MLDM Project: Handwritten Digits Recognition

Tu My DOAN - Eric DASSE - Karthik BHASKAR

Abstract

This is a report for our machine learning and data mining project which focuses on different algorithms for classifying handwritten digits such as K-nearest Neighbors (KNN) using freeman code, combination of Convolutional Neural Network (CNN) features and KNN, application of Metric learning, and finally Frequent Sequence Mining.

1. Introduction

The main objective of this project is to implement different classification algorithms like KNN, CNN and several techniques in order to improve the speed and the quality of the algorithms. To make things easier, we also provide a GUI for testing purposes.

In the following sections, we will go into details each of the topic above, also give testing results, provide some evaluations and future improvement suggestions of our implementations along with the task list and references we used for this project.

Jupyter Notebooks and python files are provided with implementations and results.

2. Dataset

For the dataset, we used MNIST dataset. Because the purpose is to recognize our own hand-drawing digits correctly especially during the demo so we have decided to draw our own digits (12) (700 digits from 0 to 9) and combine them into the MNIST training and testing. Figure 1 shows some examples from our own drawing digits set.

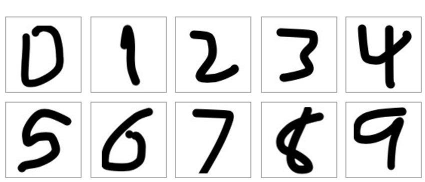

Figure 1. Sample hand-drawing digits

3. Data Reduction

For data reduction technique, we applied the Condensed Nearest Neighbors algorithm from `scikit-learn` to reducde the number of original training dataset of 60,500 examples (60,000 from MNIST and 500 from our own data). At the end we obtain a new dataset of 3,037 examples.

The reason of using CNN to reduce dataset is because when we apply KNN to classify the test digits, the algorithm has to calculate the distance among the test one with all of the digits in our training set and with the large number of our training examples, it will take a huge amount of time. So in order to tackle this, CNN is the right solution.

The reduced dataset is only used for KNN algorithm not for the Convolutional Neural Network as the training is different.

Figure 2 shows the distribution of different digits after the reduction. The number of digit three and eight are kept more than the others and somehow this also affect the accuracy of our KNN when predicting. We will discuss more about this in the Results section.

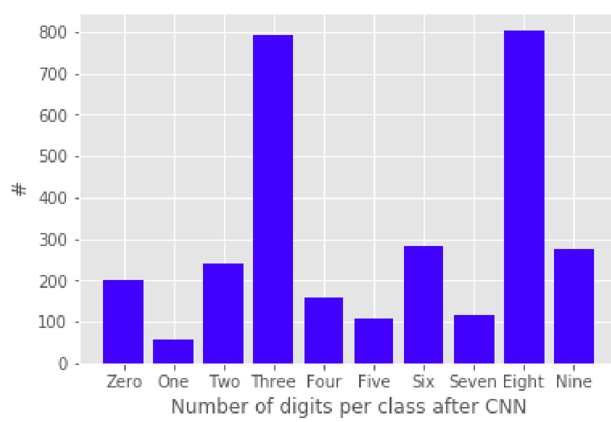

Figure 2. Distribution of digits after Condensed Nearest Neighbors

4. Classification Algorithms

In this section, we will discuss about different algorithms used for classification digits and some improvements we used to speed up the predicting time and the accuracy.

4.1. K-Nearest Neighbors with Freeman code

We represent the digits in a structured way by encoding digits into Freeman chain code and define cost function for the change of directions in the Freeman code. Edit distance algorithm was used with the help of k-strip (18) in order to find the best cost for the compared Freeman chain strings. At the beginning, we implemented Dynamic Programming in order to calculate the cost but it took extremely long to compute so we used k-strip instead to speed up the calculation.

First, the algorithm will convert the drawing image into Freeman chain code and compute the distance among this test digits with other 3,037 Freeman chain code examples in the training set. Here we set number of nearest neighbors to 5. The test digit will be assigned the label of the nearest digit which has the least cost among the 5 nearest costs. Figure 3 show an example of digit 5. As we can see that it took around 42 seconds to output the result. Actually, when we tried to test the accuracy of the algorithm with our own hand-drawing digits, the prediction time will be ranging between 30 to 60 seconds which is a bit long.

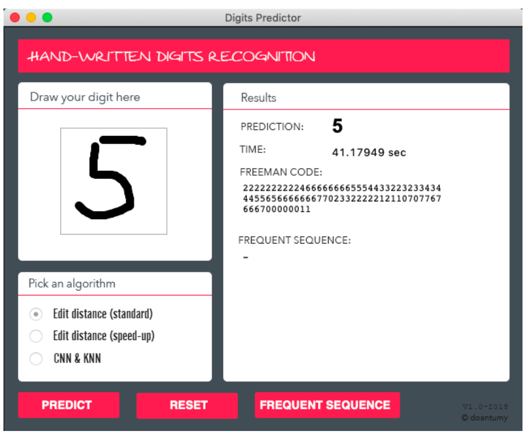

Figure 3. Prediction result with classic KNN

4.2. Speed-up KNN with K-means

With the predicting time around 30 to 60 seconds, it might not be practical in reality so we have to think of some other ways to speed up the process. At first, we tried to applied the algorithm in our lecture in first semester to speed up KNN taking advantage of triangle inequality property. But due to the randomness of choosing first 2 examples to compare the distance, the number of examples that we were able to remove were not stable hence the prediction time was not improved much. The implementation of this is still available in the Jupyter notebook for checking.

Because of the problem above, we have to find another solution. Knowing that in order to speed up the KNN we have to reduce the number of examples that will be compared the cost with the test example and there is one way to do so.

Instead of calculating the distance among the test example and all of examples in our dataset, we used K-means to cluster the condensed dataset into 10 different clusters and obviously, the result is not really good because we will have different digits in the same cluster (5).

When a new test example comes in, we will calculate the distance of this item with all of the 10 cluster centroids to find out which one is the nearest. Then we pick all examples in that nearest cluster, convert them into Freeman code and do the prediction with test item Freeman code.

Compared to the classic KNN with Freeman code on all examples, this will significantly reduce the processing time by 5-7 times.

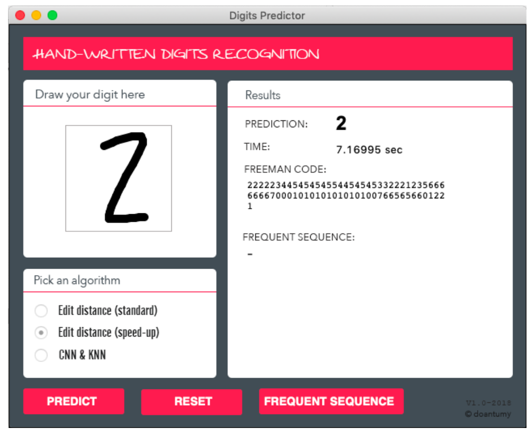

Figure 4. Prediction result with speed-up KNN

4.3. Convolutional Neural Network (CNN) features and KNN

4.3.1. CNN MODEL

To learn model with CNN, we first learned model on the MNIST dataset and then used that pre-trained model on our own data. At the beginning, we didn't add 2 `BatchNorm` layers into the model when training on our own data and the test results were not good so we improved the model and train on all layers of the pre-trained model to obtain a better result. Figure 5 shows the architecture of our CNN model with 2 added `BatchNorm` layers.

4.3.2. CNN FEATURES AND KNN

Once having the output model, we will pop out the last layer which is accounted for the prediction and feed in our own training data to get the feature outputs which will be used for KNN algorithm later (Figure 6).

To learn the classifier for KNN we utilized

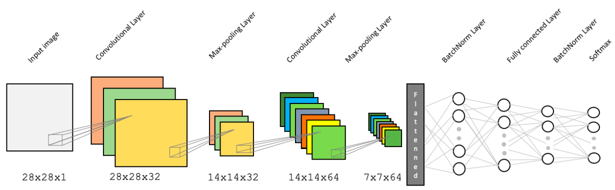

Figure 5. CNN model

`scikit-learn` package and did the grid-search to find the best parameters. Details about the implementation is available in the Jupyter notebook.

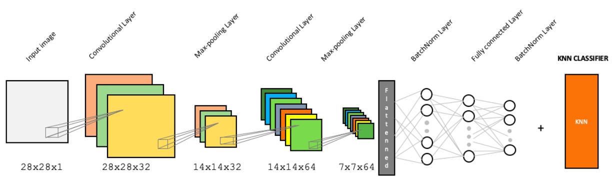

Figure 6. CNN features and KNN

If with the classic KNN using Freeman code, it took around 42 seconds to predict the same digit 5, now with CNN feature and KNN, we would only need 0.1 second to get the result.

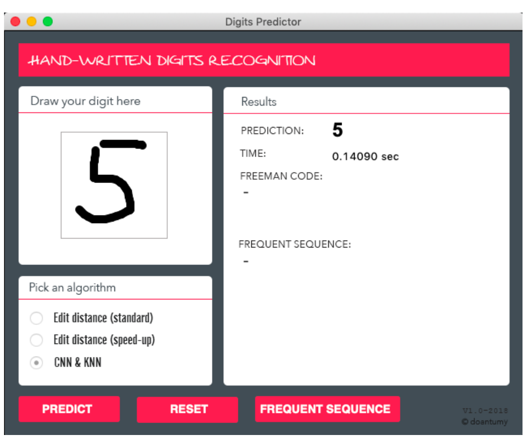

Figure 7. Prediction result with CNN features and KNN

4.4. Results

In this section we will discuss about the accuracy results of above algorithms.

- Classic KNN: We got 82.5% accuracy on our test set of 200 examples. Here we only tested on our hand-drawing digits because the purpose is to predict our hand-writing not the MNIST. However, as mentioned earlier, the main drawback of this algorithm is the long predicting time.

Algorithm	Accuracy
Classic KNN	82.5%
Kmeans and KNN	66.5%
CNN features and KNN	94.5%
LMNN and KNN	97%

Table 1. Summary of our results

- K-means and KNN: Accuracy for this dropped to 66.5% but we successfully speed up the algorithm by 5-7 times. Not surprisingly, there is a trade-off between the speed and accuracy. Sometimes the result is not as good as the prediction on all examples.

  Moreover, because the imbalanced numbers among all digits in the condensed dataset, sometimes we got wrong prediction because some digits outnumber the others making the prediction inaccurate (this can be checked in the testing result outputs in notebook). Figure 8 is an example of wrong prediction because of that.

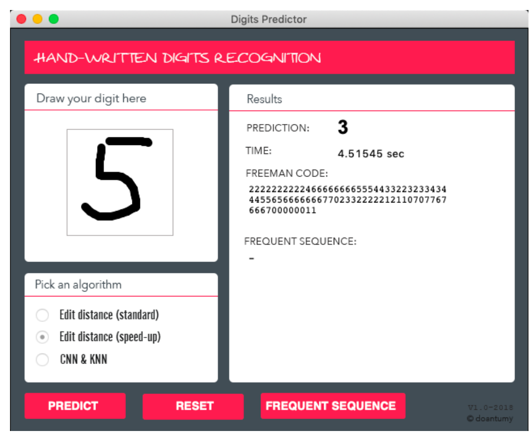

Figure 8. Wrong prediction with speed-up KNN for same digit 5

- CNN features and KNN: With this model we got very high accuracy at 94.5% on test set . Actually, this makes sense as CNN is known as one of the best algorithms to predict MNIST. Normally, we can easily get accuracy of 99% on MNIST dataset with softmax layer. Even when we only work with the features of CNN, the result is quite good.

- Large Margin Nearest Neighbors (LMNN) and KNN: As part of the project is to implement different kinds of improvement techniques so we also tried to apply LMNN (22) on the features output of CNN on our train examples. As expected, after applying LMNN, our KNN algorithm gave highest results at 97%.

5. Frequent Sequence Mining

5.1. Prefix Span

5.1.1. Algorithm

To mine frequent sequence, we used the `pymining` package with small modification based on the references (9; 10). For example, if we want to find all the frequent sequences of those 2 strings: `('124','128884')` with `minimum support=2`. With the original `pymining` we will get below results:

```
[(('1',), 2), (('1', '2'), 2),
(('1', '2', '4'), 2), (('1', '4'), 2),
(('2',), 2), (('2', '4'), 2), (('4',),
2)]
```

If we notice that the algorithm found that `('1', '2', '4')` appears twice in the strings doesn't matter how many characters stay between them and this might affect our frequent sequence for digits.

By making small modification in the original code that only accept the distance to be at most 2 characters, we won't see the `('1', '2', '4')` anymore because there are more than 2 characters between `('1', '2')` and `('4)`.

```
[(('1',), 2), (('1', '2'), 2),
(('2',), 2), (('4',), 2)]
```

If we change second string into `'12884'`, we will get `('1', '2', '4')` as our frequent sequence and also `('2', '4')`.

```
[(('1',), 2), (('1', '2'), 2), (('1',
'2', '4'), 2), (('2',), 2), (('2',
'4'), 2), (('4',), 2)]
```

5.1.2. Mining Process

The running time of the algorithm will be very long if we run it on the original dataset, to save the hassle we ran it on the condensed dataset. Outputs will be save in `pickle` files for later use. It depends on the number of examples in each digit, the `min_support_thres` and the length of the freeman code that processing time will take longer.

5.1.3. K-medoids

Because the outputs of the frequent sequence will be at many different length and we are not interested in the small length, so we get the max length among them all to filter and keep only the sequences of that length. Sometimes, we only got one sequence that has same length with max length.

With all the sequences that have the same max length, we just want to pick one of them to be the most frequent sequence of a digit, so we use `k-medoids` algorithm (8) to select the most center string. Actually, there might be more than one group of frequent sequences but here to simplify the problem, we only use 1 cluster for `k-medoids` and maybe in the future we can consider more frequent sequences as an improvement.

Once we have the most frequent sequence for each digit, we need to display them on our test digit. In reality, we rarely find the exact frequent sequence on our test digit. To tackle that, we will find only the most similar one by using the package `fuzzywuzzy` (13) which is very popular in NLP tasks and keep the most similar string to show on test digit (Figure 9).

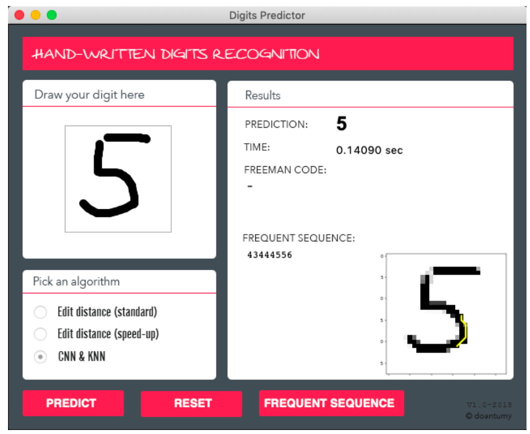

Figure 9. Display of frequent sequence on test digit

5.2. BIDE algorithm

5.2.1. Algorithm

The implementation presented here is inspired by McBurger implementation on Kaggle (6) which we proceeds to alter to fit our needs (discussed in the lines below). The major modifications consist in a brand new pre-processing to handle the variance of the pixel values and the presence of noise in some images (see figure 10). The algorithm is described as follows:

1. Preprocess the image (using the function `image_smoother` provided in the file `preprocess_image.py` of the project
2. Locate the first dark pixel of the image (it's now the current pixel)
3. From the current pixel, loop through each direction (tested according the previously taken direction) until you find a dark pixel
4. Save the direction associated to the new-found pixel at the bottom of the freeman chain code

5. Repeat from the 3rd step till you end up in the start point once again

The pre-processing is further developed in the next section.

5.2.2. Pre-processing

Like mentioned earlier, the freeman chain code represents an object in an image (our digit in this context). However, some instances of digits in the MNIST dataset have other objects than a digit (noise). They often appears as sort of "stains" (see figure 10).

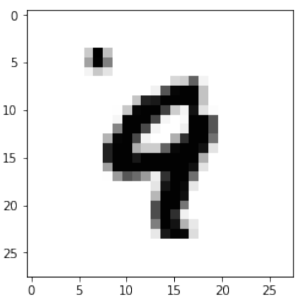

Figure 10. Instance of digit number 9 with noise

To cope with these particular noise, I make use of **median blur filter** (included in the OpenCV library). Intuitively, it makes each pixel looking like its neighbors. Concretely, this method uses a kernel (like CNN). This method scans the image pixel by pixel and replace each by the median of the neighboring pixels. The "stains" are generally surrounded by a lot of white pixels so they just fade away as illustrated in figure 11.

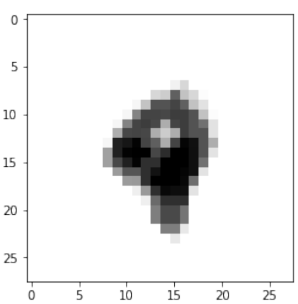

Figure 11. Instance of digit number 9 without noise. Computed with a kernel of size 5

However, this method is not without limit. In fact, the size of stains varies from an image to another. Because of this, the kernel could badly capture it and alter the image more than needed. Moreover, images that do not contain noise could be needlessly altered to the point of even losing all of its pixels (see figure 12). It is always that extreme though leading to short freeman codes.

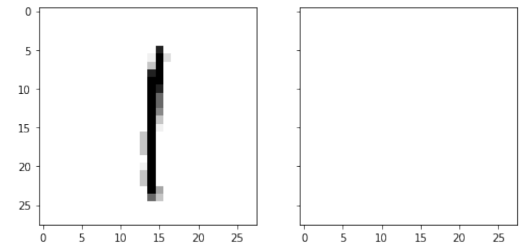

Figure 12. On the left side is the original digit. On the right side is the result of applying median blur with kernel size 5. As you can see, the entire content of the image is obliterated because the digit was too thin and thus mostly surrounded by white pixels

The second preprocessing we perform is thresholding (with **cv2.threshold** provided by OpenCV). It's done to capture the darkest pixels of the image i.e. relevant pixels. This helps capturing the "true" contour of a digit instead of a version disturbed by some light pixels that shouldn't normally be there. The value of the threshold is chosen based on the image itself. To do so, we make use of Otsu's Method to compute threshold (again provided by OpenCV).

Several other methods were considered to address the shortcomings of the preprocessing as it is right now such as what a twisted genetic method which consists in recursively computing Freeman chain code on each cluster of pixels in the image and then select the longest chain code (assuming the biggest element in the image is the digit).

5.2.3. Mining Process

For a given digit, there exists several freeman codes each associated to an instance of the digit in the MNIST dataset. Now, our second mission is to mine the sequential patterns in those digits and display the most frequent (and relevant) out of them all. To avoid redundancy and keep the same expressive power, we mine closed sequential patterns in this project. The algorithm we use is called **PrefixSpan**(3; 1) of which I implement the close sequence mining version (BIDE (2)). We make use of the library prefixspan. Details of the implementaion of the algorithm can be found in the notebook `Sequential Pattern Mining on MNIST dataset`.

6. Evaluation and Improvement

For the quality of implemented algorithms:

- In general, the KNN with Freeman code works at acceptable level but not the best one we can have at the moment. For example, for normal hand-written digits it can predict labels but if user draws in careless handwriting number, algorithm might fail to predict it. Moreover, we have two different versions of the

Freeman code the first one is not tolerable with random noise in image and the second version that can pass by the random noise such as a random dot on image, it can output the Freeman code only for the digit part, not the dot pixels. However, after testing the 2 versions on test set the first one returned better results so we kept it for our implementation (details on testing results are in the notebook).

- For Convolutional Neural Network, the model works better than the others. However, if we have more time, we will draw more digits with different variations so it will predict our hand-drawing better even when users draw digits carelessly.

- Thought we implemented preprocessing functions to handle the "stain" noise in the images while computing Freeman code, it has to be noted that it can be improved using what we will call **genetical methods**. It consists in computing Freeman code on every "cluster" of pixels in the image and select the best out of them. In that case, the best is the longest (assuming the digit is the largest object in the image)

- In this project, we performed K-medoids on the Freeman codes using k = 1 for the convenience. One way to improve it would be to represent the freeman sequences as vectors using *prototype selection* (4) and from a plot, visualize the possible number of clusters.

- Due to the time constrained, we can only create a simple GUI for testing purposes, but later this can be extended as an online version for easier use in Flask.

Talking about the project management within the team, it can be seen that we didn't manage the workload well among team members as expected and this has resulted into heavy work for some individuals. Fortunately, at the end, the work submitted is still the group submission and what really matters is how much we have learned by working on this project as knowledge will take us further in our future career path.

7. Task List

Below is the task lists done for this project:

- My: 57.0%

- Eric: 30.0%

- Karthik: 13.0%

Task lists	Done by	(%)
Dataset drawing	All	9%
Condensed Nearest Neighbors	My	4%
K-strip algorithm	Karthik	7%
Freeman Code	Eric	7%
Classic KNN	My	5%
K-means and KNN	My	5%
Convolutional Neural Net	My	25%
LMNN	My	4%
Frequent Seq. Mining [Prefix-Span]	My	8%
Frequent Seq. Mining [BIDE]	Eric	20%
GUI	My+Karthik	6%

References

[1] Jian Pei, Jiawei Han. Prefixspan: Mining sequential patternsby prefix-projected growth.Proceedings of the 17th International Conference on Data En-gineering, 2001.

[2] J. H. Jianyong Wang. Bide: Efficient mining of frequent closed sequences.Proceedings ofthe 20th International Conference on Data Engineering.

[3] N. R. MABROUKEH and C. I. EZEIFE. A taxonomy of sequential pattern mining algo-rithms.ACM Computing surveys, 43(3):24-26, 2010

[4] Barbara Spillmann, Michel Neuhaus, Horst Bunke, Elzbieta Pekalska and Robert P.W. Duin. Transforming Strings to vectors using prototype selection. Proceedings of the 2006 joint IAPR international conference on Structural, Syntactic, and Statistical Pattern Recognition, 2006.

[5] Shuhei Kishi, Speed up naive kNN by the concept of kmeans - `http://marubon-ds.blogspot.com/2017/08/speed-up-naive-knn-by-concept-of-kmeans.html`

[6] McBurger, Freeman Chain code, Second attempt - `https://www.kaggle.com/mburger/freeman-chain-code-second-attempt`

[7] McBurger, Freeman Chain code - `https://www.kaggle.com/mburger/freeman-chain-code-script`

[8] Letiantian, K-medoids - `https://github.com/letiantian/kmedoids`

[9] Barthelemy Dagenais, Pymining - `https://github.com/bartdag/pymining/blob/master/pymining/seqmining.py`

[10] Austin Schwinn, Pymining - https://github.com/amschwinn/handwritten_digit_recognition/blob/master/pymining.py

[11] Jake VanderPlas, Python Data Science Handbook - Essential Tools for Working with Data, In Depth: k-Means Clustering- https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html

[12] Draw digits website - https://sketch.io/sketchpad/

[13] SeatGeek, Fuzzy String Matching in Python - https://github.com/seatgeek/fuzzywuzzy

[14] Doan Tu My, CNN with MNIST - https://github.com/doantumy/CNN-with-MNIST/

[15] Aditya Sharma, Convolutional Neural Networks in Python with Keras - https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python

[16] Han Xiao et.al. - Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms - https://arxiv.org/abs/1708.07747

[17] Sebastian Raschka, Vahid Mirjalili, Python Machine Learning Book - 2nd edition - https://www.packtpub.com/big-data-and-business-intelligence/python-machine-learning-second-edition

[18] Group Project from 1st semester - https://github.com/ajoseph12/Protein_Alignment_Minimum_Edit_Distance

[19] Keras, Document, Convolutional Layers - https://keras.io/layers/convolutional/

[20] Keras Document - https://keras.io/getting-started/faq/#how-can-i-obtain-the-output-of-an-intermediate-layer

[21] Shibuiwilliam, Keras_Sklearn - https://github.com/shibuiwilliam/Keras_Sklearn

[22] John Chiotellis, LMNN - https://github.com/johny-c/pylmnn