INDIVIDUAL ASSIGNMENT-REPORT

Statistical and Machine Learning Approaches

Castillo Acosta Mario

Individual Assignment

Mario Castillo Acosta

Part 1- Describing and executing five models

Logistic Regression

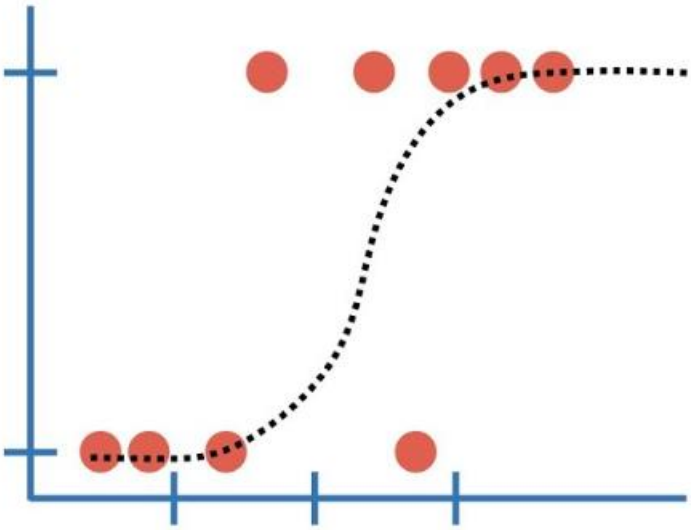
This is one of the most essential and basic algorithms in machine learning. This algorithm differs from its simplicity to implement and its great interpretability, commonly used for predicting numeric values on a timeline basis. It is an adaptation of linear regression, but instead of using continuous values, it uses discrete values to predict a class. These values are obtained thanks to the sigmoid function, which is known for its “s” form when graphed.

The model is composed of a dependent variable that can be labeled as “0” and “1”. This variable will be predicted based on one or multiple independent variables, these can be either numerical or categorical.

The end goal of a linear regression is to predict the best-fitted line within the data pattern. This is achieved by minimizing the errors in the distance between the line and the actual data, whereas, the objective of logistic regression is to predict the probability of the most accurate outcome, weather a point in the data set will be a “1” or a “0”. There are several types of logistic regression, Binary , multi class and ordinal.

In order to measure the performance of a classification we can use a cost function called log loss or cross-entrophy, that helps us evaluate the performance of the classifications. We cannot use the same metric function, mean squared error, as we do in linear regression, simply put because it is not linear.

One possible downside of this model is that it might tend to overfit when the number of observations is less than the number of features.



Random Forest

This algorithm can be used for both classification and regression, and it is similar to a decision tree, in the sense that it is composed of multiple decision trees. At the end the results of these individual trees are averaged in order to obtain a result.

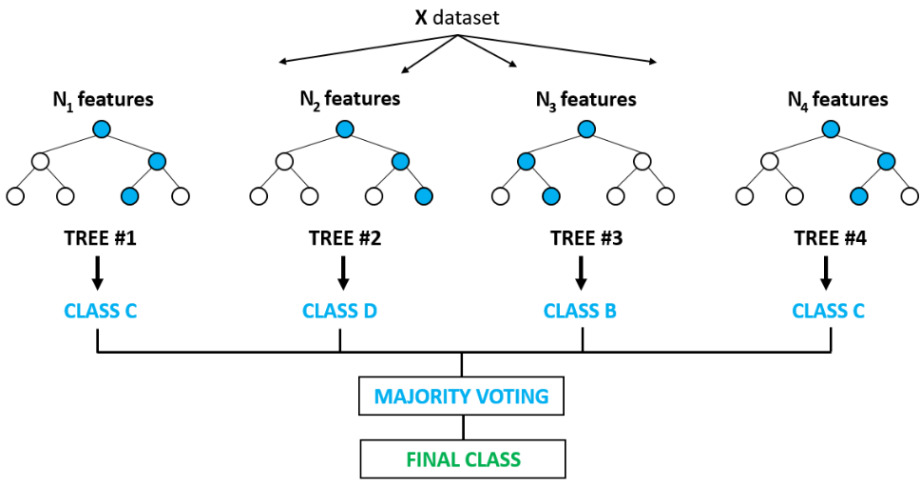
The model structure is composed of a root node, which contains all the data before the splitting starts, and subsequently creating different sub-nodes. Decision nodes are the parts that keep splitting into more nodes based on different questions asked to the data. When the node cannot further split, it is called a terminal node, which holds a result. In order to make the decision to split, the model uses mean square error and chooses the lowest value.

The overall benefit of this model is its capacity to lower variance and create an accurate prediction. Compared with a decision tree, which can highly overfit because it is based only in the result of one tree.

At the end, each individual tree comes up with their own results, based on a random subset and particular variables, but all together, the results of each individual tree contributes to a more accurate result with less error.

In order to further tune a random forest and avoid overfitting with we can adjust hyper-parameters, like maximum amount of tress and features considered by each splitting node.

Another benefit of random forest is the utilization of statistical techniques for selection and division of data called Bagging or Bootstrapping. In summary, these techniques help the model to have less bias results, by how the data is selected for the analysis of each tree.



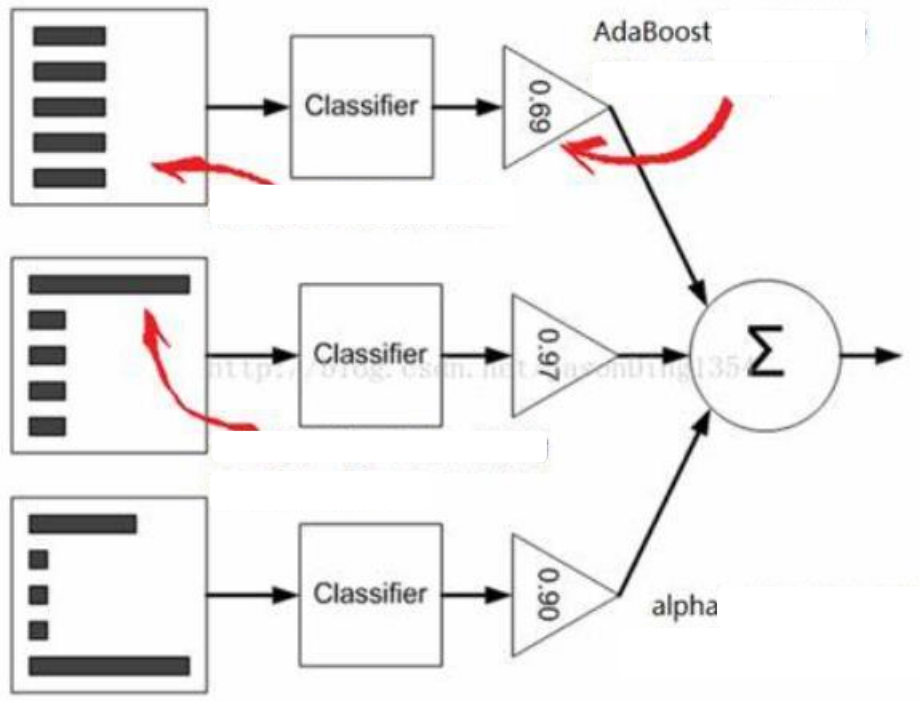
Adaptive Boosting (Adaboost)

The basic idea behind Adaboost is to build a decision tree based on the error of the previous trees in order to make a more accurate prediction, but for classification and regression. It is commonly used along different algorithms, particularly Random forest. The algorithm uses "weak learners" that are the very basic components of decision trees called stumps. The benefit of this model is that once all the weak learners results are converged, it becomes a "Strong learner", which is nothing more than a more accurate result based on the average of multiple results.

Its structure is based on decision trees, which will give a result based on the particular sample they are evaluating. Afterwards, this decision tree or stump, will be evaluated, and depending on its evaluation the stump will receive more or less weight. This will be done to every stump analyzing a subset of the data. The key procedure is that wrong predictions will be trained again, so the new stump will improve based on the error of the previous stump.

Another important point is that some trees have more weight than others do, at the start of the model, every tree has the same weight, but as the models are being trained and results are being analyzed weight changes depending on the performance of the stump.

Ada boost or adaptive boosting is part of an ensemble technique known as boosting, which ultimate goal is to combine different base models to create an optimal prediction.

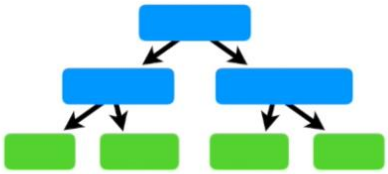


Gradient Boosting

Gradient boosting uses the same principal as Adaboost in Random forest. It is an algorithm focus on a loss function based on square errors to be optimized. In its process, it utilizes weak learners as well, that together create a more accurate or less bias result.

In essence, it first create some predictions, and calculate the loss, or difference between the expected and the predicted. Then it will create a second process by means of “weak learners”, only based on the wrong prediction, so every new learner will be more accurate than the second will, because it is based on the mistakes of the previous one. This process will continue based on the threshold established.

Even though, gradient boosting and adaboost are very similar, they differ in their focus, adaboost tries to improve the performance of its learners by constantly adjusting the weight of the stumps that predicted wrong. On the other hand, gradient boosting focuses on the residuals between predicted and expected, of every learner, and then improving it in the next learner.



Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

KNN Regression

K-Nearest Neighbors Classification is a, non-parametric, supervised algorithm used to predict the classification of the data (and regression too).

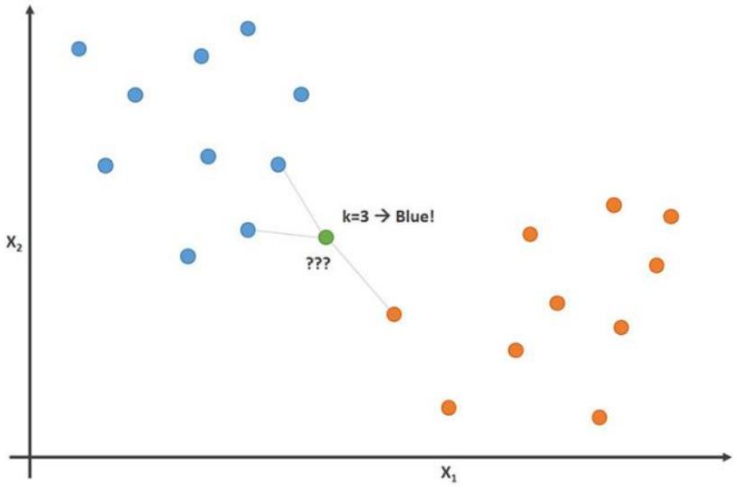
The basic idea of the algorithm is to predict the values of unknown data points based on the proximity with known data points. The average of the nearest neighbors is used for the final prediction.

First, the distance of this unknown data point, in regards to the rest of the data is calculated, and then the closest data points are selected and averaged

One important part of this algorithm is how to determine the value of “K” or the amount of clusters. One way to find “K” is to try different measures until the variation is no longer significant. For better visualization, we can just plot the reduction in variance, and find the “elbow”, which is the point where the variation does not vary significantly anymore. Other methods

In terms of the distance calculation, you can use several methods: Euclidian, Manhattan, Hamming, although this one is more for classification.

One downside of this algorithm is when the class or cluster dominates the prediction based on the “majority vote” for the clustering process. This issue is usually dealt with by assigning a weight to the contribution of the neighbor, and in that way compensate for the skewness of the data.



Part 2- Comparing, tuning and evaluating models

Models used:

- Logistic regression
- Random Forest
- Adaptive Boosting (Adaboost)
- Gradient Boosting
- KNN(regression)

Model	ACC. Test	ACC. Train
Logistic regression	0.7916	0.7973
Random Forest	0.7727	
AdaBoost	0.8014	0.8221
Gradient Boosting	0.7896	0.794
KNN	0.7339	0.9074

As shown by the results, the best performing with less overfitting model was **AdaBoost**.

Pre-Processing

After being imported, we checked the data for NA is although there was none; in general the data was pretty clean and almost ready to use.

The most work was actually devoted for feature engineering and variable selection. There were several variables created, categorical values transformed into dummies, other categorical values were grouped together and general regularization of some values.

Regarding the feature selection, the method used was fisher score, which ultimately gave the top variables responsible for the greatest weight or value on the data.

Lastly, we verified the columns both in the train and in test to make sure all the columns were present, and the proper adjustments were done.

The Model Setup Structure

The code was structured in a way that at the end we could perform a benchmark analysis.

We started by defining a resampling method for the data selection, the method used was cross validation for all of the models. Regarding iterations, which represent the amount of times we want to retrain our model, I manually adjusted the value for every model in order to explore and approximate the best results. For example, for Logistic Regression the iterations were set up at 70, while the iterations in AdaBoost was set up at five, after several tries, these were the numbers that gave a better result.

The next step was to define the model we want to use; five models were selected, although nine in total were tested, like Naive Bayes, Extreme Gradient Boosting, KSVM, and Neural networks.

Naturally, there were tuning parameters, these help the model get more accurate results, by avoiding overfitting and reducing the noise, among others. There was no hyper parameter tuned on any of the models, although the model might accept different ones.

For example, adjusting the amount of trees within the random forests or the amount of features considered by every tree, this all together would help the model avoid possible overfitting. We can have an idea if our model is over fitting, if we observe a big gap between the accuracy of training and data.

Particularly for my best performing model, AdaBoost, the most relevant hyper parameter is tuned with “N estimators”. All this has to be done before training the model.

After all the models start the training process, we use AUC in order to understand the performance of each model. As a reminder, we use ROC to understand how a model predictions performed during an iteration, and then we use AUC to understand which one was overall, during all the iterations the best performing model. Then, this best performing model is selected apart. This process happens for each one of the five different models. Ultimately, the purpose of the benchmark process is to be able to compare the best iterations of models among themselves.

Particularly in this code, as mentioned before, this process was done individually in each model, and then at the end we have the best result of each of the five different models.