Handwritten Digit Recognition using Machine Learning Algorithms and Convolutional Neural Network

Abstract

Handwritten Digit Recognition can be considered as one of the most famous problems in the field of pattern recognition. Various works are done with varying level of accuracy and application of newer techniques. Developed in the 1980s, the MNIST (Modified National Institute of Standard and Technology) Database serves as the de-facto dataset for the purpose of digit recognition. It is safe to consider MNIST as the "Hello World" for Deep Learning as well as for testing various Machine Learning models. The purpose of the project is to highlight the performance of Machine Learning algorithms on the dataset and understand how the dimensionality reduction techniques affect these performances.

Introduction

With an increase in the quantum and complexity of information, it has become difficult for humans to maintain the speed and accuracy in handling this information. Computers came to its rescue, with faster and accurate processors. Since, then, the world has never been the same. However, computers are nothing but dumb machines which in turn look up to humans to operate. While it can perform millions of calculations in fractions of a second, it can't recognize a cat from a dog, a man from a woman, or a line from a circle. Every time, they are required to be hard-coded. While certain tasks can be hard-coded, others can't be. Let's introduce the MNIST Dataset, containing handwritten digits (from 0-9). If a computer is fed with one of the samples, it will fail to produce an output. However, it can quickly recognize and process an ASCII character since the machine is pre-installed with the information. Handwritten digits don't follow static pattern. Each person can produce different versions of the digits. In fact, sometimes it becomes difficult for fellow humans to recognize others handwriting.

Machine Learning algorithms follow a different approach. They learn from the input data and infer patterns. In supervised learning approach, each data sample contains certain information along with the label. The algorithm processes the raw data and tries to predict an output. The predicted output is compared with the true output (or the true label) and a loss function is used to measure how correctly the algorithm predicted. The aim is to minimize the loss function and achieve higher accuracy. The term *learning* can be understood as obtaining weight vectors. For eg. in case of a linear classifier, a line of the form $y = wx + b$ is required to be obtained. x is the input data, b is a bias (or constant) and w is the weight that the algorithm learns. The input data is called training set. A portion of the training set is used as validation set. Usually 20% of the training set is used as validation set. The algorithm is trained on the training set (80% of the original) and the remaining (validation set) is used to evaluate the model. The test set is the set without the labels. The algorithm is required to predict the labels in the test set.

In our problem, we will predict the digits from the handwritten data. This finds application in various sectors like courier service, banks, etc. Zipcodes can be automatically recognized by computers without human intervention. The cheques can be processed with ease. This way human resource can be utilized effectively.

We have employed four Machine Learning algorithms which classify each test samples into 10 discrete labels. This is a multi-class classification problem. Support Vector Machine (SVM), Random Forest, Logistic Regression, and XGBoost are used to train the training data and performance is evaluated on the training set and validation set. Then the models are used to predict the test set and the first hundred samples of the dataset are displayed with the predicted labels. Since the datasets have 784 features (will be discussed in the next section), it is not possible to visualize them on the 2-D plot. Moreover, the size of the datasets and the features demand greater training time. Dimensionality reduction methods are useful to extract features that explain the maximum variance and make the visualization possible. The effect of dimensionality reduction on the performance of the models is also studied. Finally, Convolutional Neural Network (CNN) is used on the training data. CNN outperforms the other models, undoubtedly.

Dataset

The dataset is obtained from Kaggle.com. The site contains two separate csv files – a training set and a test set. The training set contains 785 features, while the test set contains 784 features. The training set and the test set contains 42000 samples and 28000 samples respectively. The extra feature of the training set owes to the label. In the data preprocessing step, we have extracted the labels and stored them in *y_train* (vector). The remaining columns are stored in *x_train* (matrix). The test set is stored in *x_test* (matrix). Labels are needed to be predicted from the *x_test*. Since the model evaluation is not possible without any test labels, the *x_train* and *y_train* are further divided into training and validation set. 80% of the original *x_train* and *y_train* are preserved as the *x_train* and *y_train* and the remaining 20% is used as *x_val* and *y_val*. Only the *x_train* and *y_train* are used for training. The *x_val* and *y_val* are used for the purpose of cross-validation.

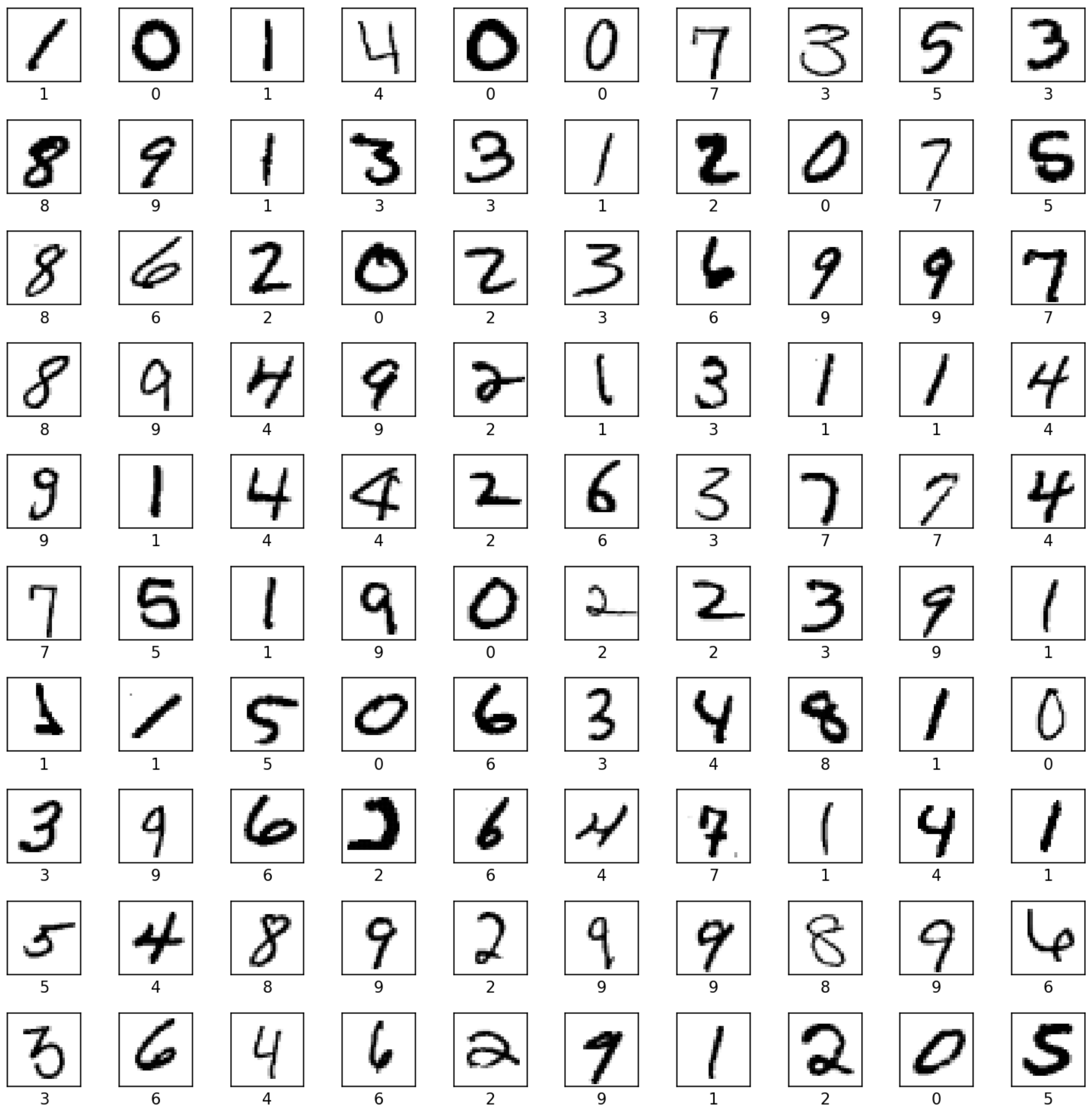


Fig.1 The MNIST Dataset (source: Kaggle.com)

Dimensionality Reduction Method

Usually, the data that we encounter contains more than two features. It's hard to make predictions with features as less as two. But it's very difficult to visualize data containing more than two or three features. Certain techniques can be used to reduce the features of the data and yet preserve the maximum information. For our purpose, we have used Principal Component Analysis (PCA) for dimensionality reduction. Large data with correlated features can be described with a smaller number of features that collectively explain most of the variability in the original data. The directions of principal components produced by the PCA are the directions in the feature space along which the original data are highly variable. PCA finds a low-dimensional representation of the dataset that contains as much as possible of the variation.

Each of the dimensions found by PCA is a linear combination of the features present in the original dataset. Consider a (n x p) dataset X. The first linear component can be found by the equation:

$$z_{i1} = \phi_{11}x_{i1} + \phi_{21}x_{i2} + \ldots + \phi_{p1}x_{ip} \quad \textit{(Eq. 1)}$$

This component will have the maximum variance and ϕ is subjected to the constraint:

$$\sum_{j=1}^{p} \varphi^{2}_{j1}=1 \quad \textit{(Eq. 2)}$$

The rest of the principal components can be computed in a similar fashion. However, it's important to note that these components will be uncorrelated to each other, i.e. orthogonal. For the purpose of visualization, we have taken 2 principal components initially.

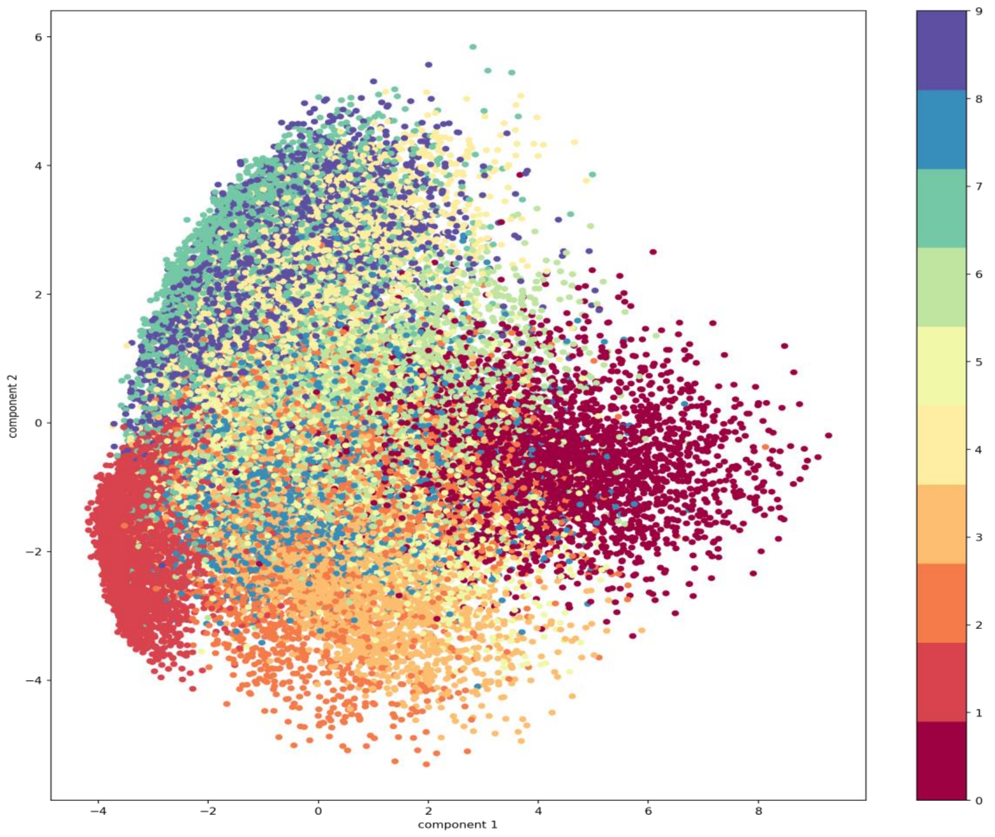


Fig.2 Two-Dimensional representation of data after applying PCA

The above scatter plot shows the distribution of the dataset in 2-D, along those two directions which describe highest variance. The color of the dots represents various digits.

However, two components are not sufficient to describe the entire dataset. We have plotted a graph for variance vs. no. of components, which will give us a fair idea about how the number of components is explaining the variance of the dataset.

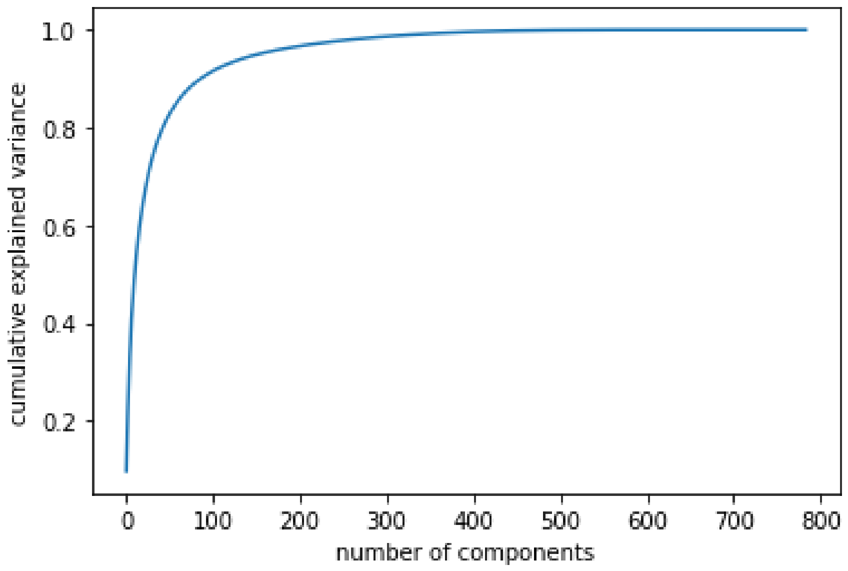


Fig. 3 Variance vs. no. of components

It can be seen from the graph that with only 2 components less than 10% of the variance is explained. This might give a fair idea about the distribution of the data, but it is useless for the purpose of training a model – we have lost most of the information. However, it can be seen that with around 100 components, roughly 90% of the variance can be explained. This is a good number! It can be found (shown in the notebook) that 153 components can explain 95% of the variance.

Why are we doing this if our goal of visualization is over? The training data (original) of interest has 784 features and 42000 samples. That's a huge number. Any machine learning algorithm will take a considerable amount of training time. With just 153 components, we can explain 95% of the variance, and the graph almost remains constant after 200-300 components. So, instead of considering all the 784 components, we can fix a % variance and apply PCA over the dataset. We have settled for 95% variance. This can reduce the training time since the number of features is less. However, we can comment on the training and validation accuracy of the model after applying PCA. With 95% variance, most of the information is expected to be retained. So, we can say that there shouldn't be much variation in the accuracy of the model. The graphs in the model selection section will clarify the doubts.

Image reconstruction after applying PCA

Given an original image, after applying PCA, how much of the information can be retained that is visually useful to us? We have applied PCA with % variance of 50, 75, and 95, and then with an inverse function, reconstructed the image.

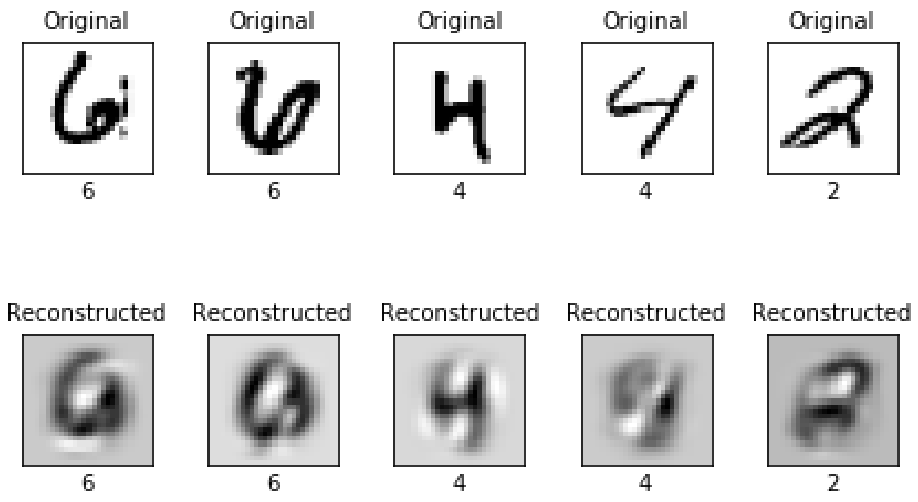


Fig. 4 PCA with 50% variance

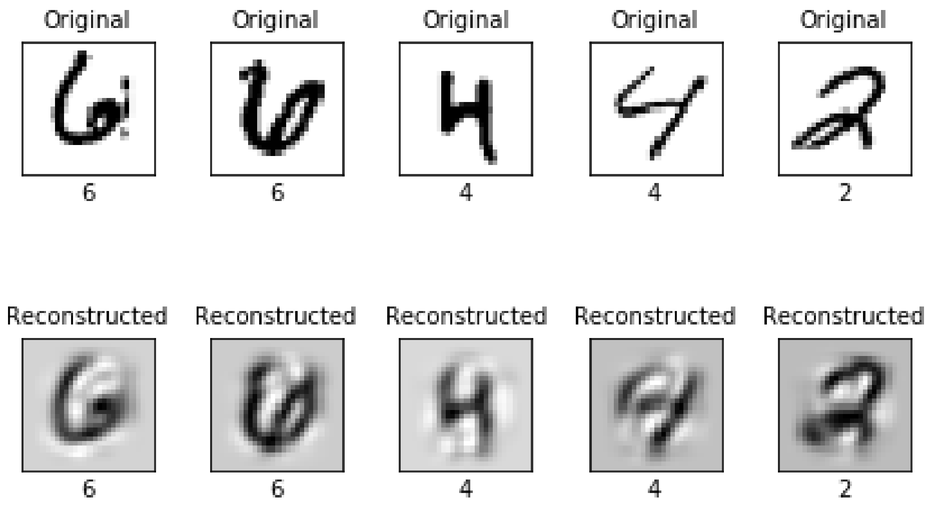


Fig. 5 PCA with 75% variance

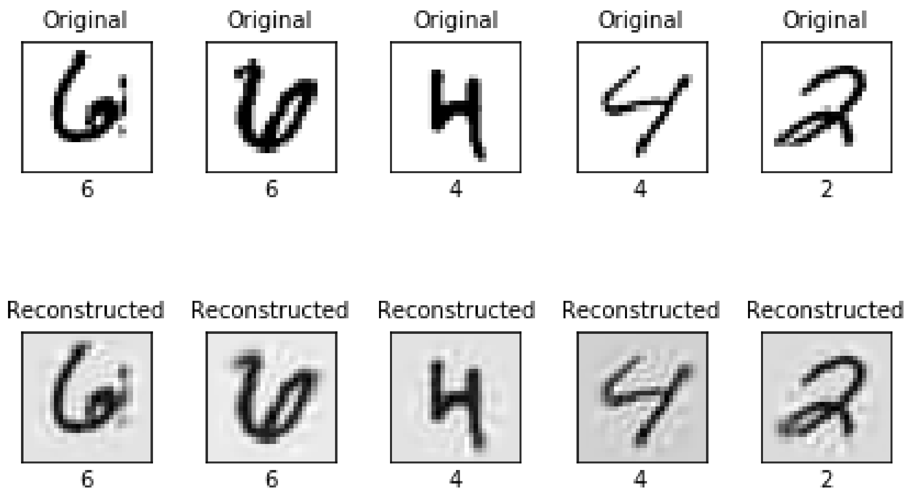


Fig. 6 PCA with 95% variance

With 50% variance, the reconstructed images are unrecognizable with a lot of distortions. 75% variance does a pretty good job. With 95% variance, it is almost identical to the original. So, we can safely consider 95% variance for training the models, and at the same time reduce the number of features by many folds. However, accuracy and training time are yet to be observed.

Normalization

Since an image is stored in the form of pixels in the range of 0-255, it needs to be normalized. Without normalization, it can cost sufficient training time. Both x_train (before splitting into validation set) and x_test are normalized by dividing by 255. This helps in obtaining the image in pixel range of 0-1.

Support Vector Machines (SVM)

Developed in the 1990s, SVM is based on maximal margin classifier. Though elegant, the maximal margin classifier can only be used on linearly separable data. The classifier is based on the concept of hyperplane. For a p dimensional space, a hyperplane is a p-1 dimensional subspace. A line serves as hyperplane for a 2-D space and a plane serves as hyperplane for a 3-D space. Higher-dimensional hyperplanes are difficult to visualize. This concept of hyperplane can be used to separate linearly-separable binary data. A separating hyperplane has following property:

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + . . . + \beta_p x_{ip}) > 0 \qquad \textit{(Eq. 3)}$$

An observation can be assigned a class depending on which side of the hyperplane it is located. Since there can be infinite number of such hyperplanes separating binary data, we consider the hyperplane which is furthest from the training observations. Considering a hyperplane, we can calculate the perpendicular distances from each of the observations. The smallest such distance is known as the margin. The maximal margin hyperplane is the separating hyperplane for which the margin is largest. It represents the mid-line of widest slab that we can insert between the two classes. The observations lying on these margins are called the support vectors and their position affects the hyperplane. The problem with maximal margin hyperplane is that they are sensitive towards the change in single observation, and therefore, can lead to overfitting. To solve this problem, Support Vector Classifier is introduced with soft margin. SVC tolerates some amount of misclassification. The optimization goals in case of SVC are:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\ \varepsilon_1,\ldots,\ \varepsilon_n,\ M}{\text{maximize}} \quad M \qquad \textit{(Eq. 4)}$$

$$\text{subject to} \quad \sum_{j=1}^{p} B_j^2 = 1 \qquad \textit{(Eq. 5)}$$

$$yi(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \varepsilon_i) \qquad \textit{(Eq. 6)}$$

$$\varepsilon_i \geq 0, \sum_{i=1}^{n} \varepsilon_i \leq C \qquad \textit{(Eq. 7)}$$

M is the width of the margin. $\boldsymbol{\varepsilon_i}$ is the slack variable that allows individual observations to be on the wrong side of hyperplane or margin. C is a budget for the amount that margin can be violated by the n observations.

The optimization goals of the maximal margin hyperplane are quite similar except that it doesn't contain any slack variable and violation budget. However, even with SVC, the problem of non-linearity is not solved. To solve this problem, we enlarge the feature space using higher-order polynomial functions of the predictors or using kernels. SVM is an extension of SVC that uses kernels. The solution of the Eq. (5-7) involves only the inner products of the observations. We can replace these inner products by K(x, x'). K is the kernel function and it quantifies the similarity of two observations.

Radial Basis Function (RBF) kernel is one of the most famous kernels used in SVM. RBF is expressed by:

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2) \qquad \textit{(Eq. 8)}$$

We have also used RBF kernel SVM on our training data.

Random Forest

Random Forest is based on tree-based methods. The predictor space is segmented into number of regions. A prediction is made based on the mean or the mode of the training observations in the region. Since the splitting rules can be summarized in a tree, this approach is called decision tree. The predictor space is divided into distinct and non-overlapping regions. For every observation that falls into the region, we make the same prediction, which is mode (in case of classification) of the response values for the training observations in the region. A top-down greedy approach is taken for splitting the predictor space. This is known as recursive binary splitting. It begins with a single space, which is then successively split, and each split is indicated via two new branches further down on the tree. At each step, the best split is made.

A classification error rate needs to be chosen to check the performance of the splits. Gini Index is one of the methods used.

$$G = \sum_{k=1}^{K} p_{mk}(1-p_{mk}) \qquad \textit{(Eq. 9)}$$

It is the measure of total variance across K classes. p_{mk} represents the proportion of training observations in the m^{th} region that are from kth class.

Entropy is an alternative to Gini index.

$$E = - \sum_{k=1}^{K} p_{mk} \log p_{mk} \qquad \textit{(Eq. 10)}$$

The decision trees suffer from high variance. Bootstrap aggregation or bagging is used to overcome this problem. In bagging, the algorithm takes many training sets from the population, build a separate prediction model using each training set and average the resulting predictions. Random Forest provides an improvisation over bagging by decorrelating the trees. It builds a number of decision trees on bootstrapped training samples. While building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.

XGboost

XGBoost is a pretty new algorithm based on gradient boosting algorithm. Boosting works similar to bagging, except that in boosting, the trees are grown sequentially. Boosting doesn’t involve bootstrap sampling; instead, it fits trees on the modified version of the original dataset. Boosting learns slowly and the construction of each trees depend on the trees that have already been grown. Gradient boosting uses gradient descent algorithm to minimize the errors in sequential models. Tianqi Chen and Carlos Guestrin introduced XGBoost in the paper titled *XGBoost: A Scalable Tree Boosting System*. In the paper the authors claimed that the algorithm is 10 times faster and scales to billions of examples in distributed system. The scalability of XGBoost is the reason behind its success. The scalability of XGBoost is due to systems and algorithmic optimizations. Parallel and distributed computing make learning faster which enables quicker model exploration. Most existing tree learning algorithms are either only optimized for dense data or need specific procedures to handle limited cases such as categorical encoding. XGBoost handles all sparsity patterns in a unified way.

Logistic Regression

Linear Regression uses:

$$Y \approx \beta_0 + \beta_1 X \qquad \textit{(Eq. 11)}$$

for predicting the output label. It uses the training set to obtain the values of the coefficient and the constant in (Eq. 11). The loss function to penalize the model is:

$$\text{loss} = \sum_{i=1}^{m} (y_i - y_i')^2 \qquad \textit{(Eq. 12)}$$

where, y_i is the true label and y_i' is predicted. This is a convex function, therefore, minimum of the function can be obtained. Gradient Descent method is used for this perform.

While linear regression is useful for predicting continuous values, it doesn't help in classification. In case of binary classification, our task is to calculate

$$\text{Probability}(Y \mid X=1) \qquad \textit{(Eq. 13)}$$

This is not possible with (Eq. 11). We need a function that converts it into the range of 0-1.

$$p(x) = \exp(\beta_0 + \beta_1 X)/1 + \exp(\beta_0 + \beta_1 X) \qquad \textit{(Eq. 14)}$$

serves our purpose. Logistic Regression uses:

$$\text{loss} = -y\log(y') - (1-y)\log(1- y') \qquad \textit{(Eq. 15)}$$

as loss function.

Convolutional Neural Network

In 1998, Yann LeCun et al. introduced LeNet-5 and its application in handwritten digit recognition in the paper titled *Gradient Based Learning Applied to Document Recognition*. They used backpropogation algorithm to learn the convolution kernel coefficients. CNN is a kind of neural network that has certain advantages over the artificial neural network. The patterns they learn are translation invariant. After learning a certain pattern in some part of the image, it can recognize it anywhere. CNN can learn spatial hierarchies of patterns. The first layer might learn small local patterns as edges; the second layer might learn other features composed of patterns from the first layer, and so on. These make CNN perfect for image recognition.

The convolution operation is defined as:

$$s(t) = (x*w)(t) = \sum_{a=-\infty}^{\infty} x(a)\, w(t-a) \qquad \textit{(Eq. 16)}$$

The input is a multidimensional array of data and kernel is a multidimensional array of parameters. Convolution operates over 3D tensors called feature maps with spatial axes (height and width) and a depth axis (channel). The convolution operation extracts patches from its input feature map and applies the same transformations to all the patches producing an output feature map. A convolution works by sliding windows of 3x3 or 5x5 in the stride of 1 over the 3D input feature map. Each such 3D patch is then transformed into a 1D vector. All of these vectors are then spatially reassembled into a 3D output map.

Max Pooling is another operation performed that downsamples the feature maps. It consists of extracting windows from the input feature maps and outputting the max value of each channel. They work by sliding windows of 2x2 in the stride of 2.

The operations in a CNN can be summarized as consisting of series of convolution and max pooling operations, flattening the feature maps (output of these operations), and then applying dense layers. In each operation certain activation function is used that transform the input. For the convolution and max pooling operation, Rectified Linear Unit (RELU) is used and in the last layer, softmax is used. Softmax determines the probability of each input from the previous layers.

RELU:

$$y = \max(0, x) \qquad \textit{(Eq. 17)}$$

Softmax:

$$\sigma(z)_i = e^{zi} / \sum_{j=1}^{K} e^{zi} \text{ for } i = 1\ldots K\text{, and } z = (z_1\ldots z_k) \in \mathbb{R}^K \qquad \textit{(Eq. 18)}$$

Model Evaluation

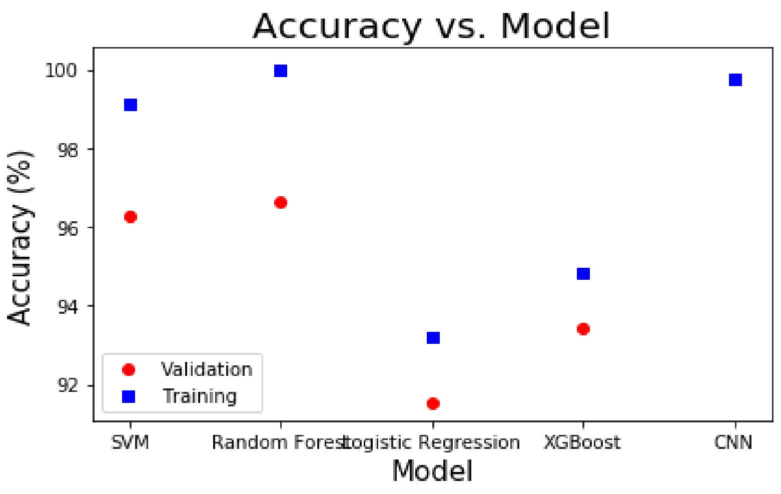
We have considered accuracy (training and testing) and training time as metrics for model evaluation. Confusion Matrix is also plotted for each model (in the notebook). For the machine learning models (SVM, Random Forest, Logistic Regression and XGBoost), two training sets are used – one with PCA applied and another without. This approach will help us understand how dimensionality reduction method such as PCA affects the performance of the algorithms and training time. Python (*version 3.6*) and Keras (*with Tensorflow backend*) are used for implementation. The models are run in Windows 10 operating system, processor Intel(R)Core(TM)i5-7200U CPU @ 2.50GHz 2.71 GHz and 8.00GB RAM.

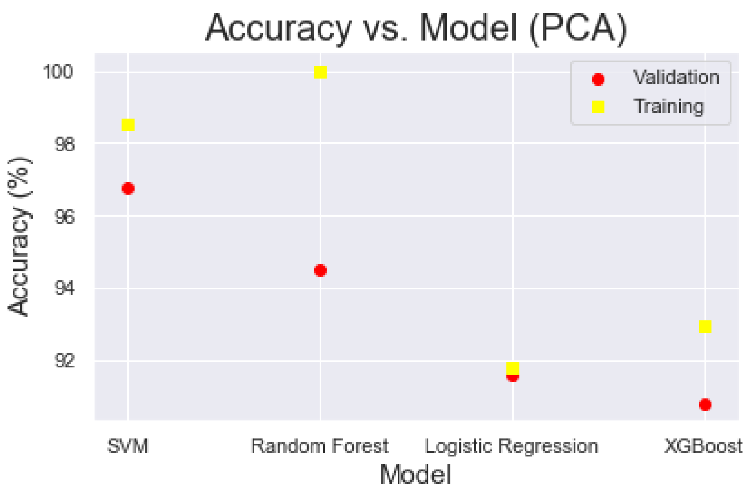
Models with various metrics for evaluation (without PCA)

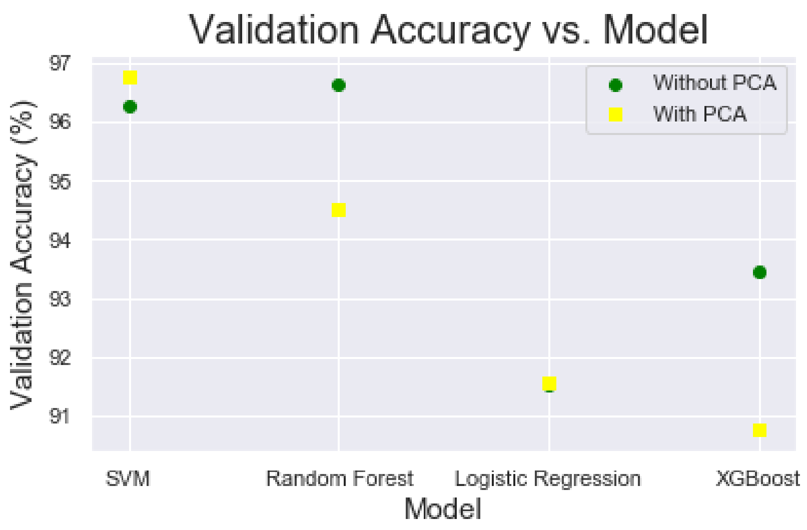
Models	Validation Accuracy (%)	Training Accuracy (%)	Training Time (sec)
SVM	96.28	99.11	131
Random Forest	96.64	100	67
Logistic Regression	91.53	93.21	33
XGBoost	93.45	94.82	429
CNN	99.78	99.78	447

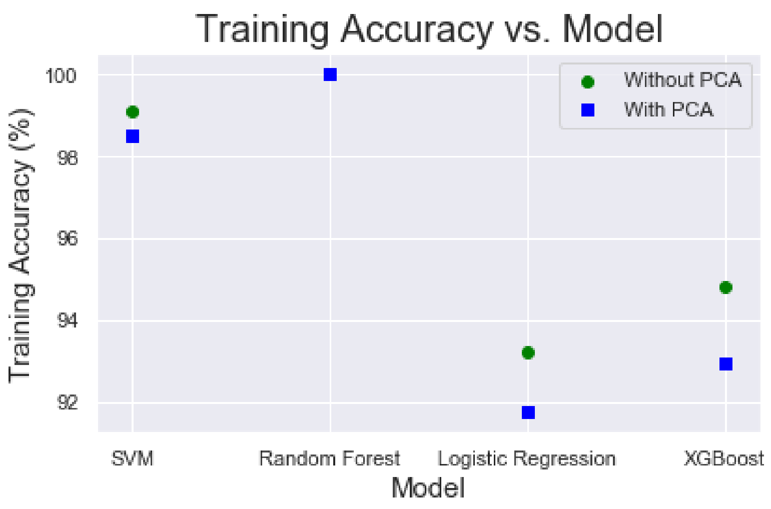
Models with various metrics for evaluation (with PCA)

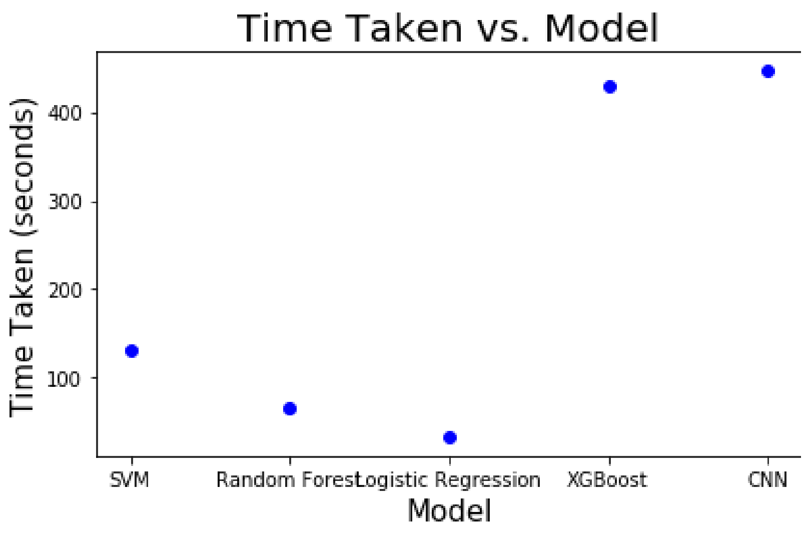
Models	Validation Accuracy (%)	Training Accuracy (%)	Training Time (sec)
SVM	96.75	98.52	32
Random Forest	94.5	100	145
Logistic Regression	91.55	91.77	28
XGBoost	90.78	92.93	300

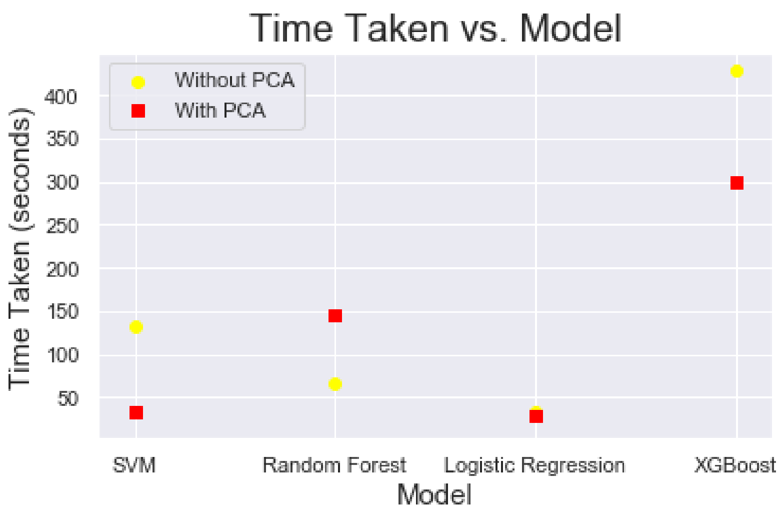

Accuracy vs. Model
Accuracy (%)
100
98
96
94
92
Validation
Training
SVM
Random Forest
Logistic Regression
XGBoost
CNN
Model

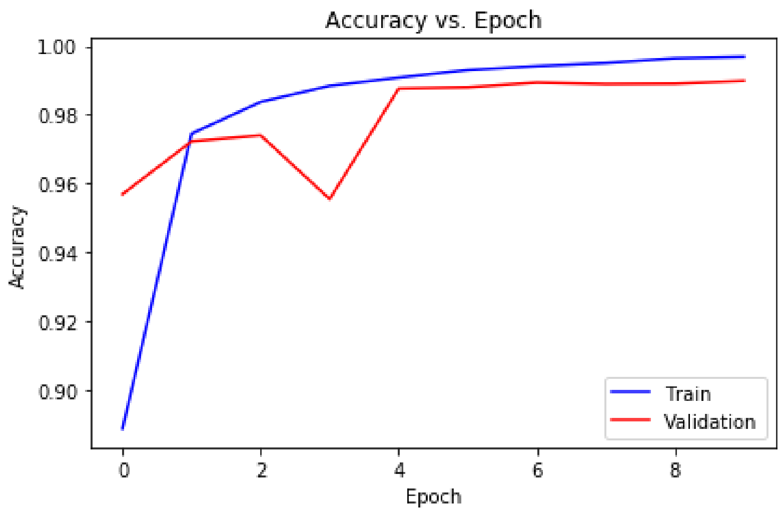

Accuracy vs. Model (PCA)
Accuracy (%)
100
98
96
94
92
Validation
Training
SVM
Random Forest
Logistic Regression
XGBoost
Model

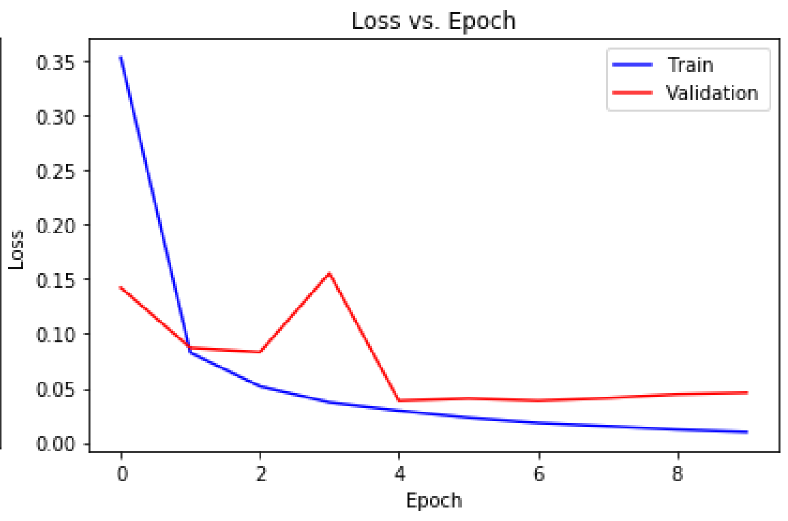

Validation Accuracy vs. Model
Validation Accuracy (%)
97
96
95
94
93
92
91
Without PCA
With PCA
SVM
Random Forest
Logistic Regression
XGBoost
Model


Training Accuracy vs. Model
Training Accuracy (%)
100
98
96
94
92
Without PCA
With PCA
SVM
Random Forest
Logistic Regression
XGBoost
Model


Time Taken vs. Model
Time Taken (seconds)
400
300
200
100
SVM
Random Forest
Logistic Regression
XGBoost
CNN
Model


Time Taken vs. Model
Time Taken (seconds)
400
350
300
250
200
150
100
50
Without PCA
With PCA
SVM
Random Forest
Logistic Regression
XGBoost
Model





From *Accuracy vs. Model (without PCA)* graph, it can be seen that CNN outperformed the rest of the models (on the validation set). In fact, the validation accuracy and training accuracy overlapped. The model generalized perfectly. Among the machine learning models, random forest has highest validation accuracy. Though it attained 100% accuracy on the training set, there is a considerable gap between training and validation accuracy. SVM performed at par random forest with minimal difference. Though XGBoost is a superior algorithm, it couldn't beat the other machine learning algorithms. Maybe parameter tuning is to be blamed. Logistic regression performed worst.

PCA is applied only on the machine learning models. *Accuracy vs. Model (with PCA)* graph plots the validation and training accuracies for various algorithms. In terms of validation accuracy, SVM is the winner and XGBoost performed worst. Random Forest has the worst generalization with a considerable gap between training and validation accuracy. However, logistic regression, though, performed average, achieved best generalization.

Validation Accuracy vs. Model and *Training Accuracy vs. Model* graphs compared the performances of the algorithms when PCA is applied to the training set and when not applied. By this, we can understand how PCA affect the performances. In SVM, PCA increases the accuracy on validation set. In the rest of the models, PCA decreased the accuracy on validation set. However, it didn't make much difference with logistic regression – in fact, they overlapped. On the training set, PCA reduced the accuracy in all the models, except for the random forest, where it overlapped completely (100% accuracy).

In terms of training time, PCA helped in faster execution, except for the random forest. In addition to the visualization purpose, PCA is expected to reduce the training time in case of a large dataset. For random forest, it proved to be terrible – both reducing the validation accuracy and the training time.

When PCA is not applied and both machine learning models and CNN are compared, CNN takes the highest training time and logistic regression takes least, though accuracy is compromised. For the CNN model, the *Accuracy vs. Epoch* graph shows that in the beginning of the epoch, it generalized poorly, but with an increase in epoch, it starts generalizing well, with slight dip in 3rd epoch. However, after 4th epoch, the training and validation accuracy remains constant with minimal difference. Loss is decreased with increase in epoch as shown in the *Loss vs. Epoch* graph, though there's a slight increase in the 3rd epoch which corresponds to the dip in the accuracy graph.

Result

From the graphs, it can be seen that CNN performed superior to the other algorithms in terms of accuracy and generalization. In fact, CNN is highly effective for the purpose of image recognition. However, SVM and Random Forest also did appreciably. Model selection methods like K-fold cross validation and GridSearch CV might have helped to attain better accuracy. While PCA helped in both visualization and reducing training time (except for Random Forest), it reduced the accuracy in most of the cases. However, SVM is an exception. If the training set is huge with hundreds of features, PCA can be used to select correlated components that best describe the data. But it should be used with caution – it can reduce the accuracy to some extent.

The future works in this direction include handwritten texts recognition in various local languages. Though considerable works have already been done, certain local languages (especially, in countries like India with more than 22 languages) need to be covered extensively. In fact, that will help in translating old manuscripts of historical significance in an efficient way.

References

1. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. *An Introduction to Statistical Learning with Application in R*. 2013. Springer.
2. Francois Chollet. *Deep Learning with Python*. 2018. Maning Publication Co.
3. Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning*. 2016. MIT Press.
4. Jake VanderPlas. *Python Data Science Handbook*. 2017. O'Reilly.
5. Tianqi Chen, Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System.* 2016. arXiv.
6. Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner. *Gradient Based Learning Applied to Document Recognition*. 1998. Proceedings of the IEEE.