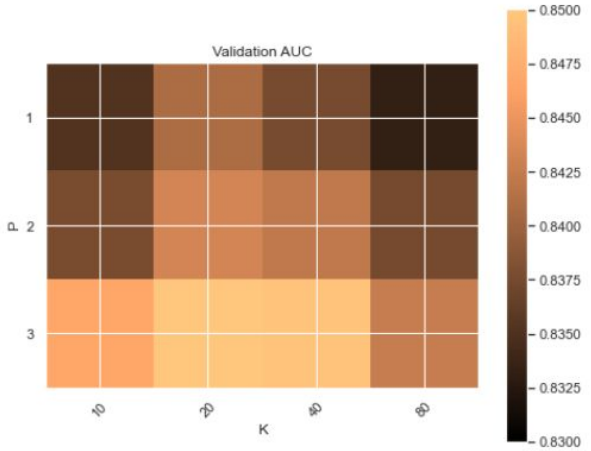

Figure 23: Grid search across distance metrics and number nearest neighbors. We find that the optimal set is 3, and 20 respectively.

Figure 24: AUC vs. dataset size.

2.6 Dataset 2 - Summary

In Table 2, we find the results of each ML model for Train AUC, Test AUC and Training Time. The NN outperformed all other models here, just as in Dataset 1. In theory, this could be related to a NN's ability to fit any function and ability to simulate theoretically infinite functions. Decision Trees and KNN's are both relatively simple functions and do not attempt to create continuous fits for the underlying hypothesis. Instead they create piecewise functions and Voronoi graphs respectively and these discontinuities in predictions could lead to some loss of AUC due to the jaggedness of the prediction landscape. Conversely, boosting, SVMs, and NNs are creating a less jagged, more complex approximation of the function. They depend on more data but possess more model complexity which is often good for fitting high dimensional data. Here, the best model would be the NN for it's high test AUC, though it does have significantly higher training time that may become less negligible with larger datasets.

Table 2: Train and Test AUC for different ML models. Note that the NN outperformed others.

Model	Train AUC (%)	Test AUC (%)	Train Time (s)
Decision Tree	83.0	82.9	0.018
NN	94.2	93.5	11
Boosting	81.7	80.8	0.8
SVM	78.1	79.1	1.46
KNN	74.1	74.2	0.081

3. Discussion

Comparing the results on Dataset 2 we see better performance for all classifiers in comparison to their performance on Dataset 1. This could be a for a few reasons:

- More data in Dataset 2 (~5x more observations, ~5x more features, and possibly more useful "signal"). With more data usually comes better performance. The models can learn to generalize better.
- An easier classification task. It is possible this is a simpler classification task. A good approach is sometimes comparing a model's performance with a human's to get a sense of baseline human performance and to what degree the ML problem is solvable. In this case, if a customer churn expert analyzed this data, one might reasonably expect that it would be easier to predict for them than an expert on Dataset 1.

Ultimately this question is just an interesting thought question and cannot be solved - they are simply different datasets. However, it looks like the neural network performed best on both datasets. This could be related to the specific choice of hyperparameters of all of the models (sub-optimal choices in other models perhaps), or random variance in the test set. However, in most cases the test AUC was fairly close to train AUC, leading one to believe this difference is beyond the realm of random variance, and the NN truly is the best here.

Other considerations that were not explored deeply during this analysis were:

- Inference time - this could be substantial for KNN in particular given larger dataset sizes. The decision tree is the fastest algorithm here for inference most likely and there are strong reasons to use this in practice when speed is required. To build onto this, NNs can be optimized for inference with filter pruning and different forms of model quantization and model distillation.
- Memory requirements for training and inference - these may impact the choice of what is the "best" model for a given application. If an SVM requires 500 Gb of RAM to train, this rules out most personal desktops from performing this task.
- Changing parameter distributions at inference time and the need to decide on a threshold for some models. Because some models predict a probability that can then be thresholded, these implicitly rely on assumptions about the distribution of the target labels. If the model assumes a dataset with more positive examples than negative, the threshold for positive predictions could be lower. If the test label distribution is much lower than the train label distribution, this could result in a high number of false negatives. Though this may not impact AUC, it may impact real world performance.
- Metrics besides AUC would be useful. Considering accuracy, PR curve, and different sensitivities and specificities at different thresholds could also prove useful in comparing these models.

References

1. Kaggle - Pima Indians Diabetes Database - https://www.kaggle.com/uciml/pima-indians-diabetes-database
2. SKLearn - https://scikit-learn.org/
3. Kaggle - Telco Customer Churn Dataset - https://www.kaggle.com/blastchar/telco-customer-churn