ECE 759: Pattern Recognition and Machine Learning

Project Report

Image Content Classification

Authors:
Nazneen Kotwal
Priya Diwakar

Instructor:
Dr. Hamid Krim

Table of Contents

Sr. No.	Title	Pg. No.
1	Project Overview	2
2	Image Classification Database	3
3	Feature Selection	4
4	Classification Algorithm	7
5	Cross Validation	11
6	Demonstration of Performances	13
7	Analysis of Results	17
8	References	19

I. Project Overview

Motivation: In machine learning, pattern recognition is the assignment of a label to a given input value. One example of pattern recognition is the image classification problem which attempts to assign each input image to one of a given set of classes. This is one of the core problems in Computer Vision that, despite its simplicity, has a large variety of practical applications.

The Image Classification Pipeline: The task in Image Classification is to take an array of pixels that represents a single image and assign a label to it. The complete pipeline can be formalized as follows:

Fig. 1 Image Classification Pipeline

1. **Input Data Organization :** The input consists of a set of N, (P x Q) images, each labeled with one of the K different classes. We flatten the images to obtain a N x (No. of pixels) matrix which serves as an input to our classifier.
2. **Pre-Processing**: In this project the images are normalized and resized to improve the system.
3. **Feature Extraction:** After the pre-processing the normalized image is given as input to the feature extraction module to find the key features that will be used for classification.
4. **Learning a Model:** The training data set is used to learn what each of the classes looks like. This step is referred to as training a classifier.
5. **Evaluation:** The quality of the classifier is evaluated by asking it to predict labels for a new set of images. In this project we use 5 fold cross validation.

Since this task of recognizing a visual concept is relatively trivial for a human to perform, it is worth considering the challenges involved from the perspective of a Computer Vision algorithm. The list of some challenges keeping in mind the raw representation of images as a 3-D array of brightness values are as follows:

- **Viewpoint variation**: Object can have multiple orientations with respect to the camera.
- **Scale variation and Deformation**: Visual objects exhibit variation in their size and can be deformed in multiple ways.
- **Occlusion**: Sometimes only a small portion of an object could be visible.
- **Illumination conditions**. The effects of illumination are drastic on the pixel level.
- **Background clutter**. The objects of interest may *blend* into their environment, making them hard to identify.
- **Intra-class variation**. The classes of interest can often be relatively broad, such as *chair*. There are many different types of these objects, each with their own appearance.

A good image classification model must be invariant to the cross product of all these variations, while simultaneously retaining sensitivity to the inter-class variations. The two classifier models implemented in this project is the SVM, with linear and quadratic kernel, and Naive Bayes algorithm.

II. Image Classification Database

It is undeniable that 80% of a data scientist's time and effort is spent in collecting, cleaning and preparing the data for analysis because datasets come in various sizes and are different in nature. The classifiers implemented in this project are tested on two extremely popular and extensively used datasets namely, MNIST and Extended Yale B.

MNIST Database: The MNIST database is a dataset of handwritten digits. For the purpose of our project we split the training and test sets equally. Each image is represented by 28-by-28 pixels, each containing a value 0 - 255 with its grayscale value. The digits have been size-normalized and centered in a fixed-size image. A sample of the MNIST database is shown in Fig. 2.

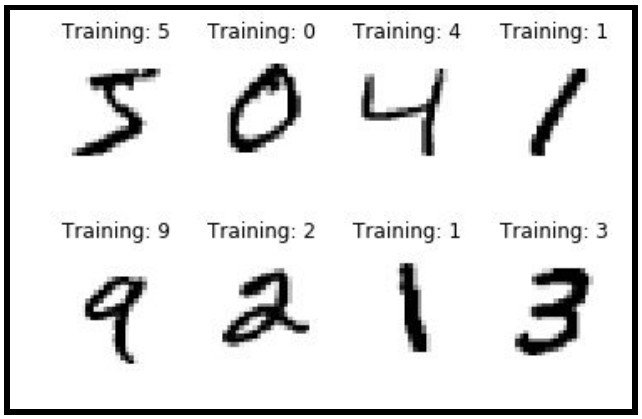


Fig. 2 Sample of MNIST Training Database

Extended Yale B Database: The Extended Yale B database includes 2,432 face images of 38 subjects captured under various lighting conditions. Each of the individuals have 64 images. A sample of the database consisting of four differently illuminated images of four distinct individuals in given in Fig 3.

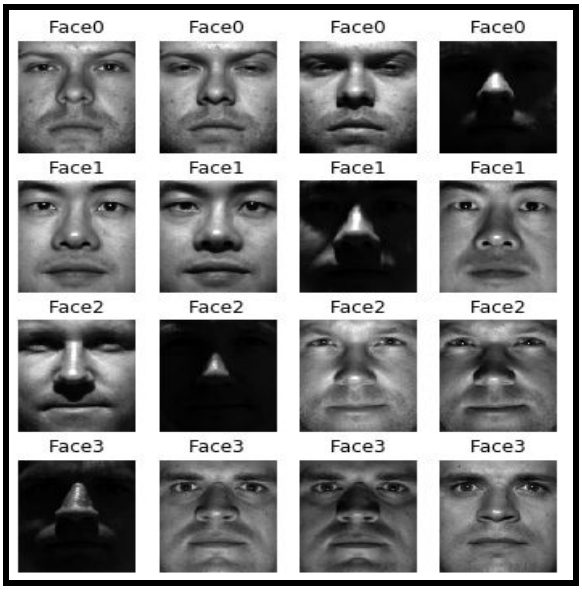


Fig 3. Sample of Extended Yale B Database

III. Feature Selection

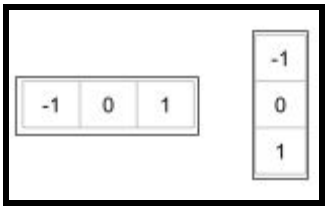
Feature selection plays a key role in many pattern recognition problems. While a great many of features can be utilized to characterize an image, only a few number of them are efficient and effective in classification. More features does not always lead to a better classification performance, thus feature selection is usually performed to select a compact and relevant feature subset in order to reduce the dimensionality of feature space, which will eventually improves the classification accuracy and reduce time consumption.

1. Feature Selection for MNIST Dataset using HOG

Histogram of Oriented Gradients: HOG are feature descriptors used in computer vision and image processing for the purpose of object detection[1]. The technique counts the occurrences of gradient orientation in localized portions of an image. HOG is based on the idea that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions.

Algorithm Overview: The steps for calculating the HOG descriptor for an image are listed below.

1. Gradient calculation: Calculate the x and the y gradient images, gx and gy, from the original image. This can be done by filtering the original image with the following kernels.



Using the gradient images gx and gy, we can calculate the magnitude and orientation of the gradient using the following equations.:

$$g = \sqrt{gx^2 + gy^2} \qquad \theta = arctan\ \frac{gy}{gx}$$

The calculated gradients are "unsigned" and therefore θ is in the range 0 to 180 degrees.

2. Cells : Divide the image into 14-by-14 cells.
3. Calculate histogram of gradients in these 14-by-14 cells: At each pixel in an 14-by-14 cell we know the gradient (magnitude and direction). Histogram of these gradients will provide a more useful and compact representation. We convert these 14x14x2 = 392 numbers into a 9-bin histogram (i.e. 9 numbers). The bins of the histogram correspond to gradients directions 0, 20, 40 … 160 degrees.
4. Block Normalization: The histogram calculated in the previous step is not very robust to lighting changes. Multiplying image intensities by a constant factor scales the histogram bin values as well. To counter these effects we can normalize the histogram.
5. Flattening into a feature vector

The HOG descriptor maintains a few key advantages over other descriptor methods. Since the HOG descriptor operates on localized cells, the method upholds invariance to geometric and photometric transformations, except for object orientation.

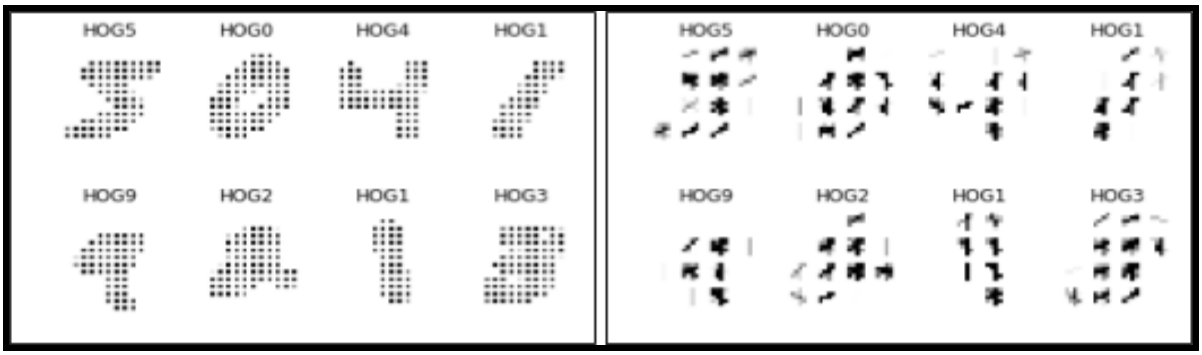


Fig. 4 a) 1 x 1 cell per block, 2 x 2 pixels per cell, b) 1 x 1 cell per block, 7 x 7 pixels per cell

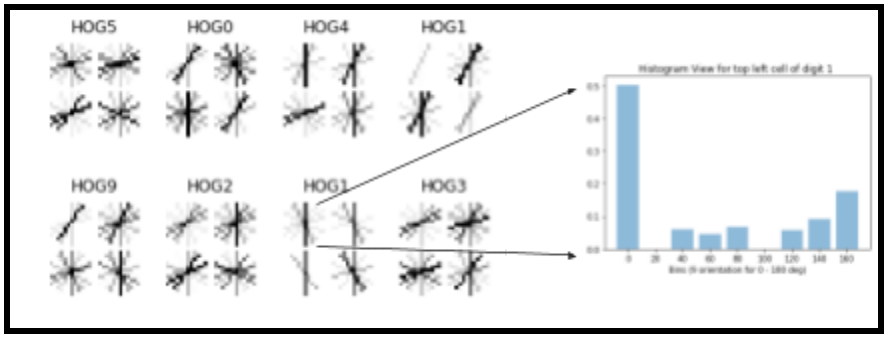


Fig. 5 1 x 1 cell per block, 14 x 14 pixels per cell

As we stated before the MNIST dataset size is 28-by-28 pixel, so we will have four blocks/cells of size 14-by-14 each. The orientation vector is set to 9. That means the HOG feature vector will be of size 4-by-9 = 36. The implemented HOG feature extraction for MNIST can be seen in the Fig. 4-5. Fig. 5 shows that a cell size of 14-by-14 does not encode much shape information, while a cell size of 2-by-2 and 7-by-7 encodes a lot of shape information but increases the dimensionality of the HOG feature vector significantly. In practice, the HOG parameters should be varied with repeated classifier training and testing to identify the optimal parameter settings.

The SVM classifier on MNIST dataset, has an average accuracy of 96% with HOG feature vectors using a 14-by-14 cell size whereas a 97% accuracy with a feature vectors using a 7-by-7 cell size. The computation time for training an SVM Classifier using a cell size of 7-by-7 was 24.5 sec whereas the computation time using a cell size of 14-by-14 was 18.16 sec. Using a tradeoff between accuracy and time we chose generation of HOG feature vectors using a cell size of 14-by-14. On using a Naive Bayes classifier on the MNIST database, the classifier performed better with feature extraction using a 14-by-14 cell size with a accuracy of 84.83% whereas a accuracy of 79.16% using a cell size of 7-by-7 to extract the features. The tables below summarize the comparison information.

Cell Size	SVM - Linear Kernel	Naive Bayes
14-by-14	96.02%	84.83%
7-by-7	96.78%	79.16%

Table 1. Comparison of Accuracy using a 7-by-7 cell size and 14-by-14 cell size

Cell Size	SVM - Linear Kernel	Naive Bayes
14-by-14	18.16 sec	0.027 sec
7-by-7	24.5 sec	0.088 sec

Table 2. Comparison of Training Time using a 7-by-7 cell size and 14-by-14 cell size

2. Feature Selection for Extended Yale B Dataset using PCA

The Principal Component Analysis (PCA) is one of the most successful techniques that have been used in image recognition and compression. PCA is a statistical method under the broad title of factor analysis. The purpose of PCA is to reduce the large dimensionality of the data space to the smaller intrinsic dimensionality of feature space, which are needed to describe the data economically. This is the case when there is a strong correlation between observed variables.

Algorithm Overview: The main idea of using PCA for face recognition is to express the large 1-D vector of pixels constructed from 2-D facial image into the compact principal components of the feature space. This can be called eigenspace projection. Eigenspace is calculated by identifying the eigenvectors of the covariance matrix derived from a set of facial images. The eigenvectors are obtained as the columns of the orthogonal matrix in the spectral decomposition of the covariance or correlation matrix, S or R. More specifically, because R is symmetric, an orthogonal matrix V exists such that

$$V'RV = D \text{ or equivalently, } R = VDV'$$

where D is a diagonal matrix whose diagonal elements are the eigenvalues. The eigenvectors are the columns of V.

A vital part of using PCA in practice is the ability to estimate how many components are needed to describe the data. This can be determined by looking at the cumulative explained variance ratio as a function of the number of components:

The cumulative proportion of sample variance explained by the first k principal components is calculated as follows:

$$\frac{\lambda_1+\lambda_2+....+\lambda_k}{\lambda_1+\lambda_2+....+\lambda_p}$$

λ_k *is the* k^{th} *eigenvalue* p, *the number of variables*

The graphical representation for cumulative proportion of sample variance is given in Fig. 6

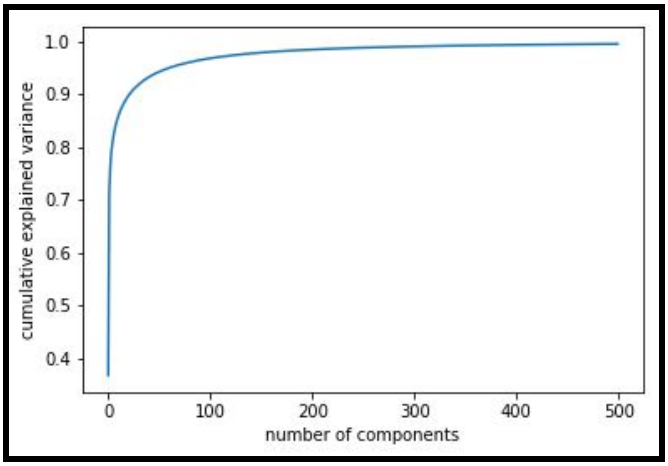


Fig. 6 Cumulative proportion of sample variance

This curve quantifies how much of the total, 32256-dimensional variance is contained within the first N components. For example, we see that with the digits the first 50 components contain approximately 90% of the variance, while you need around 400 components to describe close to 100% of the variance.

When choosing the number of principal components (k), we choose k to be the smallest value so that for example, 99% of variance, is retained. In the case of Extended Yale B database the number of components required to achieve 99% variance is 300. Fig. 7 below is a representation of the first four eigenfaces for four distinct individuals. The first eigenface account for the maximal variation of the training vectors, and the second one accounts for the second maximal variation, etc.

Fig. 7 Approximation of the original data after PCA

IV. Classification Algorithm

Instead of trying to specify what every one of the categories of interest (K classes) look like directly in code, we provide the classifier with many examples of each class and then develop learning algorithms that look at these examples and learn about the visual appearance of each class. This approach is referred to as a data-driven approach, since it relies on first accumulating a training dataset of labeled images.

1. Support Vector Machine (SVM) Classifier

In this project we train the Support Vector Machines using the Sequential Minimal Optimization (SMO) algorithm. Training a support vector machine requires the solution of a very large quadratic programming (QP) optimization problem. SMO breaks this large QP problem into a series of smallest possible QP problems. These small QP problems are solved analytically, which avoids using a time-consuming numerical QP optimization as an inner loop. The amount of memory required for SMO is linear in the training set size, which allows SMO to handle very large training sets.

SVM Multiclass Strategies: SVM classification is essentially a binary (two-class) classification technique, which has to be modified to handle the multiclass tasks in real world situations e.g. face recognition between multiple individuals or handwritten digits classification.

The pseudo-code below describes the entire SMO algorithm [2]:

```
target = desired output vector
 point = training point matrix
 procedure takeStep(i1,i2)
        if (i1 == i2) return 0
        alph1 = Lagrange multiplier for i1
        y1 = target[i1]
        E1 = SVM output on point[i1] – y1
        s = y1*y2
        Compute L, H via equations (13) and (14) in [2]
        if (L == H)
                return 0
        k11 = kernel(point[i1],point[i1])
        k12 = kernel(point[i1],point[i2])
        k22 = kernel(point[i2],point[i2])
        eta = k11+k22-2*k12
        if (eta > 0)
        {
                a2 = alph2 + y2*(E1-E2)/eta
                if (a2 < L) a2 = L
                else if (a2 > H) a2 = H
        }
        else
        {
                Lobj = objective function at a2=L
                Hobj = objective function at a2=H
                if (Lobj < Hobj-eps)
                        a2 = L
                else if (Lobj > Hobj+eps)
                        a2 = H
                else
                        a2 = alph2
        }
        if (|a2-alph2| < eps*(a2+alph2+eps))
                return 0
        a1 = alph1+s*(alph2-a2)
Update threshold to reflect change in Lagrange multipliers
Update weight vector to reflect change in a1 & a2,
if SVM is linear
          Update error cache using new Lagrange multipliers
        Store a1 in the alpha array
        Store a2 in the alpha array
        return 1
End procedure
```

```
procedure examine Example(i2)
          y2 = target[i2]
          alph2 = Lagrange multiplier for i2
          E2 = SVM output on point[i2] – y2
          r2 = E2*y2
 if ((r2 < -tol && alph2 < C) || (r2 > tol && alph2 > 0))
   {
    if (number of non-zero & non-C alpha > 1)
    {
          i1 = result of second choice heuristic (section 2.2 in [2])
          if takeStep(i1,i2)
             return 1
    }
  loop over all non-zero and non-C alpha, starting at a random
   point
    {
          i1 = identity of current alpha
          if takeStep(i1,i2)
             return 1
                    }
   loop over all possible i1, starting at a random point
    {
          i1 = loop variable
          if (takeStep(i1,i2)
             return 1
    }
    }
   return 0
End procedure
main routine:
    numChanged = 0;
    examineAll = 1;
    while (numChanged > 0 | examineAll)
    {
     numChanged = 0;
     if (examineAll)
          loop I over all training examples
             numChanged += examineExample(I)
     else
          loop I over examples where alpha is not 0 & not C
             numChanged += examineExample(I)
              if (examineAll == 1)
                 examineAll = 0
              else if (numChanged == 0)
                 examineAll = 1
    }
```

Two of the common methods to enable this adaptation include the 1A1 and 1AA techniques. The 1AA approach represents the earliest and most common SVM multiclass approach (Melgani and Bruzzone, 2004) and involves the division of an N class dataset into N two-class cases. The 1A1 approach on the other hand involves constructing a machine for each pair of classes resulting in N(N-1)/2 machines. When applied to a test point, each classification gives one vote to the winning class and the point is labeled with the class having most votes. This approach can be further modified to give weighting to the voting process. From machine learning theory, it is acknowledged that the disadvantage the 1AA approach has over 1A1 is that its performance can be compromised due to unbalanced training datasets (Gualtieri and Cromp, 1998), however, the 1A1 approach is more computationally intensive since the results of more SVM pairs ought to be computed. In this project we use the 1A1 approach.For handwritten digit classification using the MNIST dataset we require to train 45 classifiers to distinguish between 10 digits. The face recognition system will require 703 classifiers to distinguish between 38 distinct individuals.

2. Naive Bayes Classifier

In this project we train a Gaussian Naïve Bayes Classifier, based on applying Bayes' theorem with the 'naive' assumption of independence between every pair of features. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

$$P(y/x) = \frac{P(x/y)P(y)}{P(x)}$$

$$P(y/x) = P(x_1/y) * P(x_2/y) * ..... * P(x_n/y) * P(y)$$

- P(y|x) is the posterior probability of class (y, target) given predictor (x, attributes).
- P(y) is the prior probability of class.
- P(x|y) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor.

The likelihood of the features is assumed to be Gaussian. The parameters σ_y and μ_y are estimated using maximum likelihood.

$$P(x_i/y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp(\frac{-(x_i-\mu_y)^2}{2\sigma_y^2})$$

For the classification rule we use the following modified Bayes' theorem assuming independence between features:

$$P(y/x_1....x_n) \propto P(y) \prod_{i=1}^{n} P(x_i/y)$$

$$\hat{y} = arg\ max_y\ P(y) \prod_{i=1}^{n} P(x_i/y)$$

Where $P(y/x_1....x_n)$ is the posterior probability and $\hat{y}$ is the predicted class for the testing data that maximizes the posterior Gaussian distribution.

The pseudo-code below describes the entire Naive Bayes algorithm:

Training the classifier:	Testing the classifier:
Inputs : 1. N by L 2D array of training data where N is number of training samples and L is number of features of the each sample. 2. 1D array of size N, of labels associated with each training sample i.e the label represents the class of each sample. Outputs : 1. The parameters σ_y and μ_y in the form of an M by L - 2D array where M is the number of classes. So each class has mean and standard deviation associated with it. 2. The prior (probability) of each class P(yi) where i = 1,2,3…M. Start: Nlabels = No. of classes. L = No. of features For i in range of 1 to Nlabels find : $P(y_i) = \frac{(No.\ of\ samples\ for\ class\ i)}{Total\ No.\ of\ samples}$ σ_{yi} = For each class find the standard deviation for each feature , a 1D array of size L. #Substitute the zero standard deviation values with smallest standard deviation values. μ_{yi} = For each class find the mean for each feature, so we get a 1D array of size L. Return $P(y)$, σ_y *and* μ_y Stop	Inputs : 1. N by L 2D array of testing data, where N is number of testing samples and L is number of features of the each sample. Outputs : 1. 1D array of size N, of labels associated with each testing sample i.e the class predicted by the classifier. Start: For each image in testing data find: $P(y/x_1....x_n)$ the posterior probability and the class $\hat{y}$ using the formula. Return $\hat{y}$ Stop

V. Cross Validation

The most common technique for model evaluation and model selection in machine learning practice is k-fold cross-validation. It can be thought of as a crossing over of training and validation stages in successive rounds. Here, the main idea behind cross-validation is that each sample in our dataset has the opportunity of being tested. K-fold cross-validation is a special case of cross-validation where we iterate over a dataset set k times. In each round, we split the dataset into k parts: one part is used for validation, and the remaining k-1 parts are merged into a training subset for model evaluation as shown in the figure below, which illustrates the process of 5-fold cross-validation:

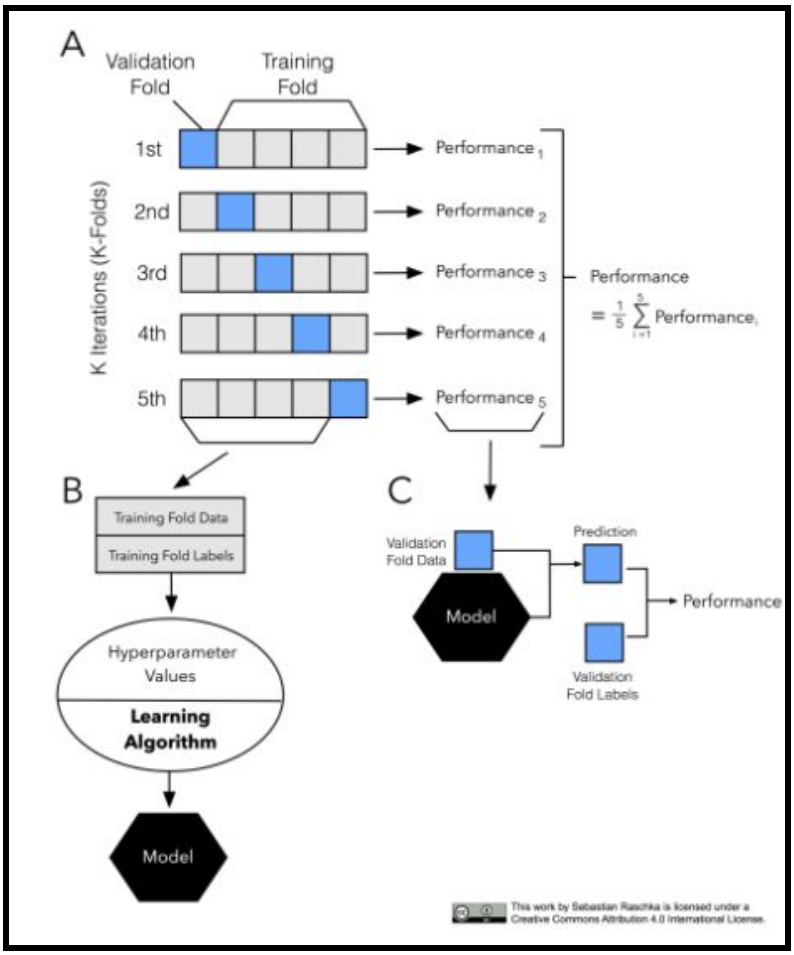


Fig. 8 Representation for 5 - fold Cross Validation

In 5-fold cross-validation, this procedure will result in five different models fitted; these models were fit to distinct yet partly overlapping training sets and evaluated on non-overlapping validation sets. Eventually, we compute the cross-validation performance as the arithmetic mean over the k performance estimates from the validation sets. To do hyperparameter tuning, iterate through a wide range of hyperparameter combinations. Then, choose the set of parameters for which k-folds reports the highest accuracy.

1. Tuning of SVM Hyperparameters

Regularization Parameter C and Kernel

SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. The kernel functions return the inner product between two points in a suitable feature space. Thus by defining a notion of similarity, with little computational cost even in very high-dimensional spaces. In the project we implement SVM classifier using three different kernels namely Linear, Quadratic and Gaussian.

The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C we could get misclassified examples, often even if your training data is linearly separable.

2. Tuning of Naive Bayes Classifier

The Naive Bayes Classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. In this project we implement a Gaussian Naive Bayes Classifier in which Gaussian distribution likelihoods were assumed. The Mean and Variance parameters were estimated using MLE. The Naive Bayes Algorithm does not have hyperparameters that can be tuned. We perform a 5 - fold Cross Validation was on the training dataset to determine the average accuracy.

VI. Demonstration of Performances

1. SVM Classifier

The SVM classifier is fitted using the best parameters obtained from tuning using the 5 - fold cross validation approach. The demonstration of the implementation for SVM Classifier is given below in Fig.9 and Fig.10 along with the test image and the predicted labels. A Testing accuracy of 95.83% was achieved for the MNIST database, whereas, the testing accuracy achieved for YaleB database was 92.85%.

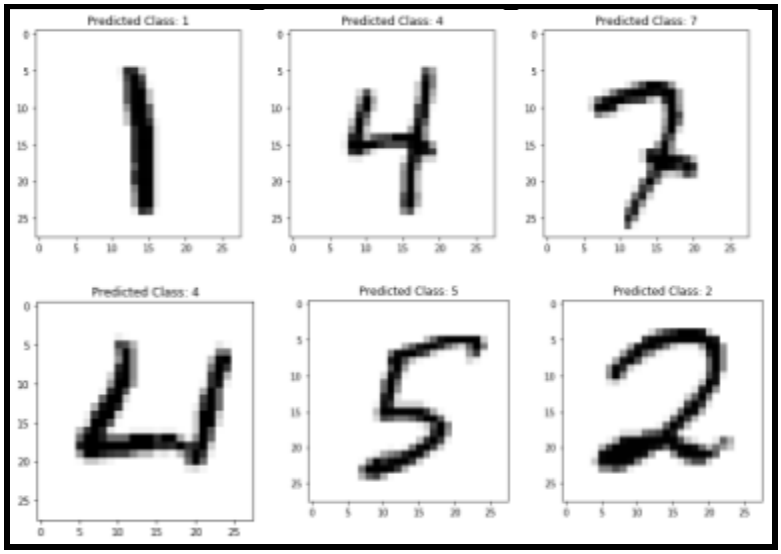


Fig.9 Demonstration on Test data for MNIST Database

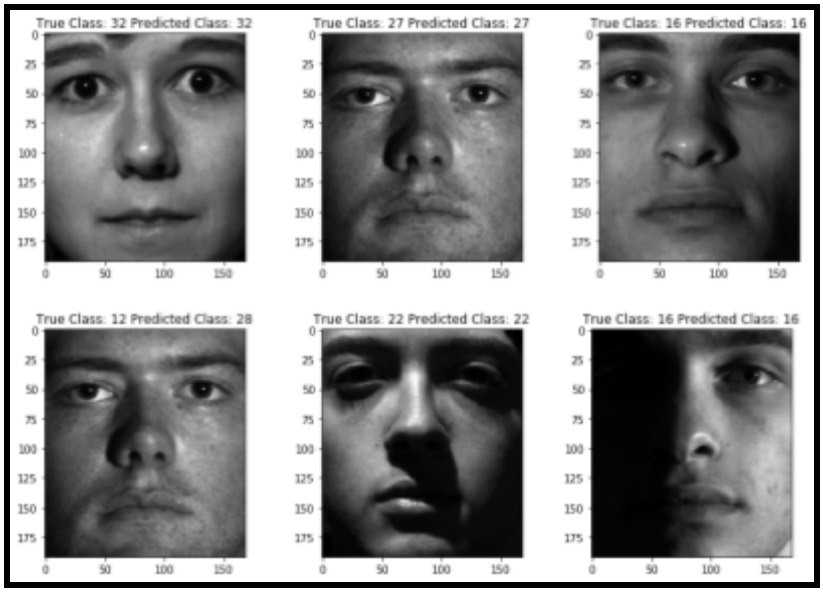


Fig.10 Demonstration on Test data for YaleB Database

The below graphs compare the performance of the SVM classifier for different values of C parameter and for different Kernels used. It is seen from the graphs in Fig. 11 that all the three Kernels behave the similarly for the MNIST dataset and the best Hyperparameter value found for Regularization constant is C = 100.

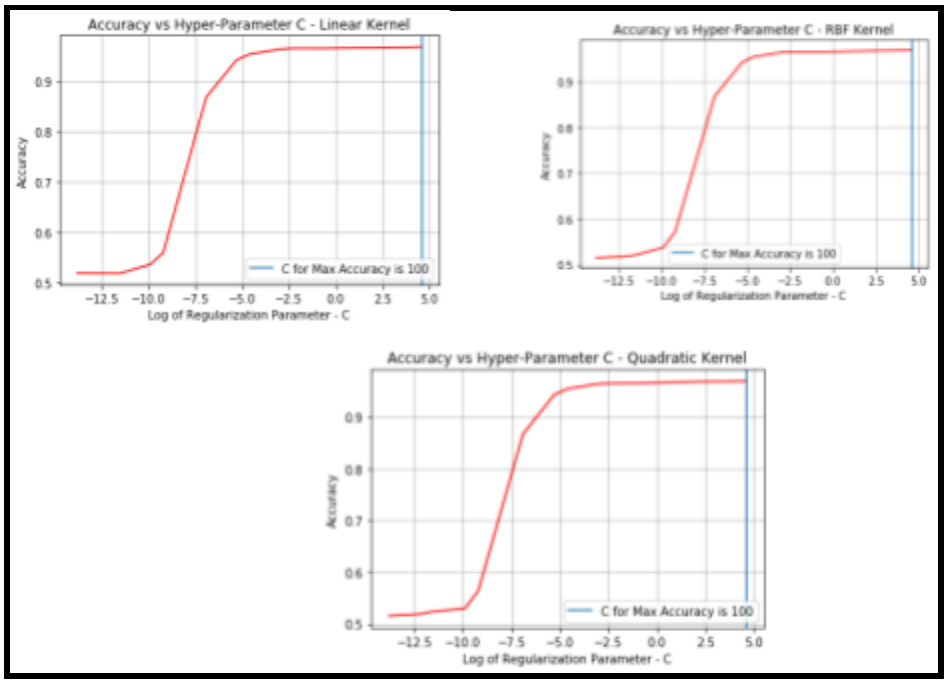


Fig.11 Performance v/s Hyperparameter for MNIST Database

Fig. 12 shows the SVM classifier performance for different hyperparameters for the YaleB database. It is found that we obtain the best performing SVM classifier using the Gaussian Kernel with a Regularization parameter value of C = 0.1.

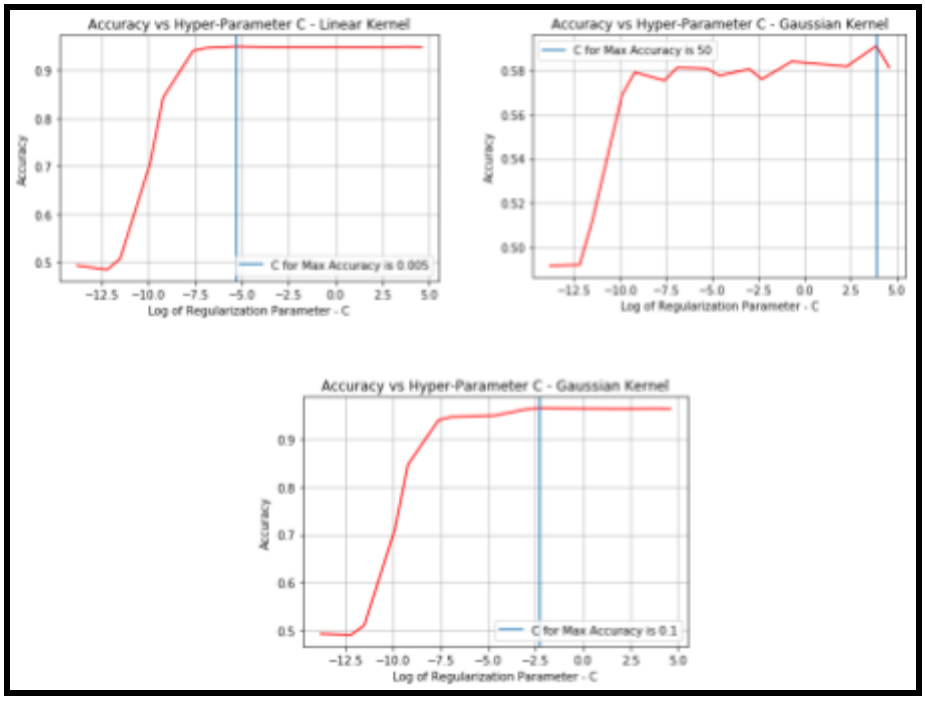


Fig.12 Performance v/s Hyperparameter for YaleB Database

2. Naive Bayes Classifier

The demonstration of the implementation for NB Classifier is given below along with the test image, the true label and the predicted label. A Testing accuracy of 84.84% was achieved for the MNIST database, whereas, the testing accuracy achieved for YaleB database was 92.85%.

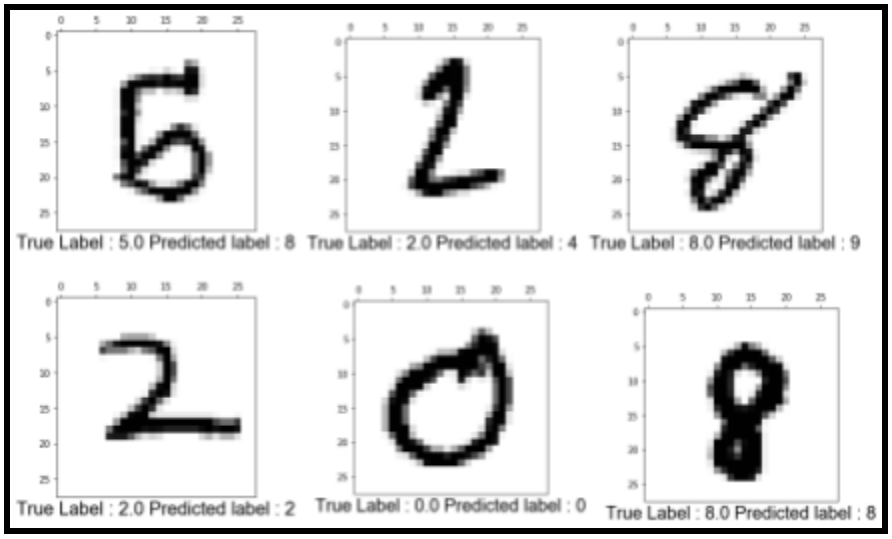


Fig. 13 Demonstration on Test data for MNIST Database

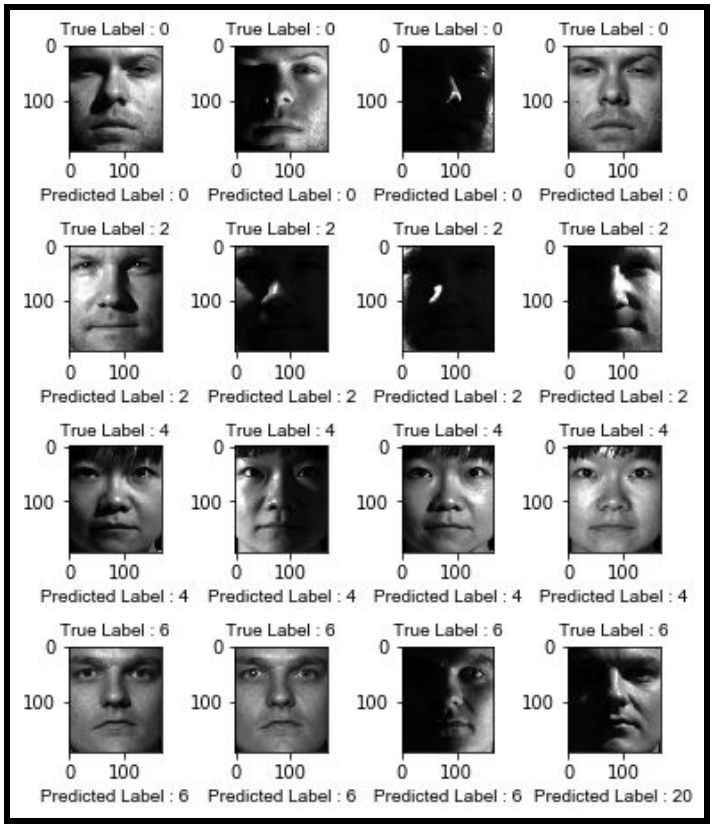


Fig. 14 Demonstration on Test data for YaleB Database

We illustrate the performance of the Naive Bayes classifier using ROC Curves. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection in machine learning. The false-positive rate is also known as the fall-out or probability of false alarm and can be calculated as (1 − specificity). Fig.15 gives the ROC Curves for MNIST Database and Fig.16 gives the ROC curves for the YaleB Database.

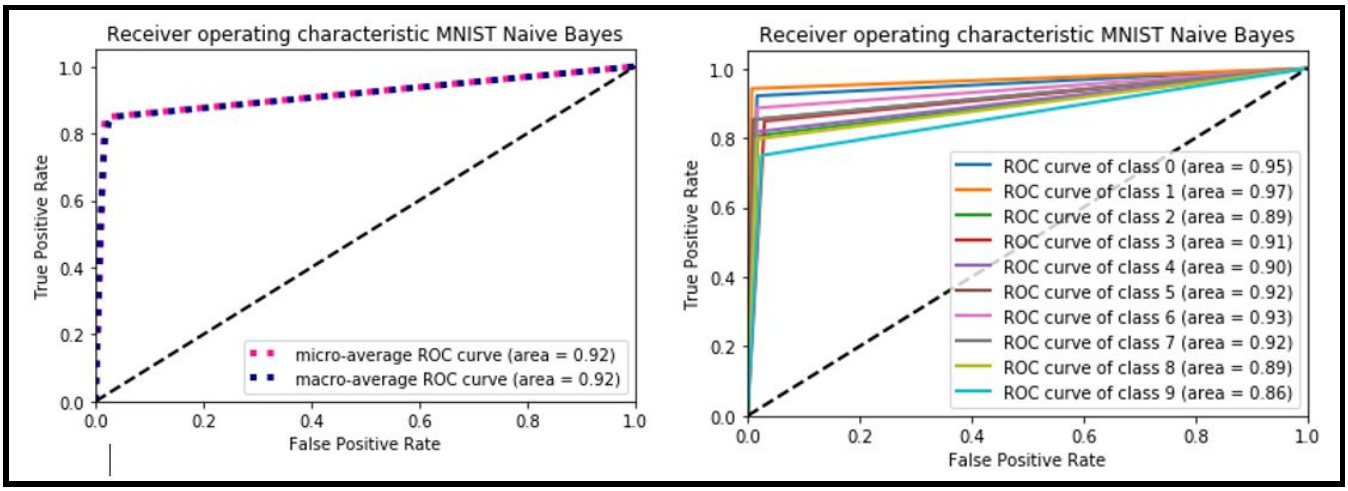


Fig. 15 ROC Curves for MNIST Database

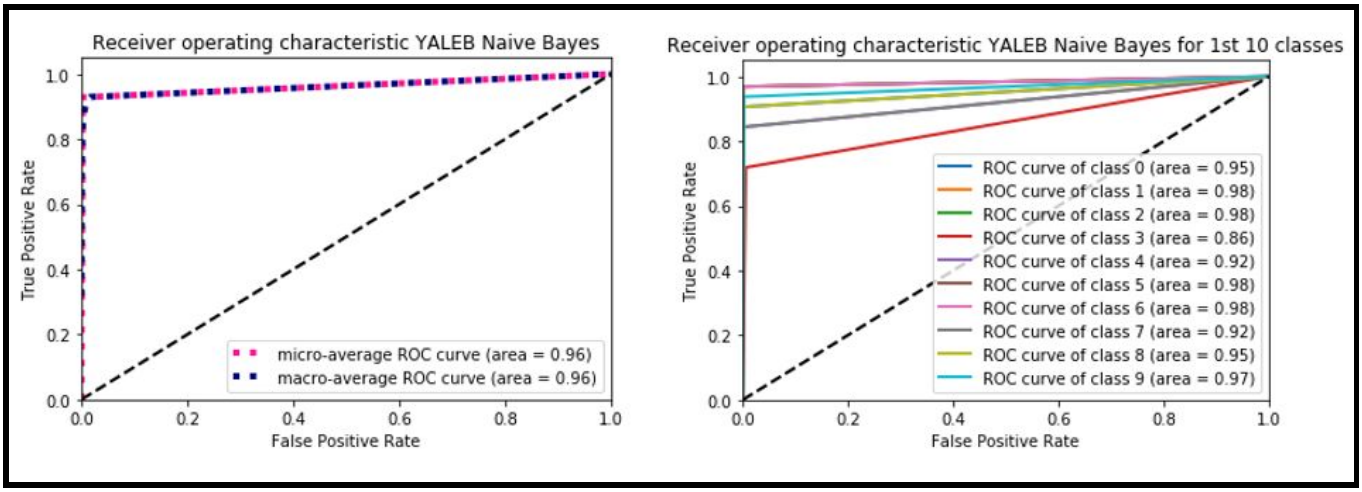


Fig. 16 ROC Curves for YaleB Database

VII. Analysis of Results

1. SVM Classifier

Based on the analysis of the results in section VI the advantages and disadvantages of using a SVM Classifier is as given below:

Drawbacks and Solutions:

1. Selecting the best kernel function for a problem can prove to be tricky. **We use a 5-fold cross validation technique to find the kernel that provides the best accuracy.**
2. For a multi class problem, we have the One vs All or One vs One approach, but no multi class SVM. **In this project we apply the One Vs One approach.**
3. Computationally complex and requires lot of memory. Learning can take a very long time for large scale problems. **A tradeoff between the time complexity and convergence rate is required.**
4. SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

Advantages:

1. Guaranteed Optimality, due to the nature of Convex Optimization, the solution is guaranteed to be the global minimum not a local minimum.
2. The Kernel-based framework is very powerful,flexible approach. SVM works very well in practice, even with small training sample sizes.
3. SVM finds the optimal separating hyperplane and can deal with very high dimensional data.
4. It is useful for both Linearly Separable(hard margin) and Non-linearly Separable(soft margin) data.

2. Naive Bayes Classifier

Based on the analysis of the results in section VI the advantages and disadvantages of using a NB Classifier is as given below:

Drawbacks and Solutions:

1. Naive Bayes classifier makes a very strong assumption of Independence on the shape of the data distribution, i.e. any two features are independent given the output class. The result of this can be bad - hence, a "naive" classifier. It can be observed even from the results of this project that the accuracy for MNIST database is 84.84% which is suboptimal. **The NB classifier can be optimal even if the assumption is violated and its results can be good even in the case of sub-optimality.**
2. If a categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a zero probability and will be unable to make a prediction. This is

often known as Zero Frequency. **To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.**

3. A third problem arises for continuous features. It is common to use a binning procedure to make them discrete, but if we are not careful we can throw away a lot of information. **The possible solution is to use Gaussian distributions for the likelihoods as done in this project.**

Advantages:

1. Very fast, efficient,does not require much memory and often produces good results. The results for YaleB database where the features are not independent, we still receive an accuracy of 92.85%. Sometimes NB turns out to be better or at least equally good as other, more sophisticated algorithms.
2. Has low affinity to over-fitting and performs well even with small training set. It performs well in case of categorical input variables compared to numerical variable(s) and for numerical variable, normal distribution is assumed.
3. Resistant to the low-importance attributes i.e. attributes that are equally distributed through the overall training set, and thus do not have significant impact on the class label.

VIII. References

[1] Dalal N, Triggs B. Histograms of oriented gradients for human detection[C]//Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. IEEE, 2005, 1: 886-893

[2] Platt, John (1998), Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines 10.1.1.43.4376

[3] Osuna, E., Freund, R., Girosi, F., "Training Support Vector Machines: An Application to Face Detection," Proc. Computer Vision and Pattern Recognition ’97, 130-136, (1997).

[4] LeCun, Y., Jackel, L. D., Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Muller, U. A., Sackinger, E., Simard, P. and Vapnik, V., "Learning Algorithms for Classification: A Comparison on Handwritten Digit Recognition," Neural Networks: The Statistical Mechanics Perspective, Oh, J. H., Kwon, C. and Cho, S. (Ed.), World Scientific, 261-276, (1995)