Joel Katz

9/24/18

Intro to Machine Learning

Byron Boots

Supervised Learning on Two Datasets: Leaf Classification and Bank Notes, Forged and Legitimate

Datasets:

The leaf dataset contains data about 40 different leaf specimens, with 14 identifying characteristics including eccentricity, average contrast, uniformity, etc., all measured with an image processor created by the donor of the data set. There are 340 training instances, with at least 6 examples from each class of leaf present. These characteristics will be used to identify the species, in this case represented by an enumerated list. This dataset presents a difficult problem that is ideally tackled with machine learning, because even though many of the leaves look very similar, through an analysis of many numerically defined characteristics, a machine learning algorithm can be trained to distinguish between these species. This is therefore a classification problem, and since there are enough data points and characteristics, it is a valid problem to study.

The banknote dataset contains genuine and forged banknotes that have been analyzed by taking a picture and using the Wavelet Transform tool to extract features from the images. The features from the Wavelet Transformed image include the variance, skewness, curtosis, and entropy. These features will be used to assess the validity of the banknote, or a Boolean output. This problem is an example of classification since the output is discrete, even with continuous inputs. Machine learning will work well on this data set because finding the relationships among these input values using human made algorithms would be incredibly challenging, but a machine learning algorithm will be able to train itself on the data and form an empirical algorithm that will be effective in practice. Also, there are 1372 training examples, so there are more than enough to properly train and test with.

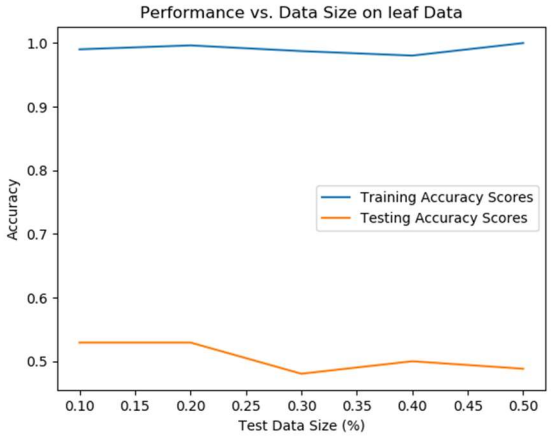
Methodology:

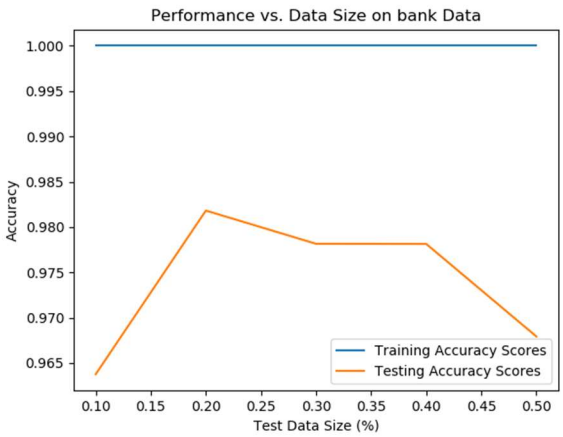
For each algorithm, I identified one or more hyper parameters that could affect the performance of the algorithm. Then, using a range of values for those parameters, I iteratively tested the algorithm over this range using the sklearn.model_selection.cross_val_score() function to analyze a model classifier. This function performs a k-fold cross-validation on the training set, which returns a list of values for the accuracy on the validation set for each test. The mean of these values is then used as an estimation for the accuracy of the model with the specified hyper parameters. Then, the maximum value for the accuracy is used to define the optimal hyper parameter values. This way, cross-validation is employed to prevent overfitting of the training data since the hyper parameters are assessed with data that the model was not trained on. As a note, the hyperparameters must be determined from only the training set, and not the testing set because if the testing set is used to pick hyperparameters, the testing data is being used to train the model and will ruin the model's legitimacy in generalizing to unclassified data. In the graphs shown, the cross-validation accuracy is the metric that is used to find the hyper parameters, and the training and test accuracy are just visualizations of how the specified hyper parameters affects

these values and nothing more, because the model is then trained using the test data with the specific hyper parameter values based on the cross-validation accuracy, not the testing or training accuracy.

Test Set Size:

The percent of data reserved as test data is often set at around 20 percent, but it is another parameter that can vary the performance of the algorithms. I used the Decision Tree algorithm to graph performance between 10 percent and 50 percent of the data reserved for testing.

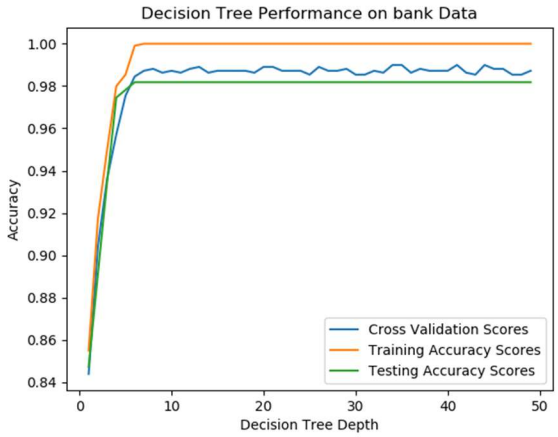




As shown by these graphs, the optimal percetnage of data to use as test data is 20 percent, especially for the bank note data set which shows significant improvement in performance at that amount.

Decision Tree:

A decision tree is a very simple model to understand as it can be followed simply based on characteristics to find the associated output. To create a decision tree model for the data, as well as all of the other algorithms, I used the scikit-learn python module, a very common library for the implementations of machine learning algorithms. The specific library I used for the decision tree is imported from sklearn.tree.DecisionTreeClassifier. This model is capable of training on and fitting a decision tree model to a set of inputs with corresponding outputs and a maximum depth of the tree. To perform pruning with this library, I iterated over the depth of the tree and took the average cross-validation error to discover the optimal depth for the tree on both data sets. This is called pre-pruning, and it prevents overfitting by limiting the depth of the tree. A tree without pruning can be too sensitive to errors in the data and therefore won't generalize well (overfitting). One can't use the test results to pick the best tree depth here because that would be using the test data to fit the model, which breaks the generalization ability of the model. Therefore, pre-pruning can be seen as simply the adjustment of a hyper parameter.



The cross-validation scores here show that as the decision tree depth increases, it reaches a point where the depth no longer increases performance. The leaf identification data and the bank note data are very different in that there are 40 different output leaf types and only two outcomes for whether a bank note is forged or not. For the decision tree, this appears to mean that the accuracy of the leaf data decision tree is much lower than the bank data tree.

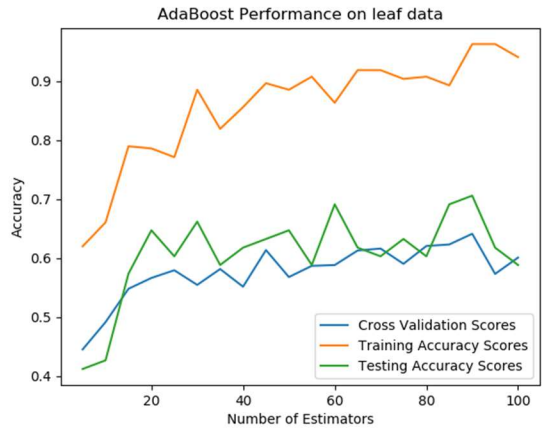
Because we are using pre-pruning and not using the test data to choose hyperparameters, the maximum accuracy for the cross-validation determined the optimal decision tree depth. Thusly, the optimal depth for the leaf data was determined to be 32, and the accuracy of the model on the training data was 1.0, and the accuracy on the test data was 0.588. For the bank data, the optimal tree depth was 34 which produced a model with 1.0 accuracy on training data and 0.982 accuracy on the test data.

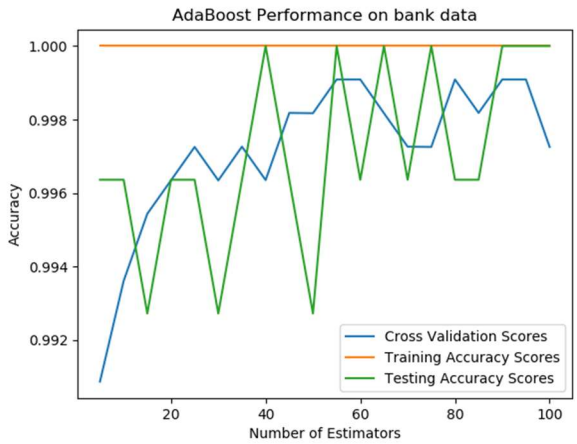
The speed of this algorithm was quite quick, taking only a few seconds to complete, especially since there was only one hyperparameter to test for, and therefore the iterative algorithm only needed to go through 50 times. I chose one through 50 for the decision tree depths for two reasons: the first is that I saw on the graphs that the data was interesting beginning from one, and that the data levelled off, and the second is that it was a quick operation, and since the data oscillated some from 10 to 50 I wanted to make sure I obtained the maximum cross-validation score value.

AdaBoost Decision Tree:

The AdaBoost algorithm is a kind of ensemble learning that uses multiple iterations of an estimator, then puts more weight on the misclassified points to recalculate an estimator, and finally uses an average of these estimators to calculate a more effective model. This algorithm can be used on many different base estimators, but for this project we are using the Decision Tree as the base estimator. The hyperparameters of this algorithm are both the maximum depth of the decision trees that are used as the base estimator, and the number of estimators calculated. To optimize both estimators, I calculated the accuracy of the cross-validation on the Cartesian product of tree depths between 1 and 4, and the estimators between 5 and 100 skipping every 5. However, it is difficult to graph the effect of two variables on the cost, so instead I used the optimal depth as calculated from before to graph the effect on the number of estimators on the final product. The depth of a tree used for AdaBoost can be assumed to be very small because the AdaBoost algorithm is well known to be most effective on smaller

and simpler estimators, and to use the variability in these estimators to produce the best result as an ensemble.





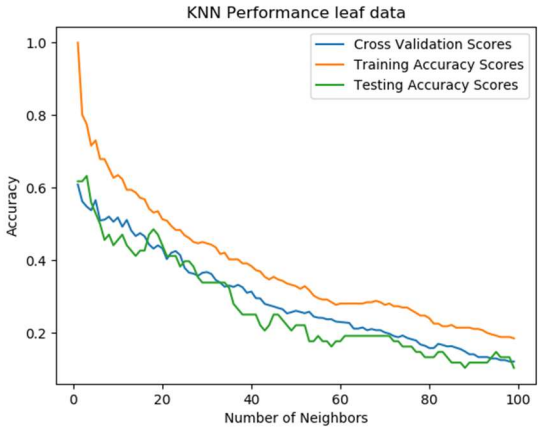
The cross-validation values for both graphs seem to show that more estimators are beneficial up until about 90 estimators, when there is a slight dip in accuracy. In fact, the optimal estimators for the leaf data was 95, and for the bank data was 40. I would have added more estimators to the test, but AdaBoost performed the slowest out of all the algorithms, and this would have taken an unfair amount of computation time as compared to the other algorithms and would have produced a skewed result in their final comparison. I believe that AdaBoost is usually intended to perform best on decision stumps, or single node decision trees, but the calculated optimal depth for the leaf and bank data were both four.

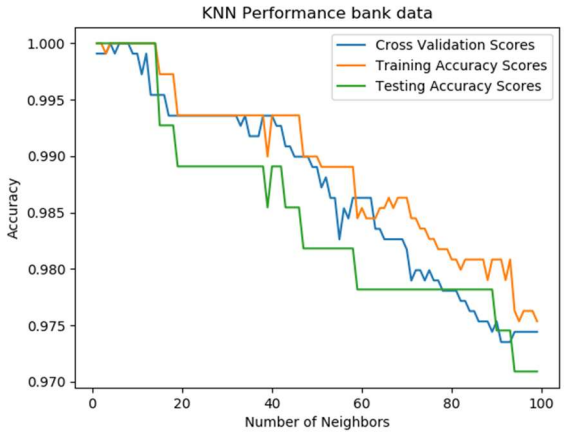
Using their respective optimal hyperparameters as calculated from the maximum cross-validation scores, I trained the model on the data and came up with the leaf data having a training accuracy of 0.93 and a test accuracy of 0.647, and the bank data having a training accuracy of 1 and a test accuracy of 1. This again shows that this model, and potentially many others, are much better at fitting the data when the output space is smaller. However, given the test accuracy is a perfect 1.0, AdaBoost seems to be very effective on the bank data set.

K Nearest Neighbors

The K Nearest Neighbors (kNN) algorithm is very easy to understand. It weights the k nearesst points based on distance, and marks that point according to the prevailing label. This model is generally highly subject to errors in the data, and the data can make the optimal k value vary widely. Therefore, the choice of K for each data set is the hyperparameter for this algorithm. The range I chose for this hyperparameter is between 1 and 100, because 1 is the minimum, and the algorithm does very poorly after k reaches 100.

These graphs (below) show a very different trend from the first two algorithms in that the accuracy seems to plummet almost immediately, and never levels off. The optimal value of k for the leaf data is 1, and for the bank data is 4. Since the leaf data has many different categories and sometimes little data within its own classification, it makes sense that neighboring points will quickly become different labels,





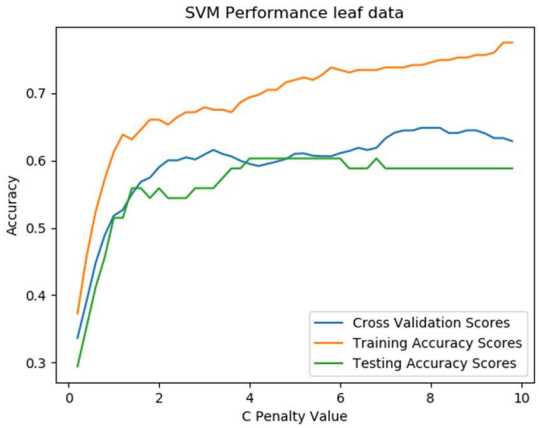
leading to an optimal number of neighbors of 1. The bank note data however has only two classifications, meaning that if the data is well clustered, the nearest neighbors are more likely to be the same classification, giving a larger k value of 4. For the leaf data, the training accuracy was 1, and the test accuracy was 0.609, and for the bank data, the training accuracy was 1 and the test accuracy was 1. Again, it seems clear that the leaf data is more difficult to fit, and that the bank data is perfectly classified.

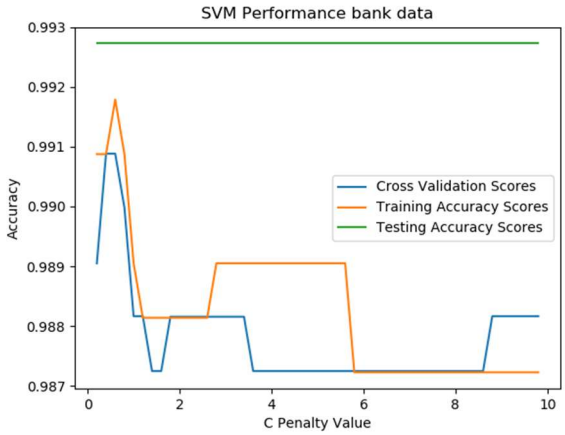
The runtime of this algorithm was much faster than most of the others, taking only a few seconds to run through all of the cross-validation iterations and the final fitting of the model. This could have been partially because there was only one hyper parameter to test, but more importantly, this algorithm is a very simple and fast one because the distance formula needed to perform it is very computationally inexpensive.

Support Vector Machine (SVM)

The Support Vector Machine (SVM) is a learning algorithm that attempts to classify data based on maximizing the margin between groups of data, and therefore improves the model's ability to classify points that are near this margin. In the case of data that is not linearly seperable, the SVM employs a soft margin that allows misclassification of points but with an associated error pentalty for misclassification. This error penalty is scaled by a constant value C, which can have a large effect on the training of a model. Hence, the value used for C is the hyperparameter for this algorithm. I chose to test this value on the range between .2 and 10. I chose these values because after a few test runs, the graphs exhibited the most change in these regions, and in my research, I frequently saw this value set to 1, so I wanted to test values around this number.

The cross-validation values for these graphs (below) exhibit very different trends. The leaf data shows a consistent increase in performance with greater c penalty until about 8 where it levels off, and the bank data prefers a very small value between 0 and 1. Actually, the optimal C value for the leaf data is 7.8, an for the bank data it is 0.4. For the leaf data, the traininng accuracy is 0.742 and the test accuracy is 0.588, and for the bank data, the training accuracy is 0.99, and the test accuracy is also 0.99. Again, the accuracy levels between the two datasets are quite striking, but also the fact that the leaf data has so many classifications means that errors in the data are more common, and potentially easier to make, so higher penalties for misclassifications will likely force the algorithm to create more complex decision boundaries because the higher level of fitting in the leaf data improves performance more than the



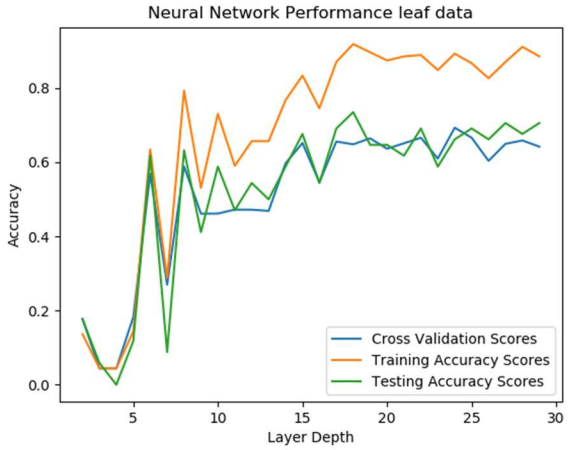


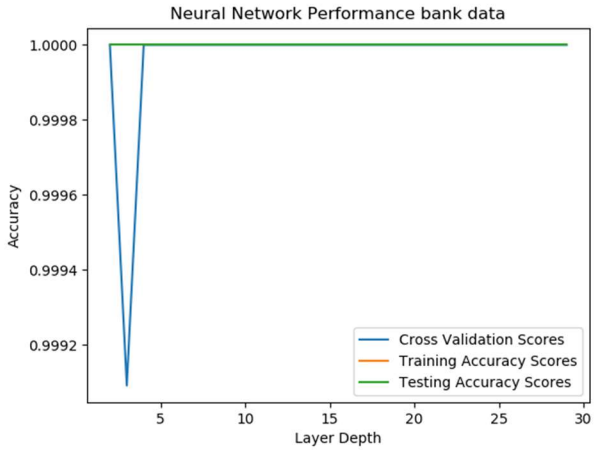
tensions of overfitting the data pull performance down. The opposite is true of the bank data, as a simple classification boundary is more likely to be effective when only two classifications exist.

The runtime of this algorithm was faster than AdaBoost, but slower than Decision Trees and kNN, if only by a few seconds. This algorithm has some complex arithmetic, and likely takes a while to find the complex decision boundaries with maximum margin in the leaf data, so it makes sense that this algorithm is slower.

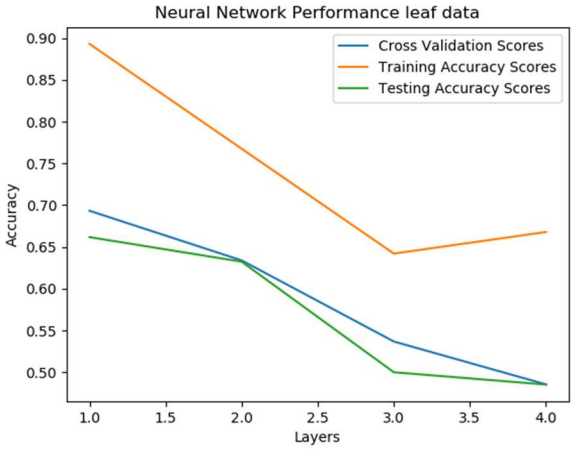
Neural Networks

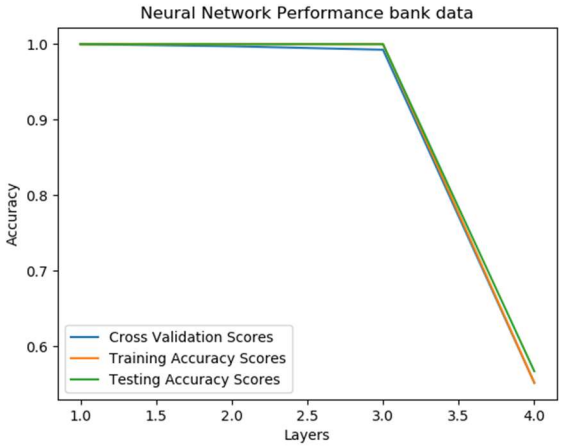
Neural Networks are one of the most well known and popular models for machine learning. They create a comparatively simplistic model of how a human brain might learn something, using nodes and activation functions, like how neurons work in the human brain. Because neural networks are an assortment of nodes, with input nodes being the input values to the network and the output nodes being a one-hot encoding of the resulting classification, the first question when defining a network is how many hidden nodes to use and how to organize them. Therein lies the hyperparameter I chose for this neural network implementation, which uses the scikit-learn MLPclassifier or multi-layer perceptron. To test varying values of this hyperparameter, I saw in my research that generally speaking most problems can be solved with only one to three separate layers, but that layers with more nodes generally improved performance. Therefore, I chose the range between one and four for the number of layers, and between 2 and 30 for the layer depth. I chose to stop at 30 for the layer depth due to computational complexity. Since there are two hyperparameters for this algorithm, but the computational complexity for testing all networks with depths between 2 and 30 and layers between 1 and 5 was too much, I decided to use one while testing depths since “One hidden layer is sufficient for the large majority of problems” (Doug, Cross Validated).





These two graphs show wildly different results. The leaf data exhibits increasing performance as the number of layers increases, but te bank data shows perfect classification very quickly, only needing a few elements to perfectly classify the data. From the maximum cross validation values, the optimal value for layer depth for the leaf data is 24, and only 2 for the bank dat a.





As predicted, since the cross-validation scores are visibly maximal at one, the optimal number of layers for both problems is only one. Therefore, the optimal hidden layer structure for the leaf data is one layer of 24, and for the bank data is one layer of 2 nodes. The training accuracy on the leaf data is 0.893, and the test accuracy is 0.662, and the training and testing accuracy of the bank data is a perfect 1.0. These are the highest accuracy figures for all of the algorithms tested. Since the leaf problem is more difficult, it required more hidden nodes to do a better job classifying the data, and conversely, the bank data only needed two hidden nodes in one layer to perfectly classify the data.

The runtime of this algorithm was relatively slow, but this is likely because I had two hyperparameters, which greatly increases the number of combinaions of hyperparameters to be tested. If I had made the initial assumption of needing only one layer, it may have sped up computation time.

Comparison

Algorithm	Leaf		Bank	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
Decision Tree	1.0	0.588	1.0	0.982
AdaBoost	0.93	0.647	1.0	1.0
kNN	1.0	0.609	1.0	1.0
SVM	0.742	0.588	0.99	0.99
Neural Network	0.893	0.662	1.0	1.0

Algorithm	Leaf Data Time (s)	Bank Data Time (s)
Decision Tree	1.785	2.231
AdaBoost	79.778	143.1
kNN	3.315	7.067
SVM	2.944	7.459
Neural Network	34.328	6.743

Based on the accuracy metrics, the Neural Network algorithm performed the best because not only did it perfectly classify the testing data for the bank notes, but it had the highest testing accuracy for the leaf data as well. It may have had a low training accuracy for the leaf data, but a high testing accuracy is more important for the general use of the algorithm. Based on time, the AdaBoost and Neural Network algorithms seem to perform quite poorly, but this is because they are the only two algorithms with two hyperparameters, and if I had used tree stumps for AdaBoost (depth of 1), and assumed only one layer for the neural network, it would have significantly reduced the number of calculations. The fastest algorithm however by far is the decision tree, needing less than half the time of any of the other algorithms.

Because the accuracy scores for the bank data is higher acros the board, the datasets clearly exhibited that the leaf data was much harder to classify than the bank data. Most likely, this is because there are 40 classifications of leaves and only two classifications of bank notes, and the bank data had several times more classified data points. More data gives the algorithms more to train on and therefore improves the model. The larger dataset for the bank data may also explain why it experienced longer computation times.

Works cited:

Brownlee, Jason. "Your First Machine Learning Project in Python Step-By-Step." *Machine Learning Mastery*, 10 June 2016, machinelearningmastery.com/machine-learning-in-python-step-by-step/.

Doug. "How to Choose the Number of Hidden Layers and Nodes in a Feedforward Neural Network?" *Cross Validated*, 2 Aug. 2010, stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw#1097.

Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

scikit-learn developers. "Scikit-Learn." - *Scikit-Learn 0.19.2 Documentation*, 2007, scikit-learn.org/stable/.