Machine Learning and Data Mining Project
Handwritten Digit Recognition

Ahmed Hassan, Omar Samir, Tho Vo

December, 2015

votuongtho@gmail.com hassan.ahmed@etu.univ-st-etienne.fr
omar.samir3000@gmail.com

Contents

Abstract

Handwritten character recognition is not a new challenge to the fields of computer vision and machine learning. It is a common task in the process of digitizing data, and a mundane effort for a human to carry out. Therefore, efficient and accurate techniques to detect, segment and classify the character are required. We focus on the part of classification of handwritten digits, under the constraint of small dataset. Our approach is to investigate different techniques of the classification pipeline, which falls into two main parts: capturing suitable features for the task, and comparing several features-algorithms combinations to evaluate the whole approach. We obtain accuracies of 94-96% for cross-validation and testing accuracies with strong classifiers such as SVM and Random Forests, on the geometric contour-based Elliptic Fourier Descriptor features. We also discuss the challenges we faced in the analysis, and our conclusion from the results of our experiments.

1 Introduction

Computer vision is one of the most challenging problems faced in machine learning. There are plethora of previous works in the field, providing different approaches to hundreds of different sub problems.

The special interest in computer vision is attributed to the fact that it is involved in daily human life tasks, and since we are in the informatics era; there is a dire need to solve every day challenges in automatics and artificial way, without the need for human intervention.

Also, the explosion of data that happened only the past few years, and is bound to continue exponentially; arises the demand on new methods to analyse and extract, in a very fast and efficient way, the information we want, with accuracies close to those of humans.

However, one common problem with the data being gathered all over the world, is that not all of it is digitized; which poses a great challenge if we want to use machine learning as a way to analyse, and learn from these data.

In this report, we address a part of this problem, better known as Handwritten Character recognition. We concentrate of the sub problem of handwritten digit recognition under the constraint of learning from a small dataset. We investigate how to obtain high accuracy classification of single digits which are handwritten by the user through our Graphical User Interface.

In section 2, we detail the different techniques and approaches we adopt to tackle the problem, going from extracting meaningful features from the data we have, to algorithms we use in classification, and establishing statistical analysis basis upon which we will analyse our experimental results. Afterwards in section 3, we introduce the structure of our dataset, then we talk a bit about how we realize our methodology through our implemented software, and finally, we explain our experimental setup. The results of our experiments are reported on section 4. And finally we list our conclusions and findings in section 5.

2 Method

The methodology that we adopt for approaching the problem of digit recognition in this project, is an investigatory one. We combine several classification algorithms, with different features extracted from the images; in order to compare and assess the advantages and disadvantages of each technique. In the following subsections, the process flow of the project is explained.

2.1 Feature Extraction

Feature extraction is one of the most important steps in the process of machine learning. Using representative features for the task, is essential to be able to describe the data in a way that induces inter-class variability, which leads to easier classification problem, and of course better accuracy.

The features we have chosen to work with are mainly: raw image pixels, encoded image contours in the form of Freeman codes, an Elliptical Fourier Descriptors built upon the image contours as well. We have also tried other features such as image histograms, and normalized chain codes; however, these are not part of the main body of this work

2.1.1 Contours and Freeman Code

Freeman codes belongs to a family of compression algorithms called Chain codes. These algorithms are useful to encode an image when it has connected components inside it. They are considered compression algorithms as they can transform a sparse matrix, to just a small fraction of the size of the image, in the form of a sequence of codes.

Freeman codes have 2 versions, 4-directional codes, and 8-directional codes. Both are fairly simple as they encode each direction with a unique number (from 0 to n-1, where n are the directions). A direction is defined in the image as the directed vector connecting two neighbouring pixels on the contour of a connected component.

In our approach, we have one digit per image, which serves as one connected component. First, we do some preprocessing for the image to make sure the digit is fully connected, and then we extract the out most pixels as the contours of the largest connected component to account for noise. Lastly, convert each direction from the contours to its corresponding Freeman code symbol, as shown in Figure 1. Figure 2 shows the transformation from raw image, to contours, to a Freeman code sequence.

2.1.2 Elliptic Fourier Descriptors

Elliptic Fourier Descriptors method is used to describe the properties of an edge, just like Freeman chain and Hartley descriptors [1]. One of its great advantages is that it is not affected by translation, rotation, or scaling of the image.

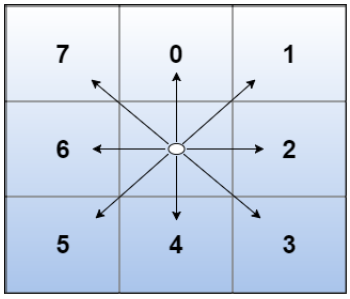
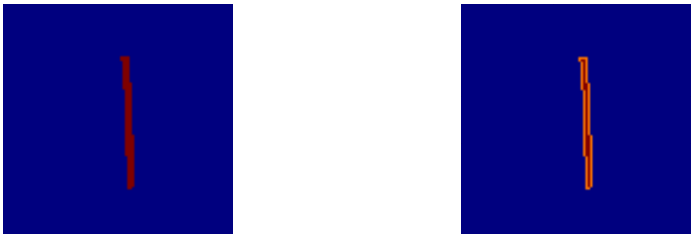


Figure 1: Freeman code representation



(a) Original image (b) Image contours

Freeman code
434444444444434444444444444444444444444444
344444444444442100000000000000000000007
000000000000000000000700000000066

Figure 2: Image Transformation

Assume we have a Freeman chain code. u_i is element i in this code. The length of this element dt_i is:

$$dt_i \ = \ 1 \ + \ (\frac{1}{2})(\frac{\sqrt{2}-1}{2})(1-(-1)^{u_i})$$

If we have n elements in the chain code, then the total length of the contour will be

$$T \ = \ t_n \ = \ \sum_{i=1}^{n} dt_i \tag{1}$$

Assume dx_i and dy_i are the projections of u_i on the X and Y axises. Their values will be

$$\begin{gathered} dx_i \ = \ sign(6-u_i) \ * \ sign(2-u_i) \\ dy_i \ = \ sign(4-u_i) \ * \ sign(u_i) \\ where \ sign(x) \ = \ 1 \ if \ \ x \ > \ 0, \\ sign(x) \ = \ 0 \ \ if \ x \ = \ 0, \\ sign(x) \ = \ -1 \ \ if \ x \ < \ 0. \end{gathered}$$

The Fourier harmonics in this case will be

$$\begin{aligned} a_n \ &= \ \frac{T}{2n^2\pi^2} \ \sum_{i=1}^{k} \frac{dx_i}{dt_i} \left(cos\frac{2n\pi t_i}{T} - cos\frac{2n\pi t_{i-1}}{T} \right) \\ b_n \ &= \ \frac{T}{2n^2\pi^2} \ \sum_{i=1}^{k} \frac{dx_i}{dt_i} \left(sin\frac{2n\pi t_i}{T} - sin\frac{2n\pi t_{i-1}}{T} \right) \\ c_n \ &= \ \frac{T}{2n^2\pi^2} \ \sum_{i=1}^{k} \frac{dy_i}{dt_i} \left(cos\frac{2n\pi t_i}{T} - cos\frac{2n\pi t_{i-1}}{T} \right) \\ d_n \ &= \ \frac{T}{2n^2\pi^2} \ \sum_{i=1}^{k} \frac{dy_i}{dt_i} \left(sin\frac{2n\pi t_i}{T} - sin\frac{2n\pi t_{i-1}}{T} \right) \end{aligned}$$

2.2 Classification

For our classification problem, we have included several classifiers, which take different approaches to the solution, such as K-Nearest Neighbours (kNN), Hidden Markov Models (HMM), Logistic Regression, Support Vector Machine (SVM), Naive Bayes classifier (NaiveBayes), Random Forest, AdaBoost, Gradient Tree Boosting (GBRT). Below is a brief about each of them.

2.2.1 K-Nearest Neighbours

K-Nearest Neighbours (KNN) is a non-parametric method used for classification and regression. Based on the distance of nearest neighbors to a test instance to classify the class of this instance in our program.

In our program, KNN receives a list of Freeman code according with label of each class. KNN will use these list to determine the class of unknown instance by comparing the distance between its Freeman code and the other instances in the training. By using k from 3 to 5, we take in account 3 to 5 closet neighbors to vote for the class of unknown instance.

Theoretically, KNN algorithm is among the simplest of all machine learning algorithms and take a lot of time to run over the sample due to time of calculation the distance between the test instance and the rest training instances.One of disadvantage point of KNN is it is hard to build a model for next prediction, it means for every new test instance, we repeat the whole process that cost time. However, in our experiment which is small with nearly 200 images, KNN still can run quickly and provide very good result in comparison with other algorithms.

2.2.1.1 Edit Distance In this section, we will introduce about how we calculate the distance in K-Nearest Neighbours (KNN) to determine the neighbors of an instance, the distance needed to be performed on two strings so we used the Damerau–Levenshtein Distance to get the distance between two Freeman code in string type.

The Damerau–Levenshtein Distance (DLD) itself (named after Frederick J. Damerau and Vladimir I. Levenshtein) is used to calculate the distance between two strings which fit smoothly with what we did in our project, because after got the contour of an image, we extracted the Freeman code in string type and at that time, we found DLD is the best choice. The idea of DLD to calculate the distance is counting the minimum operation requirements to transform one string to another one, where an operation is defined as an insertion, deletion, or substitution of a single character, or a transposition of two adjacent characters [2]

This work is done by the support of the python package pyxDamerauLevenshtein by Geoffrey Fairchild [3]

2.2.2 Hidden Markov Models

HMM is based on stochastic Markov processes, and the concept of hidden states. HMM's depend on three main components, the states, the transition matrix, and the probability matrix. Since the states are hidden, we can only observe the output state of the system at any given time (or input subsequence). Also, a set of allowed input symbols have to be defined.

In our problem, the output states are defined as the classes, the digits. While the allowed input symbols, are the set of directions of the Freeman code. The transition matrix is initialized to allow transitions between all states, and the probability matrix is initialized uniformly, and updated while training with Maximum Likelihood estimation method.

Theoretically, HMM can be used to classify efficiently the Freeman codes extracted from the digits, as it can build a probability distribution from the

relations between each subsequent symbol in any encoded figure, which is an important factor to be able to separate the codes into different classes.

2.2.3 Naive Bayes Classifier

Naive Bayes classifier is another example of a probabilistic model based classifiers. It assumes strong independence between the features, and from this assumption it tries to build a classifier, which picks the class with the highest conditional probability of the features given their reference classes, as shown in equation 2

We try to exploit Naive Bayes simple probabilistic approach, combined with the raw pixels of the images as our features, to obtain a baseline classification accuracy for reasons of comparison with other classifier/features.

$$\hat{y} = argmax_{k \in \{1..K\}} p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{2}$$

where: $\hat{y}$ is the predicted class, C_k is class label k, and $p(x_i|C_k)$ is the conditional a posteriori probability obtained from training

2.2.4 Adaboost

AdaBoost, introduced in 1995 [4], fit a sequence of weak learners (with performance just above random guessing), while modifying the weights of the data for each classifier. After fitting the weak classifiers, a majority vote is being done to the classifier in order to produce the final prediction.

The weights of the data is being initialized to the same value at the beginning $W = 1/N$. At each iteration in the AdaBoost, the weights of the incorrectly classified training data - in the previous iteration - is increased, and the weights for the training data correctly classified in the previous iteration is decreased.

2.2.5 Random Forests

Random forests is an ensemble techniques that uses decision trees, where each tree is being data from sub-sample of the training data using bootstrap sampling (sampling with replacement) [5]. In the tree construction, the node splitting is the best split on a random subset of all the features.

2.2.6 SVM

Support vector machines construct a hyperplane in high dimensional data in order to a achieve a good separation between the different classes [6]. A good hyperplane separator is the one that has the largest distance to the nearest training data of any class.

In this experiment, we use SVC. In the case of binary classification, let's assume $x_i \in \Re^p$, i = 1,...,n, and $y \in \{1, -1\}$, then SVC can be formulated mathematically as follows:

$$min_{w,b,\zeta}\ (0.5)\ \omega^T\omega + C\sum_{i=1}^{n}\zeta_i$$
$$subject\ to\ y_i(\omega^T\phi(x_i) + b) >= 1 - \zeta_i\ ,\ \zeta_i >= 0, i = 1, ..., n\ ,$$
$$C > 0$$

2.2.7 Gradient Tree Boosting

Inspired by its huge success in different *Kaggle* competitions[1], we decided to give it a try.

Gradient Tree Boosting [7] is a generalization of boosting to arbitrary differentiable loss functions. It was reported in different competitions to have the best predictive power, while being able to handle data of mixed types, and still being resistant to outliers. It main disadvantage however is that it does not scale well (we can't parallelize its execution steps).

Gradient Tree Boosting learn several weak learners, and combine them in a weighted manner to make a decision:

$$F(x)\ =\ \sum_{m=1}^{M}\gamma_m h_m(x)$$

where h_m is a weak learner (a decision tree of fixed size).

Like boosting, it fits the weak learners stage by stage

$$F(x)\ =\ F_{m-1}(x)\ + \gamma_m h_m(x)$$

where h_m is chosen in order to minimize a loss function L,

$$F(x)\ =\ F_{m-1}(x)\ + argmin_h \sum_{i=1}^{n} L(y_i, F_{m-1}(x) - h(x))$$

2.2.8 Logistic-regression Classifier

Logistic regression [5] is a linear model for classification. It was mainly intended to be a baseline classifier during this project.

In case of binary classification, and using $L2$ loss function - which we use in this project -, the mathematical formulation for the logistic regression will be:

$$min_{w,c}||w||_1 + C\sum_{i=1}^{n} log(exp(-y_i(X_i^T w + c)) + 1)$$

[1] www.kaggle.com, a website where regular competitions in machine learning and data science are being held regularly.

2.3 Statistical analysis - Wilcoxon

In order to analyze the results we get from the experiment, we perform a significance test on the results we have - will be discussed in detail in the upcoming section.

Since we don't have a previous assumption about the distribution of the data, we choose the non-parametric significance test, which test the hypothesis on the median. It is a non-parametric version of the paired t-test.

3 Experiments

3.1 Dataset

Our dataset of digits was constructed from scratch, using real handwritten digits by several individuals from inside and outside the project work group.

The dataset is divided into 10 classes, one corresponding to each digit, and it has a total of 250 digits. It is stored as gray scale images, of 100x100 pixels each in JPEG format, and custom read and write operation are implemented to load and modify it.

3.2 Implementation

3.2.1 Software Overview

The software we developed to realize this work is divided into the following modules:

1. GUI main and support modules
2. Algorithms modules:
    - Base module for all algorithms containing basic functions
    - Implementation of each algorithm on the base class
    - Module of data mining implementation
3. Feature extraction modules (Freemancode, Elliptic Fourier Descriptors)
4. Modules for running experiments and obtaining results
5. Support modules for dataset reading, data handling and preprocessing

We relied of several main packages for our implementation, which included:

- Sci-kit Learn python library [8] for machine learning algorithms, metrics, and analysis.
- An implementation for the KNN from our side in order to deal with the Freeman code and the edit distance.

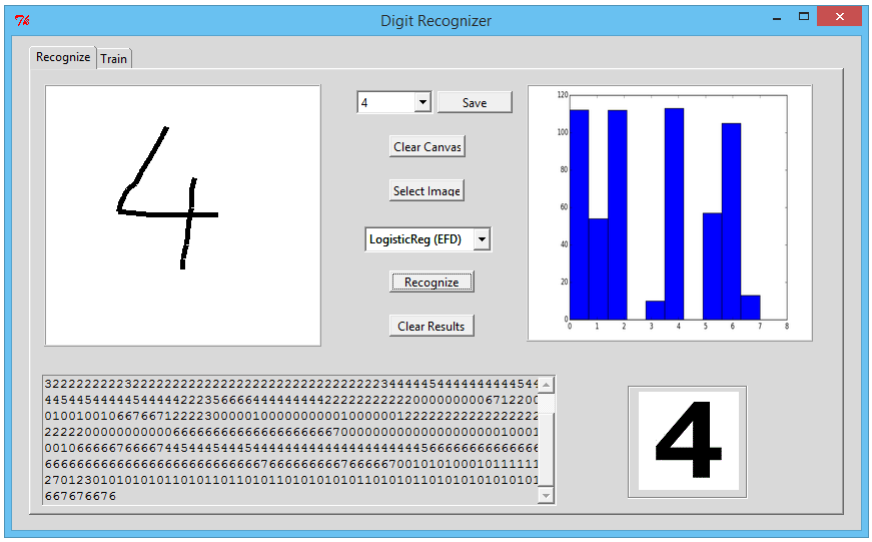


Figure 3: Graphical User Interface

- NLTK library [9] for HMM base classifier.
- OpenCV library [10] for computer vision and image processing.
- Damerau-Levenshtein string edit distance implementation in python [3]
- *Elliptic Fourier descriptors* implementation on python [2].

3.2.2 Graphical User Interface

The GUI, shown in Figure 3 included with this work, is mainly to demonstrate the different classification and feature extraction methods. The following is a brief of the features included:

- Input handwritten digits by drawing on a canvas, or by selecting an already existing image
- Save current input to the dataset of images with label
- Display freeman code and histogram of input
- Predict input digit using the algorithm of choice

[2] https://github.com/alessandroferrari/elliptic-fourier-descriptors

3.3 Early experiments

In the beginning, in order to test the pipeline of tools we would use, and also to have some idea about the possible accuracy with different data representation, we tried the following:

- Using the normal pixels as features
  - Logistic Regression: it achieved around 30% accuracy.
  - KNN: it achieved around 10% accuracy (same as random guessing).
- Bag of words: Based on the freeman code, we created a bag of word representation for each image.
  - HMM: it achieved around 10% - 20% accuracy.

3.4 Experiment

In this experiment, we had the following goals:

- Compare the performance of different algorithms, on two types of features: Freeman chain code and the Elliptical Fourier descriptor.
- Verify that cross-validation is a good estimation for the generalization error [3].
- Identify the over-fitting in the algorithms - if it exists -.

Before we address goals, we need first to tune the best possible hyper-parameters for each algorithm. We perform an extensive grid search for almost the algorithms (except for AdaBoost and Gradient Tree Boosting, due to time constraint).

Then, in order to address each of the previous goals, we observe the following:

- Comparing the algorithms: We collect the cross-validation, the training and the test scores. We make a Wilcoxon significance test between each pair of algorithms for all the three scores. We are mainly concerned with the cross-validation and the test scores comparison between different algorithms.
- Does cross-validation provide a good estimation for the generalization error? We perform cross-validation on the training data, and we test the model on a new dataset. For each algorithm, we make Wilcoxon significance test between the cross-validation and the test scores. We assume that the difference should be either insignificant most of the time.
- How to identify over-fitting? We use two methods in this case:

[3] This wasn't a goal in the beginning. But we had some concerns in the early testing from both teams about cross-validation, which lead to adding this goal

Algorithm/Features	Avg. CV Score	Avg. Train Score	Avg. Test Score	Std. CV Score	Std. Train Score	Std. Test score
AdaBoost/EFD	0.20	0.21	0.13	±0.034	±0.045	±0.055
kNN/FC	0.70	0.89	0.73	±0.023	±0.018	±0.068
GBRT/EFD	0.81	1	0.80	±0.097	±0	±0.051
HMM/FC	0.47	0.51	0.46	±0.011	±0.022	±0.064
Logistic Reg./EFD	0.95	0.98	0.96	±0.012	±0.004	±0.02
NaiveBayes/RAW	0.26	0.9445	0.23	±0.035	±0.018	±0.055
Random Forest/EFD	0.94	1	0.94	±0.012	±0	±0.029
SVM/EFD	0.96	0.99	0.96	±0.007	±0.003	±0.025

Table 1: Algorithms scores

  - Observe the difference between the training error and the test error. If the difference is huge, then this indicate the possibility of over fitting. However, this is not explicit measure.
  - We draw the learning curve for each algorithm (by increasing the number of training instances step by step).

In order to collect the relevant scores for this experiment (cross-validation, testing, training), we do the following:

- For each algorithm, Repeat for 50 times (to get a population sample, for the statistics):
  - Shuffle all the data
  - Split the data into 80%-20%
  - Perform cross validation on the 80%
  - Score the model on the 80% (to get the training error - not for this step in the experiment).
  - Score the resulting model on the 20% (new, unseen data).

The results are presented in the next section.

4 Results

In this section, we report the results of our experiments for each algorithm, as well as the result of comparative analysis of all algorithms

4.1 Results of each algorithm

First, we try to assess some metrics for each algorithm combined with its corresponding input features. Table 1 shows the cross-validation, training and testing accuracy scores of each algorithm. [4]

[4]RAW refers to raw pixels, FC refers to Freeman codes, and EFD refers to Elliptic Fourier Descriptors

Algorithm/Features	CV vs. Train	CV vs. Test	Train vs. Test
AdaBoost/EFD	0.0029	1.677e-08	3.78e-09
kNN/FC	7.55e-10	0.13	7.52e-10
GBRT/EFD	7.55e-10	0.95	7.09e-10
HMM/FC	4.77e-09	0.41	0.00071
Logistic Reg./EFD	7.55e-10	0.207	2.08e-08
NaiveBayes/RAW	7.55e-10	0.002	7.52e-10
Random Forest/EFD	7.55e-10	0.308	1.9e-09
SVM/EFD	7.55e-10	0.62	9.03e-08

Table 2: Individual algorithm statistics

Compared Algorithms	CV	Train	Test
SVM vs. Random Forest	1.75e-09	9e-10	0.00054
SVM vs. Logistic Reg.	6.62e-08	3.41e-08	0.02
Random Forest vs. Logistic Reg.	0.0031	3.72e-10	0.1609

Table 3: Comparative algorithms statistics

We also generate the confusion matrices of the used algorithms (except kNN) to check the strength and weakness of each classifiers on our dataset. Figure 4 shows the confusion matrices as heat maps for easier visualization.

However, to be able to judge the performance correctly, and without bias, we need to use the statistics of these previous results. We use Wilcoxon test, and we show the p-values in table 2

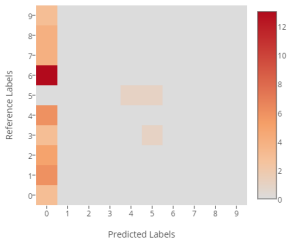
For further investigation, we evaluate the learning process of each algorithm on the dataset, by plotting its learning curves. Figure 5 the sum of the learning curves of the algorithms.

4.2 Comparative results

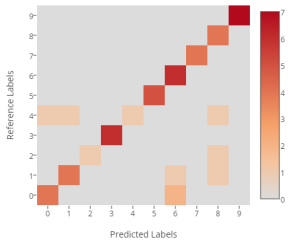
Following the results of each individual algorithm, we report the results of the performances of algorithms against each others. This serves the purpose of understanding which approach is more suitable for the handwritten digit recognition task.

Figure 6 shows the cross-validation, train, and test accuracies in one plot.

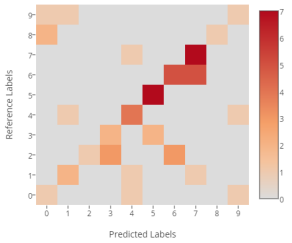
Also, as in the individual analysis part, we calculate the statistics of each pair of algorithms for non-biased judgement, which is in table 3. However, we report the non trivial p-values of algorithm pairs, and all other unreported values are considered trivial.



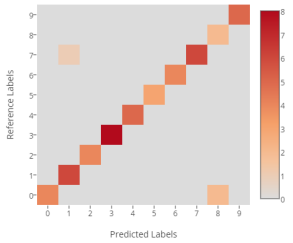
(a) AdaBoost



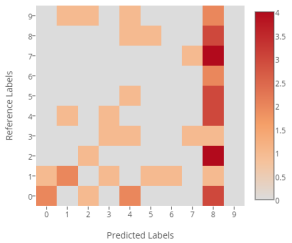
(b) GBRT



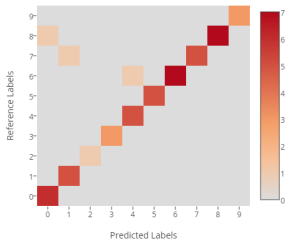
(c) HMM



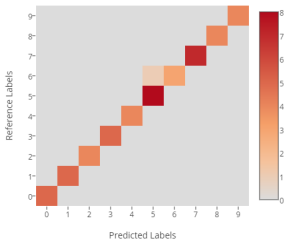
(d) Logistic Regression



(e) NaiveBayes

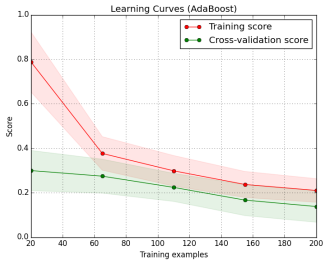


(f) Random Forest

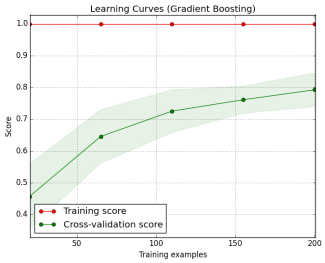


(g) SVM

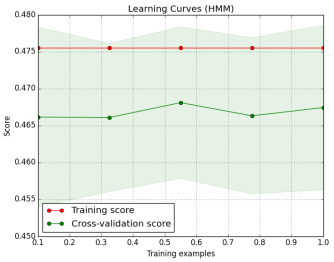
Figure 4: Confusion Matrices



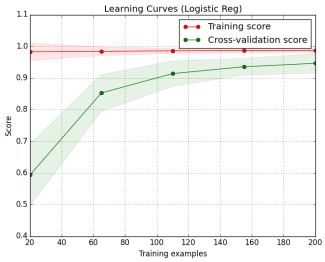
(a) AdaBoost



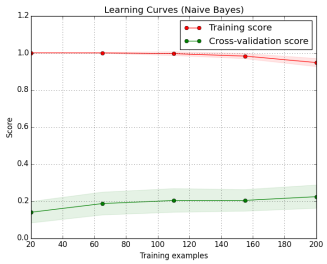
(b) GBRT



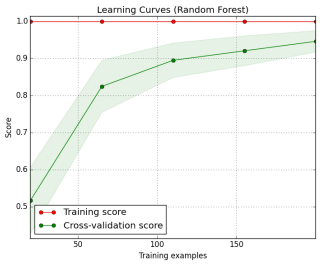
(c) HMM



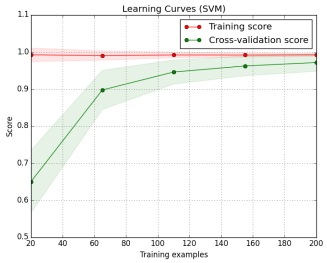
(d) Logistic Regression



(e) NaiveBayes



(f) Random Forest



(g) SVM

Figure 5: Learning Curves

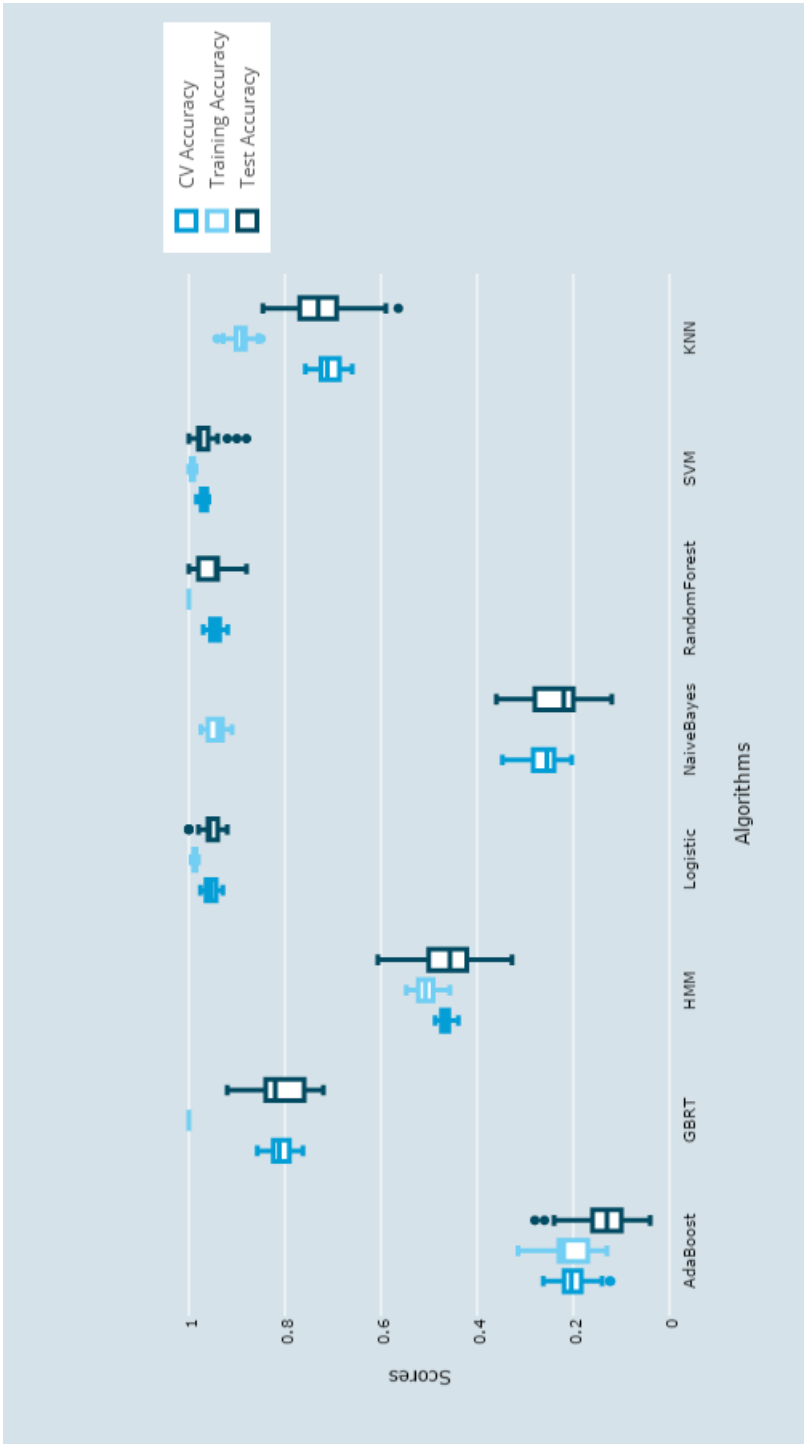


Figure 6: Comparison of accuracies of all algorithms

4.3 Analysis of results

Doing simple analysis on the previous results, we discover the following:

- The highest three classifier in terms of cross-validation and testing accuracies are:
  1. SVM
  2. Logistic Regression
  3. Random Forest
- The lowest three classifiers in terms of cross-validation and testing accuracies are:
  1. AdaBoost
  2. Naive Bayes
  3. HMM
- In only 2 cases, there is a significant difference between cross-validation and testing scores (AdaBoost, p=1.677e-08, and Naive Bayes, p=0.002); which indicates that cross-validation does estimate the generalization error well enough.
- There is a very clear over-fitting problem with the Naive Bayes classifier, which can be attributed to the use of raw pixels as direct features, as the Naive Bayes classifier tends to fit the probabilities over the training data which is clearly different in testing.

4.4 Data Mining

After doing several classification, we want to know there is any interesting part of the data to discover and from this, we can try to improve our algorithms to have a better results.

Our idea is we will try to discover the most common subsequence between each class and between whole dataset. By using this most common subsequence, we may recognize the label of a number by its Freeman code quickly than normal.

Below is the list of most common subsequence for each class of number and for the whole dataset:

- 1 : 4444
- 2 : 2222222
- 3 : 22222222
- 4 : 00000000
- 5 : 122222

- 6 : 4544
- 7 : 6666666666
- 8 : 544444
- 9 : 222
- 0 : 44
- Whole dataset: 44

As we can see the length of the most common subsequence of each class is too small in compare with the whole length of the Freeman code of a test image which is usually more than 150 or 200 characters so if we want to use this approach to improve the other algorithm, actually, it is not very effective.

In reality, we found that there is some very long most common subsequence of each class but to cover the whole class, the length is always very short. We believe the problem is the noise of the data because of human drawing mistake that make the image of the number have more unnecessary shape or unexpected point. If we can focus on these subsequence and accept some errors tolerance, the opportunity to improve our algorithms is quite high.

5 Conclusion and future work

This experiment was very rich for us in order to go deeper in the topics we studied so far in this masters.

Several conclusions can be noticed from this experiment:

- Testing on the raw pixels directly made us feel the problem of not having good features. The use of freeman code with HMM and KNN proved to be rewarding. The accuracy of the models using freeman code has increased a lot compared to the other models.
- Fourier descriptors provided us with the ability to have a fixed length set of features. This was useful in order to use these new features with classifiers like SVM and Random Forests.
- Cross-validation proved to be a very good estimator for the generalization error. We had some doubt about this in the beginning. As explained the experiment, to test this hypothesis, we had split the data to 80%-20% parts, and we performed the cross-validation on the 80% part, and then test the model on the 20% part. We noticed that in most of the cases, the results of the cross-validation and the test scores are statistically similar, which confirms our hypothesis.

On the other hand, there was a particular issue we couldn't resolve. As can be seen the results, most of the algorithms performed very well while using the

Fourier descriptors. However, these high results is inconsistent with what we see when we experiment with our GUI directly (by drawing a picture, and let it classify). In this case, we were not impressed by the performance of almost all the algorithms using the Fourier descriptors. We checked a lot the steps of the experiments and how we get the results in order to make sure the experiment is as fair as possible. To add to our confusion, these results (on the Fourier descriptor) are different with the results of the other team (Sami), where they report low accuracy results for these algorithms.
Also, we are concerned with the results of HMM on the Freeman chain code. When we experiment with our GUI directly GUI, we find the HMM has a very good performance, compared to the KNN. Yet, the results experiment show the performance of the HMM to be worse than we anticipated. A further investigation in these issues needs to be made.

For the future, we are planning to:

- Stratified sampling: During the experiment, we used random sampling to bootstrap from the dataset. However, since the number of images per label is small, random sampling could lead to unbalanced dataset. This can lead to large variance in the results of the classifiers. We didn't make a test for this hypothesis. In order to compensate for this, we ran the experiment many times with different random seeds (50 times), and we build the conclusions based on that. In a future modification on the experiment, we plan to use stratified sampling instead to eliminate this issue.

- Ensemble methods: We have monitoring *Kaggle* winners in the last period, and the techniques they use in order to win these highly competitive competitions. It seems that all agree on the power of ensemble methods (in particular, stacking), and combining it with the high predictive ability of the *Gradient tree boosting* method. We want to investigate more these point, since we believe it will be rewarding.

- Use different feature extraction methods: In this experiment, we explored in detail the use of Fourier descriptors and the Freeman chain code. We also briefly experimented with the raw pixels and the bag of words (for the Freeman code). We are looking forward to explore other feature descriptors, like Hartley descriptors, SIFT,..etc.

- Grid search for *AdaBoost* and *Gradient Tree Boosting*: We added AdaBoost and Gradient Tree Boosting without exhaustive grid search for the hyperparameters - unlike the rest of the algorithms -. This is interesting for us, especially for the AdaBoost, since it had a very poor performance during this experiment.

- The best number of Fourier Descriptors: In order to decide this number, we made very few random tests in the beginning in order to find a suitable number. We experimented on only one machine learning algorithm. We found that 8 10 descriptors was good enough. In the future, we want to explore this issue in deeper detail, by considering the number of Fourier

descriptors as a hyper-parameter for all the algorithm, and perform a grid search with different numbers of descriptors.

- Deep learning: We were planning to use deep learning in this project, thanks to a new approach to deep learning called 'transfer learning'[5]. The idea is based on the assumption that if there is a good deep learning system already developed for a specific task, it will most probably work fine for another task within the same field (eg, if we have a good enough deep learning system for image classification, then it will work fine on image analysis application)[6]. The main problem was that we didn't have good enough hardware to get the results in time for this project.

[5] Refer to *graphlab* machine learning package from *Dato* company

[6] In this case, we were going to use the classic AlexNet architechture pioneered by Alex Krizhevsky et. al in ImageNet Classification with Deep Convolutional Neural Networks. It turns out that a neural network trained on 1 million images of about 1000 classes makes a surprisingly general feature extractor. First illustrated by Donahue et al in *DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition*

References

[1] B Ballaro, PG Reas, and D Tegolo. Elliptical fourier descriptors for shape retrieval in biological images.

[2] Fred J. Damerau. A technique for computer detection and correction of spelling errors.

[3] Damerau levenshtein edit distance. `https://github.com/gfairchild/pyxDamerauLevenshtein`. Accessed: 2015-12-15.

[4] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[6] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression, 2004.

[7] Jerome H Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[8] Scikit-learn: Machine learning in python. `http://scikit-learn.org/`. Accessed: 2015-12-15.

[9] Natural language toolkit. `http://nltk.org/`. Accessed: 2015-12-15.

[10] Opencv library. `http://opencv.org/`. Accessed: 2015-12-15.