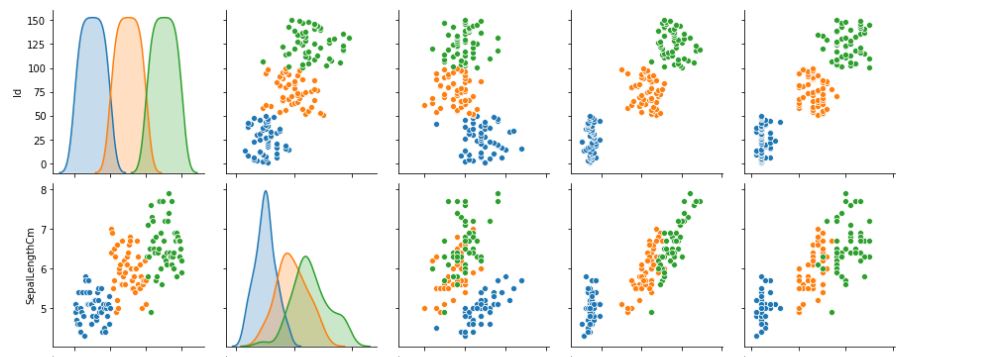
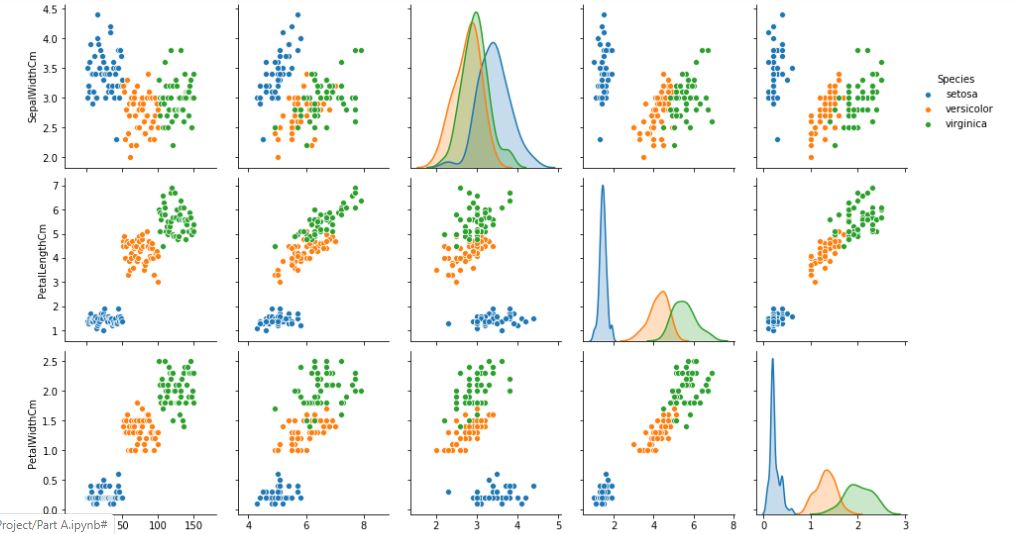
The Iris flower dataset is about Setosa, Versicolor, and Virginica irises and their characteristics. In the dataset, each of the iris species have 50 measured points of data that have the details of sepal length, sepal width, petal length, and petal width. The purpose with this dataset is to train our different algorithms – decision tree learning, backpropagation, k-nearest neighbors, k-means clustering, and a few variations of gradient descent – to be able to predict the iris type from random characteristic data. We used Jupyter Notebook to run and test our learning algorithms.

Before we can implement any of the unique algorithms, we first have to understand the data by displaying it in an easy to understand form. To do this, we examined the statistical data of the different characteristics and graphed scatterplots of all possible iris characteristic combinations (for the scatter plots, Blue is for Setosa, Orange is for Versicolor, and Green is for Virginica). The 2-D scatterplots help easily visualize that iris characteristics are the best duo combination that helps separate the data best. In this case, the sepal length and the petal length show clear differences and will be used for the algorithms. This will help the learning algorithms be more accurate and avoid superfluous noise.



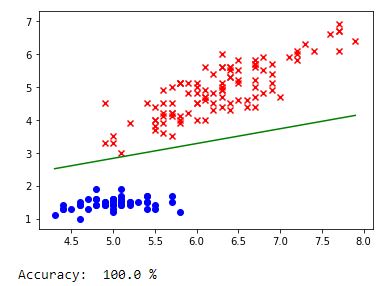
Decision tree learning was the first algorithm we implemented through scikit-learn to classify the iris dataset. Decision tree learning is a supervised learning algorithm that uses a branching tree pattern where each internal node is an input variable used to predict the external node’s (leaf) output variable. Backpropagation was the second supervised learning algorithm implemented. It works by calculating the error for the desired and predicted output before running minimalizing the difference. K-nearest was the third and final supervised learning algorithm. K-nearest creates its algorithm by matching the prediction inputs into data groups and predicts an output based on the data groups its input is closest to.

For each of these supervised learning techniques, to implement their algorithm, we split the iris dataset into datasets to be trained and datasets to check the accuracy of the trained data. Then using the algorithms pre-implemented through scikit-learn, we tested each individually for their accuracy by using the split test data and seeing if the predicted output matched the test output. Decision tree’s accuracy result was 95.56%; backpropagation’s results was 97%; and k-nearest’s result was 93%. Thus, backpropagation worked the best in this scenario.

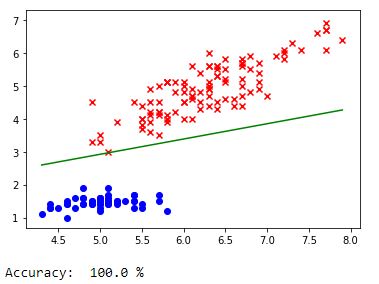
K-means clustering was the last algorithm implemented and is an unsupervised learning algorithm. This means unlike the previous supervised learning algorithms, k-means clustering is not given a desired output for a set of inputs that means it will find its own strategy to classify the data. The k-means clustering algorithm works by finding unique data point groups and their corresponding centers to categorize future and present datasets based on their proximity to the centroids of the groups.

Continuing onward, we used the iris dataset to compare the performances of three varients of gradient descent: batch gradient descent, stochastic gradient descent, and mini-batch gradient descent. Gradient descent according to “Towards Data Science” is a technique that iterates (one cycle/iteration when building the model through the chosen data is called an epoch) to discover the global/local minimum by calculating the derivative of the loss function (that ultimately reduces the error of calculated predictions). Different types of gradient descent algorithms differ through the way it is designed to discover the minimum (i.e., compute the error of prediction). For each of these three algorithms, the goal was to use sepal length and petal length input data points to predict whether the iris species was a Setosa iris (output).

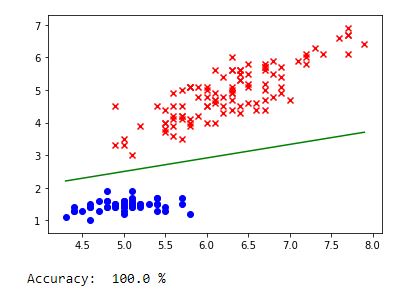
Stochastic gradient descent functions by randomly picking a data point to update the algorithm one step at a time. This effectively allows a simple convergence to the minimum where the randomness of the chosen data points helps the algorithm avoid local minimums through frequency quick updates. However, the cons to this stochastic gradient descent is that the randomness may create noise that potentially can cause the algorithm to overshoot the global minimum. Additionally, for the larger datasets, this algorithm is more computationally expensive due to its frequent updates. This variant of the gradient descent was the first one to be implemented for the Iris dataset in this project. The code used was taken from a github file made by Larix. Using a learning rate of 0.1, the time to train this neural network took 3.645 seconds and its predictive accuracy resulted in 100 %.



The unique way that batch gradient descent uses to discover the minimum is to analyze the entire dataset for each point’s individual dataset before updating the algorithm. This effectively allows a steady convergence with each iteration being run through a set of data instead of a single point. However, the cons to the batch gradient descent are it has a greater chance to hit the local minimum of the dataset (thus, arriving at an incorrect predictive network) and takes longer as it has to run through the entire dataset every iteration. This variant of the gradient descent was the second to be implemented for the Iris dataset and was implemented by modifying the base code from Larix’s stochastic gradient descent (and modified further to include momentum). Momentum was added to the batch algorithm as a way to improve its chance of not arriving at a local minimum. Using an update (learning) rate of 0.1 and a small momentum of 0.1, the time to train this neural network took 7.253 seconds and its predictive accuracy also resulted in 100 %.



Mini-batch gradient descent builds it algorithm by creating random smaller datasets from the larger dataset to run. Basically, this algorithm uses a mixture of both batch and stochastic gradient descent by utilizing small random datasets. Being the middle choice, mini-batch gradient descent has more robustness than batch descent and is more efficient computationally than stochastic descent while not exceling in either field. Mini-batch gradient descent followed the trend by using an update (learning) rate of 0.1; the time to train this neural network took 5.140 seconds and its predictive accuracy resulted in 100 %.



[The reason why the prediction accuracy of each of the gradient descent algorithms were so high could possibility be due to the large natural difference between the two characteristics (sepal and petal length) in correlation to their iris species. Changing the code to pick the iris species whose data points are in the center of the dataset results in a loss in accuracy.]

The digits dataset was another dataset that was analyzed using the three supervised learning algorithms (from Part A). The DataCamp tutorial was used for implementing the digits dataset and the previous decision tree, backpropagation, and k-nearest algorithms were altered to fit the new dataset. The results were that decision tree had the lowest accuracy at a weighted average of 88%, backpropagation had the second lowest accuracy at 96%, and k-nearest had the best accuracy at 98%.

In conclusion, the Iris dataset was utilized to implement several python algorithms that took in input values (i.e., sepal length and width) to predict an output value (i.e., species of iris). The purpose was to learn about each of the algorithms by coding and testing them in person. The code used in part A were taken from the links provided by the prompt and the code used and modified in part B mainly came from a user on github called Larix. Finally, testing and coding the AI algorithms and learning new tidbits about artificial intelligence in class and during this project were rewarding. I hope to utilize my new knowledge in the future as I explore my passion of robotics.

**Works Cited**

*All Links from the programing project*

Brownlee, Jason. “A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size.” *Machine Learning Mastery*, 19 Aug. 2019, machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/.

F, Antonio. “Linear Models Optimization.” *GitHub*, Oct. 2018, github.com/antonio-f/Programming-assignment-Linear-models-Optimization-/blob/master/Programming%20assignment%20(Linear%20models%2C%20Optimization)/week01\_pa.ipynb.

Larix. “Logistic\_Regression.” *GitHub*, June 2018, github.com/Larix/Logistic\_Regression/blob/master/logistic\_regression\_sgd.py.

Ruder, Sebastian. “An Overview of Gradient Descent Optimization Algorithms.” *Ruder.io*, Jan. 2016, ruder.io/optimizing-gradient-descent/.

S, Yang. “An Introduction to Gradient Descent.” *Towards Data Science*, Medium, May 2019, towardsdatascience.com/an-introduction-to-gradient-descent-c9cca5739307.