Read Me

Inside this zip file I have 7 text documents. These are my predicted outputs to the final project assignment.

There are also 2 subfolders. One is a subfolder holding 5 jupyter notebook .ipynb files which is the primary technology I used to complete this assignment. These files can be run through the jupyter notebook environment with no issues.

If the user does not have or does not wish to use Jupyter notebook then I have a separate subfolder of python .py files which should work in any python IDE environment. Though I used VSCode, it was also tested to work in Pycharm.

Within each subfolder for the python and jupyter code there is also a subfolder labeled “datasets” this is necessary to properly run the code as this data has been formatted through excel to remove the empty values and replace them with NaN values instead.

Each file is labeled DatasetX where X is the number of the dataset the functions and equations were performed on. In each of the classification datasets I used 4 algorithms and the code upon execution will print an accuracy test for each algorithm tested on the dataset given to me. As well as printing the predicted values of the testing data.

For the txt documents I chose the algorithms which had the highest accuracy test for each dataset

Self - Analysis

These were my attempts at predicting and solving problems given several training datasets and labels as well as testing sets to make predictions on. The first five problems were classification problems and so using classification-based machine learning algorithms was the obvious choice. Before I began making decisions as to which algorithms or methods that I would use to make my predictions I first needed to format and clean the datasets. I did this by opening each of the datasets in Excel then using the ctrl+f function to find any value of 1.00E+99, I then used the replace all function to replace these values with NaN. This was done as I knew that any entry with this value was an empty value. Once I had the data formatted it was time to begin.

As our first several problems are classification related, I narrowed the problem down to several algorithms I wanted to use as follows: Logistic regression, Support Vector Machines, Neural Networks, and Random Forest. Python provides many tools and libraries to utilize these algorithms, so I have run the datasets through each of the pre-built library functions and will explain my decisions here.

In order to test each of my functions I first split the TrainingData and TrainingLabels into 80/20 distribution in order to make predictions and gather information. For each dataset I used four different algorithms and tested their respective effectiveness.

Logistic regression:

Logistic regression named for the sigmoid function (also called the logistic function) used to describe properties of population growth. It is normally used for binary classification problems, however, multinomial logistic regression can be used to model nominal outcome variables where the log percentage of the outcomes are modeled as a linear combination of predictor variables. In other words, it’s useful when you want to classify subjects based on a set of predictor variables, which we have in this instance.

Some notes about the function called for logistic regression is the “solver= ” parameter provides the option to determine which algorithm will be used in the optimization problem. The default is ‘lbfgs’ however for datasets 1, 3, and 4 I found that ‘newton-cg’ tends to provide more accurate predictions given my split train/test datasets. That said in the case of dataset 4, it suffers from some performance issues taking much longer to perform its calculations.

Support Vector Machine:

One issue in logistic regression is that it doesn’t care whether instances are close to the decision boundary that it draws. SVM takes a given set of training examples, determines which categories the entries are marked into and then builds a model to assign new examples to one category of the other. It is determined to find the best line or hyperplane in order to separate our space into classes.

Some notes about the function called for support vector machine:

* Kernels in support vector machines change the definition of the dot product. The parameter ‘kernel’ gives us several options including ‘linear’ ‘poly’ ‘rbf’ and ‘sigmoid’.
  + The default is ‘rbf’ where each point in our induced space of Gaussian distributions becomes probability density function of a normal distribution. Of course, because this has incredible flexibility it can be very useful, however it is much more time consuming and has higher risk of overfitting than the other methods.
  + Next is ‘poly’ which induces space of polynomial combinations of the given features. It’s not as flexible as rbf, but there is less risk of overfitting and takes less time. Generally used as a middle ground between linear and rbf, though it has more hyperparameters to work with.
  + Linear is the simplest model, just utilizing a normal dot product. It has the highest risk of underfitting but is generally the most useful kernel to start with. After doing some research it seems the inventors of the SVM generally supported the idea that you should begin with the simplest model and only move onto a more complex model if it underfits.
* decision\_function\_shape parameter has two options one-vs-rest (ovr) and one-vs-one (ovo) I used ovo in all of my datasets due its primary use in multi-class classification problems.

Random Forest:

Is a more advanced application of the decision tree classifier. It creates multiple decision trees and merges them together in order to get a wider view of the problem and acquire a more accurate prediction. The way it works is that, in order to classify a new object given new attributes, each decision tree in our forest will conclude and give a classification. We then tally the total conclusions of each tree and take the classifications with the most occurrences and take the average difference from the output of different trees.

Some notes about the function called for random forest:

* n\_estimators parameter is used to determine the number of trees in the forest. I simply settled on 1000 for each of the datasets in this project
* max\_depth parameter does as it says and determines the maximum depth of the tree. I simply chose values between 10-15 and tested which gave better performance but didn’t test much farther.

Neural Network:

Neural networks are modeled after our own brains. They use nodes to pass information back and forth in a method reminiscent of neurons within the human brain. Here we’re using an artificial neural network to take an input, perform some calculations within a hidden layer of nodes, and then return an output.

Some notes about the function called for neural network:

* The solver parameter has three setting options ‘lbfgs’,’sgd’, and ‘adam’. ‘lbfgs’ is best used for smaller datasets and has better performance than the others. ‘sgd’ stands for stochastic gradient descent and therefore uses that method for weight optimization. I found the best performance in datasets 1 and 2 using this method. While datasets 3, 4, and 5 had the best performance with ‘adam’ which is a variant stochastic gradient-based optimizer which generally works well on larger datasets.
* Hidden\_layer\_sizes parameter is simply the number of neurons in the ith hidden layer
* For the second dataset I also found a bit of improvement by playing with the activation and learning\_rate parameters.

Finally, for the missing values estimations I simply used a prebuilt interpolation function for the first and a fillna function for the second using the median values as my replacements.