Optical Character Recognition Via Supervised Learning (K-NN & Bayesian & Parzen)

Amir M. Mousavi

Department of Computer Engineering
Shahid Rajayi University

Tehran, Iran

AmirMahmood.Mousavi@yahoo.com

Handwritten Digit Recognition is one of the most fundamental problems in designing practical recognition system. Immediate applications of the digit recognition techniques include postal mail sorting, automatically address reading and mail routing, bank check processing, etc

Machine learning and deep learning plays an important role in computer technology and artificial intelligence. With the use of machine learning, human effort can be reduced in recognizing, learning, predictions and many more areas. This article presents recognizing the handwritten digits (0 to 9) from the given dataset, comparing supervised learning classifiers like 1NN , KNN, parzen , and bayesian on basis of performance, accuracy, time, sensitivity, positive productivity, and specificity with using different parameters with the classifiers.

Keywords—Supervised Learning , KNN , K-Nearest Neighbour , Bayesian , parzen , 1NN , OCR , Optical Character Recognition , Classifiers , handwritten digits , Machine Learning .

.

I. Introduction

The recognition of handwritten characters by computer has been a topic of intensive search for many years. Handwritten numeral recognition is always the research focus in the field of image process and pattern recognition. The numeral varieties in size, shape, slant and the writing style make the research harder. The numeral character recognition is the most challenging field, because the big research and development effort that has gone into it has not solved all commercial and intellectual problems. Handwritten numeral character recognition is an important step in many document processing applications. Digital document processing is gaining popularity for application to office and library automation, bank and postal services, publishing houses and communication technology. The complexity of the problem is greatly increased by noise in the data and infinite variability of handwriting as a result of mood of writer and nature of writing. Recognition of handwritten digits has been popular topic of research for many years. The recognition of handwritten numeral character has been considerable interest to researchers working on OCR.

As mentioned before , Handwritten digit recognition problem can be seen as a subtask of the more general Optical Character Recognition (OCR) problem. However, there are some applications (e.g., postal code and bank checks reading) that are restricted to recognizing digits but require very high accuracy and speed. In addition, handwritten digit recognition problem is usually used as a benchmark for comparing different classification techniques [1].

While recognition of handwritten Latin digits has been extensively investigated using various techniques [1–8], little work has been done on Arabic handwritten digit recognition. Al-Omari et al. [9] proposed a system for recognizing Arabic digits from '1' to '9'. They used a scale-, translation-, rotation-invariant feature vector to train a probabilistic neural network (PNN). Their database was composed of 720 digits for training and 480 digits for testing written by 120 persons. They achieved 99.75% accuracy. Said et al. [1] used pixel values of the 16×20 size-normalized digit images as features. They fed these values to an Artificial Neural Network (ANN), where number of its hidden units is determined dynamically. They used a training set of 2400 digits and a testing set of 200 digits written by 20 persons to achieve 94% accuracy.

Naming conventions for different numeral systems may be confusing. Digits used in Europe and several other countries sometimes are called "ArabicNumbers"; and digits used in Arab world are sometimes called "Hindi Numbers". A different naming convention is used in this paper. Digits used in Europe will be referred to as "Latin Digits" and that used in Arab world as "Arabic Digits". It is worthwhile to mention here that Arabic and Persian handwritten digits (digits used in Iran) are similar but not identical. However, there are some writing styles for Persian digits that are very similar to Arabic which leads some researchers to consider Arabic and Persian digits to be the same [2,3]. Tables 1 and 2 show Arabic and Persian handwritten digits with different writing styles as well as their printed versions.

In this paper, we are going to study the performance of various classifiers/features combinations on the Arabic digit recognition problem. Well-known feature extraction techniques like HOG and PCA are considered, in this paper. Results are then analyzed and discussed .

Table 1 Arabic printed and handwritten digits

Latin Equivalent	0	1	2	3	4	5	6	7	8	9
Printed	٠	١	٢	٣	٤	٥	٦	٧	٨	٩
Typical Handwritten	٠	١	٢	٣	٤	٥	٦	٧	٨	٩
Other Writing Style	--	--	--	٣	--	--	--	--	--	--

Table 2 Persian printed and handwritten digits

Latin Equivalent	0	1	2	3	4	5	6	7	8	9
Printed	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
Typical Handwritten	٥	۱	۲	۳	۴	۵	۶	۷	۸	۹
Other Writing Style	٠	--	٢	٣	٤	٥	٦	--	--	--

In this paper we have proposed to provide a comprehensive tutorial and survey about review of the supervised learning and its classifiers including KNN , parzen window , Bayesian and 1NN , which is discussed in sections II to VI Then a brief explanation of OCR problem and feature extraction methods are in sections VII and V . Finally. The results of the implementation are presented VIII and the conclusions are discussed in section IX .

II. Supervised Learning

Supervised learning is typically done in the context of classification, when we want to map input to output labels, or regression, when we want to map input to a continuous output. Common algorithms in supervised learning include logistic regression, naive bayes, support vector machines, artificial neural networks, and random forests. In both regression and classification, the goal is to find specific relationships or structure in the input data that allow us to effectively produce correct output data. Note that "correct" output is determined entirely from the training data, so while we do have a ground truth that our model will assume is true, it is not to say that data labels are always correct in real-world situations. Noisy, or incorrect, data labels will clearly reduce the effectiveness of your model.

When conducting supervised learning, the main considerations are model complexity, and the bias-variance tradeoff. Note that both of these are interrelated.

Model complexity refers to the complexity of the function you are attempting to learn—similar to the degree of a polynomial. The proper level of model complexity is generally determined by the nature of your training data. If you have a small amount of data, or if your data is not uniformly spread throughout different possible scenarios, you should opt for a low-complexity model. This is because a high-complexity model will overfit if used on a small number of data points. Overfitting refers to learning a function that fits your training data very well, but does not generalize to other data points—in other words, you are strictly learning to produce your training data without learning the actual trend or structure in the data that leads to this output. Imagine trying to fit a curve between 2 points. In theory, you can use a function of any degree, but in practice, you would parsimoniously add complexity, and go with a linear function.

The bias-variance tradeoff also relates to model generalization. In any model, there is a balance between bias, which is the constant error term, and variance, which is the amount by which the error may vary between different training sets. So, high bias and low variance would be a model that is consistently wrong 20% of the time, whereas a low bias and high variance model would be a model that can be wrong anywhere from 5%-50% of the time, depending on the data used to train it. Note that bias and variance typically move in opposite directions of each other; increasing bias will usually lead to lower variance, and vice versa. When making your model, your specific problem and the nature of your data should allow you to make an informed decision on where to fall on the bias-variance spectrum. Generally, increasing bias (and decreasing variance) results in models with relatively guaranteed baseline levels of performance, which may be critical in certain tasks. Additionally, in order to produce models that generalize well, the variance of your model should scale with the size and complexity of your training data—small, simple data-sets should usually be learned with low-variance models, and large, complex data-sets will often require higher-variance models to fully learn the structure of the data.

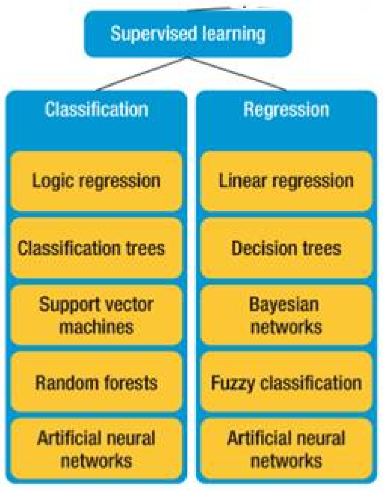


Figure 3. Categories of Supervised Learning

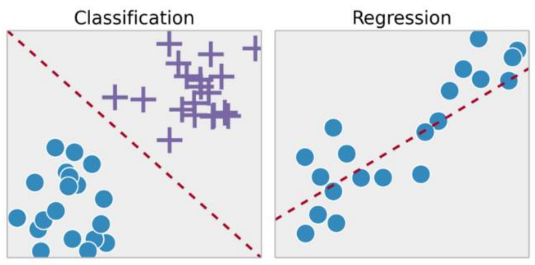


Figure 4. Classification vs Regression

III. K-Nearest Neighbour

KNN is the non-parametric method or classifier used for classification as well as regression problems. This is the lazy or late learning classification algorithm where all of the computations are derived until the last stage of classification, as well as this, is the instance-based learning algorithms where the approximation takes place locally. Being simplest and easiest to implement there is no explicit training phase earlier and the algorithm does not perform any generalization of training data. The direct solution would be when these are nonlinear decision boundaries between classes or when the amount of data is large enough. Input features can be both qualitative and quantitative in nature. Whereas output features can be categorical values which are typical classes seen in data. KNN explains categorical value using majority votes of K nearest neighbors where the value for K can differ, so on changing the value of K, the value of votes can also vary.

✓ **Algorithm :**

a) Compute the distance metric between the test data point and all labeled data points.
b) Order the labeled data points in increasing order of distance metric.
c) Select the top K labeled data points and look at class labels.
d) Look for the class labels that majority of these K labeled data points have and assign it to test data points.

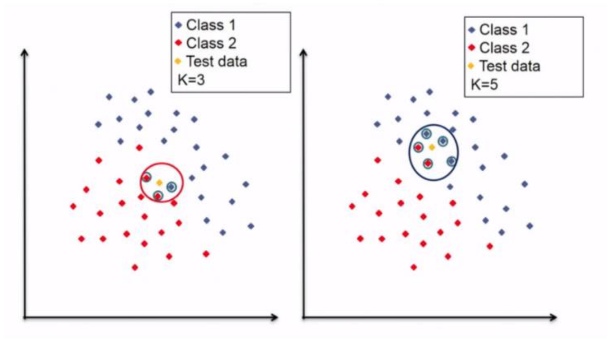
✓ **Algorithm :**

a) parameter selection- Best choice of K depends on data. The larger value of K reduce the effect of noise on classification but makes the decision boundaries between classless distinct. The smaller value of K tends to be affected by noise with clear separation between classes.

b) Presence of noise

c) Feature selection and scaling- It is important to reduce irrelevant features. When the number of features is too large and suspected to be highly redundant, features extraction will be required. If the features are carefully chosen then it is expected that the classification will be better.

d) Curse of dimensionality



Case 1 (K =3) and Case 2 (K = 5)

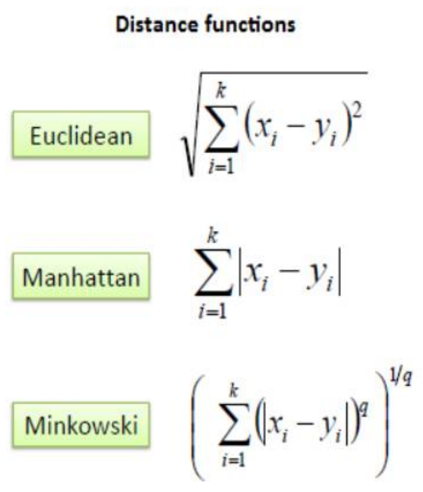
Figure 5

To understand better, take look at the different values for K in figure 5 . Says in case 1, value for K is 3. Then the class for the test data point would be of red color among the classes for red and blue. For K = 5 in case 2, then the predicted class would be of blue color from the KNN algorithm. So for changing the value of K, the output for the test data point can also vary. So it's necessary to choose the value of K wisely. The large value for K can reduce the overall noise but there is no guarantee.

✓ Distance Functions

Different distance functions used in KNN are :

i. Euclidean function
ii. Manhattan function
iii. Minkowski
iv. Hamming distance
v. Mahalanobis distance



Distance functions

Euclidean $$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan $$\sum_{i=1}^{k}|x_i - y_i|$$

Minkowski $$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$$

Distance functions in KNN

IV. Naïve Bayesian

Naive Bayes is a simple, yet effective and commonly-used, machine learning classifier. It is a probabilistic classifier that makes classifications using the Maximum A Posteriori decision rule in a Bayesian setting. It can also be represented using a very simple Bayesian network. Naive Bayes classifiers have been especially popular for text classification, and are a traditional solution for problems such as spam detection.

The goal of any probabilistic classifier is, with features x_0 through x_n and classes c_0 through c_k, to determine the probability of the features occurring in each class, and to return the most likely class. Therefore, for each class, we want to be able to calculate P(c_i | x_0, …, x_n).

In order to do this, we use Bayes rule. Recall that Bayes rule is the following:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

n the context of classification, you can replace A with a class, c_i, and B with our set of features, x_0 through x_n. Since P(B) serves as normalization, and we are usually unable to calculate P(x_0, …, x_n), we can simply ignore that term, and instead just state that P(c_i | x_0, …, x_n) ∝ P(x_0, …, x_n | c_i) * P(c_i), where ∝ means "is proportional to". P(c_i) is simple to calculate; it is just the proportion of the data-set that falls in class i. P(x_0, …, x_n | c_i) is more difficult to compute. In order to simplify its computation, we make the assumption that x_0 through x_n are conditionally independent given c_i, which allows us to say that P(x_0, …, x_n | c_i) = P(x_0 | c_i) * P(x_1 | c_i) * … * P(x_n | c_i). This assumption is most likely not true—hence the name naive Bayes classifier, but the classifier nonetheless performs well in most situations. Therefore, our final representation of class probability is the following:

$$\begin{aligned} P(c_i|x_0,\ldots,x_n) &\propto P(x_0,\ldots,x_n|c_i)P(c_i) \\ &\propto P(c_i)\prod_{j=1}^{n} P(x_j|c_i) \end{aligned}$$

Calculating the individual P(x_j | c_i) terms will depend on what distribution your features follow. In the context of text classification, where features may be word counts, features may follow a multinomial distribution. In other cases, where features are continuous, they may follow a Gaussian distribution.

Note that there is very little explicit training in Naive Bayes compared to other common classification methods. The only work that must be done before prediction is finding the parameters for the features' individual probability distributions, which can typically be done quickly and deterministically. This means that Naive Bayes classifiers can perform well even with high-dimensional data points and/or a large number of data points .

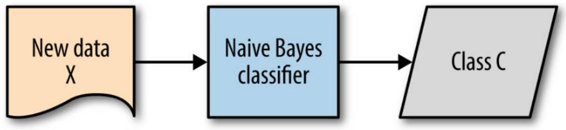


Figure 6

Now that we have a way to estimate the probability of a given data point falling in a certain class, we need to be able to use this to produce classifications. Naive Bayes handles this in a very simple manner; simply pick the c_i that has the largest probability given the data point's features.

$$y = \underset{c_i}{argmax}\ P(c_i)\prod_{j=1}^{n} P(x_j|c_i)$$

This is referred to as the Maximum A Posteriori decision rule. This is because, referring back to our formulation of Bayes rule, we only use the P(B|A) and P(A) terms, which are the likelihood and prior terms, respectively. If we only used P(B|A), the likelihood, we would be using a Maximum Likelihood decision rule.

V. Parzen Window

Parzen windows classification is a technique for nonparametric density estimation, which can also be used for classification. Using a given kernel function, the technique approximates a given training set distribution via a linear combination of kernels centered on the observed points. In this work, we separately approximate densities for each of the two classes, and we assign a test point to the class with maximal posterior probability.

The resulting algorithm is extremely simple and closely related to support vector machines. The decision function is:

$$f(\mathbf{X}) = sign(\sum y_i K(\mathbf{X}_i, \mathbf{X})),$$

The Parzen windows classification algorithm does not require any training phase; however, the lack of sparseness makes the test phase quite slow. Furthermore, although asymptotical convergence guarantees on the perfomance of Parzen windows classifiers exist [Duda and Hart, 1973], no such guarantees exist for finite sample sizes.

Parzen windows can be regarded as a generalization of k-nearest neighbor techniques. Rather than choosing the k nearest neighbors of a test point and labelling the test point with the weighted majority of its neighbors' votes, one can consider all points in the voting scheme and assign their weight by means of the kernel function. With Gaussian kernels, the weight decreases exponentially with the square of the distance, so far away points are practically irrelevant. The width σ of the Guassian determines the relative weighting of near and far points. Tuning this parameter controls the predictive power of the system. We have empirically optimized the value of σ.

VI. OCR

Optical character recognition, usually abbreviated to OCR is the mechanical or electronic translation of images of handwritten, typewritten or printed text (usually captured by a scanner) into machine-editable text or computer process-able format, such as ASCII code. Whenever a page is scanned, it is stored as a bit-mapped file. When the image is displayed on the screen, we can read it. But it is just a series of dots for the computer. The computer does not recognize any "words" on the image.

OCR makes the computer read these words. It looks at each line of the image and determines which particular character is represented by dots. OCR is a field of research in pattern recognition, artificial intelligence and machine vision. Optical character recognition (using optical techniques such as mirrors and lenses) and digital character recognition (using scanners and computer algorithms) were originally considered separate fields. Because very few applications survive that use true optical techniques, the OCR term has now been broadened to include digital image processing as well.

OCR uses more than one approach when it comes to recognising text. The most basic way the technology distinguishes characters from pictures is through a technique known as pattern recognition. This involves a computer comparing objects within an image to letters already stored within its software. In other words, the software is equipped with a library of characters and the computer will search for the same patterns within your work and recognize when it finds a match.

The problem with pattern recognition, at least for our purposes, is that it cannot detect handwritten text. No one writes in Times New Roman, after all. Thankfully, as the technology has become more sophisticated, it increasingly relies on a different tactic known as feature extraction.

Rather than trying to recognize full letters, feature extraction occurs when a computer detects certain features (lines and loops, for example) and understands that they signify a character. The letter ‘H’, for instance, will be picked up by the software whenever it detects two vertical lines joined in the middle by a smaller, horizontal line.

This technique means that a computer’s ability to recognize characters is not constrained to a limited number of fonts. From here, it can be trained to detect even handwritten text.

VII. Feature Extraction

In machine learning, pattern recognition and in image processing, feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. Feature extraction is a dimensionality reduction process, where an initial set of raw variables is reduced to more manageable groups (features) for processing, while still accurately and .completely describing the original data set

When the input data to an algorithm is too large to be processed and it is suspected to be redundant (e.g. the same measurement in both feet and meters, or the repetitiveness of images presented as pixels), then it can be transformed into a reduced set of features (also named a feature vector). Determining a subset of the initial features is called feature selection. The selected features are expected to contain the relevant information from the input data, so that the desired task can be performed by using this reduced representation instead of the complete initial data. to extract the OCR's . feature we used PCA and HOG

A.) HOGhistogram of oriented gradients(

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of edge orientation histograms, scale-invariant feature transform descriptors, and shape contexts, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.

The essential thought behind the histogram of oriented gradients descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing.

The HOG descriptor has a few key advantages over other descriptors. Since it operates on local cells, it is invariant to geometric and photometric transformations, except for object orientation. Such changes would only appear in larger spatial regions. Moreover, as Dalal and Triggs discovered, coarse spatial sampling, fine orientation sampling, and strong local photometric normalization permits the individual body movement of pedestrians to be ignored so long as they maintain a roughly upright position. The HOG descriptor is thus particularly suited for human detection in images , we used it in our OCR system to extract the features.

The data used to train the classifier are HOG feature vectors extracted from the training images. Therefore, it is important to make sure the HOG feature vector encodes the right amount of information about the object. The extract HOG Features function returns a visualization output that can help form some intuition about just what the "right amount of information" means. By varying the HOG cell size parameter and visualizing the result, you can see the effect the cell size parameter has on the amount of shape information encoded in the feature vector:

The visualization shows that a cell size of [8 8] does not encode much shape information, while a cell size of [2 2] encodes a lot of shape information but increases the dimensionality of the HOG feature vector significantly. A good compromise is a 4-by-4 cell size. This size setting encodes enough spatial information to visually identify a digit shape while limiting the number of dimensions in the HOG feature vector, which helps speed up training. In practice, the HOG parameters should be varied with repeated classifier training and testing to identify the optimal parameter setting .

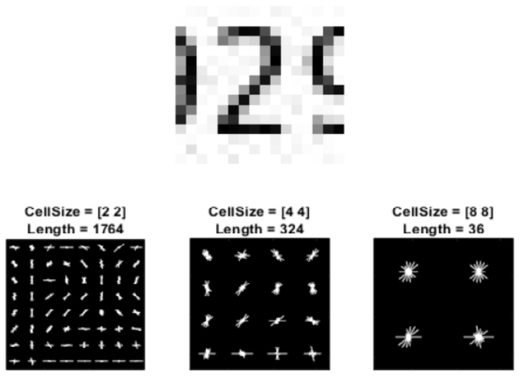


Figure.6. cell size of applied HOG on digits

The visualization on the figure 5 shows that a cell size of [8 8] does not encode much shape information, while a cell size of [2 2] encodes a lot of shape information but increases the dimensionality of the HOG feature vector significantly. A good compromise is a 4-by-4 cell size. This size setting encodes enough spatial information to visually identify a digit shape while limiting the number of dimensions in the HOG feature vector, which helps speed up training. In practice, the HOG parameters should be varied with repeated classifier training and testing to identify the optimal parameter setting.

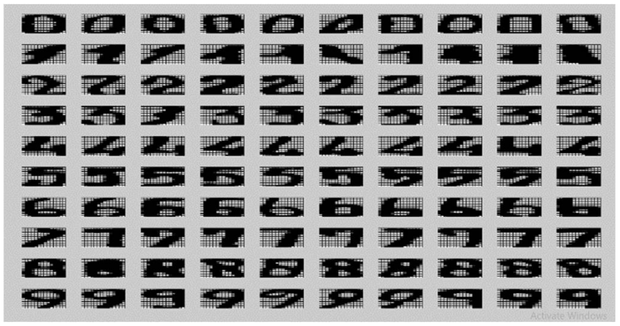


Figure 5. Extracted feature

B. PCA (Principal Component Analysis):

In statistics, principal components analysis (PCA) is a technique that can be used to simplify a dataset., more formally it is a linear transformation that chooses a new coordinate system for the data set such that the greatest variance by any projection of the data set comes to lie on the first axis (then called the first principal component), the second greatest variance on the second axis, and so on. PCA can be used for reducing dimnesionalty in a dataset while retaining those characteristics of the dataset that contribute most to its variance by eliminating the later principal components (by a more or less heuristic decision). These characteristics may be the "most important", but this is not necessarily the case, depending on the application.

PCA has the speciality of being the optimal linear transformation subspace that has largest variance. However this comes at the price of greater computational requirement. Unlike other linear transforms, the PCA does not have a fixed set of basis vectors, Its basis vectors depend on the data set.

Assuming zero empirical mean (the empirical mean of the distribution has been subtracted from the data set), the principal component w_i of a dataset x can be calculate by finding the eigenvalues and eigenvectors of the covariance matrix of x, we find that the eigenvectors with the largest eigenvalues correspond to the dimensions that have the strongest correlation in the dataset. The original measurements are finally projected onto the reduced vector space.

PCA can be thought of as fitting an n-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a principal component. If some axis of the ellipsoid is small, then the variance along that axis is also small, and by omitting that axis and its corresponding principal component from our representation of the dataset, we lose only a commensurately small amount of information.

To find the axes of the ellipsoid, we must first subtract the mean of each variable from the dataset to center the data around the origin. Then, we compute the covariance matrix of the data, and calculate the eigenvalues and corresponding eigenvectors of this covariance matrix. Then we must normalize each of the orthogonal eigenvectors to become unit vectors. Once this is done, each of the mutually orthogonal, unit eigenvectors can be interpreted as an axis of the ellipsoid fitted to the data. This choice of basis will transform our covariance matrix into a diagonalised form with the diagonal elements representing the variance of each axis. The proportion of the variance that each eigenvector represents can be calculated by dividing the eigenvalue corresponding to that eigenvector by the sum of all eigenvalues.

This procedure is sensitive to the scaling of the data, and there is no consensus as to how to best scale the data to obtain optimal results.

PCA is mathematically defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

Consider a data matrix, X, with column-wise zero empirical mean (the sample mean of each column has been shifted to zero), where each of the n rows represents a different repetition of the experiment, and each of the p columns gives a particular kind of feature (say, the results from a particular sensor).

Mathematically, the transformation is defined by a set of p-dimensional vectors of weights or coefficients {\displaystyle \mathbf {w} _{(k)}=(w_{1},\dots ,w_{p})_{(k)}} that map each row vector {\displaystyle \mathbf {x} _{(i)}} of X to a new vector of principal component scores {\displaystyle \mathbf {t} _{(i)}=(t_{1},\dots ,t_{l})_{(i)}} , given by :

$$t_{k(i)} = \mathbf{x}_{(i)} \cdot \mathbf{w}_{(k)} \qquad \text{for} \qquad i = 1, \dots, n \qquad k = 1, \dots, l$$

in such a way that the individual variables {\displaystyle t_{1},\dots ,t_{l}} of t considered over the data set successively inherit the maximum possible variance from x, with each coefficient vector w constrained to be a unit vector.

.

VIII. Implementation Results

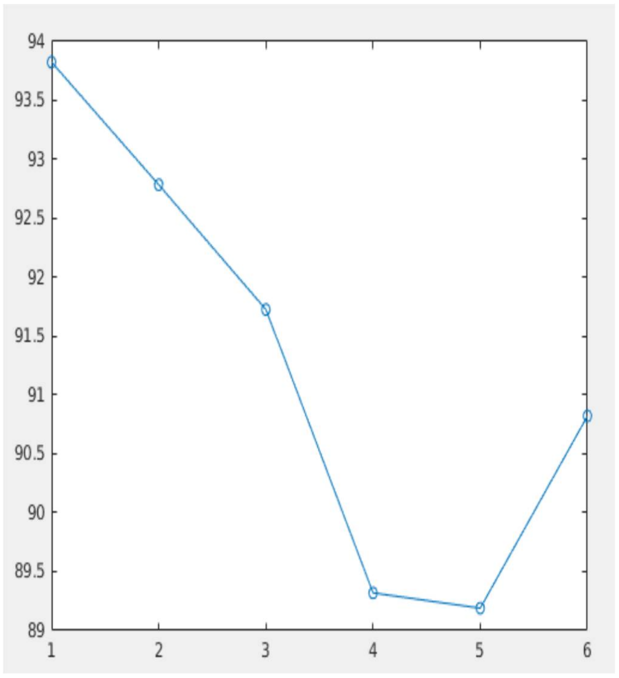
In this section we are going focus on checking our assumptions. So far we have learned how to perform KNN Bayesian and parzen . so let's see the results of , : implementation

A. NN

We developed ١NN classifier by calculating distance of test sample to train samples and find nearest neighbor of training sample and choose the class as result. To obtain this , first we calculated absolute subtraction of each ١٠x٦ feature vector of test and training sample and summation of the whole result in each row and column to calculate a single value distance of final result the winner class is the one which has minimum distance to test sample here is the result of this classifier over ٥٠٪ test data up on all classes . including error class is ٩٤٪

B. K-NN

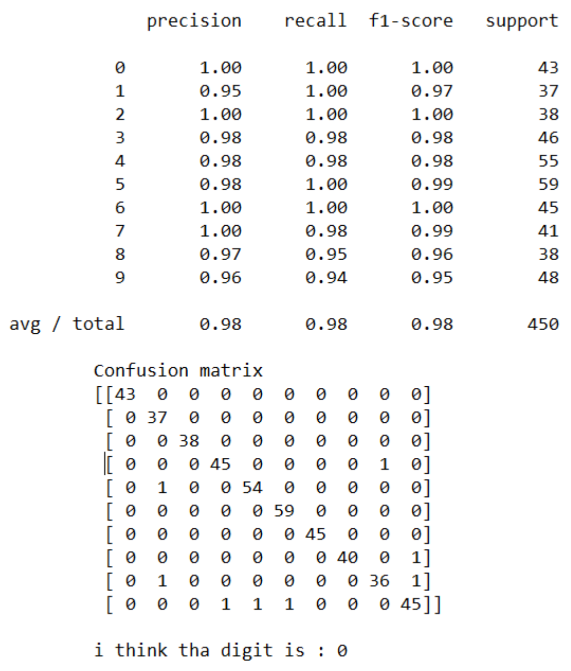
All we done in this classifier is similar to ١-NN but, at the end of function we choose the winner class by k nearest neighbor and maximum occurrence of a class. In table ١ we can see the results of accuracy after evaluating ٥٠٪ of data .as validation set



K	Accuracy
١	٩٤/١%
٣	٩٢/٣%
٥	٩٢%
٧	٩١/٧%
٩	٩٠/٩%
١١	٨٩/٧%
١٣	٨٩/١%

.Table ٣

training data points: ١٢١٢)
validation data points: ١٣٥
(testing data points: ٤٥٠

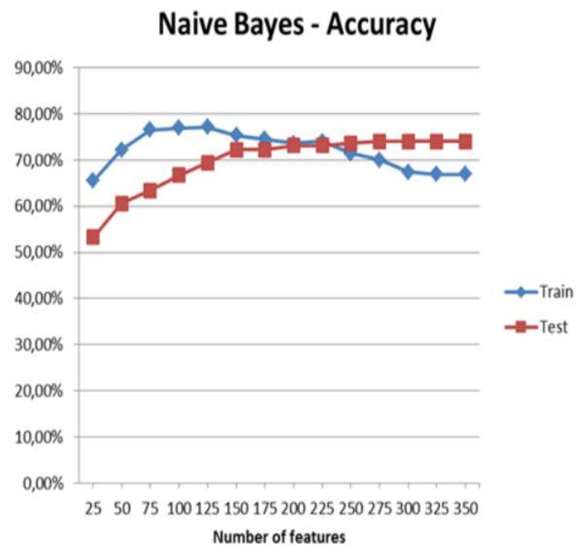


```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        43
          1       0.95      1.00      0.97        37
          2       1.00      1.00      1.00        38
          3       0.98      0.98      0.98        46
          4       0.98      0.98      0.98        55
          5       0.98      1.00      0.99        59
          6       1.00      1.00      1.00        45
          7       1.00      0.98      0.99        41
          8       0.97      0.95      0.96        38
          9       0.96      0.94      0.95        48

avg / total       0.98      0.98      0.98       450

Confusion matrix
[[43  0  0  0  0  0  0  0  0  0]
 [ 0 37  0  0  0  0  0  0  0  0]
 [ 0  0 38  0  0  0  0  0  0  0]
 [ 0  0  0 45  0  0  0  0  1  0]
 [ 0  1  0  0 54  0  0  0  0  0]
 [ 0  0  0  0  0 59  0  0  0  0]
 [ 0  0  0  0  0  0 45  0  0  0]
 [ 0  0  0  0  0  0  0 40  0  1]
 [ 0  1  0  0  0  0  0  0 36  1]
 [ 0  0  0  1  1  1  0  0  0 45]]

i think tha digit is : 0
```

Figure ٨ / the output of KNN

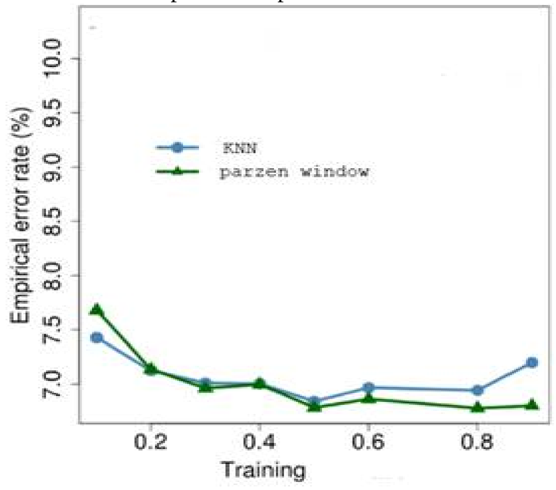
✓ : Bayesian

As a specific research to implementing bayze classification we first should calculate PDF of each class individually from all training set data. To achieve this we first should evaluate our ١٠x٦ feature vector because if we don’t reduce the diminutions before we calculate PDF we have to evaluate ٦٠ PDF for each class and this make calculation time slower. This process is done with PCA analysis of data , first we create a matrix of ٦٠ feature variable for all training samples and run PCA algorithm over it to reduce .dimensions



✓ : Parzen window

:Here is the comparison of parzen and KNN



IX. CONCLUSION

In this paper,we have evaluated the performance of a number of feature sets and classification techniques on the problem of recognizing Arabic digits. we studied KNN , baysian , parzen window classifiers and observed how they . worked on Arabic digits

REFERENCES

[1] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE 86(11), 2278-2324 (1998).

[2] Dong, J.: Comparison of algorithms for handwritten numeral recognition. Technical report, CENPARMI, Concordia University, 1999

[3] digit recognition: benchmarking of state-of-the-art techniques. Pattern Recognit. 36, 2271–2285 (2003).

[4] Zhang, P., Bui, T., Suen, C.Y.: Hybrid feature extraction and feature selection for improving recognition accuracy of handwritten numerals. Proc. 8th ICDAR, pp. 136–140, 2005,

[5] eow, L., Loe, K.: Robust vision-based features and classification schemes for off-line handwritten digit recognition. Pattern Recognit. 40(6), 1816–1824 (2007). .

[6] Belongie, S., Malik, J., Puzicha, J.: Shape matching and object recognition using shape contexts. PAMI, vol. 24, no. 24, 2002

[7] Lauera, F., Suen, C., Blocha, G.: A trainable feature extractor

[8] for handwritten digit recognition. Pattern Recognit. 36, 2271–2285 (2003) .

[9] Gorgevik, D., Cakmakov, D.: An efficient three-stage classifier forhandwritten digit recognition. Proc. 17th ICPR, pp. 1051–4651 (2004)

[10] Al-Omari, F., Al-Jarrah, O.: Handwritten Indian numerals recognition system using probabilistic neural networks. Adv. Eng. Inform. 18(1), 9–16 (2004).

[11] C. Cortes and V. Vapnik, "Support-vector networks," Machine learning, vol. 20, pp.273-297, 1995.

THE END