APANPS 5335 Final Report: Exploring the Use of Various Machine Learning Methods for Classification of the MNIST Dataset

Highest FOM: 0.038

Matthew Vtha
Columbia Univeresity
School of Professional Studies:
Applied Analytics
New York, NY, United States
mv2705@columbia.edu

***Abstract* — Pursuant to Columbia University's Machine Learning: Concepts and Applications Course (APANPS55335), this paper explores the use of various Machine Learning techniques to perform the supervised learning task of classification on the MNIST dataset. This paper will explore both the rationale behind each method used, and the performance generated from each method. Ultimately, the use of a multi-layered Convolutional Neural Network (CNN) was able to generate the best performing model, as measured by the classification metric of accuracy and in terms of FOM.**

I. Introduction

The use of Machine Learning algorithms to solve the supervised learning task of Classification has been Ill documented, and widely implemented across different industries throughout the world. This paper will explore how different algorithms perform for Classification. Specifically, this paper implements the Kmeans, Logistic Regression, Support Vector Machine, Decision Tree, Random Forest, and Convolutional Neural Network algorithms to solve this task. Two iterations of the Logistic Regression, Support Vector Machine, Decision Tree, and Random Forest algorithms Ire implemented in this paper: The first iteration did not apply dimensionality reduction techniques to the dataset, whereas the second iteration did implement dimensionality reduction techniques. The primary method of dimensionality reduction employed was Princnipal Component Analysis (PCA). All analyses performed for this project Ire completed using the Python programming language.

II. The Dataset

The dataset used for this study was the MNIST Dataset. The dataset consists of 70,000 hand-drawn images of digits numbered from 0-9. In this sense, the problem becomes a multi-class classification problem in which the algorithms Ire evaluated on whether they could correctly predict whether the image of a given number betIen 0-9 was correctly labeled as such. As is common in Machine Learning applications, the dataset is split into a training set test-set. The training set contained 60,000 images, while the test set contained 10,000 images. The training set was used to train the algorithms discussed above, while the test set was used to evaluate the performance or accuracy of the algorithms. In efforts to simulate real-world applications of such algorithms, the paper further tests the algorithms on author-drawn digits to evaluate the performance of the algorithms on data extraneous to the dataset.

III. K-Means Algorithm

A. Background

The first algorithm implemented in this paper was the K-Means algorithm. K-means is a clustering algorithm (generally used for unsupervised learning problems) that takes a predefined, user-inputted number of clusters (in this case, 10, given the 10 digits) and assigns each observation to a particular cluster. The algorithm works by generating cluster centers, or centroids, that are randomly initialized and represent the average of observations per cluster. For each observation, the algorithm computes the sum of squared distance betIen all observations. This distance measure in this implementaiton, and most generally used, is the Euclidean distance measure. For each cluster, after running multiple iterations, the algorithm tries to minimize the distance betIen each obesrvation and its given centroid, updating each centroid based on this minimization. The algorithm stops when there is convergence and the centroids no longer need to be updated.

B. Implementation

The algorithm was implemented using the Sklearn MiniBatchKmeans to help run the model more efficiently. As mentioned previously, the data was fitted to 10 clusters a-priori. To assess the accuracy of how this model clustered images to a centroid that represents the correct label of the image, one must first iterate through the clusters and index the points of the clusters

based on the labels that the Kmeans generated. One can further infer which label the cluster represents by identifying the most common label associated with that cluster.

C. Evaluation

After running the algorithm on both the training and test set, one can observe that the training set has an accuracy of .58, while the test set has an accuracy of .52. These performance metrics not satisfactory and it would not be wise to implement a machine learning application with such a low accuracy. This may in part be due to the fact that Kmeans is a clustering algorithm, that aims to cluster data together. This can be contrasted with a supervised learning algorithm, as those that will subsequently be presented, that are trained to predict classes rather than cluster observations together.

IV. Logistic Regression

To improve upon the performance of the Kmeans model, one can implement a series of supervised learning algorithms to attempt to predict the correct label per image.

A. Background

The Logistic Regression is often regared as one of the simplest, most explainable models that can be used for classification. Logistic regression can be compared to a linear regression intuitevely. The Linear Regression attempts to take observations, and generate coefficients that will minimize the error between a model's predicted value and the actual values from the data observations. The Logistic Regression on the otherhand, uses the sigmoid function to change the output of that model from a value, to a probability between 0 and 1. In such a fashion, the logistic regression maximizes the likelihood of all the training samples.

B. Implementation and Evaluation

The Logisitc Regression model was implemented using the Sklearn package in Python, and using the LFBGS solver to maximize efficiency in run time. The algorithm was further implemented using a C parameter of 1.0, and 100 max iterations. After fitting the model on the training data, the model was evaluated using the test set and generated an accuracy score of .91. Clearly, this model performed much better than the Kmeans algorithm.

V. SVM

A. Background

The SVM (support vector machine) algorithm is a supervised learning algorithm that is able to classify data by generating an optimal hyperplane for linear classification. The SVM algorithm is very useful in that finding the optimal hyperplane to separate the data can be mathematically proven, as opposed to the logsitic regression which essentially approximates the optimal outcome. More sepcifically, the distance from an obersvation to the hyperplane can be calculated and thus the hyperplane can be instered into the data to maximize the distance between the observations and the hyperpalne. This maximization of distance attempts to maximize a distance known as the 'hard margin' in the event that the data is linear. If the data is non-linear, this margin is called a 'soft-margin'.

B. Implementation and Evaluation

The SVM algorithm was implemented using the Sklearn package in Python, specifying a "poly" kernal, and using a grid search to find optimal values of the SVM_C and SVM_Gamma parameters. The C parameter represents the cost of missclassification, whereas the gamma parameter of Gaussian Kernel is used to handle non-linaer classification. Similarly, the "poly" kernel allows for polynomials and non-linearity over the feature space. Given these parameters, the SVM generated an accuracy score of .92 on the test data. It is important to note, that given the computational expense of running an SVM model, the model was only trained on 5000 samples of training data. Even given this small training size, the model still outperformed both the Logistic Regression and Kmeans algorithm.

VI. Decision Tree

A. Background

The Decision Tree algorithm can be used for both classification and regression within supervised learning. The algorithm works by recursively splitting a dataset into different nodes, branches, and leafs, aiming to generate homogenous groups of data based on the label of each observation. The performance of splitting the data can be measured by understanding the purity or impurity of each node. The more impure the node, the less effective the splitting was. Certain measures for such impurity are the Gini Index, and Entropy.

B. Implementation and Evaluation

The algorithm was implemented using the Sklearn package in Python, specifying to measure the impurity of each split by using the Gini index, and having the max depth of tree of 32. Ultimately, the resulting accuracy of this model to predict the label of the MNIST digits on the test dataset was .72. This algorithm performed significantly worse than the Logistic Regression and SVM model. One potential explanation for this performance is that it is difficult to prevent overfitting in Decision Tree models. Moreover, the model tends to perform better with smaller trees rather than larger trees, which may have complicated this model given that there are 784 features per observation.

VII. Random Forest

A. Background

Rather than use one decision tree, the logical next step to enhance poor performance would be to try to combine a larger subset of decision trees to generate a more accurate score. This is precisely what the Random Forest algorithm does. This is referred to as an ensemble method. In such a fashion, once can generate many baseline models and establish higher performing model through plurality voting of the multiple classes output per each model. Random Forest in particular, is a type of bagging algorithm that randomly draws from the training set with replacement when building each different baseline model that is part of the overall ensemble. Due to this random drawing of features, the base models are more independent of each other and theoretically will help the model generalize better onto unseen data.

B. Implementation and Evaluation

This model was implemented using Python's Sklearn's RandomForest Classifier. Similar to the Decision Tree algorithm, Random Forest contains many parameters that can be tuned to generate a higher performing model. For this reason, different values of n_estimators, max_features, max_depth, min_samples_split, and min_sample_leaf were examined using a randomized grid search. Ultimately, the model that performed the best contained an n_estimator parameter of 800, min_samples_split of 5, min_sample_leaf of 1, max_features of 1, and a max_depth of 100. When performed on the test set, the accuracy for the model was .96.

VIII. Algorithm Comparison

A. Comparison Table

The table below provides the advantages and disadvantages of the algorithms referenced above:

TABLE I. Algorithm Comparison

Algorithm	***Advantages***	***Disadvantages***	***MNIST Dataset Application***
Kmeans	- Scales well with larger datasets - Provides convergence of centroids - Generalizes well to new data - Efficient to train	- Have to choose k manually – may not be the optimal choice - Trouble generalizing to data where clusters exhibit wide variability - Trouble dealing with outliers - Difficulty with large feature space	- Low accuracy not surprising given that this algorithm is typically used for clustering not classification - Large feature space complicates ability to distinguish different clusters - Only 1 possible value of K given 10 digits
Logistic Regression	- Performs well with linearly seperable data - Provides coefficient size and direction for interpretation - Efficient to train	- Assumes linearity between target and features - Easy to overfit, especially with larger feature space -	- Given the efficiency of training, this model serves as an ideal baseline - Yet large feature spaces may complicate algorithm - Assumption of linearity poses a problem given the success of more complex models used herein

Algorithm	*Advantages*	*Disadvantages*	*MNIST Dataset Application*
SVM	- Works well given clear separation of classes - Handles high dimensionality well - More efficient than some more complex algorithms - Regularization parameter to help prevent overfitting - Kernel for non-linear data - Mathematically provable	- No probabilistic interpretation of output - Does not handle overlapping classes well - Less efficient than Logistic Regression - Searching for ideal parameters can be computationally expensive	- The SVM makes for a highly appropriate choice for this study given the fact that it handles high dimensional data well, and that there are parameters to prevent overfitting. Similarly, the kernel trick will help to deal with nonlinearity. However, there is some computation expense required to generate optimal parameters.
Decision Trees	- No normalization required - Less preprocessing - Handles null values - Easy to understand given splits	- More computationally expensive than Logistic baseline - Harder to prevent overfitting - Small changes in tree structure lead to worse performance - Biased responses re information gain with more classes	- Not the most ideal model to use given its instability regarding changes in tree structure and outcomes, and inaccuracy given higher number of classes to predict
Random Forest	- Handles high dimensional data very well - Bagging helps model to generalize - Provides feature importance	- Computationally expensive - Many parameters to train	- Great choice for model, but difficult to train given computational expense

IX . Dimensionality Reduction

A. Background

The primary dimensionality reduction technique employed by this paper was Principal Component Analysis (PCA). PCA takes highly dimensional data (i.e. MNIST Dataset with 784 features, one per pixel) and linearly transforms the data into less dimensions. This transformation is done by computing the covariance matrix of a datasets features, and decomposing the matrix into its eigenvectors and eigenvalues. Eigenvalue represents the scalar multiple of the eigenvector. After sorting eigenvectors in descending order with respect to their eigenvalues, the first eigenvector will account for the most variation in the data. In this manner, once can access a fraction of the datasets features by incorporating eigenvectors rather than the original features.

B. Implementation and Evaluation

The above technique was applied to the dataset using the Python's Sklearn package. After running an initial PCA asserting 784 features, one could identify that ~350 eigenvalues comprised 95% of the variability of the data. As such, it was possible to use almost half of the total number of features to run the analysis.

The below table provides a summary of the accuracy generated from the Classifications using principal components rather than features:

Algorithm	**Non-PCA Accuracy**	**PCA Accuracy**
Logistic Regression	0.91	0.92
SVM	0.92	0.74
Decision Tree	0.72	0.66
Random Forest	0.96	0.95

As one can see from the table above, the PCA-Logistic regression performed better than the original logistic regression. As the logistic regression handles a smaller feature space better, this result is not surprising. Similarly, the performance of the Random Forest model with PCA components performed only .01 worse than the non-PCA feature set. The result of the PCA-SVM on the other hand, was surprisingly low, registering at .74.

X. Deep Learning

A. Background

The use of Neural Networks comprised of neurons, activation functions, backpropogation, and in the case of Deep Learning, hidden layers, has become ubiquitous amongst Data Scientists as they attempt to solve problems from Image recognition, to time series prediction, to standard classification problems. In this vein, this paper implements a Convolutional Neural Network (CNN) to predict labels associated with the MNIST Dataset. The basis of neural networks and deep learning consists of a set of inputs, weighted by a set of weights, that are first passed forward through a network of neurons. This forward pass involves the use of an activation function, such as the logistic regression sigmoid function, to squash the net input to generate output from the network. There is then a backwards pass known as backpropogation, where the gradient at each neuron is taken with respect to the weights and biases, to ensure that error is minimized. Error can be measured by SSE, cross entropy, or other customized loss functions.

The CNN alters the traditional workflow of a neural network. CNN's use convolutions, rather than general matrix multiplication in at least one of the layers of the network. A convolution of the function f * g is the integral product of the two functions after one is reversed and shifted. CNN's for image analysis typically possess a convolutional layer first, taking into account input volume size, zero padding around the edges of a picture, and stride or number of pixels to shift by, and depth or number of filters. They then often contain a pooling layer, that reduces dimensionality of the picture by applying an aggregate operation (i.e. Max, Avg, Sum) to a subsample of the data. Next, the network applies an activation function known as ReLU (Rectified Linear Unit) to replace all negative pixel values by 0. Lastly, the network is comprised of a fully-connected layer connected to all activations of the previous volume.

B. Implementation and Evaluation

The network this paper employs first implements a convolutional layer with 32 filters. It subsequently adds a second activation layer using the ReLu activation function. After the activation function, another Convolutional layer is added with 32 filters. After the use of a second activation function, the network uses a Max Pooling layer to to down-sample the input for dimensionality reduction. The network also includes multiple instances of batch normalization to make sure that the scale of each layer stays the same.

This workflow is repeated for additional deep layers, with the exception that in this second iteration of the above network structure, the Convolutional layers use 64 filters rather than 32. After the second iteration mentioned in the last sentence, a fully connected dense layer is added to connect all activations from the previous volume. Within the fully connected layer, a Dropout layer is also used to help reduce overfitting. Lastly, there is a softmax layer to transform the output into a probability distribution. The summary output from the model can be found in the image below:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
batch_normalization_1 (Batch (None, 26, 26, 32)        128
_________________________________________________________________
activation_1 (Activation)    (None, 26, 26, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
batch_normalization_2 (Batch (None, 24, 24, 32)        128
_________________________________________________________________
activation_2 (Activation)    (None, 24, 24, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
batch_normalization_3 (Batch (None, 10, 10, 64)        256
_________________________________________________________________
activation_3 (Activation)    (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 64)          256
_________________________________________________________________
activation_4 (Activation)    (None, 8, 8, 64)          0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_6 (Activation)    (None, 10)                0
=================================================================
Total params: 597,738
Trainable params: 596,330
Non-trainable params: 1,408
```

Moreover, the network also uses Data Augmentation through ImageDataGenerator to help the model generalize on additional data. This data augmentation trains images in batches of 64 at a time.

The network further used categorical cross entropy as the loss function, the Adam optimizer, and accuracy as performance metric.

To assess the performance of this structure, several iterations Ire performed using different training sizes and model parameters. The below table summarizes the results:

CNN Iterations						
Training Size - Images	P1	Test Accuracy P2	FOM	Activation Function	steps_per_epoch	epochs
60,000	1	0.994	0.506	relu	60000//64	5
10,000	0.18	0.9908	0.0992	relu	60000//64	5
6,000	0.1	0.9146	0.1354	relu	6000//64	5
6,000	0.1	0.3792	0.6708	sigmoid	6000//64	5
1000	.016	.97	0.038	relu	60000//64	5

From the table above, the iteration of the CNN network above that was trained using 1,000 rows generated the lowest FOM. Other interesting results included a ~8% decrease in performance for 6,000 steps per epoch vs 60,000 steps per epoch. Not surprisingly, the iteration that replaced the relu activation function saw a tremendous decrease in performance.

XI. Author Drawn Images

A. Background

The MNIST dataset is well known throughout the machine learning community. Yet, to test the performance of the algorithms used in this analysis, a more robust method of evaluation exists where one can draw images, and feed them into the algorithms as a new test set. Thus, 50 new images were drawn consisting of 5 images per digit, for the digits 0-9.

The below table summarizes the results from running the algorithms on the new test set:

New Test Set Author Drawn Images - Performance	
Algorithm	Accuracy
Kmeans	0.42
Logistic Regression	0.1
Logistic Regression - PCA	0.02
SVM	0.1
SVM - PCA	0.06
Decision Tree	0.14
Decision Tree - PCA	0.14
Random Forest	0.1
Random Forest - PCA	0.04
CNN - 60,000 Training Samples	0.81

It is interesting to note that only the CNN generated a performance above .5, with a performance of .81. This may have been due to the fact that the hand drawn images were drawn with sharpie, which may have translated poorly to image analysis in Python. However, it is also worthwhile to note that the CNN is also the most complex algorithm deployed above, and involves the most feature engineering out of the techniques used.

XII.

A. FOM Table for All Techniques used: The below table highlights the most successful algorithm used in this paper:

FOM Scores - All Techniques Used			
Algorithm	**P1**	**P2 (Accuracy)**	**FOM**
Kmeans	1	0.5244	0.9756
Logistic Regression	1	0.9172	0.5828
Logistic Regression - PCA	1	0.9217	0.5783
SVM	0.083	0.92	0.1215
SVM - PCA	0.083	0.74	0.3015
Decision Tree	1	0.72	0.78
Decision Tree - Tree	1	0.66	0.84
Random Forest	0.016	0.96	0.048
Random Forest - PCA	1	0.95	0.55
CNN - 60k	1	0.994	0.506
CNN - 10k	0.18	0.9908	0.0992
CNN - 6k	0.1	0.9146	0.1354
CNN - 6k (Sigmoid)	0.1	0.3792	0.6708
CNN - 1k	0.016	.97	0.038

References

Bhattacharyya, Saptashwa. "Support Vector Machine: MNIST Digit Classification with Python; Including My Hand Written Digits." Medium, Towards Data Science, 12 Apr. 2019, towardsdatascience.com/support-vector-machine-mnist-digit-classification-with-python-including-my-hand-written-digits-83d6eca7004a.

"Coding Games and Programming Challenges to Code Better." CodinGame, www.codingame.com/playgrounds/37409/handwritten-digit-recognition-using-scikit-learn.

Efebozkir. "Efebozkir/Handwrittendigit-Recognition." GitHub, github.com/efebozkir/handwrittendigit-recognition/blob/master/decisiontreefile.py.

Galarnyk, Michael. "Logistic Regression Using Python (Scikit-Learn)." Medium, Towards Data Science, 5 Nov. 2019, towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a.

Galarnyk, Michael. "PCA Using Python (Scikit-Learn)." Medium, Towards Data Science, 4 Nov. 2019, towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60.

Hvass-Labs. "Hvass-Labs/TensorFlow-Tutorials." GitHub, github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/02_Convolutional_Neural_Network.ipynb.

Jaju, Saurabh, et al. "Guide to t-SNE Machine Learning Algorithm Implemented in R & Python." Analytics Vidhya, 9 Oct. 2019, www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/.

"k-Means Advantages and Disadvantages | Clustering in Machine Learning." Google, Google, developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages.

Katariya, Yash. "Applying Convolutional Neural Network on the MNIST Dataset." Applying Convolutional Neural Network on the MNIST Dataset – Yash Katariya – CS Grad @ NCSU, yashk2810.github.io/Applying-Convolutional-Neural-Network-on-the-MNIST-dataset/.

NeuralNet, MYO. "The MNIST Dataset of Handwritten Digits." The MNIST Dataset of Handwritten Digits, 1 Jan. 1970, makeyourownneuralnetwork.blogspot.com/2015/03/the-mnist-dataset-of-handwritten-digits.html.

Violante, Andre. "An Introduction to t-SNE with Python Example." Medium, Towards Data Science, 30 Aug. 2018, towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1.

Wdlv. "Wdlv/Clustering-on-the-MNIST-Data." GitHub, github.com/wdlv/Clustering-on-the-MNIST-data/blob/master/Clustering_Classification.ipynb.

willkwillk 49311 gold badge44 silver badges1010 bronze badges, et al. "Why Is t-SNE Not Used as a Dimensionality Reduction Technique for Clustering or Classification?" Cross Validated, 1 May

1968, stats.stackexchange.com/questions/340175/why-is-t-sne-not-used-as-a-dimensionality-reduction-technique-for-clustering-or.