Comparing Classification Algorithms

Sudip Tribhanga Dey
Machine Learning Intern
AI Technology and Systems
www.ai-techsystems.com
deysudip089@gmail.com

Abstract: **Machine learning algorithms are widely used in the analyses of high-dimensional gene expression datasets for gene selection (Hao and Weiss, 2016). In general, analysis methods are grouped into two groups, namely, selection of important variables and identification of subgroup structures associated with phenotypes. Taxonomic and gene function profiles are high-dimensional datasets with alike statistical parameters. Machine learning algorithms are mathematical model mapping methods used to learn or uncover underlying patterns embedded in the data. Machine learning comprises a group of computational algorithms that can perform pattern recognition, classification, and prediction on data by learning from existing data (training set).**

Keywords: Machine Learning, Mapping method, Classification, Prediction

INTRODUCTION

What is Machine Learning?

In the statistical context, Machine Learning is defined as an application of artificial intelligence where available information is used through algorithms to process or assist the processing of statistical data. While Machine Learning involves concepts of automation, it requires human guidance. Machine Learning involves a high level of generalization in order to get a system that performs well on yet unseen data instances.

Why should statistical agencies consider machine learning?

Machine learning is a relatively new discipline within Computer Science that provides a collection of data analysis techniques. Some of these techniques are based on well-established statistical methods (e.g. logistic regression and principal component analysis) while many others are not.

Most statistical techniques follow the paradigm of determining a particular probabilistic model that best describes observed data among a class of related models. Similarly, most machine learning techniques are designed to find models that best fit data (i.e. they solve certain optimization problems), except that these machine learning models are no longer restricted to probabilistic ones.

Therefore, an advantage of machine learning techniques over statistical ones is that the latter require underlying probabilistic models while the former do not. Even though some machine learning techniques use probabilistic models, the classical statistical techniques are most often too stringent for the oncoming Big Data era, because data sources are increasingly complex and multi-faceted. Prescribing probabilistic models relating variables from disparate data sources that are plausible and amenable to statistical analysis might be extremely difficult if not impossible. 5. Machine learning might be able to provide a broader class of more flexible alternative analysis methods better suited to modern sources of data. It is imperative for statistical agencies to explore the possible use of machine learning techniques to determine whether their future needs might be better met with such techniques than with traditional ones.

Types of Machine Learning Algorithm I have Compared:

I. Decision Trees

Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

From a high level, decision tree induction goes through 4 main steps to build the tree:

1. Begin with your training dataset, which should have some feature variables and classification or regression output.
2. Determine the "best feature" in the dataset to split the data on; more on how we define "best feature" later
3. Split the data into subsets that contain the possible values for this best feature. This splitting basically defines a node on the tree i.e. each node is a splitting point based on a certain feature from our data.
4. Recursively generate new tree nodes by using the subset of data created from step 3. We keep splitting until we reach a point where we have optimized, by

some measure, maximum accuracy while minimizing the number of splits / nodes.

Step 1 is easy, just grab your dataset!

For step 2, the selection of which feature to use and the specific split is commonly chosen using a greedy algorithm to minimize a cost function. If we think about it for a second, performing a split when building a decision tree is equivalent to dividing up the feature space. We will iteratively try out different split points and then at the end select the one that has the lowest cost. Of course, we can do a couple of smart things like only splitting within the range of values in our dataset. This will keep us from wasting computations on testing out split points that are trivially poor.

For a regression tree, we can use a simple squared error as our cost function:

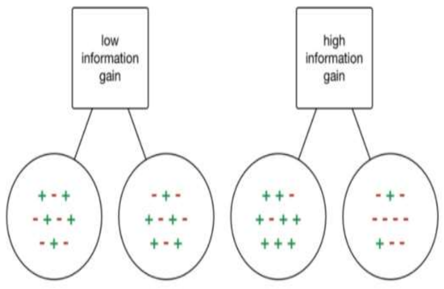
$$E = \sum (Y - \widehat{Y})^2$$

Where Y is our ground truth and Y-hat is our predicted value; we sum over all the samples in our dataset to get the total error. For a classification, we use the *Gini Index Function:*

$$E = \sum (p_k * (1 - p_k))$$

Where pk are the proportion of training instances of class k in a particular prediction node. A node should *ideally* have an error value of zero, which means that each split outputs a single class 100% of the time. This is exactly what we want because then we know, once we get to that particular decision node, what exactly our output will be whether we are on one side of the decision boundary or the other.

This concept of having a single class per-split across our dataset is known as *information gain*. Check out the example below.



If we were to choose a split where each output has a mix of classes depending on the input data, then we really haven't *gained* any information at all; we don't know any better whether or not a particular node i.e. feature has any influence in classifying our data! On the other hand, if our split has a high percentage of each class for each output, then we have *gained* the information that splitting in that particular way on that particular feature variable gives us a particular output!

Now we could of course keep splitting and splitting and splitting until our tree has thousands of branches…. but that's not really such a good idea! Our decision tree would be huge, slow, and overfitted to our training dataset. Thus, we will set some predefined stopping criterion to halt the construction of the tree.

The most common stopping method is to use a minimum count on the number of training examples assigned to each leaf node. If the count is less than some minimum value then the split is not accepted and the node is taken as a final leaf node. If all of our leaf's nodes become final, the training stops. A smaller minimum count will give you finer splits and potentially more information, but is also prone to overfitting on your training data. Too large of a min count and you might stop to early. As such, the min value is usually set based on the dataset, depending on how many examples are expected to be in each class.

II. Boosted Trees

Gradient boosting is one of the most powerful techniques for building predictive models. The first realization of boosting that saw great success in application was Adaptive Boosting or AdaBoost for short. The weak learners in AdaBoost are decision trees with a single split, called decision stumps for their shortness.

AdaBoost works by weighting the observations, putting more weight on difficult to classify instances and less on those already handled well. New weak learners are added sequentially that focus their training on the more difficult patterns.

Gradient boosting involves three elements:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

1. Loss Function

The loss function used depends on the type of problem being solved.

It must be differentiable, but many standard loss functions are supported and you can define your own.

For example, regression may use a squared error and classification may use logarithmic loss.

A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function that may want to be used, instead, it is a generic enough framework that any differentiable loss function can be used.

2. Weak Learner

Decision trees are used as the weak learner in gradient boosting.

Specifically, regression trees are used that output real values for splits and whose output can be added together, allowing subsequent models outputs to be added and "correct" the residuals in the predictions.

Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or to minimize the loss.

Initially, such as in the case of AdaBoost, very short decision trees were used that only had a single split, called a decision stump. Larger trees can be used generally with 4-to-8 levels.
It is common to constrain the weak learners in specific ways, such as a maximum number of layers, nodes, splits or leaf nodes.
This is to ensure that the learners remain weak, but can still be constructed in a greedy manner.

3. Additive Model

Trees are added one at a time, and existing trees in the model are not changed.
A gradient descent procedure is used to minimize the loss when adding trees.
Traditionally, gradient descent is used to minimize a set of parameters, such as the coefficients in a regression equation or weights in a neural network. After calculating error or loss, the weights are updated to minimize that error.

Instead of parameters, we have weak learner sub-models or more specifically decision trees. After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss (i.e. follow the gradient). We do this by parameterizing the tree, then modify the parameters of the tree and move in the right direction by (reducing the residual loss.

Gradient Boosting Trees

The algorithm for Boosting Trees evolved from the application of boosting methods to regression trees. The general idea is to compute a sequence of (very) simple trees, where each successive tree is built for the prediction residuals of the preceding tree. As described in the General Classification and Regression Trees Introductory Overview, this method will build binary trees, i.e., partition the data into two samples at each split node. Now suppose that you were to limit the complexities of the trees to 3 nodes only: a root node and two child nodes, i.e., a single split. Thus, at each step of the boosting (boosting trees algorithm), a simple (best) partitioning of the data is determined, and the deviations of the observed values from the respective means (residuals for each partition) are computed. The next 3-node tree will then be fitted to those residuals, to find another partition that will further reduce the residual (error) variance for the data, given the preceding sequence of trees.
It can be shown that such "additive weighted expansions" of trees can eventually produce an excellent fit of the predicted values to the observed values, even if the specific nature of the relationships between the predictor variables and the dependent variable of interest is very complex (nonlinear in nature). Hence, the method of gradient boosting - fitting a weighted additive expansion of simple trees - represents a very general and powerful machine learning algorithm.

III. Random Forest

The algorithm of Random Forest:

Random forest is like bootstrapping algorithm with Decision tree (CART) model. Say, we have 1000 observation in the complete population with 10 variables. Random forest tries to build multiple CART models with different samples and different initial variables. For instance, it will take a random sample of 100 observation and 5 randomly chosen initial variables to build a CART model. It will repeat the process (say) 10 times and then make a final prediction on each observation. Final prediction is a function of each prediction. This final prediction can simply be the mean of each prediction**.**

What is Random Forest algorithm?

First, Random Forest algorithm is a supervised classification algorithm. We can see it from its name, which is to create a forest by some way and make it random. There is a direct relationship between the number of trees in the forest and the results it can get: the larger the number of trees, the more accurate the result. But one thing to note is that creating the forest is not the same as constructing the decision with information gain or gain index approach.
The author gives 4 links to help people who are working with decision trees for the first time to learn it, and understand it well. The decision tree is a decision support tool. It uses a tree-like graph to show the possible consequences. If you input a training dataset with targets and features into the decision tree, it will formulate some set of rules. These rules can be used to perform predictions. The author uses one example to illustrate this point: suppose you want to predict whether your daughter will like an animated movie, you should collect the past animated movies she likes, and take some features as the input. Then, through the decision tree algorithm, you can generate the rules. You can then input the features of this movie and see whether it will be liked by your daughter. The process of calculating these nodes and forming the rules is using information gain and Gini index calculations.
The difference between Random Forest algorithm and the decision tree algorithm is that in Random Forest, the process es of finding the root node and splitting the feature nodes will run randomly.

How Random Forest algorithm works?

There are two stages in Random Forest algorithm, one is random forest creation, the other is to make a prediction from the random forest classifier created in the first stage. The whole process is shown below, and it’s easy to understand using the figure.
Here the author firstly shows the Random Forest creation pseudocode:

1. Randomly select “**K**” features from total “**m**” features where **k << m**
2. Among the “**K**” features, calculate the node “**d**” using the best split point
3. Split the node into **daughter nodes** using the **best split**
4. Repeat the **a to c** steps until “l” number of nodes has been reached
5. Build forest by repeating steps **a to d** for “n” number times to create **“n” number of trees**

n the next stage, with the random forest classifier created, we will make the prediction.

The random forest prediction pseudocode is shown below:

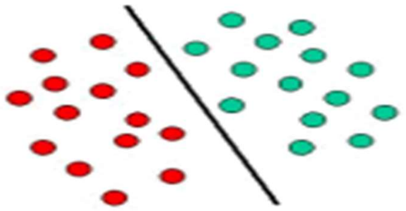
1. Takes the **test features** and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)
2. Calculate the **votes** for each predicted target

3. Consider the **high voted** predicted target as the **final prediction** from the random forest algorithm
The process is easy to understand, but it's somehow efficient.

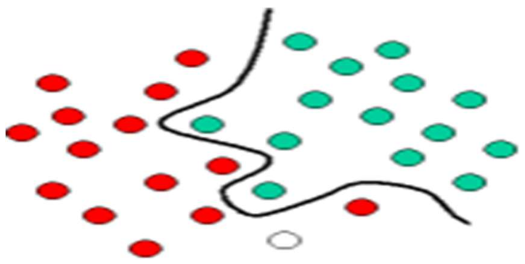
IV. Support Vector Machine

Support Vector Machines (SVM) Introductory Overview

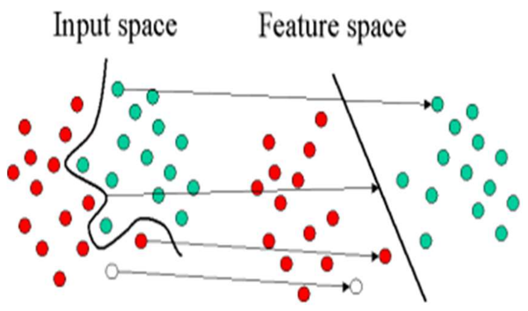
Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. A schematic example is shown in the illustration below. In this example, the objects belong either to class GREEN or RED. The separating line defines a boundary on the right side of which all objects are GREEN and to the left of which all objects are RED. Any new object (white circle) falling to the right is labeled, i.e., classified, as GREEN (or classified as RED should it fall to the left of the separating line).



The above is a classic example of a linear classifier, i.e., a classifier that separates a set of objects into their respective groups (GREEN and RED in this case) with a line. Most classification tasks, however, are not that simple, and often more complex structures are needed in order to make an optimal separation, i.e., correctly classify new objects (test cases) on the basis of the examples that are available (train cases). This situation is depicted in the illustration below. Compared to the previous schematic, it is clear that a full separation of the GREEN and RED objects would require a curve (which is more complex than a line). Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers. Support Vector Machines are particularly suited to handle such tasks.



The illustration below shows the basic idea behind Support Vector Machines. Here we see the original objects (left side of the schematic) mapped, i.e., rearranged, using a set of mathematical functions, known as kernels. The process of rearranging the objects is known as mapping (transformation). Note that in this new setting, the mapped objects (right side of the schematic) is linearly separable and, thus, instead of constructing the complex curve (left schematic), all we have to do is to find an optimal line that can separate the GREEN and the RED objects.



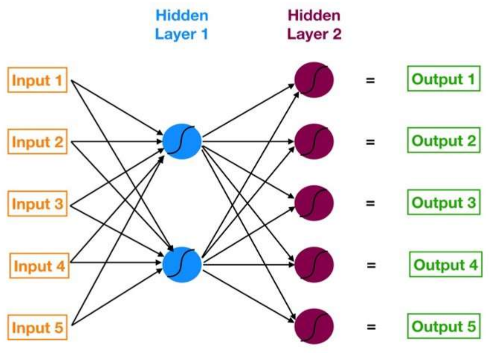
V. Neural Networks

What is a Neural Network?

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Neural networks can adapt to changing input; so, the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.
Application of Neural Networks
Neural networks are broadly used, with applications for financial operations, enterprise planning, trading, business analytics and product maintenance. Neural networks have also gained widespread adoption in business applications such as forecasting and marketing research solutions, fraud detection and risk assessment.
A neural network evaluates price data and unearths opportunities for making trade decisions based on the data analysis. The networks can distinguish subtle nonlinear interdependencies and patterns other methods of technical analysiscannot. According to research, the accuracy of neural networks in making price predictions for stocks differs. Some models predict the correct stock prices 50 to 60 percent of the time while others are accurate in 70 percent of all instances. Some have posited that a 10 percent improvement in efficiency is all an investor can ask for from a neural network.
There will always be data sets and task classes that a better analyzed by using previously developed algorithms. It is not so much the algorithm that matters; it is the well-prepared input data on the targeted indicator that

ultimately determines the level of success of a neural network.

Let’s start with a really high-level overview so we know what we are working with. Neural networks are multi-layer networks of neurons (the blue and magenta nodes in the chart below) that we use to classify things, make predictions, **etc**. Below is the diagram of a simple neural network with five inputs, 5 outputs, and two hidden layers of neurons
.



Neural network with two hidden layers
Starting from the left, we have:

1. The input layer of our model in orange.
2. Our first hidden layer of neurons in blue.
3. Our second hidden layer of neurons in magenta.
4. The output layer (a.k.a. the prediction) of our model in green.

The arrows that connect the dots shows how all the neurons are interconnected and how data travels from the input layer all the way through to the output layer.
Later we will calculate step by step each output value. We will also watch how the neural network learns from its mistake using a process known as backpropagation.