CSC11 Bonus Project

Kaggle Competition

By: Alexander Kam Student #:1000529537

12/20/2016

Titanic: Machine Learning from Disaster

Problem Description

For this kaggle competition we are expected to create an analysis which will predict which passengers will survive the titanic tragedy. The data that we are provided is the training and test sets given to us in a excel spreadsheet. Within both the training and testing spreadsheet we are given a number of features. We are given the passenger ID, survival status, passenger class, name, sex, number of siblings or spouse aboard, number of parents/children aboard , ticket number, passenger fare, cabin and port of embarkation in the training set. In the testing set we are given the same thing minus the survival status which we will be calculating with our model. Therefore the total number of features is 12, but not all 12 of these features are relevant to the model that we are trying to produce. For example, the name the passenger ID and the ticket number have nothing to do with who survives and who is going to die in the titanic crash. The size of the test data is 418 people while the size of the training data is 892 people.

Baseline Method

For the baseline method of analysis I chose to implement a KNN solution from scratch using Python as the programming language. I decided to go with the KNN solution for my first attempt because it was one of the first model selections we learned in class and I thought that this would be one of the better outcomes too for approaching this data set. In particular what we did in the beginning of the code to set up the KNN was that we had to go through all the different variables and fills in the missing data with the averages. For the KNN I chose the hyper parameter, k to be equal to 3 to produce the results. By using the KNN classifier our job was to calculate the Euclidean distance between different data points in order to classify the data into different groups. In order to calculate the Euclidean distance we basically sum the distance between each point then return the square root of that distance. In addition, we have the getNeighbour function within the knn file in order to find the neighbors to the set to generate a basic model. In addition, we have the correctness function which will check how correct the estimation of our code is. For this baseline cross validation was not done actually, instead I played around with the hyper parameter seeing how a different choice of K would change the outcome of my model and how correct it is going to be.

Advanced Baseline from Kaggle Community

Looking through the different posts on the kaggle community I found that most people use sklearn which is a python extension that has built in function for all the different types of learning models and cross validation techniques. The main solution that I found online that provided the best possible outcome for the titanic problem is using the random forest classification process. This method was not taught in class and is something that I learned from experimenting with the different types of models provided with sklearn. Upon further research I found that there is a relationship between random forests and the nearest neighbor algorithm

(KNN). They can both be considered to be weighted neighborhoods schemes. Meaning that they are built from a training set that makes predictions y for new data points x' by looking at the neighbors of the point, formalized by a weight function W. In addition, by continuing my research on the random forest algorithm I have realized that it is regarded as one of the most accurate learning algorithms that are available. In addition, it is able to run on large sizes of data. However, the major drawback to using the random forest is that it has been observed to over fit for specific data sets. From the solution that I found on the kaggle community it just uses a basic random forest algorithm provided by sklearn. However, as I am doing further research and from my understanding we can improve this output by using a cross validation method as well to improve our results.

Improvements

The kernel that I found online used a random forest to generate its prediction. As stated above we know a weakness for the random forest classifier is that it may over fit to a certain dataset. In order to prevent this from not occurring my solution was to employ a cross validation technique. Cross validation was something that was taught during class and is a way for assessing how the results of a statistical analysis will generalize to an independent data set. We don't wish to over fit to the specific dataset that we are given for the assignment. Therefore, by employing cross validation we are able to achieve a better baseline model for many different data sets. In addition, to just using the random forest sklearn I also tried multiple different types of modeling methods. This included trying naïve Bayes, the KNN classifier built in from sklearn. I wanted to try naïve Bayes because it was something we learned in class and was something that I was familiar with, but the outcome was that the solution that it provided was not as good as the solution that random forests with cross validation provided us with. In particular the cross validation technique that I used was the K-fold method which was taught to us in class during lectures and tutorials. For the k-fold method we divide the sample size in k groups of samples, called folds of equal size. The prediction function is learned using k-1 folds and the fold left out is used for testing. However, as I did more research on the problem and looked at additional cross validation methods I found out about the cross_val_score function that is included in sklearn and ended up using that for my final improvement because it is supposed to give the best score overall on different data sets. In addition, the KNN classifier performed arguably the same was the knn that was implemented by myself in the previous section. This is just a check for myself to see how my implementation goes versus an implementation done by someone else. In addition, in hindsight I found that instead of playing with the hyper parameter myself I could have used cross validation from sklearn to do the validation and achieve better results.

Things Learned

Throughout the process of completing this titanic challenge I have learned many different things that have to do with machine learning and programming. In particular, the most significant machine learning algorithm that I learned was the random forests method that was implemented in my improvement of the baseline solution. In addition, I extended my understanding of the existing methods that we learned in class and how they were not the optimal solution for solving this problem. In addition to learning a new method of classification I also researched a new cross validation method called cross val score that is located in the sklearn package within python. In conclusion, the titanic assignment was able to teach me a new method of model learning and cross validation.

<u>House Prices: Advanced Regression Techniques</u>

<u>Problem Description</u>

For the House Prices kaggle competition the goal of the model is to predict what the final house price for which the home will sell for. We are provided with training and test data set from the Kaggle competition website that provided us with a multitude of features. Within the data there are 79 different features that explain a particular house. Going into all these features would be unreasonable, if you want to find out more about each specific feature I suggest navigating to the kaggle website and downloading the data description file that is provided with the competition and that will explain every piece of data. What I can tell you is that the training data had 1460 different houses in the set and the test data had 1459 different houses in the set. In addition, for this data set we are not able to drop as many features as the titanic example because of how we are supposed to generate models based on as many features as possible. However, we can split the training data into a training set and a validation set.

<u>Baseline Method</u>

The baseline implementation of this model first requires us to manipulate the data so that we are able to fill in the missing values and split the training data into a training set and a validation set. Afterwards we apply our linear regression model from the linearRegression.py file with the training and testing sets. For the initial attempt I chose to do a linear regression model because of how I believe house prices should work. If a house has a better feature that it will increase the price of the house accordingly, therefore the linear regression model would be best suited for this sort of data set. However, after applying the linear model we can find that it does not exactly work this might be because my initial assumption was wrong about the features having a linear trend with the housing prices for the homes. This is because if there is no linear relation in the model, a linear regression line will never be able to accurately predict what the housing price is going to be. The basis for linear regression is to find a line that will go through as many data points as possible and satisfies the equations $y = xB + e$. This can be done mathematically by calculating mean, variance, covariance and coefficients for the formula then putting it all together again to achieve the $y = xB = e$ format.

<u>Advanced Baseline from Kaggle Community</u>

After checking the different kernels from the Kaggle community I found an interesting one that used a regression method called lasso regression. This is a method that was not taught in lectures or tutorials before so I had to do further research on the lasso regression method online. In general the lasso regression analysis is a method that performs both variable selection and regularization to enhance accuracy for the dataset. Going through research on the lasso regression model we can see that the model doesn't require the human to think and is an automatic procedure. However, using lasso would mean that the final product does not have to output a straight line and can be curved. Another problem with using the lasso model is that it is a dimension reduction algorithm. As a result, sometimes when the features are chosen they are not chosen correctly or accurately. From this advanced baseline solution we can see that it is an improvement from the linear regression that I created in the previous section. However, it is evident that this is not necessarily the best model and additional methods could be done to the model to improve its final results.

Improvements

Looking for ways to improve the result of the lasso regression was difficult if I solely focused on the material that we covered in lectures and tutorials. However, by expanding and looking into sklearn I was able to look at the different types of boosting algorithms that they had built into the python package. I began experimenting with the different types of boosting algorithms that was included in the package such as Ada Boost and Gradient Tree Boosting. However, I found that these methods were still lacking when comparing it with a new boosting method called XG boost. I came across this boosting method after going on the internet and reading about different types of boosting algorithms and found that many winners of previous competitions used XG Boosting versus the Ada Boost and Gradient Tree Boosting. In addition, before actually trying to combine boosting with lasso regression I tried them all separately and found that the outcome is not as accurate as combining a boosting algorithm with the lasso regression algorithm. In conclusion, I found that XG boosting is the best suit for when combining the algorithm with lasso regression. Upon further readings and research about XG boost I found that XG Boost is short for "Extreme Gradient Boosting", which is a type of gradient tree boosting as in the sklearn package. The goal of the XG boosting algorithm is to create a data set that is portable, accurate and scalable.

Things Learned

During the process of creating a model for the housing prices for homes I learned a few different concepts to machine learning. Expanding from what I have learned in class about linear regression, I was able to explore lasso regression in the advanced baseline method that I found on the kaggle kernel. In addition, for improving the lasso regression I looked into different boosting algorithms and found that the most common and widely used now is XG boosting. While looking into boosting techniques I also came across a different type of algorithm called bagging algorithms. This is the approach you take when random samples of data, build learning algorithms and take sample means to find bagging probabilities. In conclusion, the housing prices taught me yet another new learning model and boosting method, which expanded on what we covered in lectures and tutorials.