ELEC 390

Mini Project Report

Group 25
Andy Li: 20124884
William Du: 20100436

Section 1: Introduction and Motivation

To begin with, the topic of the project is recognizing handwritten digits with machine learning. In nowadays, more and more people are using tablet pens to take notes and write on their tablets or other electronics, even though pencil and paper are still widely used for various reasons (price, availability……etc). To improve the user experience of handwritten using tablet and make it more efficiently used, it is important to have what a person has written down accurately recognized by the machine, and hence it is significant to train the machine to properly recognize handwritten notes (including letter, digit……etc). The dataset used in this project comes from the MNIST database. The MNIST dataset is an acronym standing for Modified National Institute of Standards and Technology dataset. It is comprised of 60,000 small square 28x28 pixel grayscale images of handwritten single digits between 0 and 9. The task is to classify a given image of a handwritten digit into one of the 10 classes representing integer values from 0 to 9, inclusively. The detail of the dataset will be discussed in Section 2. In long term, with the advance of technology, the use of electronics and high-tech devices will eventually dominate the society. Without a doubt, machine learning is an indispensable part of the development of modern technology.

Recognizing digits accurately through machines is not as easy as it sounds to be. Although every digit has a “standard” shape which is defined by mathematicians long ago, the way that every digit is written down will be different for different people. It is almost impossible to be digitally identical if the same digit is written by different people. In other words, the variation of the same digit viewed by the machine is relatively high. Without using machine learning, one way for recognizing digits is to simply set ranges and tolerances for every digit. When a handwritten digit got converted digitally, if the digit is within the pre-defined range and tolerance, the digit is recognized as the correct digit, otherwise, it will not be recognized. This method is simple and is considered sufficient in some cases. However, the accuracy of recognition of this kind of method is quite low in general since the ranges and tolerances for every digit has been pre-defined and it is uneasy to change. On the other hand, with machine learning, the ranges and tolerances for the digits would vary accordingly with the dataset given to it. With enough dataset as inputs, the machine will be able to develop a model which best suits the case. Unlike the method mentioned before, the accuracy of correctly recognizing the digits is higher with machine learning, since the machine learns how most of the people would write the digits in general and improves the model.

Section 2: Tools, Software and Dataset

Firstly, Python is the language used for creating the machine learning code for this project. Python has been chosen not only because it is relatively easy to learn and use for people with programming experiences, but also the fact that it has a wide community and a number of great libraries designated for machine learning which makes it the best choice for this project.

On the other hand, libraries and tools including Sklearn, NumPy, MatPlot, and Jupyter notebook are used in this project. To start with, Sklearn is one of the most powerful and useful libraries for machine learning in Python, it contains a lot of efficient tools for statistical modeling and machine learning, such as regression, classification, and clustering……etc. In this project, libraries including "train_test_split", "load_digits", "metrics" and "svm"……etc are imported from the Sklearn. For concision, the definitions of and functions in these libraries will not be provided in this report. Furthermore, for the classification of the digits, the digits will be stored and analyzed in the form of arrays, therefore, the library "NumPy" is chosen. "NumPy" is a Python library used for working with n-dimensional array. It also has functions for working in the domain of linear algebra, matrices and Fourier transform……etc, which in general quite useful for this specific machine learning project. Another Python library called "MatPlot" is chosen for creating 2D plots for the digit array, including the confusion matrix and accuracy plot (accuracy vs k value). Moreover, Jupyter notebook, an open-source IDE, was chosen for this project. Jupyter notebook is more versatile compared to other IDE such as PyCharm, and it has the ability to perform data visualization in the same environment. Besides, it allows users to shared live documents and codes, which makes it very attractive for this machine learning project and it allows the group members to work efficiently together.

As mentioned in Section one, the dataset used in this project comes from the MNIST database. It is one of the largest and most popular databases of handwritten digits that is widely used in the field of machine learning and image processing. It has been used for years and proven to be suitable and useful for the machine learning topic of recognizing handwritten digits. In term of the result of this project, which will be discussed further in Section 3, confirmed the fact that the dataset from MNIST is indeed a great choice for training machines to recognize handwritten digits.

Section 3: Discussion of ML Techniques

Results and discussion section should include a discussion and reasoning on the 2 ML techniques you used (reasoning could be as easy as these X and Y techniques provided better results). In addition, you have to provide figures (2 minimum) and results that may include but are not limited to, F1 score, AUC, accuracy, and confusion matrix. (2 pages, 10 marks).

SVM and KNN:

Support-Vector Machine (SVM) and K-Nearest Neighbors (KNN) are the two machine learning techniques used. Both SVM and KNN are supervised learning algorithms, meaning "they rely on labeled input data to learn a function that produces an appropriate output when given new unlabeled data" [1]. SVMs are very simple yet powerful algorithms that produce results with significant accuracy and less computation power. SVMs can be used for both regression and classification tasks but is widely used in the classification of objects. "The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space (N – the number of features)" [2] in order to separate and classify the data points. The dimension of the hyperplane will depend on the number of features or classes. The hyperplane chosen should maximize the margin distance between data points of classes. Maximizing the margin distance makes it so that future data points can be classified with more confidence. These hyperplanes can be thought of as decision boundaries.

KNN is also a simple machine learning algorithm used to solve both regression and classification problems. The KNN algorithm follows the belief that similar data will exist in close proximity to one another. Although there are other ways of calculating the distance, the one we chose to use, and very popular and familiar choice, is the Euclidean distance. As the name would imply, the K value is selected by running "the KNN algorithm several times with different values of K and choosing the K that reduces the number of errors [encountered]" [1] while maintaining accuracy in predicting data.

SVM and KNN were chosen because they both solve classification problems such as our handwritten digits MNIST dataset. Classification problems have discrete outputs with no middle ground. Both are also simple to implement algorithms. These are also two approaches to problems that are quite different, and we wanted to see if there was an algorithm that performed better than the other.

SVM Results:

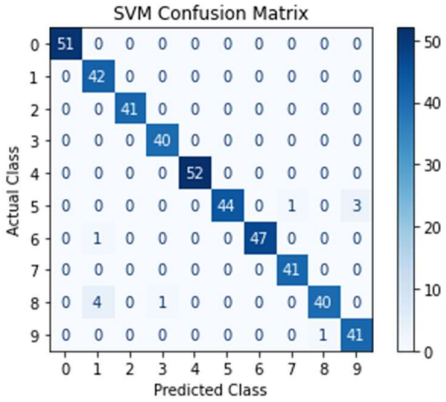

Figure 1: Confusion Matrix for SVM Model

From the confusion matrix, it can be seen from Figure 1 that the algorithm performs quite well, making mistakes only when predicting the digits 5, 6, 8, and 9.

Figure 2: Precision, Recall, and F1-Score for SVM Model

Precision quantifies the number of correct positive predictions made and thus calculates the accuracy for the minority class. Knowing this, when looking at the precision score for the SVM model, it can be seen that the model has the most difficulty with the number 1, often confusing it with the numbers 6 and 8 as seen in the confusion matrix. Overall, the SVM gets an average precision score of 0.975 out of 1.

Recall quantifies “the number of correct positive predictions made out of all positive predictions that could have been made” [3]. Thus, recall provides insight into the missed positive predictions. It can be seen that the SVM model has problem with the digits 8 and 5 the most. Overall, the SVM gets an average recall score of 0.976 out of 1.

The F1-Score is simply a combination of both precision and recall into a single measure. “Alone, neither precision [nor] recall tells the whole story” [3]. Traditionally, the F1-Score is calculated as follows:

$$F - Score = \frac{2 \times Precision \times Recall}{Precision + R} \quad (1)$$

As seen in the figure above, the SVM model overall performs quite well with an average F1-Score of 0.975, but there is clearly room for improvement.

Although accuracy is not the most reliable measurement for assessing machine learning models, it can be noted that the SVM got a very good accuracy score of 0.975555…

KNN Results:

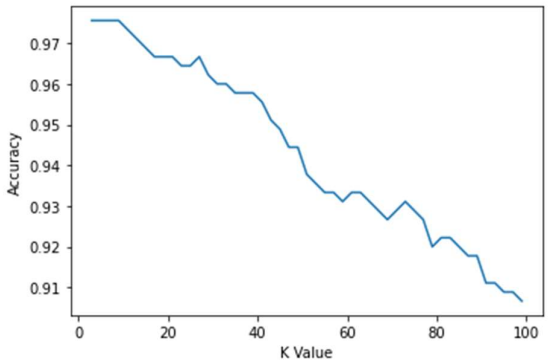

Figure 3: Graph of KNN Accuracy for Different K Values

When looking at the accuracy of the KNN model as K increases, it is clear that the model becomes less and less accurate as K increases, and our highest accuracy occurs when K is 3. Therefore 3 was chosen as the K value in the model.

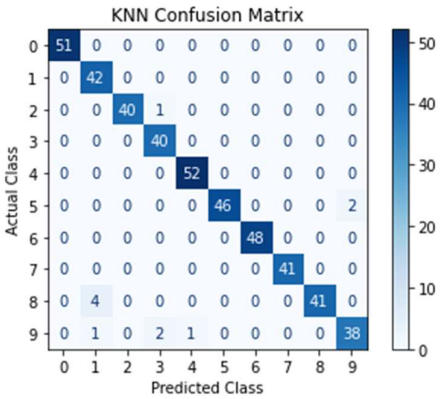

Figure 4: Confusion Matrix for KNN Model

When observing the confusion matrix for the KNN model in Figure 4, it is clear that it performs similarly to the SVM model making mistakes in only a handful of digits. Those digits being 1, 3, 4, and 9.

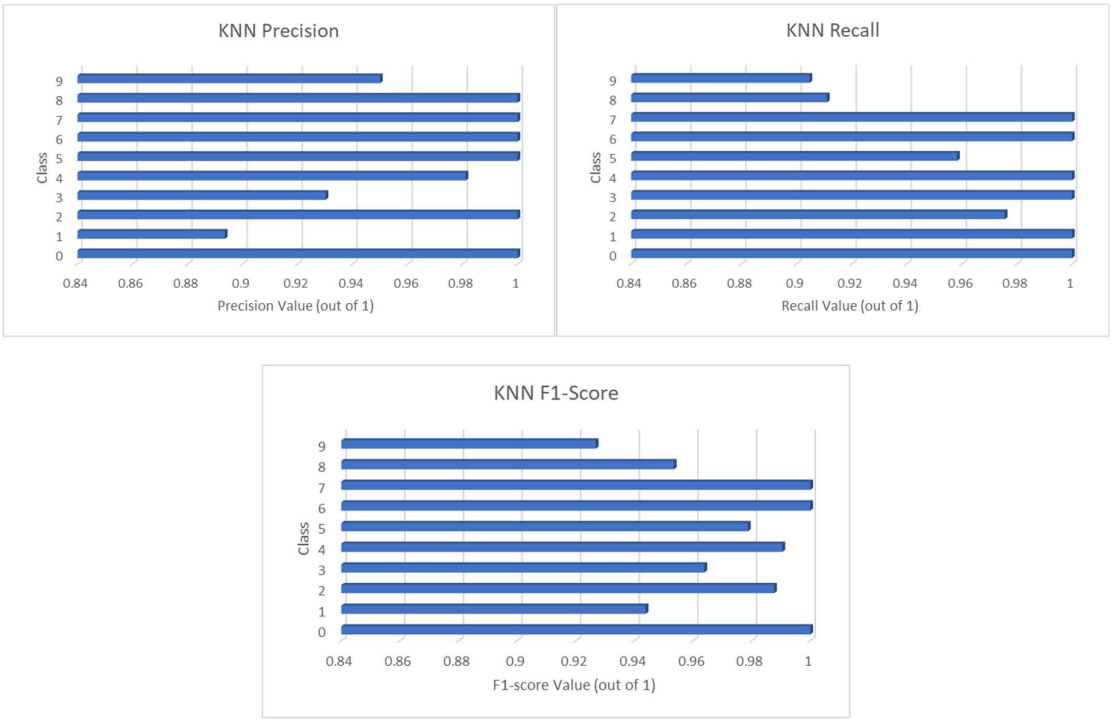

Figure 5: Precision, Recall, and F1-Score for KNN Model

The KNN model, like the SVM model, also makes the most mistakes with the number 1, confusing it with the numbers 8 and 9 as can be seen by the confusion matrix in Figure 4. The KNN model obtained a slightly higher overall score of about 0.976 out of 1 compared to the SVM's 0.975.

With an average recall score of about 0.975 compared to SVM's 0.976, the KNN model performs slightly worse in this area. It had the most trouble with the digits 9 and 8 instead of 8 and 5.

As seen in the figure above, the SVM model overall performs quite well with an average F1-Score of approximately 0.975, but there is clearly room for improvement.

Combining these two, the KNN model obtained an F1-score of approximately 0.974. This is less than SVM's F1-score of 0.975, suggesting that the SVM model performs better compared to the KNN model. However, the KNN model received the same accuracy score of 0.975555…

Section 4

The summary and conclusion section should wrap up the project findings and insights (half page, 5 marks)

In summary, although the approach used to solve problems by the two models are different, they obtained very similar results. As mentioned in the previous section, the SVM model obtained a slightly higher F1-Score which would suggest that it is a slightly better model when compared to the KNN model. Although the SVM performs slightly better, "it comes at the cost of significantly increased training time and the additional complexity of selecting a kernel function" [4]. It should be noted however that once the model is trained, this disadvantage is not much of an issue. From the figures and results show in Section 3: Discussion of ML Techniques, the models seem to perform relatively well considering their respective computational complexity and make few mistakes. However, it can also be seen that there is room for improvement because of these mistakes. After some research, apart from simply using more sophisticated algorithms, machine learning models generally increase in performance and accuracy through a variety of techniques such as combining multiple ML techniques, cross validation, or simply increase the size of the dataset. However, getting scores of above 97% is already considered very good.

References

[1] O. Harrison, "Machine Learning Basics with the K-Nearest Neighbors Algorithm," towards data science, 10 September 2018. [Online]. Available: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761. [Accessed 8 March 2021].

[2] R. Gandhi, "Support Vector Machine — Introduction to Machine Learning Algorithms," towards data science, 7 June 2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. [Accessed 8 March 2021].

[3] J. Brownlee, "How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification," Machine Learning Mastery, 3 January 2020. [Online]. Available: https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/. [Accessed 8 March 2021].

[4] B. Levinson, "Comparing classifiers on the MNIST Data Set," Ben Levinson, 20 December 2017. [Online]. Available: https://benlevinson.com/projects/comparing-knn-svm-mnist. [Accessed 9 March 2021].