Submission Description

test.csv: The test file from Kaggle. Not actually used, but was supplied so I’m including it.

train.csv: Training file. Used for training and testing.

AddingPointsRegression.ipynb: A custom attempt at tweaking Polynomial Regression. Tries to add new instances to the data (all features, and a label), by interpolating points into the dataset. Uses a recursive “dimensional average breakdown”, to try and add these points. Tested for smaller visualizable dimensions graphically, and the logic should apply to higher dimensions, perhaps with less accuracy (curse of dimensionality). The hope is that these added points will be more or less centered around a desirable curve of best fit, to help facilitate said curve being drawn. Seems to neither hurt or improve accuracy significantly. From testing on 2D, the less of these points there are, the more centered around the desired “best fit” area they are. Deeper (recursively) generation of points offers less centered points. This behavior was expected, which is why I implemented a min_samples variable to the function to help limit the depth and splitting of subspaces. The goal is to add a small amount of meaningful points to help guide the training. This tradeoff is likely the reason no substantial change (good or bad), is seen in the accuracy. May be more useful for smaller less dimensional data sets though.

FuzzyRegression: Another custom attempt at tweaking Polynomial Regressors. The motivation was to take many strong learners (Polynomial Regressors fitted to data without regularization), and generalize them to make them weaker. Initially each learner was trained on a non-overlapping subset of the data (call this S_i). Then for subsequent training, the entire training set was used. Although the labels for each learner were different. Every learner received the correct labels for their data from S_i, but received “fuzzy labels” for the rest of their data. Supplied by the collection of every other learner predicting values for their S_i. This was really entirely experimental. I wanted to see what would happen if the labels for every learning changed over time. The idea was that initially these fuzzy labels would be accurate to the true labels (since each learner is initially overfitting to their S_i). Over time, each learner would generalize and become weaker. It is assumed each learner would become less accurate on data in their S_i, but more accurate for all other points. This would result in the fuzzy labels becoming “fuzzier and fuzzier” as time went on. I thought maybe perhaps this would cause the learners to generalize more, since they are potentially being trained on fuzzier data. Of course data that is so inaccurate is not beneficial. That’s why I was imposing early stopping after every iteration of running gradient descent for a few times with a low learning rate. Once the average of all learners was less than a previous run, stop the algorithm and return the learners. The results are in the notebook. I show the results of each learner initially, and after weakening. Since each learner is initially trained on a smaller subset of data, the increase in accuracy surely could be solely from the learners being exposed to more data after weakening. It’s hard to say what influence the method has with only these results. No further testing was done.

InitialAttempt.ipynb: First attempt at implementing learners and cleaning data different ways. *Note there is data leaking when cleaning the data. I didn’t know about fit to train data only, and then transforming test data. Makes perfect sense, just overlooked it initially.* Wrote some custom functions for cleaning the data. Dealing with NaNs and categorical data in various ways. I used OneHot encoding, and also 2 custom functions I implemented. To take non-ordinal categorical data and impose an order on it. Either by rank (assigning values 1,2,3,...,n), or proportionally which assigns values between the range 0 and 1. Various plots were used to observe relationships between the data. A good span of standard estimators were used including: Linear, Polynomial, Random Forest, Gradient Boost, Ridge Regression, Neural Network. Accuracy evaluations on a few metrics and included for each. Everything is also heavily commented to narrate the thought process while producing this notebook.

SecondAttempt.ipynb: Another attempt at cleaning data and implementing estimators. This time no data leakage occurred! Much of the same things we're done as in the *InitialAttempt,* except in a cleaner fashion. Additionally Scalers and Imputers were used. *Note that splitting the categorical and numerical data, cleaning each separately and recombining was easier and cleaner*. I also attempt to find optimal parameters for a 3 hidden layer neural network by using scipy's dual_annealing optimizer.

Data_description.txt: Description of data supplied by Kaggle. Was referenced during data cleaning.

Project_Proposal.pdf: Just attaching the Project Proposal again.

Final Comments

Cleaning data is a very comprehensive and potentially long process. It is the foundation of how well your learning algorithms will learn. I believe the well adopted expression is "garbage in, garbage out". If a particular model isn't producing sought after results, not only can you tweak the model or change algorithms, you can also clean and prepare the data in a different way as well. Neither (different preparation or algorithms) are a magical fix, but both are methods to be well aware of when building models. Hyperparameter tweaking is of course important, but don't think you have to be stuck in that domain and can't take a step back and change the algorithm or data prep as well. Take the time to explore relationships in your data, through visualization or statistical means. Understanding your data makes you as a practitioner more effective in cleaning it and implementing the best algorithm. Looking forward to learning a lot more in this field.