# Pattern Recognition Final Project

Data Set(s): Indoor Wireless Localization

Jiangrun Chen, jiangrun@usc.edu

May 8, 2020

# 1 Abstract

As we all know, GPS is the most common outdoor localization technology in our daily life. However, when we are indoors, we may find that the GPS signal we receive is very weak. That's why the indoor wireless localization technology is popular today. For example, in an emergency, everyone wants to be accurately located by the rescuers, on which floor or in which room. And in business, customers in the mall can be directed to the goods which they want to purchase using indoor wireless localization technology. The dataset I use contains the WiFi signal strength received from 7 routers, from which we can predict the location of a user (who is in one of 4 rooms). Since we already have training dataset and test dataset, I don't need to separate them. I preprocess the datasets and use PCA method to try to reduce some features. Then I use Naïve Bayes, SVM, k Nearest Neighbor, Artificial Neural Network and cross validation method to train my classifiers and finally use these models to test on the test dataset. I compare the performance of different models with different parameters to get the best testing result. The best accuracy on test dataset =0.9875. It comes from KNN model with optimal K=3.

# 2 Introduction

## 2.1 Problem Assessment and Goals

The Indoor Wireless Localization dataset contains the WiFi signal strength received from 7 routers, which is classified into 4 classes. My goal is to apply different models with different parameters to train dataset and test on test dataset to try to find the best model with highest accuracy.

# 3 Approach and Implementation

## 3.1 Preprocessing

The datasets are in csv file format, so I use csv.reader function to read the data into strings and then transfer them into float format. By observing the dataset, we can find the first column is the data label, the rest columns are features. I separate the labels and features for further use. Since all the feature data are negative numbers, I use StandardScaler function to normalize the training dataset according to its mean and deviation, then use this scaler to normalize the test dataset. Actually, in Pattern Recognition, I think normalization is to make the features between different dimensions have a certain comparison in value, which can greatly improve the accuracy of the classifier. The models and techniques used in my project are more likely to beneficial from the normalized data, like SVM, PCA and Artificial Neural Network.

## 3.2 Feature engineering

Not applicable.

## 3.3 Feature dimensionality adjustment

For this project, I first normalize the train dataset and test dataset. I find there are 7 features in datasets, so I try to use PCA to reduce dimensions to 5 which does not make datasets lose much information. PCA is a method that projects the original high-dimensional data onto a low-dimensional space and makes its variance as large as possible. Then I use several models to classifier the data and compare their performance. To choose 5 features from 7, I compare different parameters to see their performance. For example, in linear perceptron model, when all features and 5 features are used, the accuracy on train dataset are 0.98375 and 0.985, which means feature reduction kindly improve some performance. When I choose 4 features, the performance goes down. That's why I finally choose 5 as my PCA parameter.

## 3.4 Dataset Usage

I only use training dataset to calculate the normalizing factor and construct classifiers. Because we should do nothing to test dataset before we use the models to finally test the unknown data. Then I use the same normalizing factor to normalize the test dataset. By using cross validation to get the best parameters, I separate train dataset into 5 folders and run them for 50 times. For example, I use cross validation to choose C and gamma in SVC, choose k in kNN and choose best layer in Artificial Neural Network. After comparing the results, I choose the best parameters which give best performance to finally applied on test dataset. For feature reduction, I use PCA which can project the original high-dimensional data onto a low-dimensional space.

## 3.5 Training and Classification

### 3.5.1 Perceptron

For Perceptron classifier, I use model Perceptron from sklearn.linear_model. Perceptron aims to minimize the criterion function, i.e., minimize the number of misclassified data points. I first use default initial weight vector which is all zero. Among the parameters of perceptron classifier, "tol" is related to the stopping criterion in the perceptron algorithm. The parameter "max_iter" determines the backup halting condition. It represents the maximum number of passes over the training data (aka epochs).

Then I run the perceptron for 100 times. Every time the starting weight vectors are randomly chosen. I pick the run that has the best performance on the training set and use that best initial weight vectors to test on test dataset.

3.5.2 Naïve Bayes

For Naïve Bayes, I use model GaussianNB() from sklearn.naive_bayes. PCA is applied to compare performance between all features and 5 features. The prior probability which corresponds to the prior probability P of each category is the only one parameter of the GaussianNB model. This value is not given by default, if it is not given then P (Y = Ck) = Mk / m. Where m is the total number of samples in the training set, and Mk is the number of samples in the training set whose output is the kth category.

3.5.3 SVM

3.5.3.1 SVM with linear kernel

C and gamma are the parameters of the SVM model. C is the cost of misclassification as correctly. Gamma is the parameter of a Gaussian Kernel (to handle non-linear classification). For linear kernel, gamma is set to default. Because in project description it says the dataset is not very likely linear, I don't dig much into C's choice. I choose C=1 here. PCA is applied to see the difference between all features and 5 features.

3.5.3.2 SVM with RBF kernel

C and gamma are the parameters of the SVM model. To choose them, I use cross validation to more optimally design the SVM classifier. I use 5-fold cross-validation on the training dataset and run it for 50 times. In each run the accuracy average is calculated and saved. Finally, I use the best model to test on test dataset. PCA is applied to see the difference between all features and 5 features.

3.5.4 5NN

For kNN model, I choose k=5 which is the default value at first. Other parameters are all set to default here. I think it's a proper parameter to improve the performance of the model. I use KNeighborsClassifier model from sklearn.neighbors. PCA is applied to see the difference between all features and 5 features.

3.5.5 kNN (optimize k)

We know that the performance of the model varies due to the change of parameter k in kNN model. To choose k, I use cross validation to more optimally design the kNN classifier. I use 5-fold cross-validation on the training dataset and run it for 10 times. In each run the accuracy average is calculated and saved. Finally, I use the best model to test on test dataset. PCA is applied to see the difference between all features and 5 features.

3.5.6 Artificial Neural Network

Artificial Neural Network can be regarded as a multi-layer perceptron. For MLP Classifier, there are several parameters like activation, solver and hidden layers. I first set 5 different layers as [(100), (10,10), (10, 10, 10), (10, 10, 10, 10), (10, 10, 10, 10, 10)]. We know that the performance of the model varies due to the selection of the layer. To

choose the best layer, I use cross validation to more optimally design the MLP classifier. I use 5-fold cross-validation on the training dataset and run it at each layer for 1 time. Finally, I use the best model to test on test dataset. For parameter activation I set it to default. PCA is applied to see the difference between all features and 5 features. In MLP classifier, I compare the performance when parameter slover is equal to 'lbfgs' or 'adam'. The default solver 'adam' works well on training time and verification scores on relatively large data sets (containing thousands of training samples or more). However, for small data sets, 'lbfgs' can converge faster and perform better. Although our dataset contains more than thousands of training samples, it shows maximum iterations (200) reached and the optimization hasn't converged yet if I use 'adam'. That's why I finally choose solver to be 'lbfgs' which also gives better performance.

# 4 Analysis: Comparison of Results, Interpretation

3.5.1 Perceptron

(1) Final Weights=

```
[[-10.27074078    1.68105673   -2.35680043  -11.71585126   -8.97398297
    1.54914589    0.96920402]
 [ 10.55273452   -1.13861529   -1.62929934    4.3466213    -3.82701414
    0.46975837    1.23976869]
 [  2.20527918    3.86928158    2.48562875    1.08689955   -2.75773785
   -3.22519379   -1.39095532]
 [ -5.13316731    0.03106976    2.91758253   -5.03153029    8.96185559
   -1.37469166   -2.89918087]]
```

(2) train accuracy = 0.95125

(3) test accuracy = 0.93

Interpretation: Perceptron is a linear classifier, which aims to minimize the criterion function, like minimize the number of misclassified data points. As we learn from the data description that the dataset is not very likely linear, the result tells us that the performance is good enough for linear classifier. I use it to briefly look at the feature characters first, and I believe other non-linear classifier will bring me better results. The best accuracy on test set=0.93.

3.5.2 Naïve Bayes

(1) Gaussian: accuracy on test data (all features)= 0.9775

confusion matrix :

```
 [[ 99   0   1   0]
 [  0  94   6   0]
 [  1   1  98   0]
 [  0   0   0 100]]
```

(2) Gaussian: accuracy on test data with PCA (5 features) = 0.9425
confusion matrix :
[[ 92 0 8 0]
[ 0 94 6 0]
[ 2 5 91 2]
[ 0 0 0 100]]

(3) Bernoulli: accuracy on test data= 0.9025
confusion matrix :
[[ 95 0 2 3]
[ 0 87 13 0]
[ 1 17 79 3]
[ 0 0 0 100]]

(4) Bernoulli: accuracy on test data with PCA = 0.735
confusion matrix :
[[89 6 2 3]
[ 0 82 18 0]
[ 9 41 33 17]
[ 6 0 4 90]]

Interpretation: We can find the performance for Naïve Classifier is not very good. That is because the Naive Bayes model assumes that the features are independent of each other. This is not always true in daily life. When the number of features is relatively large or the correlation between attributes is large, the classification effect is not good. When the feature correlation is small, Naive Bayes has better performance. And we can find the performance of GaussianNB model is better than BernoulliNB. Also, with feature reduction, the model performance goes down for less input information, especially for BernoulliNB model. The best accuracy on test set=0.9775.

3.5.3 SVM
3.5.3.1 SVM with linear kernel
(1) With C=1, accuracy on train dataset= 0.98375
Accuracy on testset = 0.9775
confusion matrix :
[[ 99 0 1 0]
[ 0 93 7 0]
[ 0 1 99 0]
[ 0 0 0 100]]

(2) With C=1, accuracy on train dataset with PCA= 0.985
Accuracy on testset with PCA= 0.9725
confusion matrix :
[[ 99 0 1 0]
[ 0 94 6 0]
[ 0 3 96 1]
[ 0 0 0 100]]

3.5.3.2 SVM with rbf kernel
(1) Using SVM rbf kernel with cross validation:
For the best pair, gamma = 0.022229964825261943
C = 10.985411419875572
with mean cross-validation accuracy = 0.9868750000000001
and standard deviation = 0.0012500000000000178
Accuracy on testset = 0.9775
confusion matrix :
[[ 99 0 1 0]
[ 0 95 5 0]
[ 0 3 97 0]
[ 0 0 0 100]]

(2) Using SVM rbf kernel with cross validation&PCA:
For the best pair, gamma = 0.0030888435964774815
C = 568.9866029018293
with mean cross-validation accuracy = 0.985625
and standard deviation = 0.010752906583803288
Accuracy on testset = 0.97
confusion matrix :
[[ 99 0 1 0]
[ 0 94 6 0]
[ 0 4 95 1]
[ 0 0 0 100]]

Interpretation: As we state, the dataset is kindly non-linear. When we compare the results between SVM linear kernel and rbf kernel, we can find non-linear classifier behaves better. And when PCA is applied, the performance goes down very little which can ignore. Using PCA here is a good choice. The best accuracy on test set=0.9775.

3.5.4 5NN

(1) train dataset accuracy for 5NN: 0.9841249999999999
test dataset accuracy for 5NN: 0.98
confusion matrix :
[[ 99 0 0 1]
[ 0 95 5 0]
[ 1 1 98 0]
[ 0 0 0 100]]
(2) train dataset accuracy for 5NN with PCA: 0.9814375
test dataset accuracy for 5NN with PCA: 0.965
confusion matrix :
[[99 0 1 0]
[ 0 93 7 0]
[ 1 2 95 2]
[ 0 0 1 99]]

Interpretation: It's very important to choose parameter K in KNN model. If the value of K is too small, once there are noise components, they will have a greater impact on the prediction. For example, K equals 1, once the nearest point is noise, there will be a large deviation. Smaller K means that the model becomes complex and more likely to overfitting. If the value of K is too large, the approximate error of learning will increase. At this time, the data point far away from the input target will also contribute to the prediction, making the prediction not correct. After having try, I choose k=5. When PCA is applied here, the accuracy on test dataset goes down. 'All features' perform better. The best accuracy on test set=0.98.

3.5.5 kNN (optimize k)

(1) Best accuracy on train dataset: 0.9857500000000001 with k = 3
accuracy on test dataset: 0.9875
confusion matrix :
[[100 0 0 0]
[ 0 97 3 0]
[ 0 1 99 0]
[ 0 0 1 99]]
(2) With PCA, best accuracy on train dataset: 0.9817343749999999 with k = 6
accuracy on test dataset: 0.965
confusion matrix :
[[99 0 1 0]
[ 0 93 7 0]
[ 1 2 95 2]
[ 0 0 1 99]]

Interpretation: This time I choose k from 3 to 11 and apply it to train dataset using cross validation. K varies if different number of features is chosen. When PCA is applied here, the accuracy on test dataset goes down. 'All features' perform better. The best accuracy on test set=0.9875 with best k=3.

3.5.6 Artificial Neural Network(MLP Classififier)

Layers = [(100),(10,10),(10, 10, 10), (10, 10, 10, 10),(10, 10, 10, 10, 10)]

(1) hidden layer = 100 accuracy on train dataset(avr) = 0.9774999999999998

Best accuracy = 0.9774999999999998 with hidden layer = 100

accuracy on test dataset: 0.9775

confusion matrix :

```
[[ 99   0   1   0]
 [  0  95   5   0]
 [  1   2  97   0]
 [  0   0   0 100]]
```

(2) hidden layer = (10, 10) accuracy on train dataset(avr) = 0.97

Best accuracy = 0.9774999999999998 with hidden layer = 100

accuracy on test dataset: 0.98

confusion matrix :

```
[[ 99   0   1   0]
 [  0  96   4   0]
 [  1   2  97   0]
 [  0   0   0 100]]
```

(3) hidden layer = (10, 10, 10) accuracy on train dataset(avr) = 0.9799999999999999

Best accuracy = 0.9799999999999999 with hidden layer = (10, 10, 10)

accuracy on test dataset: 0.965

confusion matrix :

```
[[ 97   0   1   2]
 [  0  94   6   0]
 [  0   4  95   1]
 [  0   0   0 100]]
```

(4) hidden layer = (10, 10, 10, 10) accuracy on train dataset(avr) = 0.976250000 0000001
Best accuracy = 0.9799999999999999 with hidden layer = (10, 10, 10)
accuracy on test dataset: 0.9825
confusion matrix :
[[ 98 0 1 1]
[ 0 96 4 0]
[ 0 1 99 0]
[ 0 0 0 100]]

(5) hidden layer = (10, 10, 10, 10, 10) accuracy on train dataset(avr) = 0.98062 50000000001
Best accuracy = 0.9806250000000001 with hidden layer = (10, 10, 10, 10, 10)
accuracy on test dataset: 0.9775
confusion matrix :
[[ 99 0 1 0]
[ 0 95 5 0]
[ 0 3 97 0]
[ 0 0 0 100]]

(6) With PCA, hidden layer = 100 accuracy on train dataset(avr) = 0.973749999 9999999
Best accuracy = 0.9737499999999999 with hidden layer = 100
accuracy on test dataset: 0.9525
confusion matrix :
[[99 0 0 1]
[ 0 91 9 0]
[ 0 3 95 2]
[ 0 0 4 96]]

(7) With PCA, hidden layer = (10, 10) accuracy on train dataset(avr) = 0.975
Best accuracy = 0.975 with hidden layer = (10, 10)
accuracy on test dataset: 0.965
confusion matrix :
[[ 97 0 2 1]
[ 0 93 7 0]
[ 0 4 96 0]
[ 0 0 0 100]]

(8) With PCA, hidden layer = (10, 10, 10) accuracy on train dataset(avr) = 0.97 37499999999999
Best accuracy = 0.975 with hidden layer = (10, 10)
accuracy on test dataset: 0.9675
confusion matrix :
[[99 0 1 0]
[ 0 93 7 0]
[ 1 2 97 0]
[ 0 0 2 98]]

(9) With PCA, hidden layer = (10, 10, 10, 10) accuracy on train dataset(avr) = 0.971875
Best accuracy = 0.975 with hidden layer = (10, 10)
accuracy on test dataset: 0.95
confusion matrix :
[[98 0 2 0]
[ 0 92 8 0]
[ 0 6 92 2]
[ 0 0 2 98]]

(10) With PCA, hidden layer = (10, 10, 10, 10, 10) accuracy on train dataset(avr) = 0.971875
Best accuracy = 0.975 with hidden layer = (10, 10)
accuracy on test dataset: 0.9475
confusion matrix :
[[98 0 2 0]
[ 0 90 10 0]
[ 1 6 92 1]
[ 0 0 1 99]]

Interpretation: We can find that accuracy at different layer is different. It's hard to tell which layer will always have the best performance. We can just compare the results and say which one is the best for this case. And I feel a little difficult to set the layers to be most reasonable. From the results, we can see that when PCA is applied here, the accuracy on test dataset does not only go down but also goes up sometimes which implies it's a complex model. The best accuracy on test set=0.9825 at hidden layer = (10, 10, 10, 10) with all features.

# 5 Summary and Conclusion

Among all models, the best accuracy on test dataset =0.9875. It comes from KNN model with optimal K=3. Since the dataset is kind of linear but not very, we can see that performance of linear classifier like Perceptron is good enough, but performance of non-linear classifier is better. Also, from the confusion matrix we can see the elements in the diagonal are large which implies the dataset accuracy is high. The accuracy may drop a little using PCA since the input information decreases, but it can save computing time. From this project, I learn that the datasets have several characters. When I compare the performance of each classifier, I can clearly see them. All classifiers can show us the characters of the dataset in some way. What's more, I learn that it's important to choose proper parameter for the classifier sometimes due to algorithm and sometimes due to the result. The performance of the classifier varies a lot from it.

# 6 References

[1] cite from 'https://fishc.com.cn/thread-106393-1-1.html'
[2] 'sklearn.decomposition.PCA', cite from ' https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html'
[3] cite from 'https://blog.csdn.net/demm868/article/details/103052766'
[4] cite from 'https://blog.csdn.net/shuibuzhaodeshiren/article/details/96436731'
[5] cite from 'https://blog.csdn.net/weixin_41990278/article/details/93169529'
[6] cite from 'https://github.com/liu578/news_popularity_classifier/blob/master/code.py'
[7] cite from 'https://blog.csdn.net/weixin_38278334/article/details/83023958'