Supervised Learning
CS 7641

Abel Aguilar
aaguilar61@gatech.edu

1 INTRODUCTION TO DATA

1.1 Pima Indians Diabetes Database

For my first dataset I chose to use the Pima Indians Diabetes Database. There are a couple of characteristics to this data that make it an interesting dataset to use for machine learning. One of the reasons is because it is a binary classification dataset, either the patient has diabetes or the patient does not. This combined with the fact that it has a lot of features, 8, for the number of samples, 768, make it a great dataset to test each algorithm's ability to make correct classifications. The features themselves also add a degree of challenge. The dataset has a mixture of numerical and categorical features, it will be interesting to see how each algorithm handles these features. Another reason this dataset is interesting is because the dataset is unbalanced, there are far more examples of patients without diabetes then patients with diabetes. This adds an extra challenge to our machine learning algorithms as a simple accuracy rating will not be enough to determine how the algorithm performs.

1.2 Iris Species

The other dataset I chose to focus on for this assignment is the Iris Species dataset. This is a very popular dataset that presents its own set of interesting factors to test the machine learning algorithms. One aspect of this dataset that makes it interesting is that it is a multi classification dataset. Each sample can be 1 of 3 different classifications. This provides the algorithms with a different challenge than a binary classification. The next factor that makes this dataset interesting to work with is the fact that it is a relatively small dataset, only 150 samples and 4 features. This will be able to show how quickly an algorithm can learn with a limited number of examples. And lastly, because the features are all numerical and the output is a string, this adds a different challenge to the algorithm in terms of classification.

2 EXPERIMENTAL METHODOLOGY

In order to test the 5 algorithms with this assignment, Decision trees, Neural networks, Boosting, Support Vector Machines, and k-nearest neighbors I need to have an experimental methodology on how to approach this problem. In this report I will be focusing on 3 main methodologies, Hyperparameter Tuning, Model Evaluation, and Visualization.

For Hyperparameter Tuning I used cross validation with a wide range of hyperparameters to find which model works best to address each dataset. As each algorithm has many different hyperparameters that can be tuned and changed this process was the most challenging.

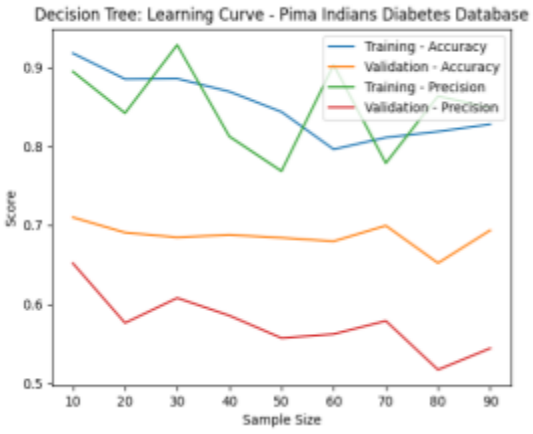
Here for model evaluation it is important to understand the dataset to make sure that your models are actually performing well as opposed to giving the appearance of success. This is the case especially with Pima Indians Diabetes Database because of its unbalanced nature. To address this I used other methods of testing for success, like precision and recall.

And lastly for visualization, for each different test run at a different hyper parameters or sample sizes I plotted these results against each other to truly be able to see what the algorithm is doing and spot any trends that might help me in my analysis.

3 ALGORITHM RESULTS

3.1 Decision Trees

3.1.1 Pima Indians Diabetes Database (PIDD)



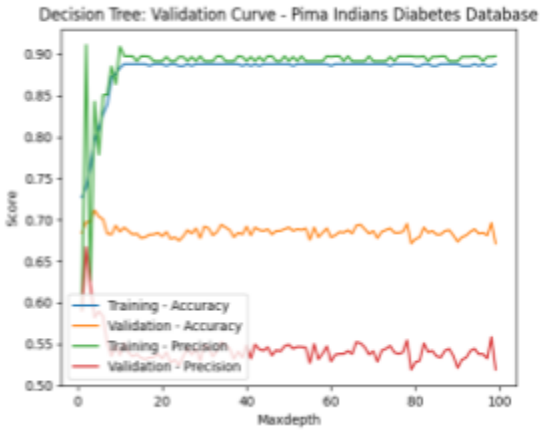


Figure 1- Decision Tree Learning and Validation Curve (PIDD)

For my decision tree I used 2 different curves to address the performance of my algorithm on the PIDD set. On the left in figure 1 you can see how the algorithm performs on different percentages of the training data passed in. This is to test how quickly it can learn and see if too much data ever leads to overfitting. First to look at accuracy. The original dataset I picked is rather small, and as you can see the graph shows that with small amounts of data it performs very well on its training set and not as well with the validation set. As you see the amount of data passed increase you see the performance decrease for a good period of time on both training and validation. This is most likely because as more data is passed in the examples are getting more complicated and the algorithm has to deal with examples it has not seen before. This trend begins to reverse towards the end of my graph. This suggests to me that it was moving towards a point where it had seen all new types of examples and it was starting to recognize them better and the accuracy would continue to increase for both in and out of sample. I suspect that this trend would continue with larger datasets. As for precision you can see that even when accuracy is doing better than 90% the precision stays low. This is most likely because the data is unbalanced in its early training, our algorithm is going to suggest nondiabetic for most samples because there aren't many diabetics. You can see this kind of behavior with precision staying around 60% for Validation and 70% for training. But again both trends start to reverse towards the end of the graph which suggests to me that I simply did not have enough data for the algorithm to get to the optimal results.

For my validation curve to choose to look at maxdepth as my hyperparameter. It looks to me that after a max depth of about 5 my algorithm stops improving. This is most likely because as the max depth increases this allows decision trees to get more complicated and it begins to overfit to the training data. Keeping max depth low seems to be going to generalize my tree to most data. You can see this in figure 1 with both accuracy and precision staying at the same performance for both in and out of sample data. This suggests to me that in order to not overfit we should keep max depth to about 5.

3.1.2 Iris Species (IS)

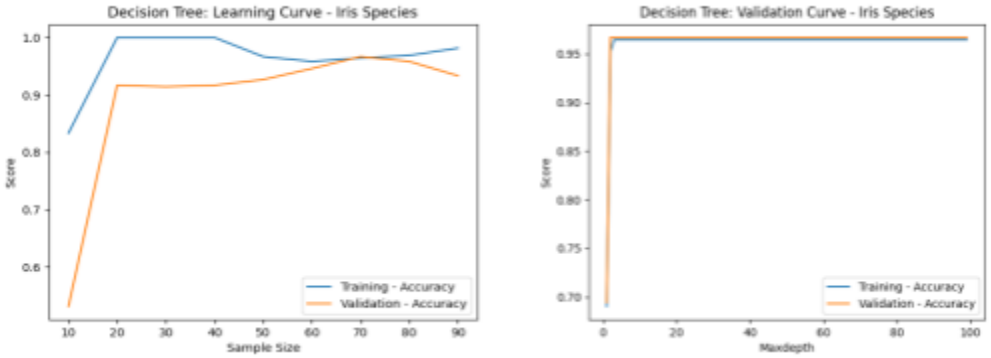


Figure 2- Decision Tree Learning and Validation Curve (IS)

For IS, you can see that as for both training and validation, as the number of examples goes up so does the accuracy of this classification. The graph does seem to suggest that over fitting could be taking place at around 80%. However since the data I used for this was rather small I feel like we would need to test this with larger numbers of data to see if that trend continues.

Similar to the previous example, max depth does not improve the performance after about 5. The reason is similar to the last example. The more max depth the more likely the training will over fit with the training set. This also suggests to me that the data is not all that complicated and that even though we are allowed to make the tree longer, our data does not warrant anything more complicated.

3.2 Neural Networks

3.2.1 Pima Indians Diabetes Database (PIDD)

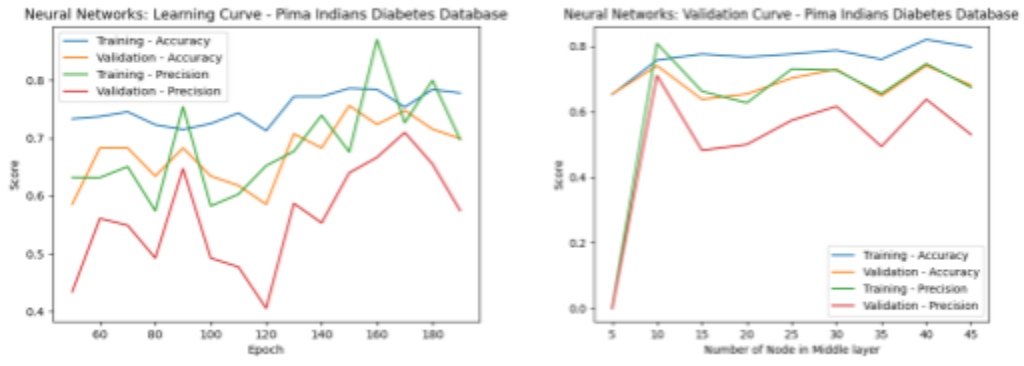


Figure 3- Neural Networks Learning and Validation Curve (PIDD)

From my analysis it seems to me that neural networks benefit from having a large amount of epochs. As you can see in the first graph, you can see many expected outcomes like precision tends to be lower than accuracy and training sets do better then validation sets, however one trend that seems to stay constant is that the more epochs that are passed in the better the algorithm does. Unfortunately, the data I chose is not long enough to test out my theory that this approach would also work with large complicated data sets.. However this does seem to be a characteristic of neural networks. The more epoch the more trips the algorithm gets to learn the data.

For my validation I decided to test out the hypermater of the width of the middle layer of my neural network. And, as I expected, when you do not have a lot of complexity in your algorithm, thus less nodes, the worse you are able to generalize to a topic. However as you can see the trend does look to be reserving at the end of the graph, this suggest s to me that we can not over complicate our network either as that may lead to over-fitting. Overall it looks to me like neural networks do the best job of fitting this data. Both precision and accuracy stay very high and one does not sacrifice the results of another, which shows me that neural networks are a good approach for unbalanced data.

3.2.2 Iris Species (IS)

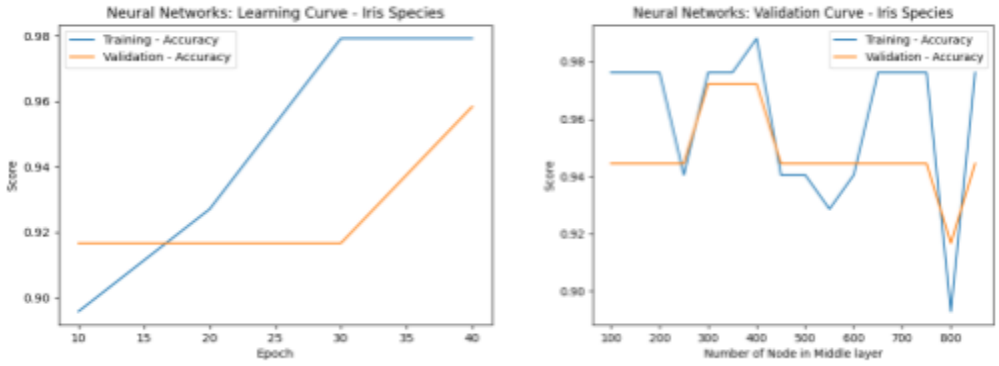


Figure 4- Neural Networks Learning and Validation Curve (IS)

For IS you can see that the more interaction that takes place, the better the algorithm does. The training examples around 20 epoch reach the maximum and

after that the validation begins to increase. I suspect that with more epoch the validation is going to continue to improve and most likely hit a maximum accuracy as well and stop improving like the training set did. This leads me to believe that in neural networks there is an optimal number of epochs for any given learning problem. After a certain amount the accuracy stops increasing and possibly even begins to overfit. It looks like I did not have enough epoch to reach this point.

For the second graph in figure 2. It looks like because the data I have is so small that adding more nodes to the middle layers really does not make much of a difference. The graph for both training and validation both only move between 92% and 94% which is not a large difference. This moves up and down with no clear direction. This again suggests that once the optimal number of nodes is found, adding more nodes only adds complexity and no accuracy.

3.3 Boosting

3.3.1 Pima Indians Diabetes Database (PIDD)

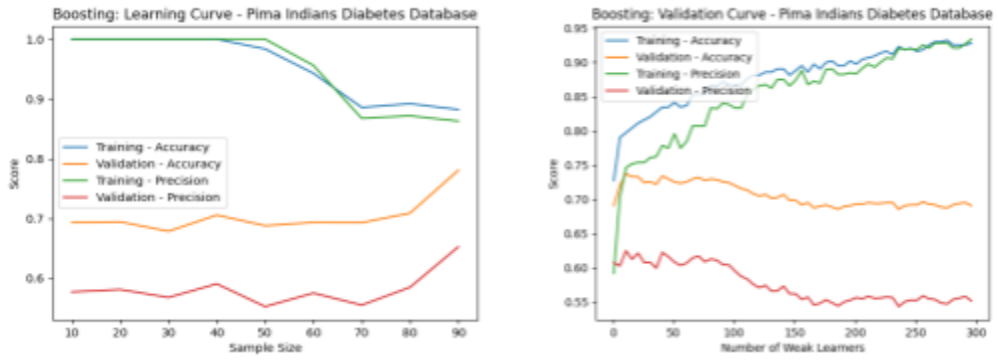


Figure 5- Boosting Learning and Validation Curve (PIDD)

Looking at the figure above it looks like with boosting the more examples that the algorithm has to work with the better it gets. The algorithm starts high for training in both accuracy and precision and as more examples are passed in we see that algorithm begins to generalize. Training accuracy and precision both decrease slightly and validation sets begin to perform much better. I would assume that as the data sets gets larger the better the performance will get, which matches my intuition for boosting

On the right in figure 5 we see that increasing the number of Weak Learners helps improve precision and accuracy for our training data but our validation sets do not seem to improve too much and maybe even fall a little. This could be because of limitations with the boosting method I choose. I chose adaboost which uses random forest as a weak learner. It could be that random forests are not the best suited for this data. While boosting does not tend to overfit on its own, if the underlying method can over fit it is likely that can affect boosting. Having too many weak learners could be what is affecting this algorithm from performing better.

3.3.2 Iris Species (IS)

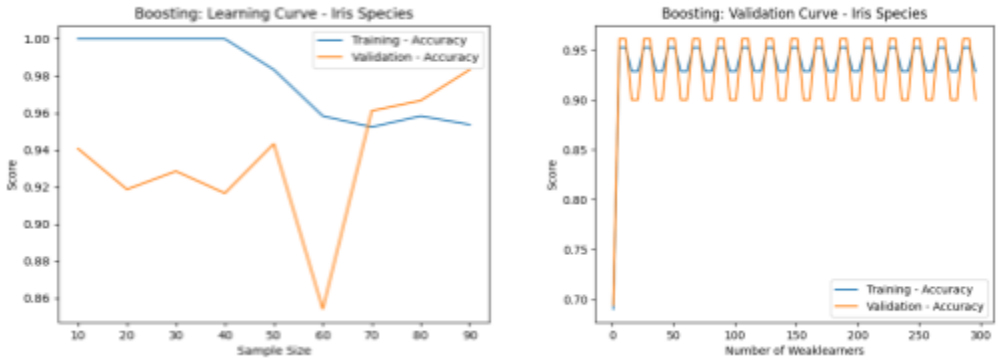


Figure 6- Boosting Learning and Validation Curve (IS)

Similar to the previous example, as the number of examples increases the better the algorithm gets at predicting. In fact boosting generalized so well that validation accuracy passed training accuracy in the final bit of the graph. This may not be the case all the time if more data is added but I suspect that the accuracy for both will continue to increase and then hover around 99%. This algorithm performed very well here.

For the second graph in figure 6 it looks like once an optimal number of weak learners are added the accuracy stops increasing and adding more only adds complexity. Because of the short nature of my data it reaches the optimal number of weak learners pretty early and just stays between 90% and 95%.

3.4 Support Vector Machines

3.4.1 Pima Indians Diabetes Database (PIDD)

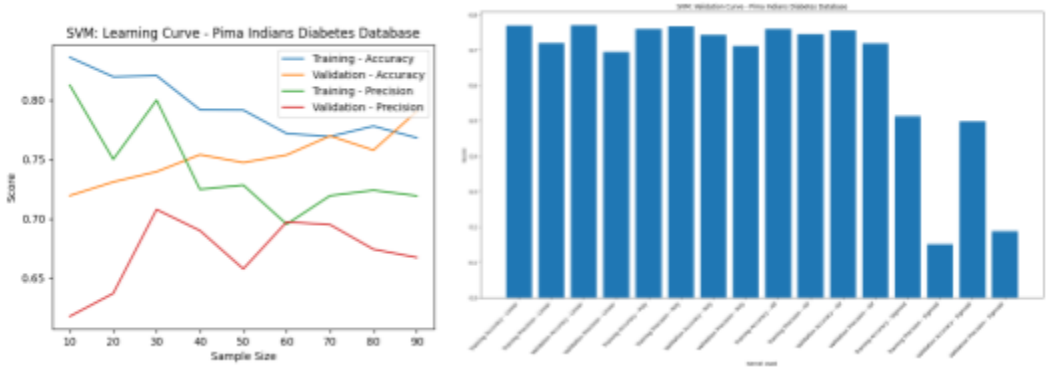


Figure 7- SVM Learning and Validation Curve (PIDD)

For SVM we can see that when the learner is only given a small number of samples it does not generalize well and training accuracy and precision both do well but for the validation, it does not as well. This trend moves in the direction of a more generalized algorithm the more samples that are passed in. We see that accuracy and precision both reach similar levels for training and validation the closer we get to 100% data. This suggests that SVM generalizes very well with many examples. In order to see if overfitting is a concern we would need to add more data to see when the trend goes.

For this example it seems that linear, ploy, and rbf all do relatively well in this classification example. Sigmoid is the only one that performs rather poorly. This could be that sigmoid introduces unnecessary complexity to a rather simple problem. Linear, ploy, and rbf all do well because of the linear nature of their form. Linear, which seems to have performed the best, was able to find a linear separator to classify binary data. This also keeps precision high as well leading to an over strong performance.

3.4.2 Iris Species (IS)

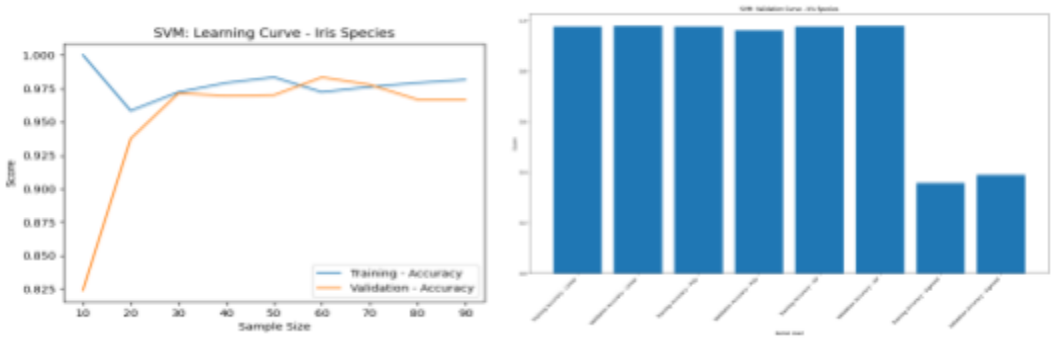


Figure 8- SVM Learning and Validation Curve (IS)

Very similar to the last example, it looks like SVM really only needed about 30% of the data to get a good generalization of the data. After that performance does not really increase and it stays very around 98% for both training and validation. This suggests that my data was simple enough to quickly learn and there is no need to spend computing power on more data.

As you can see in figure 8 the same results appeared with IS. Linear, ploy, and rbf all perform very well near 100%. Sigmoid is the only one that gives a rather poor performance. I believe this also has to do with the necessary levels of complexity that sigmoid adds.

3.5 K Nearest Neighbors

3.5.1 Pima Indians Diabetes Database (PIDD)

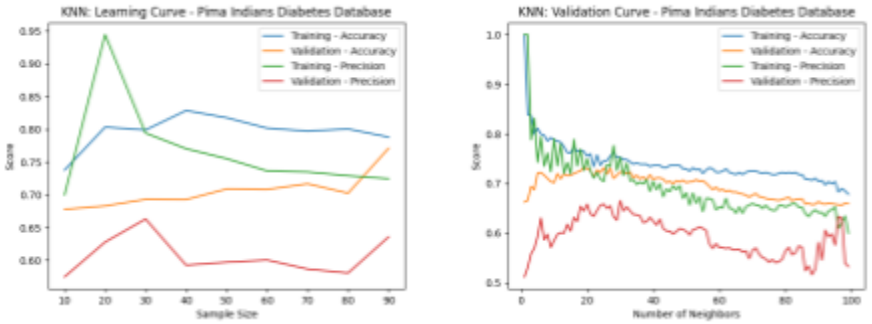


Figure 9- KNN Learning and Validation Curve (PIDD)

As you can see, adding more samples really does not make much of a difference in KNN after a certain point. It looks like according to figure 9 adding more samples kept accuracy and precision relativity stable. This makes sense because if most data points are already pretty close, which in this case they are, adding more neighbors might change the final result slightly but it won't add much new information since we are predicting based on neighbors.

Also with figure 9 on the right we see that increasing k (the number of neighbors we take into account) only makes the algorithm worse. This is because it generalizes too much. However not having enough k may lead to overfitting. In this example it looks like the optimal number of neighbors is around 20. This

gives a good estimate for validation, even with accuracy and precision, without overfitting to the training data.

3.5.2 Iris Species (IS)

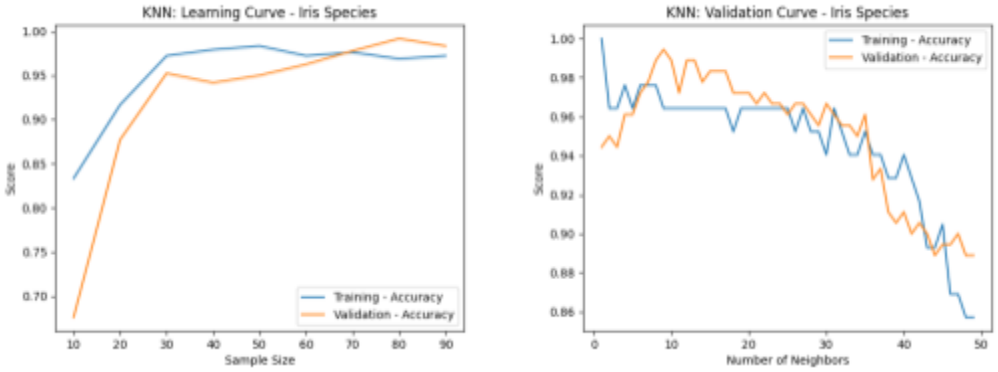


Figure 10- KNN Learning and Validation Curve (IS)

IS is a little different from PIDD because it has much less data to work with. Because of this as the total number of samples increases so does the performance of the algorithm. I do believe however that if the number of samples kept growing it would reach a point like in the previous example where it stops improving and hovers around a specific score.

In this dataset it makes sense that the higher K is the faster the performance is going to fall. This makes sense because the amount of data is so small that the more you include the more general the answer becomes and soon it will simply return an average of the entire data set. It seems optimal here is around 10 K.

4 COMPARE RESULTS

Best Results

	PIDD	**IS**
Decision trees	Accuracy:0.75064935064 Precision: 0.7033628368	Accuracy:0.96666666667
Neural networks	Accuracy:0.74025974025 Precision:0.74193548387	Accuracy:0.96666666667
Boosting	Accuracy:0.77532467532 Precision:0.70897401842	Accuracy:0.96666666667

Support Vector Machines	Accuracy:0.77922077922 Precision: 0.75407530454	Accuracy:0.98666666667
k-nearest neighbors	Accuracy:0.75324675324 Precision: 0.66975546356	Accuracy:0.97333333333

5 CONCLUSION

In conclusion, It looks like for this set of data Support Vector Machines perform the best, both in terms of accuracy and precision for both datasets. However the dataset set we have determines what is the best algorithm to use. Learning how to manage and treat datasets and learning what are the pros and cons of each algorithm is key to getting a good outcome with machine learning.

6 REFERENCES

1. Brownlee, Jason. “Your First Deep Learning Project in Python with Keras Step-by-Step.” MachineLearningMastery.Com, 16 Aug. 2022, machinelearningmastery.com/tutorial-first-neural-network-python-keras/ .
2. “How Does Scikit-Learn Compute the Precision Score Metric?” Saturn Cloud Blog, 9 Sept. 2023, saturncloud.io/blog/how-does-scikitlearn-compute-the-precision-score-metric/.
3. Keldenich, Tom. “Cross Validation - the Tutorial How to Use It - Sklearn.” Inside Machine Learning, 3 Jan. 2023, inside-machinelearning.com/en/cross-validation-tutorial/.
4. Learning, UCI Machine. “Iris Species.” Kaggle, 27 Sept. 2016, www.kaggle.com/datasets/uciml/iris.
5. Learning, UCI Machine. “Pima Indians Diabetes Database.” Kaggle, 6 Oct. 2016, www.kaggle.com/datasets/uciml/pima-indians-diabetes-database.
6. louisong97. “Neural Network Approach to Iris Dataset.” Kaggle, Kaggle, 27 Aug. 2017, www.kaggle.com/code/louisong97/neural-network-approach-to-iris-dataset.

7. Navlani, Avinash. “AdaBoost Classifier Algorithms Using Python Sklearn Tutorial.” DataCamp, DataCamp, 20 Nov. 2018, www.datacamp.com/tutorial/adaboost-classifier-python.
8. Navlani, Avinash. “Python Decision Tree Classification Tutorial: Scikit-Learn Decisiontreeclassifier.” DataCamp, DataCamp, 23 Feb. 2023, www.datacamp.com/tutorial/decision-tree-classification-python.
9. Navlani, Avinash. “Scikit-Learn SVM Tutorial with Python (Support Vector Machines).” DataCamp, DataCamp, 27 Dec. 2019, www.datacamp.com/tutorial/svm-classification-scikit-learn-python.
10. Shafi, Adam. “K-Nearest Neighbors (KNN) Classification with Scikit-Learn.” DataCamp, DataCamp, 20 Feb. 2023, www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn.
11. “Sklearn.Metrics.Precision_score¶.” Scikit, scikit-learn.org/0.15/modules/generated/sklearn.metrics.precision_score.html. Accessed 24 Sept. 2023.
12. “Sklearn.Model_selection.Cross_val_score.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html. Accessed 24 Sept. 2023.
13. “Sklearn.Svm.SVC.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. Accessed 24 Sept. 2023.