Param Singh
DATA698 – Analytics Masters Research Project
Fall 2017

Comparison of Classifier Methods

Introduction

Classification is generally understood to be the act of grouping objects, places, or people into subsets, with or without overlap. It is a problem that the human brain has learned to solve from infancy, and continues to learn and improve as we age. Its impact to humanity has been tremendous even when we don't realize we are using it at all. The method of this learning, via the brain's cortex and neural pathways, has proven its sophistication as well as its drawbacks.

Statistics (i.e. Frequentist and Bayesian) has provided Mathematicians and researchers with techniques that allow us to classify various types of data in both supervised and unsupervised capacities. A technique that has in past years gained traction, and is growing increasingly more accurate at image and text classification than traditional statistical methods is Artificial Neural Networks (A.N.N). In this paper, I shall compare the implementation and performance of Statistical classifiers with Artificial Neural Networks. The intent of the results and conclusions of this paper is to identify where some methods are more applicable than others, resulting in greater analytical efficiencies when attempting new classification problems.

Statistical Classification Algorithms

Logistic Regression

Logistic Regression is a popular classification algorithm that can be used for both binary and multiclass categorization problems. In a binary classification between a positive class and a negative class, the algorithm works by computing a weighted sum of input features, creating an estimated probability, and then categorizing into one of the two classes based on that result.

In a multiclass categorization, where the outputs are categorical and can be used to identify multiple labels (three or more), its implementation is essentially a set of independent binary regressions, or a "one-versus-everything" type of approach. This is also known as a Multinomial Logistic Regression, or a Softmax Regression. The multinomial model assumes that there are no perfect predictions for any one case or categorization, and there is no need for features to be completely independent from each other. Whereas in the case of an algorithm like a Naïve Bayes classifier, having the features be independent is required.

A closer look at how binary categorization works is by comparing the results of the estimated probability $\hat{p}$. If $\hat{p}$ is fifty percent or greater it attaches a positive classification to the output, if below fifty percent it attaches a negative classification to the output.

$$\hat{p} = h_\theta(x) = \sigma(\theta^t \cdot x)$$

Logistic Regression differs from a Linear Regression in that it outputs the logistic of the result as opposed to a direct result. It achieves this using the Logistic sigmoid:

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

Once the model estimates the probability $\hat{p} = h_\theta(x)$, it uses the following to decision function to determine its predicted categorization:

$$\hat{y} = \begin{cases} 0 \; if \; \hat{p} < 0.5 \\ 1 \; if \; \hat{p} \geq 0.5 \end{cases}$$

As it stands, Logistic Regression is a proven, versatile linear classifier capable of handling both binary and multiclass categorization.

Support Vector Machine

Support Vector Machine is a popular classifier method that can be used in a number of applications including linear and non-linear classification. It can additionally be used for regression with output that includes predicted values. It is suited for complex, high-dimensional datasets from a small to medium volume.

At a high-level, Support Vector Machine works by having each observation represented in dimensional space. In simple examples where there are limited number of dimensions this can be plotted for visualization.

The algorithm separates the observations into categories using hyperplanes. These hyperplanes are selected by determining the amount of margin, the effective space, between the categorized observations. Margin can be considered to be "hard" or "soft", and is configured using hyper-parameters with most implementations. Hard margin is used if the data is linearly separable, and does not have outlier values. Soft margin allows you to strike a balance between the margin size and the limits on margin violations.

There is significant depth and complexity with Support Vector Machines, much of which warrants its own research that is out of the scope of this paper, but for its purpose in comparison of classifiers its implementation uses a decision function in a similar manner as Logistic Regression:

$$\hat{y} = \begin{cases} 0, & if\ \boldsymbol{w}^T \cdot x + b < 0 \\ 1, & if\ \boldsymbol{w}^T \cdot x + b \geq 0 \end{cases}$$

Where w represents a vector of feature weights, and b is a bias term. The decision function can be thought of as an intersecting plane where the function is equal to 0 with bounds that are parallel to the decision function that form the margin.

In a linear Support Vector Machine, the training objective is to find the optimum values of w and b that either make the margin as wide as possible while avoiding margin violations, or by limiting the margins. This is an important concept of how Support Vector Machine works, and is one that you will also see applied in a similar fashion with Artificial Neural Networks.

Support Vector Machines are adept with datasets that are in high-dimensional spaces, and is known to be versatile with a wide domain of classification problems. It's known limits are that it does not directly provide probability estimates (a cross-fold can be leveraged in that case), and can have overfitting issues if the number of features is greater than the samples in the sample set.

Stochastic Gradient Descent

Stochastic Gradient Descent is part of the family of gradient descent algorithms, most common are mini-batch and batch gradient descent algorithms. Stochastic Gradient Descent is popular due to its speed, relative to the other descent algorithms such as Full-Batch and Mini-Batch Gradient Descent.

Gradient Descent is an optimization algorithm that is used to find the minimum of a function. Because of the nature of gradient descent for finding minima, it is versatile and can be applied to numerous optimization problems. The algorithm works by adjusting parameters over iterations to minimize a cost function. It does this by measuring the local gradient of the error function, with respect to the feature vectors and bias term.

The algorithm can be initialized by random selection of features, termed random initialization, and then improved upon each step or iteration. The learning rate that is used as a hyperparameter to determine how large each step should be is an important concept with Gradient Descent. Too large and the gradient will diverge (exploding gradient), too small and training time will increase or may not converge.

$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n$$

In this formulation, $Q_i(w)$ represents the loss function at the i-th sample and n as the step size or learning rate taken during descent. In the case of Stochastic Gradient Descent the algorithm uses a single sample.

The Stochastic Gradient Descent differs from other Gradient Descent approaches such as Full-Batch and Mini-Batch, because instead of using the entire training set or batch-subsets to calculate gradients at each step, it picks a random set of features in the training set and calculates the gradients only on that set of features. The clear pro with this approach is its speed, the tradeoff is that its initial iterations will not be very smooth and will only slow towards the end or bottom of the descent.

The approach of minimizing a cost function iteratively over time is an important concept that is leveraged in another classifier algorithm – Artificial Neural Networks.

Artificial Neural Networks

What are Artificial Neural Networks?

Nature has inspired many innovations in science and technology, including Artificial Neural Networks. Surprisingly, the idea of Artificial Neural Networks have been around longer than

most think, the following table list some significant milestones and papers in the development of the concept and methods of contemporary neural networks.

Significant Historical Research

1958 – Rosenblatt proposed Perceptrons (Rosenblatt, 1958, 1962)

1986 – Early Backpropagation Network (Rumelhart et al., 1986b)

1996 – Sigmoid Belief Network (Saul et al., 1986b)

1998 – LeNet-5 (LeCun et al., 1998b)

2006 – Deep belief network (Hinton et al., 2006)

2009 – Deep Boltazmann machine (Salakhutdinov and Hinton, 2009a)

2014 – GoogleLeNet (Szegedy et al., 2014a)

Artificial Neural Networks are modeled after biological neurons, comprised of simple building blocks of cells, dendrites, synapses, and axons that are formed into a larger network containing billions of these neurons and their connections. Artificial neurons are modeled and organized in a similar fashion; a neuron can have one or more binary inputs and one binary output.
In the case of the Artificial Neural Network, the dendrite or connection to a neuron has a weight associated to it. It acts like a biological synapse - transmitting a signal from an input cell or a preceding layer in the neural network. The neuron acts like nucleus or signal processor, taking that input, executing a function, and then passing it along to an output, another layer, or backwards, or to another layer in the network for convolutional and recurrent networks.

Machine Learning can be generally categorized into supervised, unsupervised, and reinforcement learning techniques. Neural networks are considered a form of reinforcement learning where the

ANN learns over time (epochs) and the model is improved iteratively. The mechanics of this iterative, reinforced learning model are two movements: Forward-pass or forward propagation form the input layer to the output layer, and Backpropagation where we have the results and step backward through the layers and update the weights. These operations in combination with learning techniques are the method used to train an Artificial Neural Network.

Forward Propagation

Forward propagation generally requires that inputs to a neural network be processed through a computational "hidden" layer, like a tensor with an activation function, such as a Logistic Sigmoid, Softmax, or Rectified Linear Unit (ReLu). Those outputs are then summed and fed to an output layer where they are again processed to produce a result. In the context of classification, this would produce a label or categorization. When training a model, the algorithm determines the error amount and then proceeds with the Backpropagation step to adjust weights and attempt to minimize the error.

If the processing operation seems familiar it is the same technique employed by the Support Vector Machine algorithm: $\boldsymbol{w}^T \cdot x + b$

Backpropagation

At a high level, backpropagation involves taking the values of the output from the forward pass, computing a cost, and then flowing backward through the network to compute the gradient (a cost function with respect to the network parameters, weights).

Computing the gradient can often be computationally expensive, but backpropagation accomplishes this by using the Chain Rule of Calculus in a recursive manner to the error function partial derivative:

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ij}^k}$$

Where $\frac{\partial E}{\partial a_j^k}$ represents the error, and the second term is computed from the activation function of a given node.

Strictly speaking, backpropagation is the method used to compute the gradient, but another algorithm – for example, Stochastic Gradient Descent – is used to perform the actual learning using the gradient. Similar to Stochastic Gradient Descent, a learning rate, or step value is used over each iteration to minimize the error.

Perceptrons

Perceptrons were a fundamental innovation, and it is the first model that can be classified as neural network. Relative to contemporary neural network architectures, it is somewhat simple in its design.

It is constructed out of input tensors, or nodes, with a single hidden layer, and an output layer. The neurons in the network are considered to be fully connected, and can also contain a bias neuron.

It is trained by using Hebb's rule where in biological neurons if a neuron triggers another neuron, the link or connection between those two neurons grows stronger. In computational terms for the

Perceptron, the connection weight between two neurons grows stronger when they have the same output.

In implementation, the neural network is fed one training instance at a time and creates predictions. For the neurons that had increased correct predictions, the algorithm reinforces or increases that connection. The following illustrates the Perceptron learning rule:

$$w_{i,j} = w_{i,j} + \eta(y_{j-} \hat{y}_j)x_i$$

The connection weight between the ith input neuron and jth output neuron($w_{i,j}$) is summed with the η - the learning rate, multiplied by the difference in the actual output of the neuron and its target output, multiplied by the ith value of the current training instance.

Perceptrons are powerful, but have some drawbacks in the same vein as other linear classification methods like Logistic Regression (e.g. cannot perform Exclusive-or, XOR, classification). Researchers found that by stacking Perceptrons, forming a Multilayer Perceptron, solves some of these shortcomings.

Multilayer Perceptron

Similar to a perceptron, a multilayer perceptron has an input layer, and output layer. Where it is different than a Perceptron is that it has one or more layers between the input and output layers called hidden layers.

A hidden layer, or linear threshold unit (LTU) computes a linear combination of its inputs and if the result exceeds a threshold it outputs a positive class, otherwise its output will be a negative

class. When a neural network has more than two hidden layers, it is considered a deep neural network.

One advantage of the multilayer perceptron model over its predecessor is that it its output layer can be a linear unit, and not just logistic. This allows the network to solve problems such as linear regression or estimation.

It is the combination of Forward Propagation, Backpropagation, and learning techniques that make Neural Networks a powerful method for classification or estimation. Some of the downsides are the amounts of architecture and design considerations that are a natural part of using a Neural Network, for example structure of the network, the number of hidden neurons, activation functions, and step size or learning rate configuration. Subtle changes in these parameters can often lead to issues such as vanishing gradients or exploding gradients which are often difficult to troubleshoot.

Dataset 1 - MNIST

The MNIST dataset is commonly used for evaluating the performance of image classification algorithms. It contains labeled observations of handwritten digits, values zero through 9, created by high school students and employees of the US Census Bureau. It contains 60,000 training images and 10,000 test images; each image is labeled with the digit it represents.

Each image is 28x28 pixels, which results in 784 features, a feature for each individual pixel. A feature represents the intensity of the pixel from 0-255. The images presented are black (0) with

increased intensity going up to white (255). This may seem like the image is inversed but from the perspective of a machine learning algorithm it simply indicates the absence, or presence and intensity of data.

The MNIST images are in grayscale, however in other datasets the data for color images are represented in RGB values as additional feature-values.

This dataset is interesting from a machine learning perspective because of the variations that can be observed with some digits. For example, a "7" can sometimes be confused with a "1" or a "2", a "5" with a "6", or a "3" with a "8". A representative sample of these variations are included in the MNIST dataset. It is observed that even the best models produced by Google and Yann LeCun that can outperform humans by a narrow margin (> 94% accuracy), also have trouble with observations where there is variability of how the images are written.

It can be postulated that with additional scrutiny with labeling of training datasets, either as a secondary pass requiring manual inspection (a la Amazon's Mechanical Turk efforts), and/or an additional dimension to the data with a probability or weighting associated to known variances, it could yield additional gains with recognition accuracy.

Despite the nature of handwritten digits and its variability, it is advantageous to leverage this dataset to provide a baseline against other research and performance due to its wide adoption rate.

Dataset 2 – 20 Newsgroups Dataset

The 20 newsgroups dataset contains approximately 18000 newsgroup posts on 20 categories, or topics. Like MNIST, this dataset is often used as a common evaluation dataset , but for text classification problems.

This dataset is interesting from a machine learning perspective because it contains sparse features. Sparse features, implemented as sparse matrices, are efficient with datasets with a large number of categories. Instead of storing large amounts of dimensional space with zeros in a dense format, a sparse matrix instead stores the location of the non-zero values. This format is commonly used for features associated to a large corpus of words and provides efficiencies when working with a large number of categories.

Note that use of text data by a classifier algorithm requires that the data be expressed in vector format, or "vectorized" to have feature vectors available as input. There are various methods and implementations available to perform vectorization of text data (e.g. tf-idf, bag of words, word2vec), but for simplicity and working with a canonical dataset that is a known quantity for baseline text classification I have leveraged the already-vectorized form of this dataset available through the sci-kit learn package. The vectorizer used for this dataset is tf-idf (term-frequency-inverse document frequency).

Performance Measures

There are a variety of performance measures used in machine learning, some are generic, others can be used for specific algorithm types (regression, or classification), and some can be algorithm-family specific. The following performance measures are suited for both multi-label

classification and are implemented canonically across all of algorithm being evaluated in this paper.

Accuracy

Accuracy measures the closeness of predictions to their target values. It is defined as the sum of the true positive (TP), divided by the sum of the true negative (TN), true positive (TP), false negative (FN), and false positive (FP).

$$accuracy = \frac{TN + TP}{TN + TP + FN + FP}$$

Precision

Precision can be considered to be the accuracy of positive predictions. It is defined as true positive (TP), divided by the sum of true positive (TP) and false positive (FP) values:

$$precision = \frac{TP}{TP + FP}$$

Recall/Sensitivity

Recall or Sensitivity can be considered to be the ratio of positive instances that are correctly identified by the classifier. It is defined as the true positive (TP), divided by the sum of the true positive (TP) and false negative (FN) values.

$$recall = \frac{TP}{TP + FN}$$

Precision and Recall/Sensitivity are often paired together as performance measures as precision on its own can be used to paint a false analysis. For example, a true positive prediction over one test value, a 1 out of 1, would result in a precision of 100% Recall/Sensitivity provides you with

a ratio of the positive instances that are correctly identified by a model. However, if we increase the instances fed to the model to increase Recall, we end up reducing Precision. This is considered the Precision-Recall tradeoff.

F1-score

The F1 score can be considered a combination of the Precision and Recall metrics.

$$F1 = \frac{TP}{TP + \frac{FN + FP}{2}}$$

It works by providing a harmonic mean of precision and recall. Where regular mean treats all values equally, harmonic mean applies more weight to lower values. As a result of its formulation, you typically get a high F1 value when both the Precision and Recall are high for a model.

Experiment 1 Classifier Comparison – MNIST

Utilizing the classifiers mentioned above: Multinomial Logistic Regression, Support Vector Machine, Stochastic Gradient Descent, Perceptron, and Multilayer Perceptron I trained a model for each using a training subset of the MNIST dataset. As an additional comparison, I also implemented a sample Google TensorFlow Deep Neural Network. Following this I created predictions and captured the following performance measures for each:

Table 1 - Results of Experiment 1

	Accuracy	**Precision**	**Recall**	**F1-Score**
Multinomial Logistic Regression	91.6%	92%	92%	92%
Stochastic Gradient Descent	88.9%	89%	89%	89%
Support Vector Machine	88%	88%	88%	88%
Perceptron	89%	89%	89%	89%
Multilayer Perceptron	92.6%	93%	93%	93%
TensorFlow Deep Neural Network	92.1%			
TensorFlow Convolutional Neural Network[1]	99.2%			

Analyzing the results, Neural Networks (Multilayer Perceptron, Deep Neural Network, and Convolutional Neural Network) outperformed all of the other classifiers; the Convolutional Neural Network vastly outperformed all others.

Due to its nature, the single Perceptron model ran into some difficulty where the data was not linearly separable. Stochastic Gradient Descent and Support Vector Machine had similar performance due likely to the same issue. Some adjustments to hyperparameter settings may yield incremental results, but likely would not be able to break the threshold set by the Multinomial Logistic Regression model.

The performance of the Multinomial Logistic Regression model can likely be attributed to the simplicity of the algorithm, and the dataset. The MNIST images are in grayscale and by their nature somewhat binary, pixels are activated or not with varying shades of grey for intensity. In a one-vs-rest approach the algorithm will systematically apply its decision function against the

[1] Due to implementation issues and computational limitations, this model was ultimately not included in the accompanying iPython notebook. The reference implementation included on the TensorFlow website provided the accuracy reading used in this table (https://www.tensorflow.org/get_started/mnist/pros)

thresholds set by the rest of the labels. It is not the most efficient computation, but this method along with the dataset are likely the reasons for its success.

With a dataset that contains for example color images of numbers with varying sets of contrast Multinomial Logistic Regression would likely fall behind an algorithm such as Support Vector Machine that is more efficient in high-dimensional space.

Experiment 2 Classifier Comparison – 20 Newsgroups Dataset

Taking the same approach with Experiment 1, I trained an almost identical set of algorithms with the 20 Newsgroups dataset. Google's TensorFlow framework has some support for sparse matrices, but I was ultimately unsuccessful with getting a similar Deep Neural Network implement successfully using the "vectorized" form of the 20 Newsgroups Dataset.

Table 2 - Results of Experiment 2

	Accuracy	**Precision**	**Recall**	**F1-Score**
Multinomial Logistic Regression	82%	82%	82%	82%
Stochastic Gradient Descent	87.2%	88%	87%	87%
Support Vector Machine	89.4%	90%	89%	89%
Perceptron	98%	88%	86%	87%
Multilayer Perceptron	37%	45%	38%	34%

The results of this experiment have more variation than in the first experiment. Most surprising are the results of the Multilayer Perceptron. In the first experiment it performed well, using the same hyperparameters in this experiment however, the Stochastic Optimizer was unable to converge after 200 iterations. This means that the algorithm was unable to minimize the loss, which caused the poor results observed. It could be argued that additional steps, set by the

learning rate hyperparameter, may allow it to converge eventually, but it is not a guarantee of success.

Initial observations may seem like the Perceptron model performed well just based on accuracy, not the case when we look at its precision, recall, and F1 score. This is likely due to an accuracy paradox where the true negative may be less than the false negative instances.

Multinomial Logistic Regression came in on the lower end of performance, this likely has to do with the complexity of the data – both in form, sparse features, and in additional categories relative to the first experiment.

Stochastic Gradient Descent performed admirably, but the stochastic nature of its implementation may be attributed to its performance measures. It can be postulated that a Full-Batch or Mini-Batch Gradient Descent approach might fare somewhat better.

Support Vector Machine performed well relative to the other algorithms in this experiment. Its nature of working well with high-dimensions and with multiple classes are likely attributing factors to its success in this experiment.

Conclusions

In this paper, I presented an overview of Statistical Classifiers and Artificial Neural Networks, compared their implementations, and analyzed their results in two experiments. The datasets used were structurally and categorically different from one another to ensure that no one algorithm had a distinct advantage over another.

It can be concluded that Artificial Neural Networks did generally outperform the other methods in both experiments. In Experiment 1, not factoring the TensorFlow implementations, the Multilayer Perceptron – an Artificial Neural Network – outperformed the other models. In Experiment 2 the Perceptron performed well, albeit with some degree of accuracy paradox. If the

Multilayer Perceptron did eventually converge, I posit that greater performance gains would be eventually be recorded. Additionally, if a larger dataset was evaluated, the Support Vector Machine model would likely not perform as well as it did in this experiment.

However, this does not definitively conclude that Artificial Neural Networks will always outperform Statistical Classifiers. One vector not explored as a performance measure in this paper, is computational cost. The effort of implementing and executing the Neural Network models was significantly higher relative to all of the other classifiers – to the point where it was no longer feasible executing on a personal computer. Additionally, due to their complexity, it is often difficult to determine what specific hyperparameters need to be adjusted, and by what values. The optimal number of hidden layers to use in a Neural Network model will often strike up vigorous debate with practitioners, and from the heads of research is still an area that is open to interpretation.

There is no denying that the future of machine learning has Artificial Neural Networks at its forefront. Computational costs in the cloud are being reduced year-after-year, implementation frameworks are less disparate and converging to standardized offerings in the forms of Caffe2, Theano, and TensorFlow, and Graphics Processing Unit (GPU) hardware acceleration is becoming commoditized. These factors will eventually ensure wider adoption and implementation of Neural Networks.

Citations and References

Texts

Deep Learning by Goodfellow, Benigo, and Courville. ISBN 9780262035613

Hands-On Machine Learning with Scikit-Learn and TensorFlow by Aurélien Géron. ISBN 9781491962299

Code and datasets

TensorFlow.org

Scikit-learn.org

https://github.com/paramrsingh/MS-Project/