Study and Implementation ofClassifiers

**Abstract**

**The goal of this project** is to develop a pattern recognition system that operates on a  
given real-world dataset. In this work we develop various classifiers to work  
on a real-world data set chosen and classify the samples into the corresponding classes. In  
this project I have chosen the “Wireless Indoor Localization” dataset to work on. Through the project I have implemented various classifiers like Navie Bayer, SVM, KNN, SGD, Random Classifier. The dataset was first processed so that the classifiers can be trained. After training each of the classifier, cross validation was performed to validate and evaluate the performance of the classifier. The all classifiers gave the similar classification result on my dataset of choice.

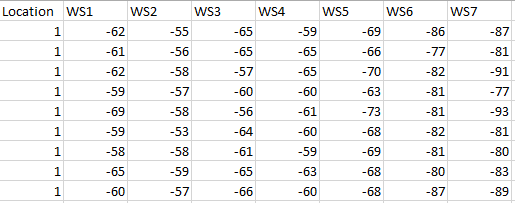
**Introduction**

Pattern recognition is the science of making inferences from perceptual data, using tools  
from statistics, probability, computational geometry, machine learning, signal processing,  
and algorithm design. A Pattern Recognition system is a system that is developed to  
perform pattern recognition automatically. In this project we develop a pattern  
recognition system to efficiently classify a real world dataset.

The project report has been structured in the following manner . At first I have presented  
the details of the dataset. The next section gives the implementation details of each  
classifier with the results obtained. Lastly a comparative study has been performed on the  
classification results obtained by running each classifier on the given real world data set.

**Description of the dataset and Pre-processing**

The initial step in implementing the project was to choose a proper dataset to perform the  
classification . In order to do this I did my initial research on the datasets provided and  
chose the “Wireless Indoor Localization” dataset for my project. The dataset has around 2000 data samples and 7 features. The data was collected from an indoor WiFi system which has 8 Wifi routers at various locations. column 2~8 refer to the measured signal strength (dB, integers) at 7 wireless sensors (WS1-WS7 below) (routers); column 1 is the user location (class). There are 500 data points for each location (class). Thus, there are 7 input variables (features) and 4 classes (user locations).



Also as part of preprocessing , I have tried to implemented the normalization process on the features but there is no better classification results. The normalization was done based  
on the standard scaler(such that its distribution will have a mean value 0 and standard deviation of 1).

**Language and Tool box Used for Implementation:**

The entire project was implemented using Python and certain other pattern  
recognition library. I have used the scikit-learn, panda, matplotlib, tensorflow for implementing the classifiers.

Ski-learn Function Used:

preprocessing.StandardScaler: normalization process

GausianNB: Navie Bayer Model classifier

svm: SVM classsifer

pca: feature-space dimensionality adjustment

cross\_val\_score: cross validation score

The above functions were used to implement the classifiers . The implementation details are given in the sections below.

Another tool box I have used for implementing my project is the tensorflow library. This library was used to train perceptron.

I have used panda library to import and parse csv data.

**Cross Validation:**

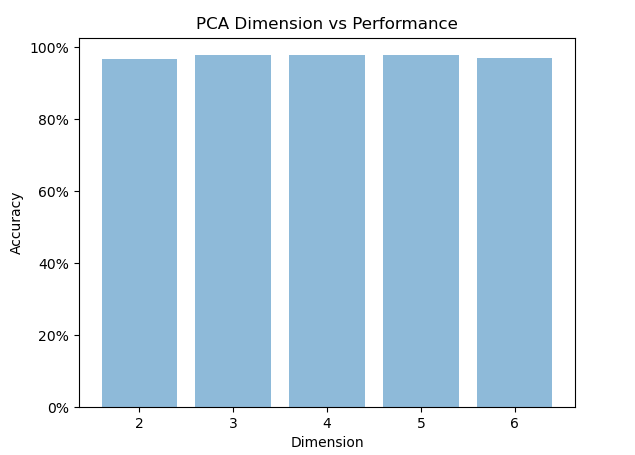
For Each of the classifiers I have implemented cross-validation to test and validate the  
performance of the classifier and in some cases also to determine the best feature  
dimension which is then used used for feature dimension reduction.

I implemented cross-validation in order to evaluate the classifier performance so that we do not corrupt our training and test data . The 1600 samples of training data was divided into 8 sets of samples D1 through D8 . In an iterative process I considered one of these 8 sets to be the validation set and the remaining 1400 samples to be the training set .

During each iteration I changed the validation set so that we can determine the error for each set of samples and evaluate the performance of the classifier. The classifier giving the minimum error was then chosen as the best and its performance was evaluated using the 400 samples of test data.

**Feature Reduction:**

Feature Dimension is an effective way of reducing the complexity of implementation and other curses of high dimensionality. The PCA and FLD are very common and well known methods of implementing feature reduction. In this project I have implemented a types of feature dimension reduction techniques the Principal Component Analysis(PCA).



**Classifer:**

**Random Assignment Classfier:**

The random assignment classifier assigns classes to samples using randomization without  
any criterion or probabilistic models. In this problem I have implemented random class  
assignment with prior information also where the classes assigned to the samples are  
randomized but this randomization is controlled by the class probability .I have  
implemented random class assignment using the DummyClassfier function in Python and  
assigned random numbers between 1-4 (4 class problem) to the 1600 samples . Then I have  
compared the randomly assigned classes to the actual class of the samples and calculated  
the percentage error in classification . I have calculated the probability of each class which gives us the class priors. The probability was calculated as the ratio of frequency of each class to the total number of samples. The class prior information was calculated from the training data . Using this information I then randomly assigned the classes to the test data samples and obtained the following results.

Random Final Test Score = 0.2675

Random Cross Validation Scores = [0.28 0.205 0.28 0.205 0.29 0.18 0.24 0.285]

Random Cross Validation Accuracy: 0.25 (+/- 0.08)

The percentage error varies every time we run this code as we are randomly assigning classes which changes each time giving varying misclassification errors.

Random Confusion Matrix:

[[25 21 28 26]

[18 29 20 33]

[24 29 26 21]

[25 24 23 28]]

**Baseline Classfier:**

In this project, I have implemented a baseline classifiers, Navie Bayer Classifier.

**Navie Bayer Classfier:**

The Bayes Classifier is a parametric type of statistical classifier . It assigns classes to  
achieve minimum error in classification. The decision rule for a Bayes classifier is given as follows :

I trained the classifier using the training dataset based on sckit-learn’s GaussianNB . The implementation details of cross-validation was given in the previous section.

Navie Bayer Final Test Score = 0.9775

Navie Bayer Cross Validation Scores = [0.98 0.98 0.985 0.995 0.99 0.99 0.985 0.98 ]

Navie Bayer Cross Validation Accuracy: 0.99 (+/- 0.01)

Navie Bayer Confusion Matrix:

[[ 99 0 1 0]

[ 0 94 6 0]

[ 1 1 98 0]

[ 0 0 0 100]]

**K-nearest neighbor(KNN) density based classfier:**

KNN classifier is a non-parametric statistical classifier that counts the k-nearest features to  
determine the class of a sample

KNN Final Test Score = 0.9825

KNN Cross Validation Scores = [0.975 0.98 0.99 0.995 0.99 0.99 0.97 0.985]

KNN Cross Validation Accuracy: 0.98 (+/- 0.02)

KNN Confusion Matrix:

[[ 99 0 1 0]

[ 0 96 4 0]

[ 1 1 98 0]

[ 0 0 0 100]]

**Stochastic Gradient Descent(SGD):**

SGD is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) [Support Vector Machines](https://en.wikipedia.org/wiki/Support_vector_machine) and [Logistic Regression](https://en.wikipedia.org/wiki/Logistic_regression). Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.

Gradient Descent Final Test Score = 0.9425

Gradient Descent Cross Validation Scores = [0.915 0.975 0.91 0.955 0.94 0.945 0.91 0.935]

Gradient Descent Cross Validation Accuracy: 0.94 (+/- 0.04)

**Support Vector Machine:**

The support vector machine is a useful classification technique which classifies data by  
creating a hyperplane in a high or infinite-dimensional space, which can be used for  
classification, regression, or other tasks. Each instance in the training set contains one  
“target value” (i.e. the class labels) and several “attributes” (i.e. the features or observed  
variables). The goal of SVM is to produce a model (based on the training data) which  
predicts the target values of the test data given only the test data attributes.

The SVM classifier is implemented using scikit-learn library package. Once we have  
these datasets we then train the classifier using the svm.SVC().fit in scikit-learn.

SVM Final Test Score = 0.975

SVM Cross Validation Scores = [0.975 0.975 0.985 0.995 0.975 0.985 0.98 0.97 ]

SVM Cross Validation Accuracy: 0.98 (+/- 0.02)

SVM Confusion Matrix:

[[ 98 0 1 1]

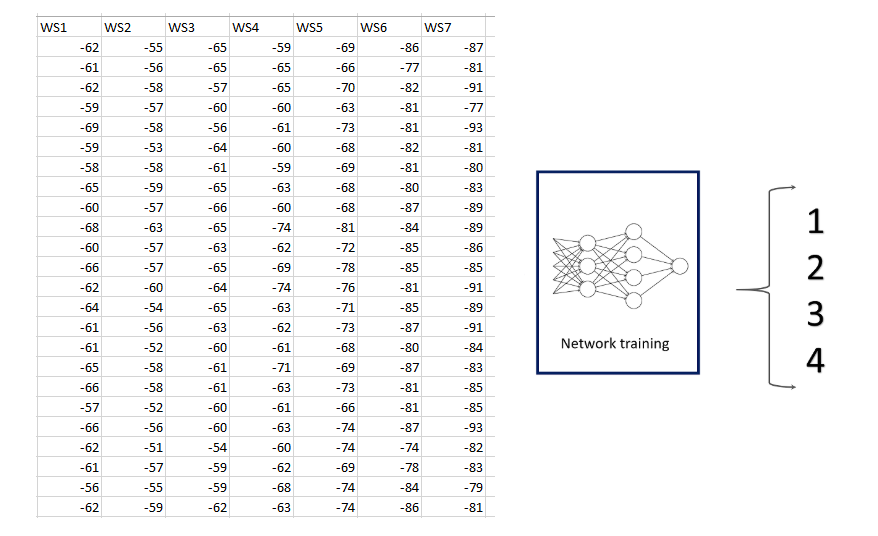
[ 0 94 6 0]

[ 0 2 98 0]

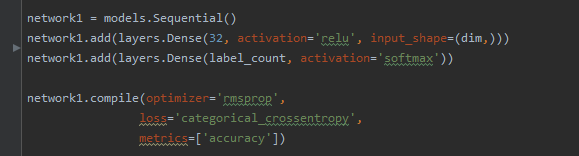
[ 0 0 0 100]]

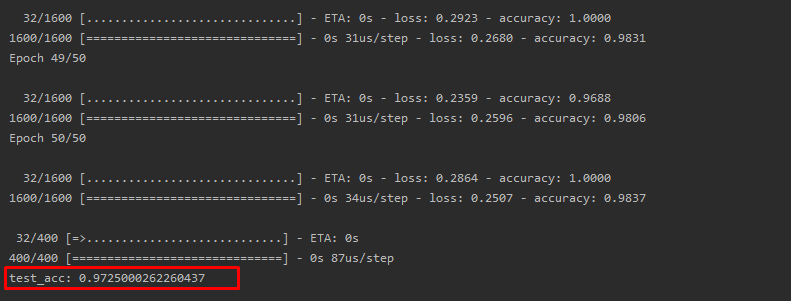
**Neural Network Classfier:**

The main structural feature of RegularNets is that all the neurons are connected to each other.



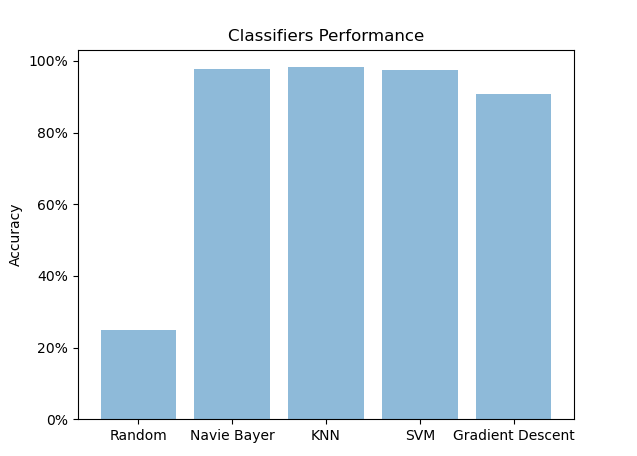
In training, 1st layer has 7 neutron, 2nd layer has 32 nenutron, 3rd layer has 4 neutron with softmax. And optimizer is based on RMSPro optimizer, and loss function is based on categorial cross entropy funciton,





**Conclusion:**

This section of my report summarizes my learning and results obtained through the  
course of this project .Analyzing the error rate of classification of each of the classifiers ,  
we can observe that the all of classfiers gives us the similiar classification accuracy  
for the test data of a given dataset , 97.5 %.



Among the baseline classifiers I select the Navie Bayer Classifier. The performance of  
each classifier is based on their test data classification. We can see that having random class assignment with prior information performs only about 25% accuracy. Comparing the feature dimension reduction results , we can see that the Principal Component Analysis gives no better classification rate than normal classification.  
For all the classifiers the error classification rate for each dimension was plotted and  
shown. These plots were generated for each iteration.

**Apppendix:**

**PYTHON CODE:**

================== Main.py ==========================================

from sklearn.dummy import DummyClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn import svm  
from sklearn.linear\_model import SGDClassifier  
from sklearn import preprocessing  
from sklearn.decomposition import PCA  
  
import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.ticker as mtick  
  
# Load the Pandas libraries with alias 'pd'  
import pandas as pd  
from util import \*  
  
# Load Train / Test Data  
df = pd.read\_csv(r'D\_Train1.csv')  
train\_data = df.to\_numpy()  
  
X\_Train = train\_data[:, 1:]  
Y\_Train = train\_data[:, [0]]  
Y\_Train = Y\_Train.ravel()  
  
f\_dim = len(X\_Train[0])  
  
scaler = preprocessing.StandardScaler().fit(X\_Train)  
  
  
df = pd.read\_csv(r'D\_Test1.csv')  
  
test\_data = df.to\_numpy()  
  
X\_Test = test\_data[:, 1:]  
Y\_Test = test\_data[:, [0]]  
Y\_Test = Y\_Test.ravel()  
names = ["Random", "Navie Bayer", "KNN", "SVM", "Gradient Descent"]  
  
classifiers = [  
 DummyClassifier(strategy="stratified"),  
 GaussianNB(),  
 KNeighborsClassifier(n\_neighbors=4),  
 SGDClassifier(loss="hinge", penalty="l2", max\_iter=1000),  
 svm.SVC(decision\_function\_shape='ovr',kernel='linear', C=1),  
]  
  
# iterate over classifiers with standard setting  
score\_list = []  
for name, clf in zip(names, classifiers):  
 score = train\_evaluate\_classfier(name, clf, X\_Train, Y\_Train, X\_Test, Y\_Test)  
 score\_list.append(score \* 100)  
  
x\_pos = np.arange(len(names));  
plt.bar(x\_pos, score\_list, align='center', alpha=0.5)  
plt.gca().yaxis.set\_major\_formatter(mtick.PercentFormatter())  
plt.xticks(x\_pos, names)  
plt.ylabel('Accuracy')  
plt.title('Classifiers Performance')  
plt.show()  
  
# Feature Reduction analysis for SVM classifier  
print("================ Feature Reduction ================")  
score\_list = []  
for pca\_dim in range(2, f\_dim):  
 pca = PCA(n\_components=pca\_dim)  
 pca.fit(X\_Train)  
  
 print("PCA Dimension = ", pca\_dim, pca.explained\_variance\_ratio\_)  
 X\_Train\_Transform = pca.transform(X\_Train)  
 X\_Test\_Transform = pca.transform(X\_Test)  
  
 # svm  
 clf = classifiers[3]  
 score = train\_evaluate\_classfier("SVM: PCA Dim = " + str(pca\_dim), clf, X\_Train\_Transform, Y\_Train, X\_Test\_Transform, Y\_Test)  
 score\_list.append(score \* 100)  
  
x\_pos = np.arange(len(score\_list))  
plt.bar(x\_pos, score\_list, align='center', alpha=0.5)  
plt.gca().yaxis.set\_major\_formatter(mtick.PercentFormatter())  
plt.xlabel('Dimension')  
plt.ylabel('Accuracy')  
plt.xticks(x\_pos, range(2, f\_dim))  
plt.title('PCA Dimension vs Performance')  
# plt.autoscale(axis='y',tight=True)  
plt.show()

================== tensorflow\_test.py ==========================================

from keras.utils import to\_categorical  
  
from keras import models  
from keras import layers  
  
import pandas as pd  
  
df = pd.read\_csv(r'D\_Train1.csv')  
train\_data = df.to\_numpy()  
  
X\_Train = train\_data[:, 1:]  
Y\_Train = train\_data[:, [0]]  
Y\_Train = Y\_Train.ravel()  
Y\_Train = Y\_Train - 1  
  
df = pd.read\_csv(r'D\_Test1.csv')  
  
test\_data = df.to\_numpy()  
  
X\_Test = test\_data[:, 1:]  
Y\_Test = test\_data[:, [0]]  
Y\_Test = Y\_Test.ravel()  
Y\_Test = Y\_Test - 1  
  
X\_Train = X\_Train.astype('float32') / 100  
X\_Test = X\_Test.astype('float32') / 100  
train\_labels = to\_categorical(Y\_Train)  
test\_labels = to\_categorical(Y\_Test)  
  
dim = len(X\_Train[0])  
label\_count = len(train\_labels[0])  
  
network1 = models.Sequential()  
network1.add(layers.Dense(32, activation='relu', input\_shape=(dim,)))  
network1.add(layers.Dense(label\_count, activation='softmax'))  
  
network1.compile(optimizer='rmsprop',  
 loss='categorical\_crossentropy',  
 metrics=['accuracy'])  
  
  
  
network1.fit(X\_Train, train\_labels, epochs=50, batch\_size=32)  
test\_loss, test\_acc = network1.evaluate(X\_Test, test\_labels)  
print('test\_acc:', test\_acc)