Report Lab 1
Pattern Recognition

NTUA 9th Semester

Efthymios Tzinis 031212007
Konstantinos Kallas 03112057

November 7, 2016

Introduction

In this work we prepared steps 10 till 16 from the Laboratory Exercise 1. The main purpose of this Exercise includes the implementation of an automatic visual recognition of hand - written digits 0-9. The steps below are presented with a given explanation for each one and images wherever it is necessary for a better perspective. We endeavored the usage and experimentation of different classifiers and deviant selections of splitting data, training and testing techniques. We tried to reach the recent state of the art of handwritten digits recognition which is currently around 99.8% as have been stated in [1] which was achieved through the usage of a Deep Neural Network. Comments in our codes could be quite helpful in further and deeper understanding of our implementation. Steps (1-10) have been introduced in our previous report for the preparation and are considered superficial for co-existing in this report.

Important Notes for the Code folder: We tried to produce a lucid code folder which contains only the source files of our scripts and it is cleared from the given data or the data produced during our experimentation. The wandering reader can always take advantage of our *Makefile* which includes the usage of our individual scripts. Matlab, Bash and Python scripts are responsible for various functionality (Making proper .arff files, training and testing classifiers, implementations of classifiers, etc.). The catholic hierarchy of our folder demands a separate folder for each classifier and a results subfolder respectively. The other folders are named in order to be self-explanatory for the purpose of our laboratory exercise (scripts, confusion_matrices, etc.). The input folder should contain the initial test and train files We portend that a potentially interested user could be perturbed by the vast number of scripts but we would be grateful to elucidate any of his misconceptions and amenable to upgrade our code following his lead.

Step 10

In this step, the sub goal is quite straightforward and the a-posterior probabilities were computed for every class by counting the occurrences of each one and divided by the total amount in our training data set.

Step 11

a)

Taking advantage from the pre-computation of all the variances and mean values of each category we implemented a Naive Bayes Classifier. We worked in Matlab environment for completing this task and used **mvnpdf()** for representing the Gaussian kernels in $\mathbb{R}^N$. Where N = number of features, so in our case $N = 256$ equal to the pixel number. The matrix of variances is considered as a diagonal matrix which contains only the auto-correlation elements of every attribute. In order to conduct a positive definite form of this matrix, which is a prerequisite for our implementation, we found the diagonal zero values and we increased the

values of these cells (only 2 in total of 256) by the norm-2 of the eigenvectors corresponding to the zeros eigenvalues.

b)

In this way we achieved a total recall of this method which is not so satisfactory.

$$Naive\ Bayes\ Recall = 75.83\%$$

Step 12

The repetition of the last step but by assuming constant variance for all the pixels equal to 1, demonstrated a gentrification in the total recall.

$$Naive\ Bayes\ Recall\ With\ Constant\ Variation = 81.76\%$$

We postulate that the reasons of this phenomenon are based on the nature of our data set. In particular, our data set is comprised by features that stimulate a very tiny variance and consequently the Gaussian which is created and represents this feature turns itself into a spike. Thus, the computation of the integral of the total probability deviates from the real value because of the number precision Matlab offers. Moreover it seems that because of these non-standardized pattern of variances the model could actually face severe difficulties when it tries to predict instances which are comprised by many features of lower variances and of course this results to lower classification ratio.

Step 13

a)

The implementation of NNR-1 was quite easy and the decision of our classifier is based only on the nearest neighbor's class. Every instance is classified in the same way as the nearest instance of the trained data set. In this section we trained our model with the top 1000 instances of our training data set and we assessed it on the top 100 of the respective test data set. The metric of distance used was the typical Euclidean distance.

b)

The evaluation of this partial test is demonstrated accordingly:

$$NNR1\ Recall\ Partial = 93.00\%$$

Step 14

a)

Following the aforementioned implementation we evaluated our model on the whole test and train data sets.

b)

The total evaluation of our NNR1 model can be shown below:

$$NNR1\ Recall\ Total = 95.02\%$$

c)

A naive but efficient implementation of the KNN model is proposed. The decision of our classifier is based now not only on the nearest neighbor's class but on the K nearest classes. A majority vote algorithm of linear time was implemented. Therefore, the majority of the K-classes instigate the final decision for every instance. If there is no such majority the instance is classified according to one of the classes whose percentage is the highest, arbitrary. Although we used deviant values for K, the best results were shown for $K = 3$. We evaluated the 3NN on the whole data set once again:

$$3NN\ Recall\ Total = 93.12\%$$

It can be easily inferred that our data contain a high spatial locality and this is why the 1NN outperforms the KNN as our naive implementation loses validity in border regions between 2 classes.

d)

We propose two different ideas which would probably increase our evaluation results.

- **Different Distance Metric** Instead of using the euclidean distance we could use the cosine similarity, which is widely proven, [2] that performs better than the Euclidean distance. Although, in our model this is not verified and Euclidean distance seems a better metric for our task.
- **Weighted Majority Voting** In order to achieve a higher recall we concluded to this optimization. In our Matlab script **find_KNN.m** it can be called with or without an optimization flag and select between the naive and optimized solution. The weights sensitive implementation of the KNN just measures the validity of each class vote from the neighbors by the distance of each one. Thus, this implementation dwindles the influence of the vote from a more alienated neighbor and enhances the selection from the nearest ones. In the following table, it can be shown that 3NN has the same results as the 1NN but even for a selection of a larger K the evaluation does not increases.

Optimization	Recall
1NN with Cosine Similarity	94.82%
3NN with Weighted Voting	95.02%

Data Pre-processing

The quality of the data used in all classifiers is a key element of the classification success. Because of that, we decided to improve the quality of our data by

smoothing the "images" using a Gaussian filter. This is important because it eliminates sharp edges in the images improving uniformity. We applied this filtering to both the train and the test data and re-run all the classifiers. Below are the new classifier recalls.

	Default	Smooth
Mean	81.4%	81.9%
Bayes	78.0%	75.8%
Bayes (s=1)	81.3%	81.8%
1NN	94.3%	95.0%
KNN	90.7%	92.2%

We can see that all classifiers, except Bayes, achieve a better overall recall. Thus for the rest of the project we used the smoothed data instead of the default ones.

We also decided to select the prevalent data features removing the ones that do not encapsulate further information. In order to achieve this we had to sort features depending on the amount of information they offer to the classification. Initially we tried to remove the corner features, because they are mostly dark for all digits. However this approach led to worse classification results so we abandoned it. We then chose variance as a metric of information. We calculated the variance of each feature on all the train data and kept only the features whose variance exceeded a threshold. We experimented with different thresholds and found out that removed features were indeed close to the co'rners as we initially thought. However in the end removing features did not improve the classification results at all so we abandoned this approach.

Step 15

a)

In this section we include deviant classifiers and assay them with our data set. We embarked a more professional way of testing classifiers. We used **WEKA** because its database offered a wide variety of classifiers and ways of feature selection. We tested thoroughly the recall of many classifiers but we concluded that the best among them were Multi Layer Perceptron (MLP), 1NN and Random Forest. We also checked an SVM approach with different kernels and configurations in order to achieve the best recall. the SVM we used is the open source implementation of **LIBSVM** analyzed in [3]. Finally, we did not use our implementations of Naive Bayes and KNN cause the former has very low recall rates and the latter could be inferior to a pre tested and analyzed implementation.

We achieved the highest SVM evaluation results by choosing a polynomial kernel of degree 3 and configuring the Cost parameter at 15. In our task this configuration outperforms all of our models 95.5157% and achieves a quite remarkable result with only one classifier. An RBF kernel as refer in [5] seems to be a much prudent decision for many cases of data classification but as tested the results does not prove this speculation. A supplement analysis of our choice is described below.

It seems that our data are not linear separable and thus the separation through higher dimensionality offered by SVM is a better technique. As referred in [4] the classification is a problem induced to the minimization of the term $min\{\frac{1}{2}(w^m)^T w^m + C\Sigma_{i=1}^{l}\xi_i^m\}$. Therefore, the C penalty parameter as it increases, it diminishes the possible values of our solution. In the presentation of the exercise it was suggested to check the one vs one and one vs all. LIBSVM implementation uses one-against-one and as it was proven in [4] it is a much more better technique of training in many cases as the one-against-all is rather obsolete. It is worth mentioning that Random Forest Algorithm achieves such a high result with minimum training and testing time. comments and results for our previous classifiers can be found on previous steps. In this way of thinking we could try some more classifiers which take advantage of non-linearities and are highly expressive. A manifold propagation through the different layers of a Deep Neural Net could actually subordinate substantially our effort but it excesses the purpose of this laboratory exercise.

b)

Here we present the total recall for all of our classifiers trained and tested with the aforementioned method of smoothing as pre processing.

Classifier	Recall
MLP -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a	94.57%
Random Forest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1	93.72%
SVM RBF	93.87%
SVM Sigmoid	91.92%
SVM -polynomial 3rd degree and C=15	**95.51%**

Step 16

In this section we combined previously introduced classifiers in order to achieve a better classification accuracy. We begin by approaching the problem in a naive way, by choosing the majority vote between 3 classifiers. Then we tried to combine our classifiers' confidence scores using an aggregate function. In the end we combined the confidence scores and used a classifier for our final result.

In order to choose which classifiers to combine together we inspected the confusion matrices to make sure each one classifies better different digits. The three chosen classifiers' confusions matrices are presented below:

$$SVM = \begin{vmatrix} 354 & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 2 & 0 \\ 0 & 257 & 0 & 0 & 4 & 0 & 3 & 0 & 0 & 0 \\ 1 & 0 & 185 & 2 & 1 & 1 & 0 & 1 & 7 & 0 \\ 1 & 0 & 2 & 152 & 0 & 9 & 0 & 0 & 2 & 0 \\ 0 & 2 & 2 & 0 & 188 & 1 & 2 & 1 & 1 & 3 \\ 2 & 0 & 0 & 4 & 1 & 150 & 0 & 0 & 2 & 1 \\ 1 & 0 & 2 & 0 & 3 & 2 & 162 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 4 & 0 & 0 & 139 & 2 & 1 \\ 2 & 0 & 1 & 3 & 0 & 2 & 1 & 0 & 156 & 1 \\ 0 & 0 & 0 & 1 & 4 & 0 & 0 & 0 & 0 & 172 \end{vmatrix} \quad MLP = \begin{vmatrix} 354 & 0 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & 1 \\ 0 & 256 & 0 & 2 & 1 & 0 & 3 & 1 & 0 & 1 \\ 4 & 0 & 183 & 2 & 3 & 2 & 1 & 1 & 1 & 1 \\ 0 & 0 & 3 & 151 & 1 & 5 & 0 & 1 & 4 & 1 \\ 1 & 2 & 4 & 0 & 184 & 2 & 1 & 0 & 0 & 6 \\ 2 & 0 & 0 & 6 & 0 & 148 & 0 & 0 & 1 & 3 \\ 1 & 0 & 3 & 0 & 2 & 1 & 162 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 7 & 0 & 0 & 137 & 0 & 2 \\ 4 & 0 & 1 & 2 & 1 & 3 & 1 & 0 & 151 & 3 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 2 & 0 & 172 \end{vmatrix}$$

$$1NN = \begin{vmatrix} 355 & 0 & 2 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 257 & 0 & 0 & 4 & 0 & 2 & 1 & 0 & 0 \\ 4 & 0 & 186 & 1 & 1 & 1 & 0 & 1 & 4 & 0 \\ 2 & 0 & 1 & 153 & 0 & 8 & 0 & 0 & 0 & 2 \\ 0 & 2 & 1 & 0 & 183 & 1 & 3 & 2 & 1 & 7 \\ 3 & 0 & 2 & 5 & 0 & 145 & 0 & 0 & 3 & 2 \\ 0 & 0 & 2 & 0 & 1 & 1 & 166 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 3 & 0 & 0 & 141 & 1 & 1 \\ 4 & 0 & 1 & 5 & 0 & 1 & 0 & 0 & 151 & 4 \\ 1 & 0 & 0 & 0 & 3 & 0 & 0 & 2 & 1 & 170 \end{vmatrix}$$

a)

In this first step we tried to combine the classifiers using a simple majority vote system. Each classifier "votes" for a prediction and the prediction with most votes is chosen. Special care needs to be taken on how to break ties. In our approach we decided to break ties using the **SVM** classifier that had the highest total recall. Sadly this method gave slightly worse results than just using the **SVM** classifier, so we decided to take a deeper look on the results.

$$Majority\ Recall\ (SVM, 1NN, MLP\ with\ SVM\ tie-breaker) = 95.3164\%$$

$$Majority\ Recall\ (SVM, 1NN, MLP\ with\ Random\ tie-breaker) = 95.0174\%$$

After careful examination we discovered that 94 out of 2007 occurrences were classified wrong. Out of those 94 occurrences, *70* were classified wrong by all separate classifiers! This means that we can hope (?) to achieve 70 misclassifications or 96.51% at best, considering we keep using the same classifiers. We should keep this in mind for the rest of the exercise.

b)

In this step we combined the separate classifiers by applying an aggregate function (min, max, avg, mult) at their confidence scores. We evaluated those 4 aggregate functions on the test data and the results are the following.

Aggregate Function	Recall
Average	94.97%
Maximum	94.97%
Minimum	95.02%
Multiplication	**95.12**%

As we see the results are worse than the previous step. This is caused mostly because in the previous step, we implicitly gave SVM a "larger" vote weight by breaking ties with its prediction. The only way to simulate the same effect in this step is by multiplying the SVM confidence score by a constant

c)

In this section we used the confidence scores produced from every individual classifier as a concatenated vector of attributes and trained an SVM with linear kernel in order to achieve a higher evaluation result. We split the training data set in two different percentages 80-20% and 60-40%. At first, we trained each classifier with the bigger subset and tested them on the smaller and on the whole test set also. So we produced some confidence scores for the small train data set

and the whole test data set. Additionally, we concatenated all these scores from every classifier into one vector and considered these multi dimension vectors as features. After, we used different combinations of our classifiers the naive implementation of this idea gave as **95.316%** by using as selected classifiers the 1NN and SVM and put their scores into another SVM with linear kernel (same as the given hint).

Conclusions and Feature Work

Our effort for finding the best classifier has demonstrated that we can perform better than many other multi classifier systems with proper pre-process of the data and selection - configuration of the classifier. The SVM meets the expectation of our task and achieves an even higher result from all the other classifiers united or separated. In a potential feature work we could use a much larder database of handwritten digits to train our system or try different classification algorithms. Another possible solution could be to use Fourier attributes that could provide a different representation of data and take advantage of patterns created in frequency space.

References

[1] Multi-column Deep Neural Networks for Image Classification, Dan Ciresan Ueli, Meier, Jurgen Schmidhuber, *Technical Report No. IDSIA-04-12*

[2] An Optimized K-Nearest Neighbor Algorithm for Large Scale Hierarchical Text Classification, Xiaogang Han, Junfa Liu, Zhiqi Shen, and Chunyan Miao

[3] LIBSVM: A Library for Support Vector Machines, Chih-Chung Chang and Chih-Jen Lin.

[4] A Comparison of Methods for Multi-class Support Vector Machines, Chih-Wei Hsu and Chih-Jen Lin.

[5] Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel, S. Sathiya, Keerthi.