Recognizing hand-written digits with K-nearest neighbours, Support Vector Machine and Random Forest classifiers.

Introduction

A friend of mine that is not too into technology asked me the other day how can it be that computers cannot pass CAPTCHA generated tests. CAPTCHA is an acronym for Completely Automated Public Turing Test to Tell Computers and Humans Apart (Captcha.net, 2021). While there is clearly an obstacle for computers to pass CAPTCHA's, I wanted to try how machine learning would compete in recognizing distorted images of words.

Problem Formulation

An application to machine learning problem is quite straight forward. As data I will use the MNIST dataset of handwritten digits obtained from Kaggle. Dataset is available at: <https://www.kaggle.com/c/digit-recognizer/data>. The MNIST dataset provides two datasets: "train" as well as "test" in a .csv file format.

Datapoints in this problem are images of integers ranging from 0-9. Below is a sample image of handwritten digits in the dataset. As we can see from the sample, the dataset allows a bit of distortion between the images, which is suitable for the ML problem at hand when considering computers ability to recognize distorted images of letters.

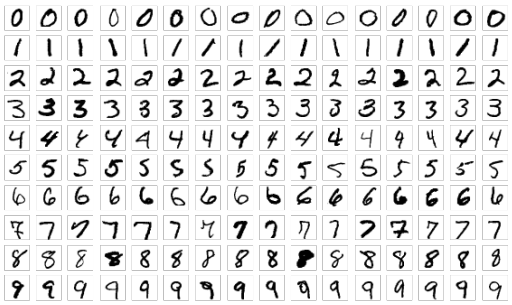

Figure 1. Wikimedia. Example of MNIST dataset.

Each datapoint is represented through an array of pixels. Each datapoint is an 28x28 image of a digit. This results in 784 pixels in total for each image. A pixel consists of a single integer value between 0-255 indicating lightness or darkness of that pixel. 255 is black, and 0 is white. Therefore, a datapoint is an image of a handwritten digit, and its features are an array of integers between 0-255 with a length of 784.

A label that we are interested is an integer between 0-9.

The training dataset provides 785 columns. The first column is reserved for an integer value between 0-9 indicating the label of that column. The rest 784 columns contain pixel values of the associated image, representing the features. The rows i.e., number of datapoints provided in the training dataset is 42000.

Method

I split the training dataset into training and validation sets with the rate of 80/20. For the sake of my computer, I'll utilize 30000 datapoints despite the dataset offering 42000 datapoints. I'll analyse the performance of a model with evaluating accuracy of each of these sets. The ML problem at hand is a classification problem. I'll test 3 different types of classifiers. With KNN, I'll measure with different 8 different values for neighbours. This results to 10 different classifiers. For each I'll evaluate their performance on the data. Evaluating the performance can be measured through different means. I'll measure a simple accuracy score of a classifier. In addition, I'll utilize scikit-learn's tools which with I can easily construct a confusion matrix as well as classification report. Also, with scikit-learn I'll be able to measure the performance of a classifier by recording the time spent on the problem for each different classifier. Therefore, we'll have throughout data which with to compare different classifiers.

First model I'll consider is K-nearest neighbours. I'll use scikit-learn as a tool to implement the classifier. The measure the Euclidean distance between two vectors in a feature space.
For K-nearest neighbours I'll measure the performance of the classifier in different number of neighbours through iterations. I'll iterate k values from 1 to 8. With k=1 I'll be confident to expect overfitting, which has to be considered when analysing and comparing training and validation error.

As mentioned in Ch. 3.13 of [MLBasicsBook], nearest neighbour methods' hypothesis space relies on training data. There is no specific loss function which with to evaluate the quality of hypothesis on K-nearest neighbours. Though, through performance evaluators such as accuracy, as mentioned above, we can evaluate the performance of the classifier.

Another model that I'll evaluate is Support Vector Machine (SVM). Again, I'll use scikit-learn. As a loss function to measure the quality of this hypothesis a Hinge-loss is used. This allows to use a ready-made library for implementing SVM, which is relatively straight forward in scikit-learn.

Also, with SVM, I'll be able to split the data in k-fold cross validation. In K-nearest neighbours I did not feel confident in other split than a single split, since I iterated through a number of different values for neighbours. Adding a k-fold cross validation into the mix is very demanding considering computing power. In constructing dataset for SVM I'll cross validate the data in a 5-fold manner. This'll negate any unlucky splits that could happen in a single split.

A third model that I'll try on my ML problem is Random Forest Classifier. Similarly, to Support Vector Machine, I'll split the data in k-fold cross validation with k-value of 5. To measure the quality of the hypothesis I'll simply measure the accuracy of the model. To obtain accuracy I can rely on scikit-learns' scoring tool which with to obtain accuracy percentages. To measure quality of a split I'll use the default measure offered by the library with is to measure Gini impurity.

```
SVC training scores:  [0.98725    0.988125   0.987875   0.98795833 0.988375  ] mean:  0.9879166666666667
SVC validation scores:  [0.973      0.97216667 0.97316667 0.9695     0.96933333] mean:  0.9714333333333334
```

Figure 2. Training and validation scores for each cross validation of SVC.

```
Random forest training scores:  [1. 1. 1. 1. 1.] mean:  1.0
Random forest validation scores:  [0.95575 0.9545  0.9575  0.95425 0.95375] mean:  0.95515
```

Figure 3. Training and validation scores for each cross validation of Random forest with 100 trees.

```
KNN training scores:  [1.0, 0.9810625, 0.982375, 0.97834375, 0.97684375, 0.9750625, 0.97440625, 0.97321875]
KNN validation scores:  [0.969, 0.9605, 0.967875, 0.966, 0.96625, 0.965875, 0.964625, 0.9645]
```

Figure 4. Accuracy of KNN with number of neighbours running from 1 until 8

Results

In the following, I'll discuss results of each figure.

Figure 2.
Support Vector Models' average training accuracy 98.79% and validation accuracy 97.14%. Which implicates a difference of 1.65% in training and validation accuracy.

Figure 3.
Random forest performed as follows. Its training accuracy was 100% which could denote overfitting. Random forest obtained an average validation error of 95.515% which results in a difference of 4.485% in training and validation accuracy.

To avoid overfitting, I tried a smaller hypothesis space by using a shorter decision tree as suggested in Ch. 6.6 of [MLBasicsBook]., I tried lowering the number of trees to 25, which resulted in training accuracy just shy of 100%. Validation error in this instance was 94.44% The difference between training and validation accuracy rose and did not fix overfitting. While the performance of the model is quite accurate, in a fear of overfitting I'll discard this model from consideration.

Figure 4.
K-nearest neighbours performed well. As can be seen from the figure, with growing the number of neighbours, overfitting could be avoided. Models after k-value of 3 onwards perform pretty similarly. For instance, model with k-values 7 and 8 had training accuracy of 97.44% and 97.32% respectively. Validation accuracy with k-values 7 and 8 were 96.46%, 96.45%. This means difference in accuracies between training and validation sets were 0.98% and 0.87%.

From the models, I'll choose Support Vector Model, since it obtained the best training accuracy as well as validation accuracy. However, it did obtain a bigger difference in the between training and validation accuracies than KNN. Nevertheless, it's validation accuracy is still greater than K-nearest neighbours' and therefore I'll consider SVM to use as a model in testing my data. Neither model indicates overfitting, since the difference of accuracies between training and validation sets are less than one percent. I'll consider the difference as acceptable and apply the model to test data.

	precision	recall	f1-score	support
0	0.97	0.99	0.98	184
1	0.99	1.00	0.99	213
2	0.97	0.98	0.97	194
3	0.99	0.98	0.98	209
4	0.97	0.98	0.97	208
5	0.97	0.99	0.98	184
6	1.00	0.98	0.99	176
7	0.97	0.97	0.97	210
8	1.00	0.98	0.99	205
9	0.96	0.96	0.96	217
accuracy			0.98	2000
macro avg	0.98	0.98	0.98	2000
weighted avg	0.98	0.98	0.98	2000

Figure 5. Classification report for test data using SVM.

In a test data completely separate from the datasets utilized before, SVM resulted in accuracy of 97.90%. This is very close to the validation accuracy which was 97.14%. A good performance on the test data was obtained. This can also be seen from the Figure 6, in which the images of the handwritten digits are pictured with the label value associated to them above. Figure 5 illustrates the classification report for the test data

using Support Vector Machine as a model for predicting the numbers associated with pixel data.

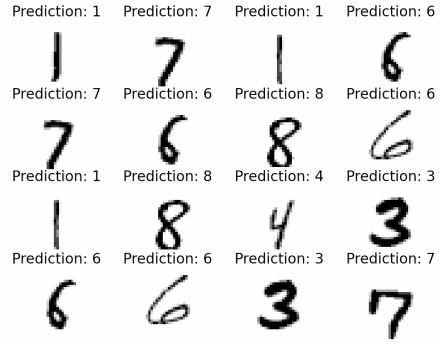

Figure 6. Test data's images and their predictions.

Conclusion

In this report we have studied 10 different models for learning the optimal hypothesis which with to estimate the value of a handwritten digit. 8 of these models are based on K-nearest neighbours classifier with 8 different k-values for the number of neighbours. In addition, Support Vector Model as well as a Random Forest classifier were tested. For the latter two, SVM and Random Forest, I used k-fold cross validation in choosing the model. From these models' SVM and k-nearest neighbours with k-value of 7 and 8 were chosen to be the best models. At the end, SVM was chosen. With SVM, an accuracy on a test set of 97.90% was achieved.

With Random forest we obtained fairly accurate predictions, but the model was discarded in a fear of overfitting. Random forests' training accuracy was 100% and validation accuracy 95.5%. These are comparable metrics to the previous two models, but a training accuracy of 100% could indicate overfitting and therefore the model was discarded from contention.

The chosen ML hypothesis predicted handwritten digits with ~98% accuracy. In Kaggle, where the data was obtained from, the most accurate predictions are obtained with utilizing neural networks. It seems that, while the models evaluated in this report are fairly accurate, even greater accuracies can be obtained with neural networks. However, I identify that the model presented in this report predicted only a single digit. In CAPTCHA's there is the addition of 28 different letters all in various different shapes. This provides even further challenges that evidently no machine learning method can solve.

Models utilizing neural networks could be considered to be good candidates for developing further in striving for solving the ML problem presented of recognizing CAPTCHA. However, a ML algorithm that could obtain 100% accuracy in recognizing CAPTCHAs could lead to further security issues, since it is a security measure recognized and used worldwide in different contexts.

References

Captcha.net, 2021. CAPTCHA: Telling Humans and Computers Apart Automatically. Available at: <www.captcha.net> Retrieved: 17.3.2021

Figure 1. Josef Steppan, 2017. A few examples of MNIST test dataset. Wikimedia. Available at: < https://commons.wikimedia.org/wiki/File:MnistExamples.png>. Retrieved: 17.3.2021

Alex Jung, 2021. Machine Learning: the Basics. (Version: 8.2.2021)