Machine Learning on Vertafore Salesfore Data

Senghuot Lim
Department of Computer Science
University of Washington
Seattle, WA U.S.A
senghuot@cs.washington.edu

Abstract

Using Machine Learning algorithm, we can write a program which can improve itself by generalizing from examples. Human evolve in a three dimensional environment, seeking pattern in a hyper dimensional space manually become very time consuming if not impossible. For this reason, it is ideal to define a methodology to automate this process for us. This paper is a documentation which will walk you through the process of applying Multiclass Adaboost, Decision Trees, and a Recommender system on Vertafore Salesfore data. The goal of this project is to classify Vertafore transaction outcomes. In order words, with a large enough training data, we should be able to intelligently predict whether or not a future transaction is going to be closed or won and recommend a similar successful transaction for comparison.

1 Introduction

This project is building on top of many various open source projects. For example, I used python library, Scikit-learn, to help me build decision trees and Numpy for speed optimization. You can find many research papers on Multiclass Adaboost and Recommender System algorithms online or at your local universities. There are probably hundreds of variation of those algorithms available and many more by next year. However, they're all follow three components: representation, evaluation, and optimization. Below, I will walk you through all the algorithms I used for this project: (1)Recommender System (2)Weak Decision Trees as a representation, and (3)Multiclass-Adaboost as both evaluation and optimization.

2 Algorithms

I want to set us up with some common terminology. Suppose, we're given a set of training data $(x_1, c_1)...(x_n, c_n)$ where the input $x_i \in \mathbb{R}^p$ and output c_i as a discrete value. Moreover, $\delta(x = 1)$ return 1 if $x = 1$, otherwise return 0. (x_i, y_i) is a single data point and $x_i = (x_{i,1}, ...x_{i,d})$. As you can see, x_i has d dimensions or features. Okay, now that we're on the same page, let's jump into the algorithms.

2.1 Recommender System

Let's briefly review Euclidean distance. The Euclidean distance $r_2(x, y)$ between two 2-dimension vectors $x = (x_1, x_2)$ and $y = (y_1, y_2)$ then the distance between those two points would be:

$$r_2(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} = \sqrt{\sum_{i=1}^{2}(x_i - y_i)^2}$$

Going back to your data set, remember our data set x_i is just a row in a large data set and it can have d dimensions or features. We can visualize each x_i in a d dimensional space, any x_j share the most common attributes with x_i if they have the smallest Euclidean distance.

$$sim(a, b) = \arg\min_b \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

$$\therefore \arg\max_b \frac{1}{1 + \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}}$$

2.2 Decision Trees

Decision trees are being split based on the idea of maximizing information gain through entropy. I used python library, Scikit-learn, to help me build decision trees. Entropy, $H(Y)$, is used to determine the chaos of data set, chaos is defined as the mixture of the each class in a data set. The goal is to decrease the mixture of classes or labels in the data set. To do this, we split the data set into many smaller data sets based on features, $H(Y \mid X)$, in hope to yield less chaos. Big information gain equals to less chaos. Therefore, we will split the data set on features that maximize information gain

By default when constructing decision trees, each sample gets equal weights. However, we need to accommodate unequal weights at each stage of the

Multiclass-Adaboost, we have to reformulate our $H(X)$ and $H(Y \mid X)$.

$$H(Y) = -\sum_{j=1}^{k}\sum_{i=1} D(i)\delta(Y = y_j)log_2 D(i)\delta(Y = y_i)$$

$$H(Y \mid X) = -\sum_{i=1}^{k}\sum_{j=1} D(j)\delta(X = x_i)\sum_{a \subset X_i} \frac{\sum_b D(b)\delta(Y = y_a)}{\sum_b D(b)} log_2(\frac{\sum_b D(b)\delta(Y = y_a)}{\sum_b D(b)})$$

$$IG = H(Y) - H(Y \mid X)$$

Footnote:

$\delta(1 = x)$ return 1 if $x = 1$ otherwise return 0.
* a is a subset of N data set split on i
* b is a subset of a which is a subset of N

2.3 Multiclass-Adaboost

The Multiclass Adaboost algorithm is an iterative process that combine many weak classiers to approximate the Bayes classifier $C(x)$. Starting with an unweighted training samples, the Adaboosts build a simple classifier, in our case we are using decision trees with depth of three to represent our weak classifiers. If a training data is miss-classified, the weight of that training data point would increase or 'boosted'. A second classifier is build using the new weights, which are no longer equal. In other words, the second tree learns from the mistakes made by its predecessor, in this case it is the first tree. It is ideal to build 500 to 1000 classifiers. An alpha score is then assign to each tree based on how well they classify the training data, and the final classifier is defined as the linear combination of all the classifiers in each stage.

An important question you might ask yourself would be, why depth of three. A quick answer is to avoid over-fitting. A longer answer, would need a quick thought experiment, let say we want to represent a Boolean function of say 10 variables. The space complexity would be $2^{10} = 1024$, it doubles on every addition of Boolean variable. Using a weak classifier of depth of 3 instead of 10 allow us to generalize from the training examples instead of hallucinating a classifier that is not grounded in reality by simply encoding random quirks in the data.

1. *Initialize the observation weights* $w_i = \frac{1}{n}$
2. For $m = 1...M$:
   (a) *Fit a classier* $T^{(m)}(x)$ *to be the training data using weights* w_i
   (b) *Compute Error*

$$error^{(m)} = \sum_{i=1}^{n} w_i * \delta(c_i \neq T^{(m)}(x_i))$$

   (c) *Compute Classifier Alpha*

$$\alpha^{(m)} = log(\frac{1 - error^{(m)}}{error^{(m)}}) + log(K - 1)$$

   (d) *Update Weights*

$$w_{i+1} = w_i * exp(\alpha^{(m)} * \delta(c_i \neq T^{(m)}(x_i))$$

   (e) *Re-normalize Weights*

$$w_i = \frac{w_i}{\sum_{i=1}^{n} w_i}$$

3. *Output*

$$C(x) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} * \delta(T^{(m)}(x) = k)$$

3 Conclusion

Our algorithm is just merely a process of the tweaking the next representation by learning from its predecessor through evaluation and optimization. A good analogy of this algorithm would be, you don't fix the teenager, instead you would give advices to the next parents to raise a stronger version of its predecessor. The algorithms only see features and each data point as numbers. It is not aware of human context or the meaning we give to each relationship between features and data points.