AN EMPIRICAL LOOK AT THE PERFORMANCE AND ACCURACY OF VARIOUS MACHINE LEARNING ALGORITHMS

BY ERIC LE FORT, B.ENG.

A Thesis Submitted to the School of Graduate Studies in Partial Fulfilment of the Requirements for the Degree Master of Applied Science

McMaster University MASTER OF APPLIED SCIENCE (2018) Hamilton, Ontario (Machine Learning)

TITLE: An Empirical Look at the Accuracy and Performance of Various Machine Learning Algorithms
AUTHOR: Eric Le Fort, B.ENG. (McMaster University)
SUPERVISOR: Professor Antoine Deza
NUMBER OF PAGES: xiv

Abstract

Acknowledgements

Contents

List of Tables

List of Figures

Introduction

Algorithms

Since a wide range of different machine learning algorithms will be discussed in this paper, it is prudent that those algorithms first be described. This section will go into detail regarding the mathematic mechanics of each of these algorithms. In addition, we will look at some of the known strengths and weaknesses of each algorithm.

Kernel Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction technique that uses an orthogonal transformation to convert a dataset's original features into new ones which are linearly uncorrelated [2]. These new features follow orthogonal vectors called principal components which capture the maximum amount of the data's variance. With n features, PCA will provide n principal components. However, by taking just a subset of the components that capture the most variance, a reduction in the problem's dimensionality can be achieved. This selection is often done by choosing as many as necessary in order from most to least captured variance until a desired percentage of the variance is captured. PCA's limitation is that since it only maps along linear axes it can lead to inadequate transformations. Kernel Principal Component Analysis (KPCA) is the answer to that problem. KPCA uses a kernel function to map the original features into the new space in a non-linear fashion [2]. Along with the advantage of being able to represent non-linear relations, KPCA has the added advantage that the extracted features do not depend on the original coordinate system used to represent the data [2].
The following process is described in [2]. The first step in performing KPCA is computing the kernel matrix, K, where k is the kernel function, as:

$$K_{ij} = f_k(x_i, x_j)$$

The following eigenvalue problem where K is the $M \times M$ kernel matrix must then be solved:

$$M\lambda\alpha = K\alpha$$

Once the above is solved, the eigenvector expansion coefficients must be normalized by requiring:

$$\lambda_k(\alpha_k \cdot \alpha_k) = 1$$

Projecting a new point, x, along the principal components can be accomplished by using the eigenvectors, α, as:

$$V^k \cdot \Phi(x) = \sum_{i=1}^{M} \alpha_i^k f_k(x_i, x)$$

K-Nearest Neighbours

K-Nearest Neighbours (k-NN) is a relatively simple classification or regression algorithm. Its technique is to first locate the k closest data points from the training dataset and then use the class of each of those points to determine the class of the unknown data point.
The distance of these points can be calculated using any one of several distance metrics. In this analysis, the chosen distance metric is the Euclidean distance and a weighted version of majority voting is used to predict the unknown class. This algorithm has a few things going for it including being simple to implement, flexible to feature and distance metric choices, handling multi-class problems naturally, and most importantly, it is quite accurate given a large, representative training set [CITATION-NEEDED]. However, this a lgorithm has some significant flaws as well. Most importantly, since this is a lazy algorithm (work is only performed when an unknown data point is to be classified), it can be resource intensive in deployment. Every time a new point is to be classified, there is a large search problem through the full training set which must be solved to find the nearest neighbours. Another difficulty of this algorithm is selecting a meaningful distance function[CITATION-NEEDED]. It is possible to mislead the algorithm if the function does not accurately reflect the similarity between points[CITATION-NEEDED].
Given a classified point p, and an unknown point u with n features, the euclidean distance between them is calculated as[CITATION-NEEDED]:

$$d(p,u) = \sqrt{\sum_{i}^{n}(u_i - p_i)^2}$$

Given a set of k neighbours, x, and distances to those neighbours, d, we can compute the vote for class i, v_i as[CITATION-NEEDED]:

$$v_i = \sum_{j=1}^{k} \frac{d_j}{d_{max}} \left[x_j \in class\ i\right]$$

Note that the square brackets are denoting the Iverson bracket. This term goes to 1 if the contents are true or to 0 if the contents are false.

Decision Tree Learning

Decision tree learning is another regression or classification algorithm. Starting with a root node containing the full training dataset, the learning phase of this algorithm works to decide how to split the data based on the value of one of the features. These splits are meant to homogenize the child nodes' data as much as possible until a stopping condition is reached. The algorithm determines the class for each leaf node as the mean class of its members[CITATION-NEEDED]. The two main splitting criteria to choose from are the Gini Impurity criterion or the Information Gain criterion. As discussed in [1], both criteria are often

equally accurate and so the selection shall be based on computational complexity. In this work the Gini Impurity criterion was selected since it only requires squaring a term compared to computing a logarithm for Information Gain.
The last step to consider is pruning. Since decision tree learning is a greedy algorithm, it is possible that it made splits that are not necessary in the larger picture. In simple terms, the process of pruning involves analyzing the leaf nodes and removing any that provide negative returns when seen from the root.
One of the largest benefits of this algorithm is the ease of comprehension of its models – even a human could make a prediction using the final tree. Other benefits include its resiliency to outliers[CITATION-NEEDED] which warrants less data cleaning, its ability to handle both numeric and categoric values naturally, and having the freedom of being non-parametric[CITATION-NEEDED].
On the other hand, the two biggest shortcomings of this algorithm is its tendency to overfit[CITATION-NEEDED] and not being suitable for continuous variables[CITATION-NEEDED]. Luckily, there are some tricks which can help avoid the former such as limiting the maximum depth of the tree or the minimum subpopulation size to consider splitting a node.
The Gini Impurity for a dataset, $I_G(p)$, of J classes where p_i is the percentage of points with class i in the dataset is calculated as[CITATION-NEEDED]:

$$I_G(p) = 1 - \sum_{i=1}^{J} p_i^2$$

The split with the lowest Gini Impurity will be chosen.

Random Forest Learning

Random forest learning is an ensemble learning method. During training, many decision trees are constructed, each with random subsets of the feature space. An unknown data point is classified by classifying the point using each decision tree and then taking the majority vote.[CITATION-NEEDED]
Random Forest inherits most of the benefits of decision trees at the cost of losing much of the ease of comprehension associated with singular decision trees. Another significant advantage is its ability to handle very large, high-dimensional datasets[CITATION-NEEDED]. This study will mainly focused on supervised learning, however it is important to note that this algorithm is also useful for unsupervised learning. Other than the loss of comprehension, another drawback is that the time to train random forests is significantly longer than singular decision trees.

Logistic Regression

Logistic Regression is a binary classification algorithm. This algorithm starts from random initial parameters and then using the loss function defined below it works to continually decrease its error rate. This allows the algorithm to learn the parameters which best separate the dataset.

One of the biggest benefits of logistic regression is its ability to provide prediction with usable likelihoods[CITATION-NEEDED]. This can be a huge help whenever knowing the model's confidence for a particular result is useful (which is often the case). However this model struggles with large feature spaces and cannot solve for nonlinearities[CITATION-NEEDED]. These issues can be assuaged by performing data transformation techniques at the cost of added development and training time.
The hypothesis, $h_\theta(x)$, with θ as model parameters computes the likelihood of a positive outcome given a certain data point, x[CITATION-NEEDED]:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

The parameters can be optimized by first defining a cost function, $C(\theta)$, and then using its derivative to perform gradient descent. This is shown below where m is the size of the training set and $y \in \{0, 1\}$[CITATION-NEEDED]:

$$C(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right]$$

$$\frac{\partial}{\partial\theta} C(\theta) = \frac{1}{m}\left[\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}\right]$$

Gradient Descent is a minimization algorithm that will update our model parameters, θ, according to our step size, α, every iteration as[CITATION-NEEDED]:

$$\theta = \theta - \alpha \cdot \frac{\partial}{\partial\theta} C(\theta)$$

Training is terminated when some stopping criteria is reached.

Naive Bayes Classifiers

Naive Bayes Classifiers are a type of multi-class classification algorithm based on Bayes' theorem (described below) with an assumption of independence among features. The assumption of independence is the source of the term "Naive" in its title[CITATION-NEEDED]. Since this algorithm typically depends on bucketing, it is often used with discrete features although with some work it can also handle continuous ones. This is achieved by replacing raw probabilities by modelling Probability Density Functions (PDFs) based on the appropriate type of distribution for the feature[CITATION-NEEDED]. Another simpler alternative is simply grouping the feature into discrete buckets and using those buckets for the analysis.
If the feature is discrete, the next step is creating a frequency table which counts the occurrences of each class for all possible values of the feature. From this table one can go further and create a likelihood table for each feature which computes the probability of positive outcomes given the value of the feature

from the frequency table. Lastly, the total probability of each class for the full dataset must also be calculated. With all this in place, Bayes' theorem allows the algorithm to determine the likelihood of a new data point being a member of a certain class[CITATION-NEEDED].
The Naive Bayes Algorithm has a couple main benefits that make it extremely useful in many practical applications. Specifically, its natural handling of multi-class problems and speed of prediction which can make it a great choice for real-time systems. On the other hand, Naive Bayes also has two potential flaws. Specifically, its reliance on the validity of the assumption that there is independence between features and being a "bad estimator"[CITATION-NEEDED]. The later just means that the actual probabilities it outputs should not be taken too literally. If these pitfalls are carefully considered before implementation Naive Bayes can still be a very strong algorithm.
Given a class, y, and a feature, x, Bayes' Theorem is defined as[CITATION-NEEDED]:

$$\Pr(y|x) = \frac{\Pr(x|y) \cdot \Pr(y)}{\Pr(x)}$$

Classification is performed using the following formula where y^* is the predicted class, C is the set of possible classes, and n is the number of features[CITATION-NEEDED]:

$$y^* = \underset{y \in C}{\operatorname{argmax}} \ \Pr(y) \cdot \prod_{i=1}^{n} \Pr(x_i|y)$$

Support Vector Machines

Support Vector Machines (SVMs) are a type of binary classification algorithm. This algorithm searches for a separating hyperplane which optimally separates the two classes of points. The optimal conditions for this algorithm is to maximize both the accuracy of the separation as well as the distance from the separating hyperplane to each of the two sets of points[CITATION-NEEDED]. SVMs can also be extended using kernels to create non-linear separating hyperplanes[CITATION-NEEDED]. The points from each of the sets that are closest to the hyperplane are known as the support vectors.
Since these support vectors are all that is necessary to store to define the trained model, this algorithm is extremely memory efficient. SVMs are also effective in very high-dimensional spaces and work well if the dataset has clear separation[CITATION-NEEDED]. On the other hand, this method is very susceptible to noise as well as being quite slow to train[CITATION-NEEDED]. Also, SVMs do not directly provide likelihood estimates which could be a deal breaker for certain applications.
SVM relies on minimizing the following equation with n data points where x is the set of training points with corresponding targets $y \in \{-1, 1\}$, λ is a parameter which determines the tradeoff between maximizing the margin size and the purity of separation between the two sets, w is the normal vector to the separating hyperplane, and the parameter $\frac{b}{||w||}$ determines the offset of the hyperplane

from the origin along w[CITATION-NEEDED]:

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(w \cdot x_i - b))\right] + \lambda||w||^2$$

Using this algorithm, one can use sub-gradient descent algorithms to solve directly[CITATION-NEEDED] but the implementation that will be used for this analysis first solves the Lagrangian dual using the separation constraints. The simplified problem then becomes solving the following optimization problem[CITATION-NEEDED]:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=2}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i (x_i \cdot x_j) y_j c_j \\ \text{subject to} \quad & \sum_{i=1}^{n} c_i y_i = 0 \\ & 0 \leq c_i \leq \frac{1}{2n\lambda} \end{aligned}$$

where c_i is defined such that:

$$w = \sum_{i=1}^{n} c_i y_i x_i$$

The decision function is then[CITATION-NEEDED]:

$$\operatorname{sgn}(w \cdot x - b)$$

Artificial Neural Networks

Artificial Neural Networks (ANNs) are a regression or classification model inspired by biological neural networks. There are numerous variations of these types of systems but this analysis will deal with a relatively simple variant: multilayer perceptron (MLP). The first step in this algorithm is defining the structure of the network. This includes the number of hidden layers, the number of nodes in each layer and the activation function associated with each of the nodes. Once the architecture has been defined, it is common to initialize the weights of the connections randomly before entering the training phase[CITATION-NEEDED]. Training involves feeding training data forward through the network (forward-propagation) in batches and then performing backward-propagation[CITATION-NEEDED]. The backward-propagation step updates the weights of the connections in order to minimize the error in relation to the previous batch that was fed through the network[CITATION-NEEDED]. Several hyper-parameters are defined which define the process of training and can affect everything from the time to train to how accurate the final model is. These hyper-parameters include the rate of training, the size of the batches, momentum, and more[CITATION-NEEDED]. Once training is complete, predictions are performed using the result of forward-propagating the features of

the unknown sample through the trained model.
It is fairly well-known that there are many persuasive arguments in favour of ANNs including their impressive track record of solving a variety of problems in industry. They have also been extensively researched over the past several years and there are various neural network libraries available in a variety of languages. On the other hand, they also have some frequently overlooked flaws. For one, the training process necessitates tuning several problem-specific hyper-parameters in a trial-and-error process before good models are obtained. The length of training itself can also be quite extensive and the cost of computing resources can become a non-trivial consideration. ANNs are also not probabilistic models and so the confidence of predictions is not directly available[CITATION-NEEDED]. Further, although there has been a significant push in this area, ANNs are still largely a black box which leads to difficulties in predicting whether the model will generalize well.
ANNs have an input layer of which the values of the nodes are defined by the input into the model. The inputs to the other layers are computed using the propagation function, $p_j(t)$, where i iterates over the set of nodes connected to node j, o_i is the output from node i, w_{ij} is the weight of the connection between node i and node j, and t is the current time step[CITATION-NEEDED]:

$$p_j(t) = \sum_i o_i(t) w_{ij}$$

The output from node i is computed using some output function, f_{out}, and the activation of the node, $a_i(t)$, as[CITATION-NEEDED]:

$$o_i = f_{out}(a_i(t))$$

The activation of node i at $t+1$ is computed using an activation function, f, which takes in the node's current activation, a_i, a constant activation threshold, θ_i, and the propagation, $p_i(t)$, as[CITATION-NEEDED]:

$$a_i(t+1) = f(a_i(t), p_i(t), \theta_i)$$

Backward-propagation is achieved by performing Stochastic Gradient Descent on the model's loss function. In particular, where α is the learning rate, C is the loss function, and $\xi(t)$ is a stochastic term, the new weights can be computed as[CITATION-NEEDED]:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha \frac{\partial C}{\partial w_{ij}} + \xi(t)$$

The result of $\frac{\partial C}{\partial w_{ij}}$ depends on the activation function at that particular node.

Method

This section addresses important details about the implementation of the algorithms being compared as well as the data these models are operating on.

A description of these details is crucial in order to fully illustrate the fairness and therefore the usability of the final conclusions. Specifically, this section will define the initial problem statement, general traits of the data (supporting software documentation found in the [BLANK] provides more in-depth descriptions), preparatory data transformations performed for each algorithm if there were any, how the algorithm's parameters were fine-tuned, and any other important implementation details.

Definition of the Problem

The research discussed in this paper revolves around a real-world problem. This

Nature of the Data

Algorithm Implementation Details

Results and Analysis

Conclusions and Future Work

References

[1] Laura Elena Raileanu and Kilian Stoffel. “Theoretical comparison between the Gini Index and Information Gain criteria”. In: *Annals of Mathematics and Artificial Intelligence* (2004). URL: `unine.ch/files/live/sites/imi/files/shared/documents/papers/Gini\_index\_fulltext.pdf`.

[2] Bernhard Scholköpf and Alexander Smola. “Nonlinear Component Analysis as a Kernel Eigenvalue Problem”. In: *Neural Computation* (1998). URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.143.2441\&rep=rep1\&type=pdf`.