Hand Written Digit Identifier

Machine Learning based Hand Written Digit Identifier

August 21, 2016

Authors:
Bhaskar Mandapaka

1 Introduction

Machine Learning is a subfield of computer science that evolved from the study of pattern recogition and computational learning theory in artifical intelligence. Machine Learning explores the study and construction of algorithms that can learn from and make predictions of data. These algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions.
An Artificial Neural Network is a part of Machine learning, a Neural Network is a network inspired by biological neural networks which are used to estimate or approximate functions.
Example applications include spam filtering, optical character recognition, hand writing recognition, search engines and computer vision.

2 Dataset and Implementation

In this paper the MNIST dataset is used. These datasets consists of gray-scale images of hand drawn digits, from zero to nine.
Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.
All the machine learning algorithms were implemented using SKLearn package of python and the neural network was implemented using Tensorflow.
The model can be saved and used at a later time using this.
Program to save the model.

```
import from sklearn.externals import joblib
#..
#..
joblib.dump(classifier, 'filename.pkl')
```

These lines save the model and can be used at a later stage for predicting results.
Program to use the saved model.

```
import from sklearn.externals import joblib
#..
#..
```

```
classifier = joblib.load('filename.pkl')
```

3 Cross Validation

Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice.
A model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (testing dataset). The goal of cross validation is to define a dataset to "test" the model in the training phase.
One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set).
In this project crossvalidation is used to validate the models.
The code for this can be found in "sklearnMLCommon.py"

4 Algorithms

Random Forest

Random forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.
In this project Random Forests was implemented using the 'sklearn' library of python.
The source code for implementing Random Forest can be found in "sklearn-RandomForest.py"
Best accuracy was achieved when there were 200 decision trees.
The accuracy achieved was: 0.9583
When timed using the linux time command
Time taken for execution: 0m29.773s

Extremely Randomized Trees

In extremely randomized trees, randomness goes one step further in the way splits are computed. As in random forests, a random subset of candidate features is used, but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule. This usually allows to reduce the variance of the model a bit more, at the expense of a slightly greater increase in bias.
In this project Random Forests was implemented using the 'sklearn' library of python.
The source code for implementing Random Forest can be found in "sklearnExtraTree.py"
Best accuracy was achieved when there were 200 decision trees.
The accuracy achieved was: 0.9652
When timed using the linux time command
Time taken for execution: 0m29.032s

Random Forest and Extremely Randomized Trees can be used for feature extraction. These extracted features can be used to train another classifier to get better results.
Following piece of code returns the features and their importance.

```
#..
classifier.feature_importances_
#..
```

K-Nearest Neighbors(KNN)

The k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space.
k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.
In this project K-Nearest Neighbors was implemented using the 'sklearn' library of python.

The source code for implementing K-Nearest Neighbors can be found in "sklearnKNN.py"
The accuracy achieved was: 0.9667
When timed using the linux time command
Time taken for execution: 2m54.630s

Support Vector Machines(SVM)

Support Vector Machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.
SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier.
In this project Support Vector Machines was implemented using the 'sklearn' library of python.
The source code for implementing Support Vector Machines can be found in "sklearnSVM.py"
The accuracy achieved was: 0.9097
When timed using the linux time command
Time taken for execution: 1m9.233s

Neural Network

Neural Network was implemented using Tensorflow.
A naive implementation Neural Networks using Tensorflow can be found in "tensorflownn.py"
This network has 3 hidden layers with 500 nodes each.
Each digit was considered as a class so 10 classes.
An Epoch is a single pass through the entire training set, followed by testing of the verification set and in this sample there were 15 epochs. The accuracy achieved was: 0.9498
When timed using the linux time command
Time taken for execution: 2m2.384s
The main problem with the above approach was if user wanted to add another layer or tweak some parameters lot of programmatic changes were involved.
Optimized version of the above program can be found at "tensorflownn-opt.py"
A model with 6 hidden layers with 500 nodes in each layer and when exe-

cuted for 30 epochs.
The accuracy achieved was: 0.9639
When timed using the linux time command
Time taken for execution: 16m18.879s

5 Results

The table below shows the results of the most accurate run of algorithms.

Algorithm	Accuracy	Time(s)
Random Forest	0.9583	29.773
Extremely Randomized Trees	0.9652	29.032
K-Nearest Neighbors	0.9667	174.620
Support Vector Machines	0.9097	69.233
Neural Network	0.9639	978.879

The graph showing various algorithms vs their Accuracy and time.

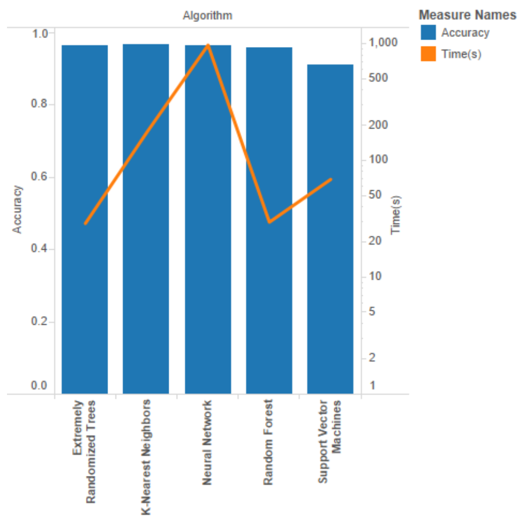

Figure 1: Algorithm vs Accuracy, Time(s)

It can be observed from the above graph and table that Random Forest and Extremely Randomized Trees are very good for this application. Their training time is less and they can predict with a very good accuracy. Exteremely Randomized Tree Algorithm's Accuracy is better than the Neural Network and gets trained very quickly.

6 References

1. https://en.wikipedia.org/wiki/Machine_learning
2. https://en.wikipedia.org/wiki/Artificial_neural_network
3. https://en.wikipedia.org/wiki/Cross-validation_(statistics)
4. https://en.wikipedia.org/wiki/Support_vector_machine
5. https://en.wikipedia.org/wiki/Random_forest
6. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
7. http://scikit-learn.org/stable/documentation.html