Eight Class Scene Classification using various Algorithms

Indrojyoti Mondal[1], *Abhishek Mamidi*[2], Advisor: *Dr.Anish Chand Turlapty* and *Dr.Rangeet Mitra*
Indian Institute of Information Technology, Chittoor, Sri City, A.P, India
indrojyoti.m15@iiits.in[1], *abhishek.m15@iiits.in*[2], *anish.turlapaty@iiits.in*, *rangeet.mitra@iiits.in*

Abstract

In this paper, eight class classification is done using various algorithms such as SVM, LMS, KLMS, MCC, MEE and PNN and compare their performances using confusion matrices. The dataset used in this paper is taken from Computational Visual Cognitive Library from MIT published by Oliva, A. & Torralba, A. (2001).Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope.

Index Terms

Multivariate Linear Regression, Brainhead, Regression

I. Introduction

MACHINE Learning has many problems, classification is one such problem of identifying to which of a set of categories (classes) a new element (observation) belongs, based on a training set of data containing elements (instances) whose class membership is known. In this paper, we are dealing with a eight class classification problem. The database that is chosen for this problem comprises of a few hundred images of scenes belonging to the some semantic category. This dataset of images are classified into eight different classes based on their similar semantic scene, i.e. into Coast & Beach, Open country, Inside City, Forest, Mountain, Highway, Street and Tall buildings using various algorithms such as SVM, LMS, KLMS, MCC, MEE and PNN and compare their performances.

II. Literature review

A. Dataset

This dataset contains 8 outdoor scene categories: Coast & beach, Open country, Inside City, Forest, Mountain, Highway, Street and Tall buildings. All of the images are in color, in jpeg format, and are 256 x 256 pixels. The sources of the images vary (from commercial databases, websites, digital cameras). There are 2600 color images, 256x256 pixels. All the objects and regions in this dataset have been fully labeled. There are more than 29.000 objects. This dataset is taken from Computational Visual Cognitive Library @ MIT published by Oliva, A. & Torralba, A. (2001).Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope.

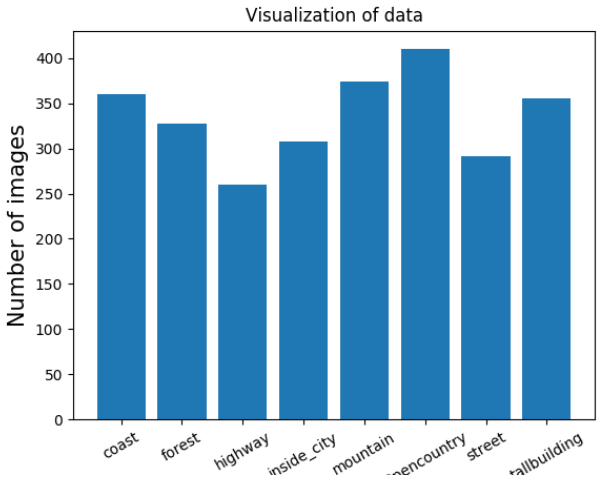

Fig. 1: Feature Extraction using bag of Visual words

B. Least Means Square Algorithm

This subsection provides a brief completeness of the LMS Algorithm. The hypothesis space consists H_1 of all the linear operators on U, denoted by $w = U \rightarrow R$. Since U is embedded in a Hilbert space, the linear operator becomes the standard inner product. [1] So, LMS minimizes the risk function:

$$\min_{w} R_{emp}[w \in H_1, Z^n] = \sum_{i=1}^{N} (y_i - w(u_i))^2 \tag{1}$$

Using Stochastic Gradient :

$$w_n = w_{n-1} + \eta e_n^a u_n \tag{2}$$

where e_n^a is called the priori error and η is the step size. Therefore, after n-step training, the weight is expressed as the linear combination of the previous and current input data weighted by the priori errors. Expressing the system in terms of inner products.

$$\overline{y} = w_n(\overline{u}) = \eta \sum_{i=1}^{n} e_i^a < u_i, \overline{u} >_U \tag{3}$$

C. Kernel LMS Algorithm

A kernel is a symmetric, positive-definitive, continuous function. $k : \mathbf{U} \times \mathbf{U} \rightarrow \mathbf{R}$. Gaussian kernel is a commonly used one.

$$(u_i, u_j) = exp(-a||u_i - u_j||^2) \tag{4}$$

By using Mercer's theorem, any kernel can be mapped to Φ such that

$$k(u_1, u_2) = \Phi(u_i)\Phi^T(u_2), \forall u1, u2 \in U \tag{5}$$

Now data u_i can be transformed into Feature space $\mathbf{F}$ as Φ_i by utilizing the above theorem. KLMS is just like LMS performed on $(\Phi(u_1), y_1), ..., (\Phi(u_N), y_N)$.

$$\Omega_n = \Omega_{n-1} + \eta e_n^a \Phi(u_n) \tag{6}$$

$$\Omega_{n-1}(\Phi(u_n)) = \eta \sum_{i=1}^{n} e_i^a k(u_i, u_n) \tag{7}$$

After N step the final relation of algorithm turns out to be

$$\Omega_n = \eta \sum_{i=1}^{n} e_i^a \Phi(u_i) \tag{8}$$

$$\overline{y} = \eta \sum_{i=1}^{n} e_i^a k(u_i, \overline{u}) \tag{9}$$

D. Maximum Correntropy criterion

In adaptive filters, the goal is to maximize correntropy between the desired signal d_i and filter output y_i.

$$J_n = \frac{1}{N} \sum_{i=n-N+1}^{n} k_\sigma(d_i, y_i) \tag{10}$$

where k_σ is the kernel with width σ and N being the number of samples in the Parzen estimate window. [3] An iterative gradient descent approach is used to search for an optimal solution.

$$w_{n+1} = w_n + \eta \nabla J_n \tag{11}$$

Substitute J_n in the above equation.

$$w_{n+1} = w_n + \frac{\eta}{N} \sum_{i=n-N+1}^{n} \frac{\partial k_\sigma(d_i, yi)}{\partial w_n} \tag{12}$$

For online mode, $(N = 1)$ approximates the gradient descent.

$$\begin{aligned} w_{n+1} &= w_n + \tfrac{\eta}{N} \textstyle\sum_{i=n-N+1}^{n} \frac{\partial k_\sigma(d_i, yi)}{\partial w_n} \\ &= w_n + \eta g(e_n) u_n \\ &= w_n + \eta exp(\tfrac{-e_n^2}{2\sigma^2}) e_n u_n \end{aligned} \tag{13}$$

in which $e_n = d_n - w_n^T u_n$ is the prediction error and $g(e_n) = exp(\frac{-e_n^2}{2\sigma^2})e_n$ for the Normalized Gaussian Kernel.

E. Minimum Error Entropy

MEE happens to be one of the most important learning criterion in information theoretic learning. [4] MEE can be formulated as

$$\Omega_k = \Omega_{k-1} + \eta \frac{\sum_{i=k-l+1}^{k} k'_h(e_k - e_i)(X_k - X_i)}{\sum_{i=k-l+1}^{k} k_h(e_k - e_i)} \tag{14}$$

where $e_k = d_k - \Omega_k^T X_k$ and $k_h(e_k - e_i) = exp(-h(e_k - e_i)^2$

F. Multi-Class SVM

Given l samples $\overline{x_1},\ y_1, ...,\ \overline{x_l},\ y_l$ with $\overline{x_i} \in R^n,\ y_i \in Y, \forall i$ and $Y = 1, ..., M$, a multiclass classifier predicts the label $y = Y$ of an unseen sample $\overline{x} \in R^n$. Multiclass SVM can be defined as combination of N independent binary classification tasks. Binary tasks are defined as an output code matrix R of size $M \times N$ and $R_{ij} \in -1, 0, 1$.

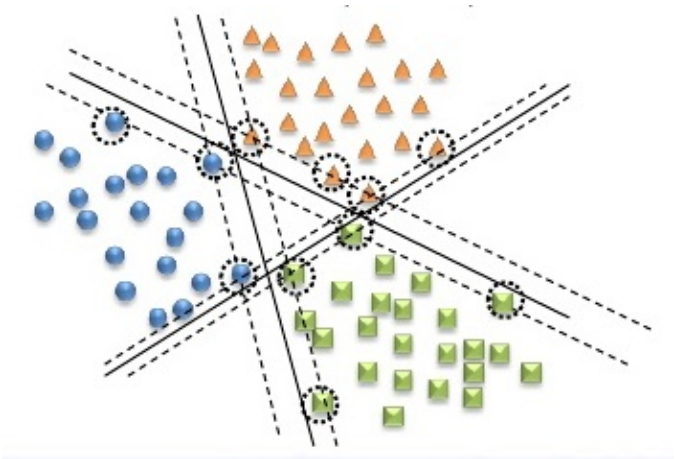

Fig. 2: Multi-class SVM

G. Probabilistic Neural Networks

In most of the neural networks sigmoid activation function is used. Aprobabilistic neural network (PNN) is one such neural network where the activation function is replaced with an exponential function. This neural network can compute non-linear decision boundaries which approach the Bayes optimal that is formed. In PNN, the operations are organized into a multilayered feedforward network with four layers:

1) Input layer
2) Hidden layer
3) Output layer

This PNN offers a tremendous speed advantage for problems in which the incremental adaptation time of back propagation is a significant fraction of the total computation time. For one application, the PNN paradigm was 200,000 times faster than back-propagation. [5]

III. Methodology

A. Feature Extraction

The main task is to convert each image as a feature vector, which represents the image. For this, we are using Bag of visual words model. Images are represented based on their frequencies of visual words. The steps involved in this feature extraction process are described below.

1) Feature extraction of each image - Extract SIFT features from each image and the dimension of each SIFT feature is 128. A image can have multiple SIFT features. The number of SIFT features vary from image to image.
2) Learning the visual vocabulary - Combine all the SIFT features of all images and apply K-means. We get K cluster centroids. These K cluster centroids are known as visual words.
3) Represent Images by frequencies of visual words: We represent each image by a K dimensional vector which is the normalized counts of visual words.

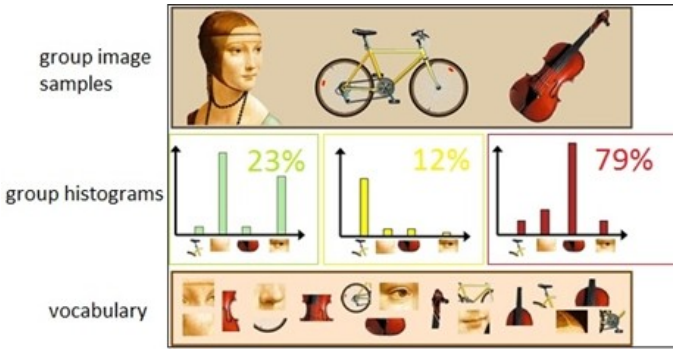

Fig. 3: Feature Extraction using bag of Visual words

IV. Implementation

We have implemented all the algorithms in Python by extracting SIFT features, learning the visual vocabulary, by applying K-means. We took K = 16 for feature extraction.

A. SVM

```
def svm_classifer(train_images, train_labels):
        rbf_svc = SVC(kernel='linear', gamma=0.1, C=1)
        rbf_svc.fit(train_images, train_labels)
        return rbf_svc
```

Fig. 4: SVM implementation in Python

B. LMS

```
mu = 0.01
w = np.zeros((1,len(X_train[0])))

errors = []
for i in range(dividing_point):
        e_i = y_train[i] - np.matmul(w,np.reshape(X_train[i], (X_train[i].shape[0],1)))
        e_i = np.reshape(e_i, ())
        errors.append(e_i)
        w = w + mu * int(e_i) * np.reshape(X_train[i], (1, X_train[i].shape[0]))

y_pred = np.matmul(X_test, np.transpose(w))
y_pred = np.round(y_pred)
```

Fig. 5: LMS implementation in Python

C. KLMS

```
mu = 0.1
sigma = 2
errors = []
e_i = y_train[0]
errors.append(e_i)
for i in range(1, dividing_point):
        sum = 0
        for j in range(i):
                num = (np.linalg.norm(X_train[j]-X_train[i])**2)*(-1.0)
                den = sigma * sigma * 1.0
                sum = sum + (errors[j] * np.exp(num/den))
        y_i = mu*sum
        e_i = y_train[i] - y_i
        errors.append(e_i)

y_pred = []
for i in range(len(X_test)):
        sum = 0
        for j in range(len(errors)):
                num = (np.linalg.norm(X_train[j]-X_test[i])**2)*(-1.0)
                den = sigma * sigma * 1.0
                sum = sum + (errors[j] * np.exp(num/den))
        y_i = mu*sum
        y_pred.append(np.round(y_i))
```

Fig. 6: KLMS implementation in Python

D. MCC

```
mu = 0.001
w = np.zeros((1,len(X_train[0])))

errors = []
sigma = 2
for i in range(dividing_point):
    e_i = y_train[i] - np.matmul(w,np.reshape(X_train[i], (X_train[i].shape[0],1)))
    e_i = np.reshape(e_i, ())
    errors.append(e_i)
    w = w + mu * np.exp((int(e_i)*int(e_i)*-1)/(2.0*sigma*sigma)) * int(e_i) * np.reshape(X_train[i], (1, X_train[i].shape[0]))

y_pred = np.matmul(X_test, np.transpose(w))
y_pred = np.round(y_pred)
```

Fig. 7: MCC implementation in Python

E. MEE

```
mu = 0.1
sigma = 2
w = np.zeros((1,len(X_train[0])))
k = 500

for i in range(dividing_point):
        num = 0
        den = 0
        e_i = y_train[i] - np.matmul(w,np.reshape(X_train[i], (X_train[i].shape[0],1)))
        e_i = np.reshape(e_i, ())
        for j in range(max(0, i-k), i):
                e_j = y_train[j] - np.matmul(w,np.reshape(X_train[j], (X_train[j].shape[0],1)))
                e_j = np.reshape(e_j, ())
                k_n = np.exp((-1.0)*((e_i-e_j)**2)*(1.0/sigma))
                den += den + k_n
                num += num + k_n * (-2.0) * (1.0/sigma) * (e_i-e_j) * (X_train[i] - X_train[j])
        #print i, num, den
        if den!=0:
                w = w - mu * (num*1.0/den*1.0)

y_pred = np.matmul(X_test, np.transpose(w))
y_pred = np.round(y_pred)
```

Fig. 8: MEE implementation in Python

F. PNN

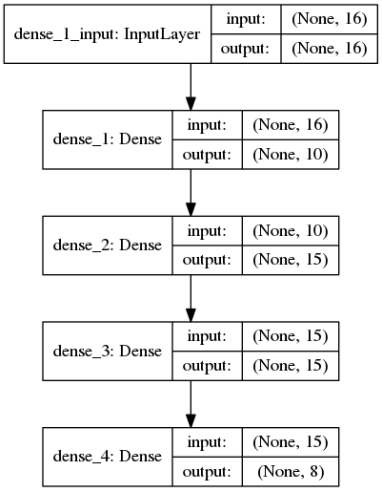

Fig. 9: Model used in PNN

V. Results

A. SVM

The accuracy obtained in SVM algorithm was the highest. The accuracy obtained is 58.9%

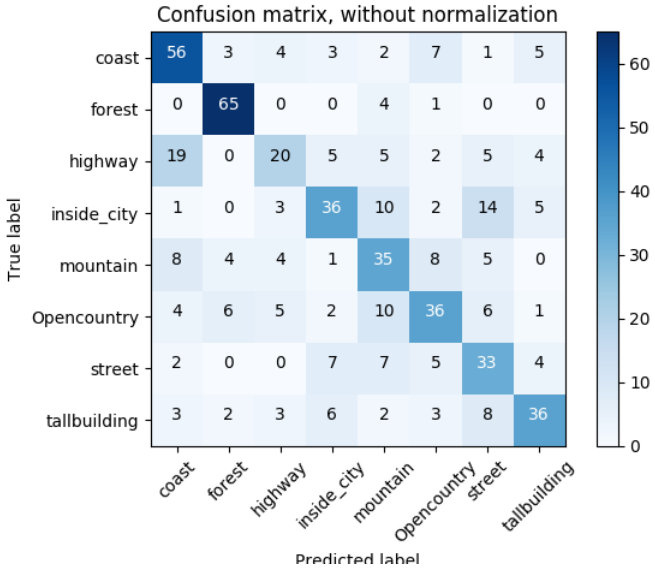

Fig. 10: Confusion Matrix for SVM

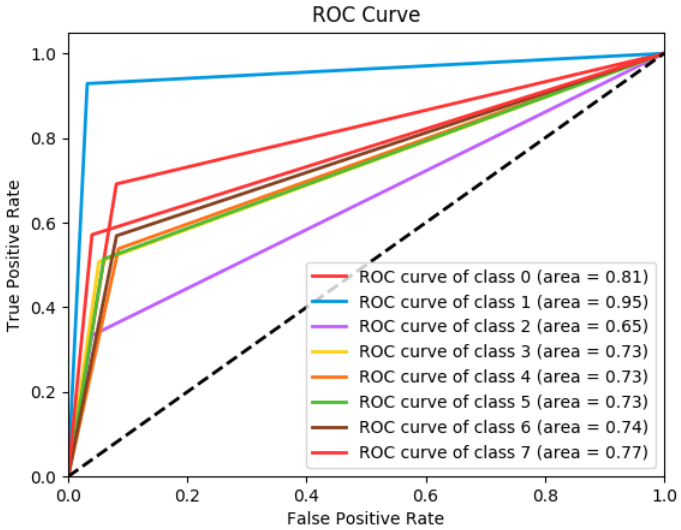

Fig. 11: ROC curve for SVM

B. LMS

The accuracy obtained in LMS is 17.65%

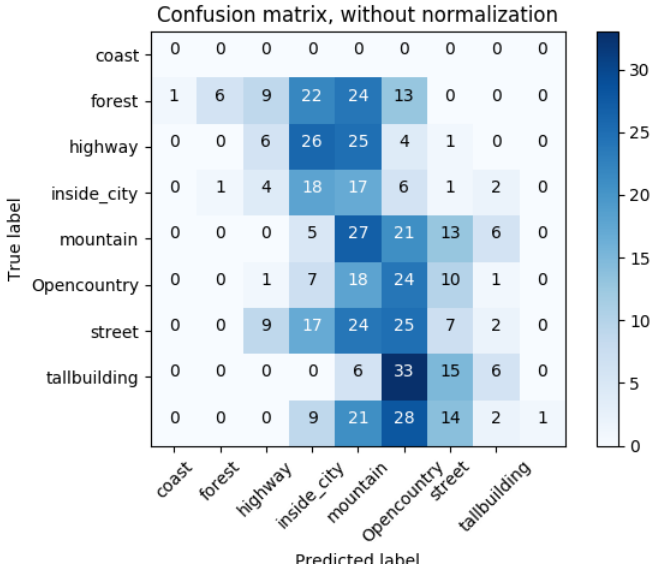

Fig. 12: Confusion Matrix for LMS

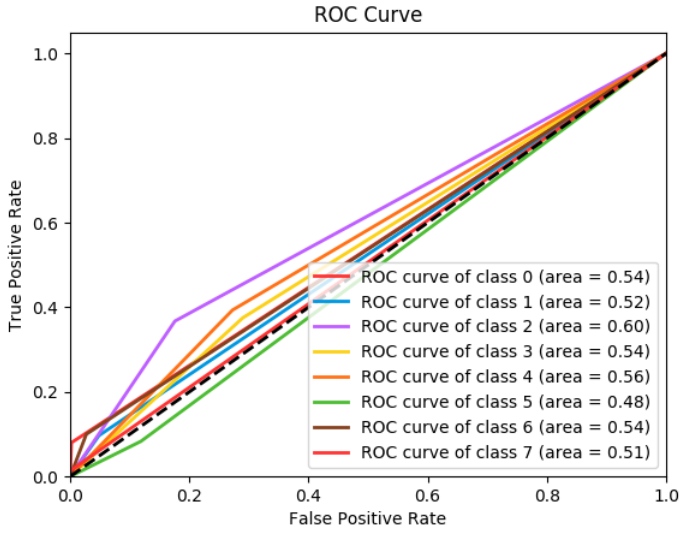

Fig. 13: ROC curve for LMS

C. KLMS

The accuracy obtained in KLMS is 19.34%

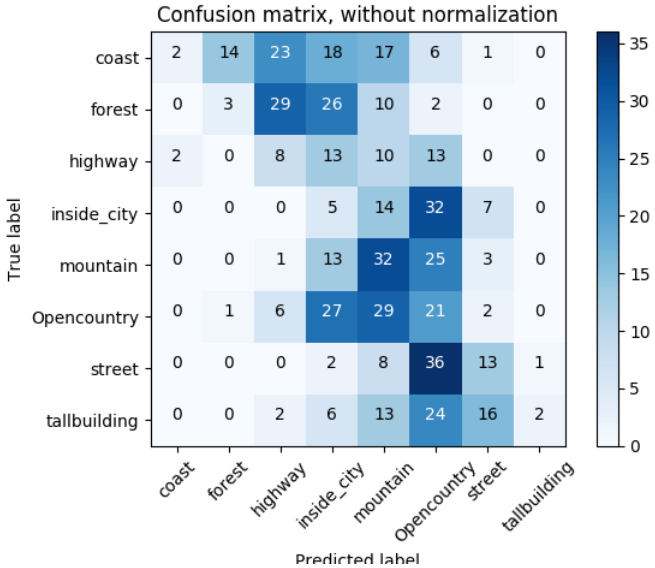

Fig. 14: Confusion Matrix for KLMS

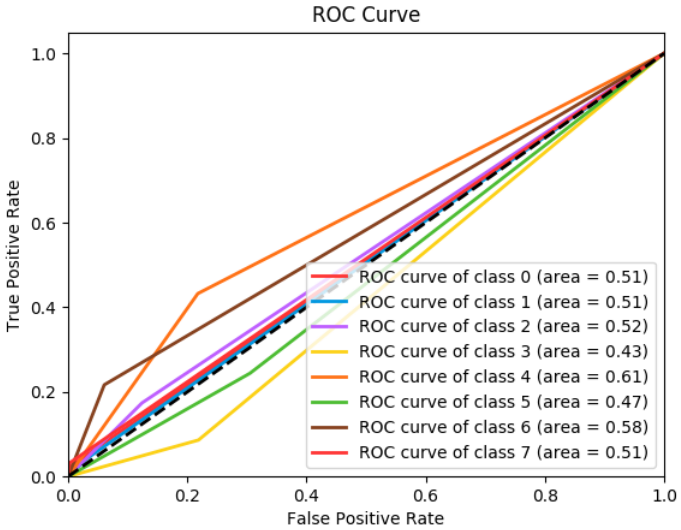

Fig. 15: ROC curve for MCC

D. MCC

Lowest accuracy is observed in MCC algorithm. The accuracy obtained in MCC is 17.47%

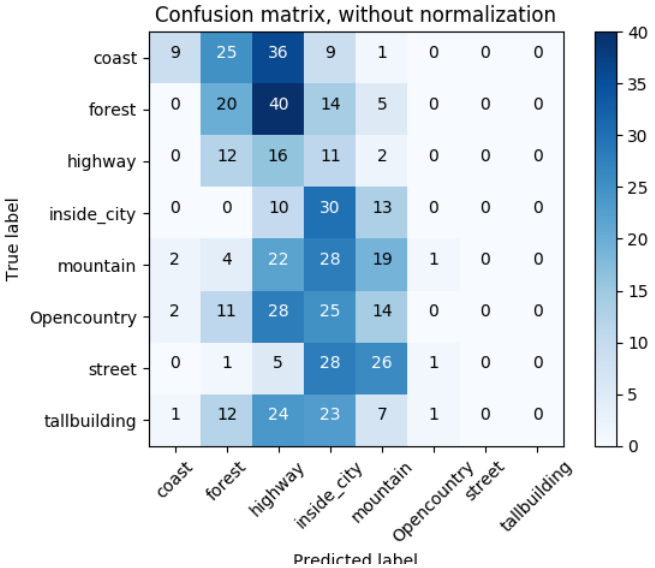

Fig. 16: Confusion Matrix for MCC

E. MEE

The accuracy obtained in MEE algorithm is 18.19%

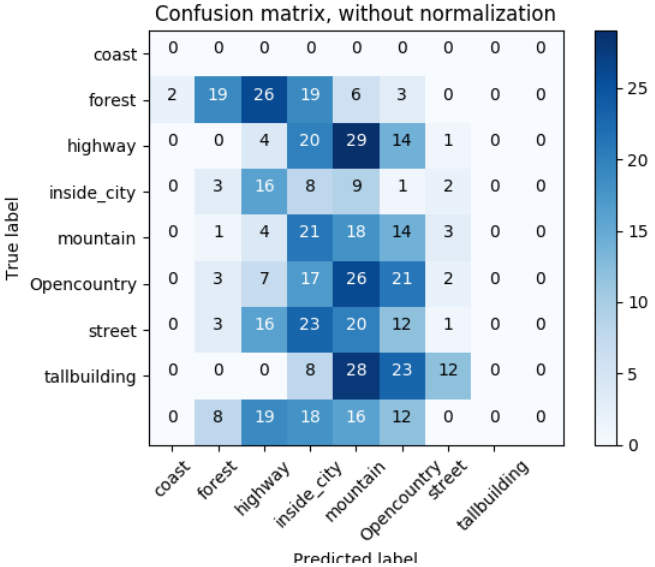

Fig. 17: Confusion Matrix for MEE

F. PNN

The accuracy obtained in PNN algorithm is 58.76%

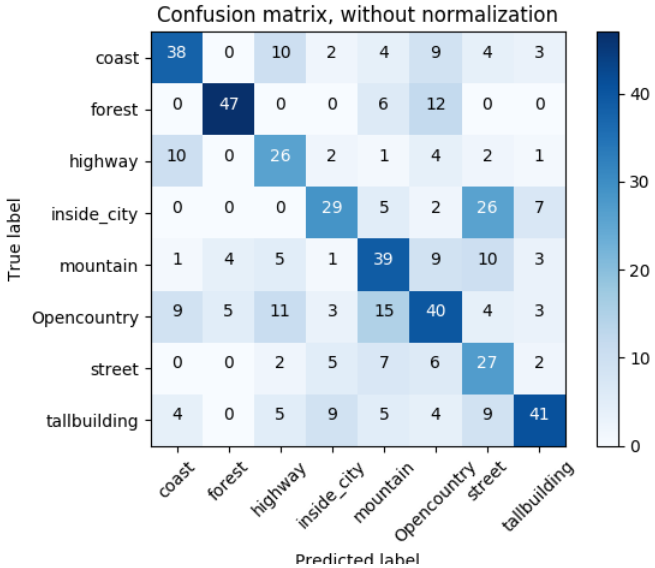

Fig. 18: Confusion Matrix for PNN

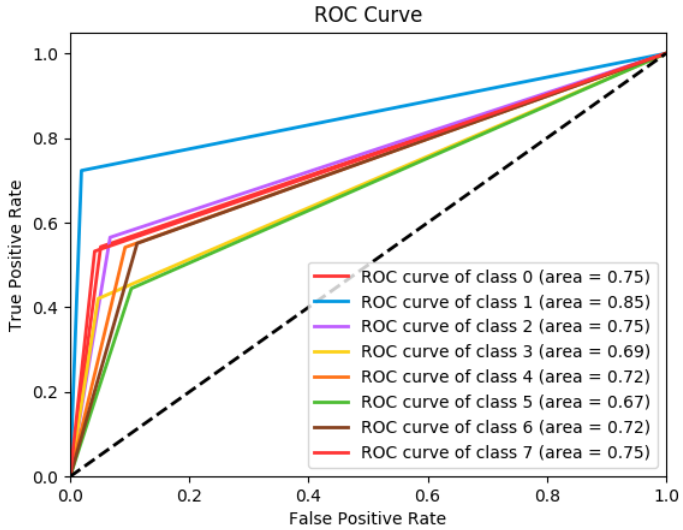

Fig. 19: ROC curve for PNN

The accuracy can be increased by increasing the value of K in multiples of 16. But it's taking more time to extract features for high value of K.

REFERENCES

[1] Liu, Weifeng, Puskal P. Pokharel, and Jose C. Principe. "The kernel least-mean-square algorithm." IEEE Transactions on Signal Processing 56.2 (2008): 543-554.
[2] Aronszajn, Nachman. "Theory of reproducing kernels." Transactions of the American mathematical society 68.3 (1950): 337-404.
[3] Zhao, Songlin, Badong Chen, and Jose C. Principe. "Kernel adaptive filtering with maximum correntropy criterion." Neural Networks (IJCNN), The 2011 International Joint Conference on. IEEE, 2011.
[4] Chen, Badong, et al. "Kernel minimum error entropy algorithm." Neurocomputing 121 (2013): 160-169.
[5] Specht, Donald F. "Probabilistic neural networks." Neural networks 3.1 (1990): 109-118.