ISTA 521 – Final Project options B

Due: Monday, December 13, 5pm

20 points of final grade

Quan Gan

Graduate

1 Introduction

In this final project option B, I am going to explore the different machine learning algorithms' performances on the MNIST [5] handwritten digit data set of computer vision. MNIST refers to Modified National institute of Standards and Technology and released in 1999. This data set is widely used for training and testing in the field of machine learning. The purpose is to identify the handwritten digit images into different digit categories from 0 to 9. In this project, I use the subset of MNIST handwritten digit data set from Kaggle Digit Recognizer competition [2]. The Kaggle provides three csv files, train, test, and sample_submission. The train.csv is the only data file I use to train and evaluate models. Five machine learning models selected to do the experiments are Naive bayes Classifier, Decision Tree Classifier, Support Vector Machine, K-Nearest Neighbors, and Convolutional Neural Network. To reproduce the experiments, please view the digit_recognizer.py file.

1.1 Experiment Setting

All models are trained by my personal computer. The system specifications are as follows:
Processor - **Intel**^{R}**Core**^{TM} i7-7700 CPU @ 3.60GHz. RAM - 16.0 GB. System type - 64-bit. Graphics - GeForce GTX 1080.

All codes are from SciKit-learn and Tensorflow libraries. Sklearn provides two Native Bayes classifier, GaussianNB and MultinomialNB, which could be used in this project. DecisionTreeClassifier, SVC, and KNeighborsClassifier are other classifiers used in this experiments from Sklearn. Tensorflow provides the Convolutional Neural Network model. The libraries' versions could be found in the requirements.txt.

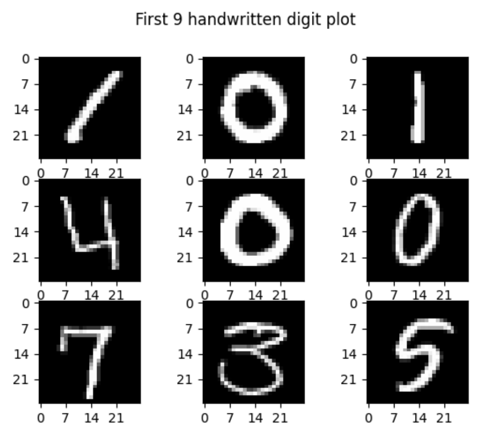
2 Data Set

The train.csv file contains 42000 observations and 785 columns. The first column is the label of the observation. So, there are 784 attributes for a handwritten digit image. Each attribute is a pixel in the image. The original image is 28 pixels in height and 28 pixels in width. If we use matrix to represent the image, it will be visualized below.

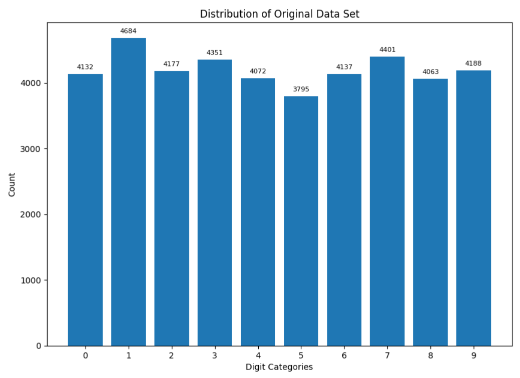
$$\textbf{Handwrite Digit} = \begin{bmatrix} column\ 0 & column\ 1 & \cdots & column\ 27 \\ column\ 28 & column\ 29 & \cdots & column\ 55 \\ \vdots & \vdots & \ddots & \vdots \\ column\ 756 & 0 & column\ 757 & column\ 783 \end{bmatrix}$$

The values for all attribute are from 0 to 255, which are the RGB color values. The values of labels are from 0 to 10. Figure 1 (a) illustrates the first 9 handwritten digits in the data set. Figure 1 (b) displays the distribution of digit categories in the data set.

Procedure: setting global variable, "DATASET == True". And run "python digit_recognizer.py" command.



(a)



(b)

Figure 1: (a) The first 9 observations in the data set. (b) The digit categories distribution in the data set

3 Models

Various models have various settings. As a result, our experiment has two steps. To begin, I train the models with various settings in order to determine the best performance settings. The original data set is split to 90% (37800 observations) as train set and 10% (4200 observations) as test set in the step one. Second, I train models with several train sets of various sizes and compare their results. The results would be demonstrated by accuracy and confusion matrix.

3.1 Naive Bayes Models

Gaussian Bayes Models [1] assumes each pixel in the image is a Gaussian distribution and the probability of each digit is equal. The likelihood of the attribute is the same as we see in the class.

$$p(\mathbf{t}|\mathbf{w},\mathbf{X},\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2\mathbf{I}}} exp\Bigg(-\frac{1}{2}(\mathbf{t}-\mathbf{Xw})^\top(\sigma^2\mathbf{I})^{-1}(\mathbf{t}-\mathbf{Xw})\Bigg)$$

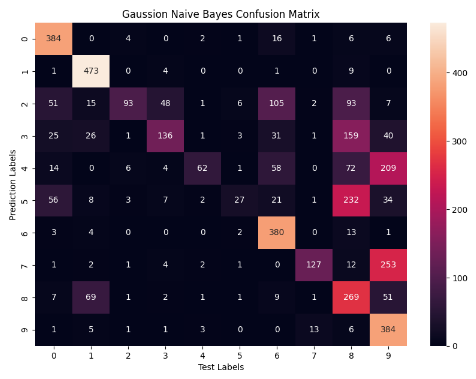
The σ^2 and $\mathbf{Xw}$ will be estimated by the maximum likelihood. However, for digit recognizer case, the distributions of each pixel are not Gaussian. So, the Gaussian Naive Bayes does not perform well. The accuracy is 55.5952% and training time is 0.4448s. The confusion matrix displays in Figure 2 (a).

Multinomial Bayes Models [1] is a naive Bayes algorithm for multinomially distributed data. The multinomial distribute is used for model feature vectors where each value represents the number of features appearing in a sample of class y. **Feature vectors** $= \theta_y = (\theta_{y1}, ..., \theta_{yn})$. y refers to the one of the class. n refers to the number of features, the number of pixels in this case. The θ_{yi} is the probability $p(x_i|y)$ for feature i. Then the probability would be estimated by a smoothed version of maximum likelihood.

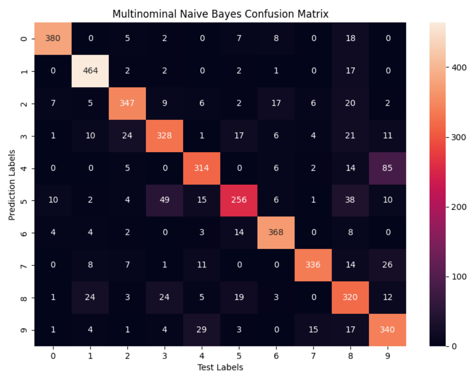
$$\theta_{yi} = \frac{\mathbf{N}_{yi} + \alpha}{\mathbf{N}_y + \alpha n}$$

$\mathbf{N}_{yi}$ is the number of times feature i appears in a sample of class y in the training set $\mathbf{T}$, and $\mathbf{N}_y$ is the total count of all features for class y. In digit recognizer case, the accuracy is 82.2143% and training time is 1.1409s. The confusion matrix displays in Figure 2 (b). By comparing these two naive bayes models, Multinomial Bayes models is more suitable for digit recognizer and will be used in experiment step 2.

Procedure: setting global variable, ”NAIVE_BAYES = True” and running ”python digit_recognizer.py” command.



(a)



(b)

Figure 2: (a) The confusion matrix for Gaussian Naive Baye. (b) The confusion matrix for Multinominal Naive Bayes

3.2 Decision Tree classifier

A decision tree classifier [7] is a prediction model that, like the literal definition, is based on a tree structure. The concept of decision tree can be comprehended easily. Figure 3 is an example of decision tree with 3 max depths. Each internal node is a binary test. The leaf nodes are the classified results. For example, we have n observations, $\mathbf{X} = X_1, X_2, ..., X_n$ and we need to identify them to m classes, $\mathbf{Y} = Y_1, Y_2, ..., Y_m$. For each observation, $x_i, 0 < i <= n$, x_i has k attributes, $\mathbf{Z} = Z_1, Z_2, ..., Z_k$. The internal binary tests would test each observations by their attributes. Depending on the outcome of the test, the observations would go to either the left or the right sub-branch of the tree. Eventually, if the tree reach the max depth or all leaves could not be split anymore, the decision tree classifier is finished.

Selecting a reasonable number for max depth parameter is very important. I use iterator way to train different decision tree classifiers from 1 max depth to 15 max depth. Figure 4 displays the accuracy of different decision tree classifiers. As we can see after 10 max depth, the accuracy has no prominent improvement. Therefore, the 10 max depth parameter would be used for step 2.

Procedure: setting global variable, "DECISION_TREE = True" and running "python digit_recognizer.py" command.

3.3 Support Vector Machine

Support Vector Machine [8] is a supervised learning method used for classification, regression, and outliers detection. In this project, we could focus on classification. The Scikit-learn library provides SVC APi refereed to C-Support Vector Classifier. Based on our data set, the classification task is the multi-class classification. The SVC Api implement the one-versus-one approach for multi-class classification. Supporting we have n classes, SVC will construct $n * (n-1)/2$ classifiers. Each classifier will identify one data point from two classes. Choosing a reasonable kernel is signification in SVM. SVC has five kernels, linear, poly, rbf, sigmoid, and precomputed. rbf is the default kernel. I will compare linear and rbf kernels and choose better kernel for this project. The linear and rbf kernel equations:

$$\begin{aligned} \textbf{linear } k(\mathbf{x},\mathbf{z}) &= \mathbf{x}^\top \mathbf{z} \\ \textbf{Gussian or rbf } k(\mathbf{x},\mathbf{z}) &= \exp\Big(-\gamma(\mathbf{x}-\mathbf{z})^\top(\mathbf{x}-\mathbf{z})\Big) \end{aligned}$$

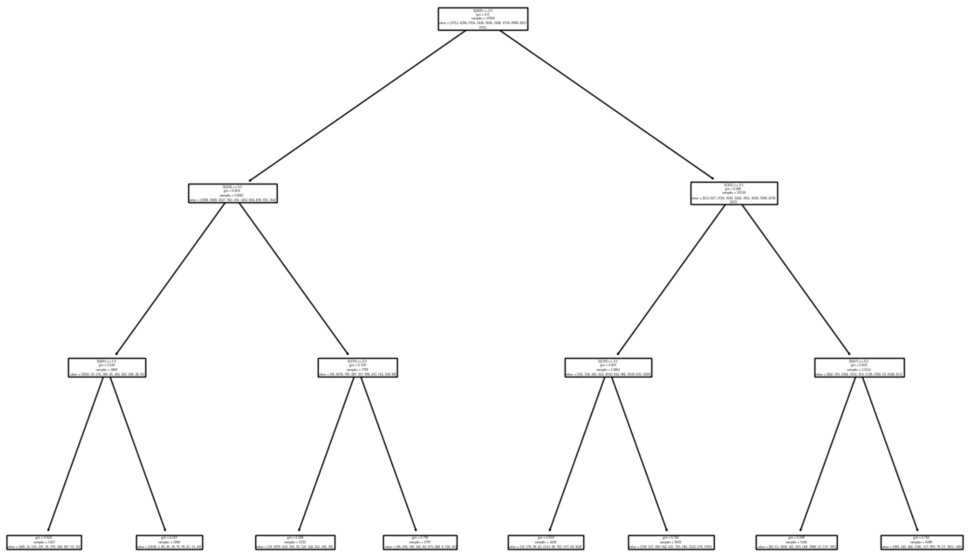


Figure 3: Decision Tree with 3 max depths

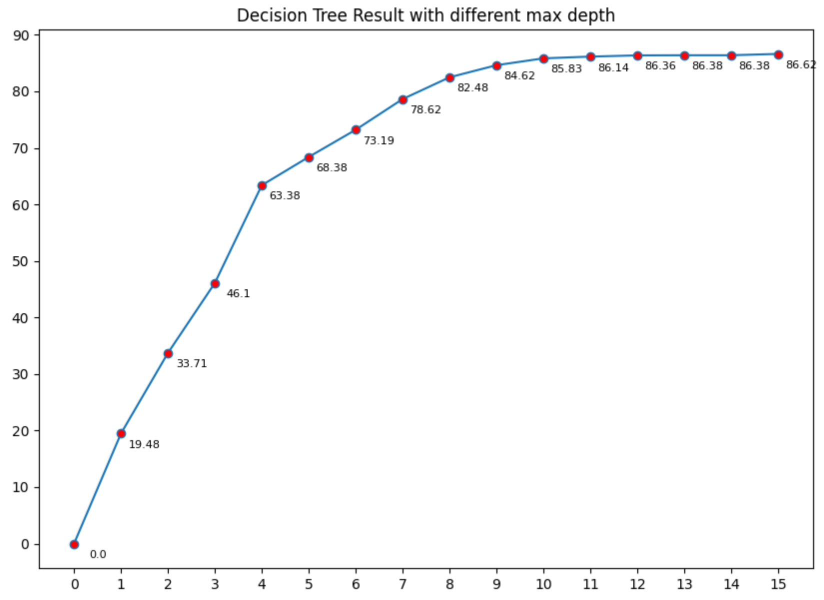


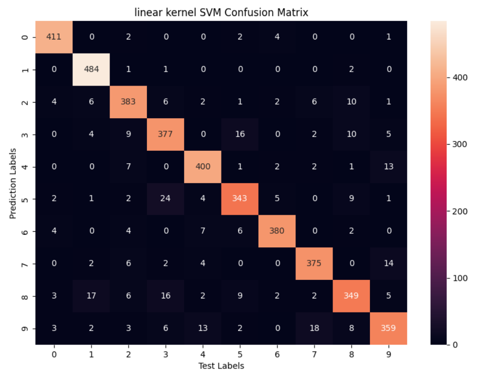
Figure 4: Decision Tree Classifier Accuracy with different max depths

For rbf kernel [6], there are two important hyperparameters, C and gamma. The gamma parameter defines how far the influence of a single training example reaches. Low values gamma means far and high values gamma means close. C parameter is the trade-off of correct classification of training examples against maximization of the decision function’s margin. The default C is 1 and default gamma is 1/(**n_features** $*$ **variance of train data**). The Cross-Validation would be used to select optimal C and gamma. I use this posting [4] as reference to select optimal C = 10 and gamma = 0.001.

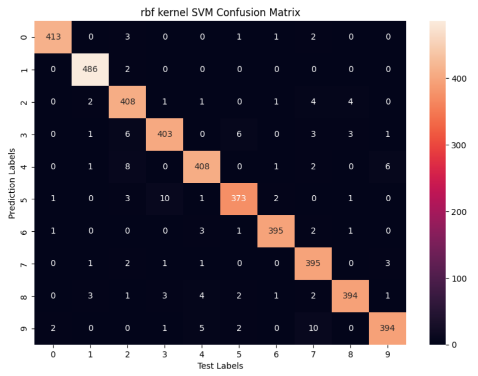
Linear kernel: The accuracy is 91.9286% and running time is 106.9007s. The confusion matrix is Figure 5 (a).

rbf kernel: The accuracy is 96.881% and running time is 116.287s. The confusion matrix is Figure 5 (b). The result show rbf kernel has better accuracy than linear kernel so the rbf kernel svm would be used for step 2.

Procedure: the range of attribute is 0-255. I first normalize the values by dividing by 255. And then, the values would be scaled by sklearn.preprocessing.scale API. setting global variable, ”SUPPORT_VECTOR_MACHINE = True”. And run ”python digit_recognizer.py” command.



(a)



(b)

Figure 5: (a) The confusion matrix for linear kernel SVM. (b) The confusion matrix for rbf kernel SVM

3.4 K-nearest Neighbors Model

KNN is a non-parametric classification method. There is no training phase for KNN. The model just stores the training data and identify the test data based on the neighbors of the test data points. K refers to the number of nearest neighbors used to vote the label of test data points. Let us assume k = 3 and the test data points as x_{new}. In the training points, the model finds 3 points that are closest to the x_{new}. The closest distances could be computed by Euclidian distance, Manhattan distance, or other distance metrics. Than, the majority class amongst these 3 neighbors would be assigned to the x_{new}. The choice of K is arbitrary but we could use cross-validation to choose best k value for the different tasks. Figure 6 shows the results of different k values for digit recognizer task. K = 1 is best choice with 96.74% accuracy and will be used in step 2.

Procedure: setting global variable, ”KNN = True” and running ”python digit_recognizer.py” command.

3.5 Convolutional Neural Network Models

CNN is a type of neural network model that has proven to be successful at image classification. CNN consists of 3 main hidden layers, convolutional layer, pooling layer, and dense layer. Figure 7 demonstrates CNN architecture [9] with two convolutional layers and two max pooling layers.

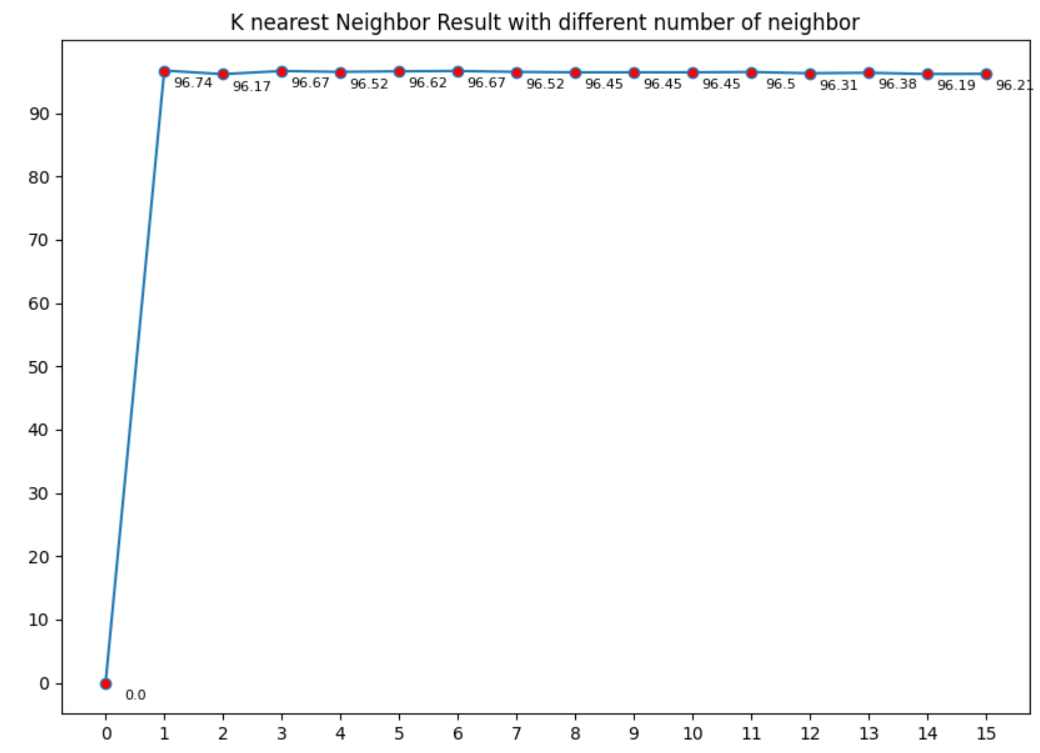


Figure 6: K Nearest Neighbors Classifier Accuracy with different number of neighbors

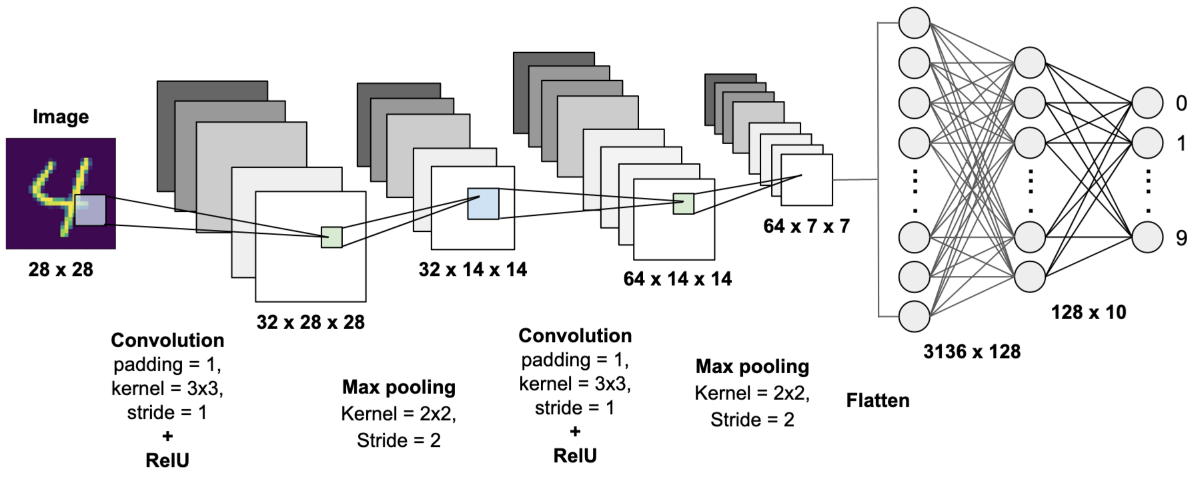


Figure 7: CNN Architecture from Towards Data Science [9]

The convolutional layers would extract features from image raw pixels for better classification. The convolutional layer has a $n \times n$ kernel matrix. The default n is 3. The elements in kernel matrix are randomly assigned. The kernel matrix would do element-wise multiplication with previous layer starting from top left corner $n \times n$ matrix and sum elements of result matrices as a new pixel stored in convolutional layer. In addition, convolutional layer contains an activation function. The most popular activation function is ReLu, which is a linear function to output the input directly if it is positive and 0 otherwise. For example,

$$\textbf{Input layer matrix} \quad = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 2 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \textbf{kernal matrix} = \begin{bmatrix} 2 & 1 & 3 \\ 4 & 0 & 2 \\ 5 & 1 & 1 \end{bmatrix}$$

$$\textbf{convolutional layer} \quad = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 13 & 11 & 0 \\ 0 & 19 & 8 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Pooling layer is used to reduce the size of image. There are two pooling methods, max pooling and average pooling. Pooling layer also has a $m \times m$ kernal matrix to select elements from previous layer. For example, the max pooling with m = 2 operates below.

$$\textbf{convolutional layer} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 13 & 11 & 0 \\ 0 & 19 & 8 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \textbf{max pooling layer} = \begin{bmatrix} 13 & 11 \\ 19 & 8 \end{bmatrix}$$

Dense layer deeply connects to the preceding layer, which means each node in the dense layer connects to every nodes in the preceding layer. Flatten layer would reshape the pooling layer to a vector as input fed to backpropagation algorithm. The output layer of CNN models is a softmax algorithm.

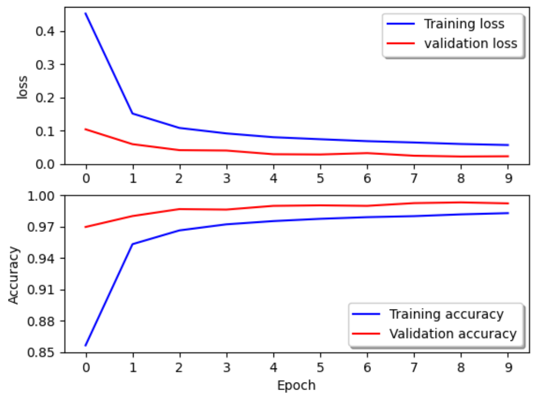
Furthermore, Dropout [3] is a popular approach used in neural net models to avoid the over-fitting. Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. Dropout will randomly ignore some nodes of layer for each training sample.

Setting and Result: My CNN model consists of first 32 filter convolutional layer with 5 x 5 kernel size and second 64 filter convolutional layer with 3 x 3 kernel size. The max pooling kernel size is 2 x 2. And the epoch is 10. The result accuracy is 99.2381%, which will be different per running because of the dropout layers. Figure 8 (a) shows the loss scores and accuracy scores of different epochs and (b) shows the confusion matrix for CNN. As we can see, more epochs could increase the accuracy and decrease the loss scores.

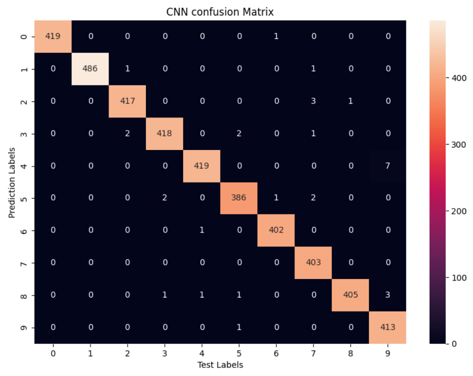
Procedure: Setting global variable, "CNN = True" and running "python digit_recognizer.py" command to reproduce the results.

4 Models Comparison

The second experiment of this project is to compare the five selected models mentioned above with different training set size. The original training set has 42000 observations, which would be split into 10500, 21000, 32500, and 42000 subsets and fed into models. At each train phrase, 5 folder cross validation would be used for evaluating Multinominal Naive Bayes, Decision Tree, Support Vector Machine, K nearest neighbor models. For Convolutional Neural Network Models, the subsets would be split into 80% as training set and



(a)



(b)

Figure 8: (a) The loss scores and accuracy scores of different epochs CNN. (b) The confusion matrix for CNN

20% as test set. Then, the CNN model would be trained by the split sets because dropout approach is already used for avoiding overfitting.

Accuracy Result: Figure 9 demonstrates the five models' accuracy across different training set. The CNN model outperforms others on all different training set and could achieve 99% accuracy. Furthermore, the train set size has no obvious impact on CNN performances. K nearest neighbor and Support Vector Machine have similar performances on the digit recognizer task and could achieve 98% accuracy. With increment of train set size, KNN and SVM performances are improved. Mutinominal Naive Bayes performs worst and the performances are not also influened by train set size because the naive nayes model is based on probability of classes and each train set has the same class distribution.

Running Time Result: Figure 10 displays the five models' running time across different training set. Because I doesn't set up GPU for the task, the running time would be slightly different each time run. From the plot, we could notice the support vector machine has longest running time because support vector machine needs to train multiple one-to-one classifier across 10 classes. The CNN model needs more running time than rest of models. The K-nearest neighbor, Naive Bayes, and Decision Tree have the similar shortest running time.

With these two results, the Convolutional Neural Network and k-nearest neighbor models are suitable for digit recognizer task. The CNN could achieve the highest accuracy. The KNN could spend less running time but maintain the relatively high accuracy.

Procedure: Setting global variable, "MODEL_COMPARE = True" and running "python digit_recognizer.py" command to reproduce the results.

5 Limitation and Discussion

We could still improve each model's performances by adjusting their hyperparameters just like the C and gamma parameters for support vector machine. the Convolutional Neural Network would be impacted by the choices of layers. This project just researches on which algorithms are good for digit recognizer task. In the future research, we could fine-tuning the CNN model or KNN model to achieve better performances or to recognize not only digit but also different characters. Furthermore, we need to smaller train set size to compare the models. Over 10k observations may be still too large to be compared.

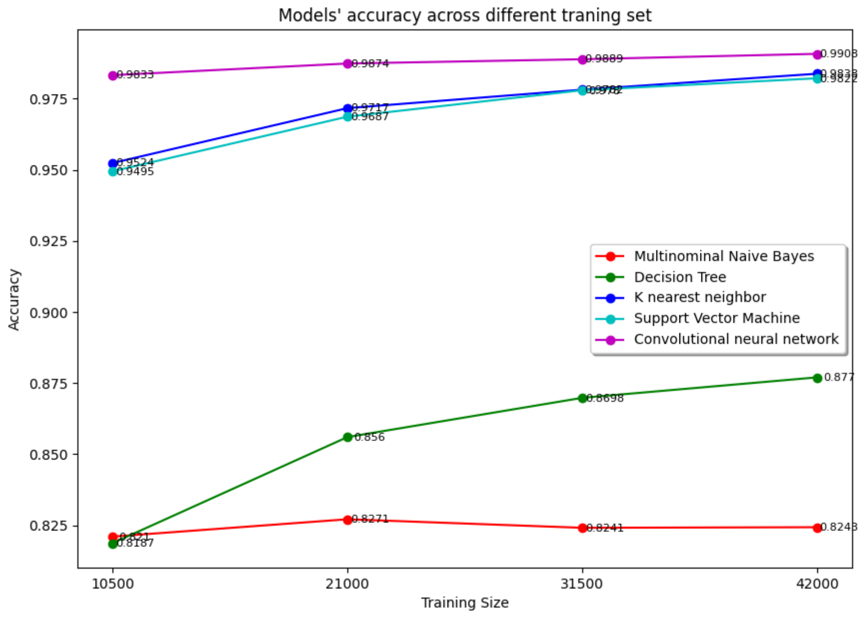


Figure 9: Five models' accuracy in experiment 2

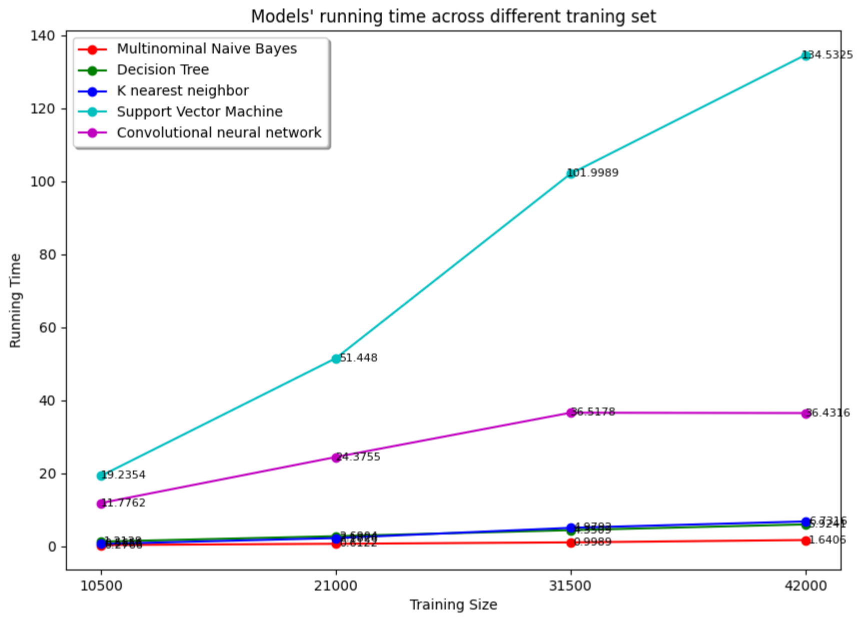


Figure 10: Five models' running time in experiment 2

References

[1] 1.9. Naive Bayes. https://scikit-learn/stable/modules/naive_bayes.html. Accessed: 2021-11-22.

[2] Digit Recognizer. https://kaggle.com/c/digit-recognizer. Accessed: 2021-11-21.

[3] A gentle introduction to dropout for regularizing deep neural networks. https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/. Accessed: 2021-11-27.

[4] Mnist digit recognition using svm. https://www.kaggle.com/nishan192/mnist-digit-recognition-using-svm. Accessed: 2021-11-27.

[5] MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. http://yann.lecun.com/exdb/mnist/. Accessed: 2021-11-21.

[6] Rbf svm parameters. https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html. Accessed: 2021-11-27.

[7] sklearn.tree.decisiontreeclassifie. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier. Accessed: 2021-11-27.

[8] Support vector machines. https://scikit-learn.org/stable/modules/svm.html#svm-classification. Accessed: 2021-11-27.

[9] K. Patel. Mnist handwritten digits classification using a convolutional neural network (cnn). https://towardsdatascience.com/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn-af5fafbc35e9. Accessed: 2021-11-27.