Predict Indoor Location By Using Wifi Signal Strength (assignment for Spring 2020)

Data Set : Indoor Wireless Localization

Yijing Jiang, yijingji@usc.edu

May 8, 2020

1 Abstract

This project is to locate the user by using wifi signal strength information. I use *Indoor Wireless Localization* dataset with 4 classes and 7 features to solve this problem.

GNB (Gussian Naive Bayes) is chosed as baseline model, and will be compared with linear version and rbf version of SVM (Support Vector Machine), MSE(Mean Square Error), Perceptron Learning and KNN (K Nearest Neighbors) algorithm.

For feature preprocessing, consider normalization method. For feature adjustment, consider feature combination, PCA (Principal Components Analysis) and FLD (Fisher's Linear Discriminant) methods.

For each comparison of methods, models or parameters selection, use cross validation method. Final best accuracy on test set is 0.9825, using KNN considering 4 nearest beighbors.

1.1 Problem Assessment and Goals

I choose the *Indoor Wireless Localization* dataset with 7 features, 4 classes. It has 2000 labeled datas(1600 training data, 400 test data). Here, features are Wifi signal strength from 7 routers to the user, classes are 4 different rooms where the user is in. As we know, Wifi signal strength will be partially related to the distance between user and router. Also it will be effected by transport medium, send device, receive device and etc. Thus, although the phsical room space must be clearly linearly seperable, this dataset may be nonlinearly seperable. According to that, Our goal is to find a classifier to locate the room by using these Wifi signal strength data as well as possible.

2 Approach and Implementation

Use different methods in preprocessing, feature adjustment and classification to find the better choice.

- For preprocessing, compare the results using data in nonnormalized and normalized form.
- For feature adjustment, compare the results using dataset processed by PCA or LDA methods.
- For classification, optimize these classifiers: GNB(baseline), SVM with linear kernel, SVM with rbf kernel, MSE, Perceptron and KNN.

2.1 Preprocessing

For data preprocessing, we can use data normalization, adjust imbalanced data, and dealing with missing data. Here, We don't have problem with imbalanced data and missing data, so we only use data normalization.

Data normalization will adjust a well spread data into a new distribution with mean = 0 and standard variance = 1. It can solve the scale mismatch problem between features.

I use *sklearn.preprocessing.StandardScaler* in code for this processing. Nonnormalized data is just the original data.

For each model in basic version, we check the result of cross validation on original data and normalized data:

accuracy mean std	**original data**	**normalize data**	**choice of normalize or not**
GNB	**0.9846** **0.0061**	0.9845 0.0049	false
SVM linear (C = 1.0)	0.9775 0.0071	**0.98375** **0.0050**	true
SVM rbf (C = 1.0, gm = 0.1)	0.8451 0.0215	**0.9831** **0.0061**	true
MSE	0.775 0.0178	**0.77525** **0.0156**	true
Perceptron (SGD, OvR)	0.88887 0.07397	**0.94325** **0.0198**	true
KNN (K = 4)	**0.984375** **0.0050**	0.98425 0.0056	false

- We can see that for different models, we may have different choice.
  (1) For supervised learning methods such as SVM, MSE and Perceotron, normalized data have slightly better results than original data.
  (2) For statistical classification such as GNB and KNN, original data is slightly better than normalized dataset.
  (3) And in almost all the models, normalized data has smaller standard variance than original data.

- The reasons may be:
  (1) For supervised learning, we are going to minimize the criterien function, which always directly related with features value, thus a standardized features vector may help to update the criterien function.
  (2) For statistical classification, data normalization may change its original statistical distribution to a closer region, thus leading to more misclassified datas.
  (3) Also, the closer and centered dataset will cause a smaller variance when choosing different folds in cross validation.

2.2 Feature adjustment

For feature adjustment, we tend to find the best from these three methods: (1) Find best choice of features combination. (2) Transform to a new feature space by using PCA(Principal Components Analysis). (3) Transform to a new feature space by using FLD(Fisher's Linear Discriminant). Considering that SVM includes its own linear or nonlinear mapping of features, we won't check feature adjustment on SVM in this part.

- **Find best choice of features combination:**
  - Directly check for all the combination of 7 features, and choose the best combination from their results in cross validation. This may be helpful to decrease some degree of freedom when we have many features.
  - Use *itertools.combinations* to generate all the subset of 7 features.

- **PCA:**
  - Apply eigenvalues and eigenvectors of covarianve matrix to find the transform weight with

minimum mean square error between original space and target space. PCA is a linear transformation of feature space.
- Use *sklearn.decomposition.PCA* to implement this transform with normalized data. Adjust the parameter *n_components* to save number of features after transformation. Also train this process by cross validation and get accuracy mean and standard variance.

- **FLD:**
- FLD transforms original space to a new feature space with maximum distance between means of different classes and minimum sum of standard variance of different classes. Theoretically, FLD can find the best projection of features in different classes. Here we need to do FLD on 4 classes and transform original space to 1 to 3 dimensions.
- Use *sklearn.discriminant_analysis.LinearDiscriminantAnalysis* to implement this transform with normalized data. Choose the best *n_components* from 1, 2, 3 by cross validation.

For each model except SVM, compare above three methods with only preprocessing data, and choose the one with best result:(use n to represent *n_components* in the table below)

mean std	**preprocessing**	**combination**	**PCA**	**LDA**	**final choice**
GNB	original 0.9846 0.0061	**(0,1,2,3,4,6)** **0.98575** **0.005309**	n = 4 0.958875 0.0110	n = 3 0.9775 0.0071	**combination (0,1,2,3,4,6)**
MSE	normalize 0.77525 0.0156	(0,1,2,3,4,5,6) 0.7760 0.0157	n = 7 0.7745 0.0183	**n = 3** **0.7762** **0.0175**	**LDA (n = 3)**
Perceptron (SGD, OvR)	normalize 0.94325 0.0198	(0,1 2,4,5,6) 0.9329 0.0181	**n = 7** **0.9456** **0.0245**	n = 2 0.9127 0.0669	**PCA (n = 7)**
KNN (K = 4)	**original** **0.984375** **0.0050**	(0,2,3,4,5,6) 0.9833 0.0060	n = 7 0.9828 0.0072	n = 3 0.9802 0.0068	**None**

Different models will get different best selections.

(1) For combination, we still remain most of the features, which indicates that all 7 features will give some information to classification.

(2) Also for PCA, most models get best result at n = 7. As we know, every component after PCA transformation will also contribute some information. So theoretically, all the models will get best result at n = 7 in PCA, that is, use all the information without dropping. But here, for GNB methods, we get the best components at n = 4, where we drop 3 components with least contribution and get better result than use all. This may be because those dropped components will not improve training but including some misleading distribution in GNB training.

(3) For LDA, we get most models get best result at n = 3.

(4) Consider that room has position like (x,y) in real space, which is exactly 2 features. So I plot the data at n = 2 in PCA and n = 2 in LDA. If the plot can be corresponed by the real position of rooms, then it will show that class 3 is surrounded by the rest three classes. Theoretically,rooms' position and region can be linearly seperable, but I find that datas in these two plots are not totally linearly seperable, and many data near the boundary of classes are mixed with each other. Data points in FLD are more centered and less spreaded than LDA but still has little points in other classes region. This indicates that my feature adjustment is still not so perfect to find the best new feature space. On the other hand, this also indicate that these data may not be linearly seperable in current two feature spaces.

(5) For statistical classification as GNB and KNN here, they are non linear classifiers, thus projection to lower feature space may not help so much on the distribution classification.

(6) For MSE and Perceotron learning here, both of them are linear classifiers, so they rely more on feature adjustment than non linear classifier.

2.3 Dataset Usage

In selecting preprocessing and feature adjust methods above, and model parameter selection following, I use the training data set with 1600 datas.

For each comparation situation, I will run cross validation for 5 times, each time seperate training set into 5 folds by using *sklearn.model_selection.StratifiedKFold*, that is, each fold has 320 datas. Record accuracy of each fold in every time, and then use this accuracies to compute final accuracy and mean and standard variance.

For final model compare, I run the model on all the training data (1600 datas), and predict the accuracy and confusion matrix on training data. Then run this fitted model on test data, and predict the accuracy and confusion matrix on test data. Considering the test with confidence measure, I totally use test set 10 times (6 for models without confidence measure, 4 foe models with confidence measure).

2.4 Training and Classification

This part, I will train different models with preprocessing and feature adjustment methods selected above, and find their best parameters. For each model, selected preprocessing and feature adjustment methods are as following:

	Normalize	**Feature Adjustment**
GNB	false	false
SVM linear	true	false
SVM rbf	true	false
MSE	true	LDA (n = 3)
Perceptron (SGD, OvR)	true	PCA (n = 7)
KNN	false	false

- **GNB:**
  - GNB(Gaussian Naive Bayes) is a non linear classifier, which will assume the distribution of features as Gaussian distribution, and then use all the data to eatimate their Gaussian distribution parameters.
  - Here I use *sklearn.naive_bayes.GaussianNB* in python to create the classifier.
  - The parameters will computed in the process of training.

- **SVM:**
  - The goal of SVM is to find the largest classified margin. It will minimize a criterien function with constraints of correctly classifying labels by Lagrange Optimazation.
  - Lagrange Optimization will apply differnt kernel, which means a mapping of original features. If kernel is linear, then it's a linear mapping; If kernel is non linear, then it's a non linear mapping.
  - **linear SVM:** SVM with linear kernel, which is a linear mapping of features and is a linear classifier. I use *sklearn.svm.SVC* with *kernel = 'linear'* to implement. For parameter C, I use cross validation to select from 0.1 to 10.
  - **rbf SVM:** SVM with radial basis kernel, which is a non linear mapping of features and is a non linear classifier. I use *sklearn.svm.SVC* with *kernel = 'rbf'* to implement. For parameter C and *gamma*, I use cross validation to select from (0.01, 100) and (0.1, 10).
  - Change of parameters: In SVM, results of different parameters will float in a large region, thus it is difficult to find the global maximum result.

- **MSE:**
  - MSE is a linear classifier to minimize sum of the error points' distance with margin. Here I use MSE pseudo-inverse version, which is a one iteration training model.
  - Use *sklearn.linear_model.LinearRegression* to construct my own MSE_binary classifier, which is to classify two class problem. And then use *sklearn.multiclass.OneVsRestClassifier* to apply

MSE_binary to multiclass problem.
- The parameters are chosen in process of training.

- **Perceptron:**
- Here I use the perceptron learning with SGD method to optimize criterien function and OvR method to apply it to multiclass problem. It is a linear classifier with several iterations to minimize the sum of error points in augmented space.
- Use *sklearn.linear_model.Perceptron* to implement the classifier, with parameter *tol* = 1e-3 which means the stop condition.
- The parameters are chosen in process of training.

- **KNN:**
- KNN is a non parameter statistical estimation method with specified k neighbor points. It classifies the data to the class which has most points in k neighbor.
- Use *sklearn.neighbors.KNeighborsClassifier* to implement the classifier. Try the parameter *n_neighbors* in list (1,10,100,5,6,4,3).

	mean accuracy	**standard variance**
K = 1	0.983	0.0068
K = 3	0.9827	0.0091
K = 4	0.98425	0.0070
K = 5	0.9832	0.0078
K = 6	0.98225	0.0079
K = 10	0.98225	0.0079
K = 100	0.9773	0.0096

- Change of the parameter: If the parameter is too small or too big, the accuracy will decrease. This may be because if the parameter is too small like 1, then it only uses one data's information to judge its own class, it will be dangerous on the mixed area of two classes. Also if the parameter is too big like 100, then it uses extra datas that looks too much outside the mixed area, thus may lead to some misclassification.

3 Analysis: Comparison of Results, Interpretation

Performance of different models on cross validation, training set and test set. Get accuracy mean and standard variance in corss validation. Get accuracy and confusion matrix on training and test set. In confusion matrix, row index shows the actual label, column index shows the predicted label.

Compare and analyze the data in the next page:

- **Difference and similarities between cross vaildation, training set and test set:**
(1) **Accuracy:** accuracy of training set is commonly greater than cross validation and test set. The former one may be because training set has more data than cross validation on both training and label predicting process. Cross validation is trained on 4 folds (1280 datas) and tested on 1 folds (320 datas), while training set is trained and tested on all the training data (1600 datas). The latter one may be because test set has less data than training set on label predicting process, and there may be some information that is unique on test set, which has not learned by trained classifier.

(2) **Confusion matrix:**
- values on the diagnal entries, which means the number of data correctly classified, are the biggest. Other entries shows how the classifier misclassified those datas: number of datas in class i (row index) classified to class j (column index).
- There are some similar ratio at each index between training and test set for each model. This also makes sense, because we use the same preprocessing and feature adjustment on both training and test set, thus they are in the same feature space. Then if in training data, data points

at the mixed area of class 2 and class 3 are failed to be correctly classified, then in test data, there will also be similar ratio of points in this area be misclassified. In the dataset we used here, class 2 and class 3 are most easily to be misclassified.

	Preprocessing Feature Ad Param	**cross validation**	**training set**	**training confusion matrix**	**test set**	**test confusion matrix**
GNB	false false	0.9846 0.0061	**0.985**	[399 0 1 0] [0 384 16 0] [1 1 396 2] [2 0 1 397]	0.9775	[99 0 1 0] [0 94 6 0] [1 1 98 0] [0 0 0 100]
SVM linear (C = 0.99)	true false	0.98375 0.0050	**0.9843**	[399 0 0 1] [0 388 12 0] [2 5 391 2] [1 0 2 397]	0.9775	[99 0 1 0] [0 93 7 0] [0 1 99 0] [0 0 0 100]
SVM rbf (C = 0.5623, gm = 1.0)	true false	0.98375 0.0023	**0.9881**	[399 0 0 1] [0 390 10 0] [0 3 395 2] [2 0 1 397]	0.98	[99 0 1 0] [0 96 4 0] [0 3 97 0] [0 0 0 100]
MSE	true LDA(n = 3)	0.77625 0.0175	0.7768	[388 0 5 7] [1 333 49 17] [2 10 125 263] [1 0 2 397]	**0.79**	[98 0 1 1] [1 85 11 3] [1 5 33 61] [0 0 0 100]
Perceptron (SGD, OvR)	true PCA(n = 7)	0.94562 0.0245	**0.9418**	[399 0 1 0] [0 388 12 0] [1 49 324 26] [1 0 3 396]	0.925	[99 0 0 1] [0 91 9 0] [0 17 80 3] [0 0 0 100]
KNN (K = 4)	false false	0.98425 0.0070	**0.9925**	[400 0 0 0] [0 379 3 0] [4 2 393 1] [2 0 0 398]	0.9825	[99 0 1 0] [0 96 4 0] [1 1 98 0] [0 0 0 100]

- **Accuracy compared between different models:**
  (1) **cross validation set:**
  - Accuracy on cross validation set:

$$GNB > KNN > SVM(rbf) > SVM(linear) > Perceptron > MSE$$

GNB , KNN and SVM(rbf) are non linear classifiers; SVM(linear), Perceptron and MSE are linear classifiers. Thus in this non linearly seperable problem and the final feature space is also nonlinearly seperable, non linear classifiers perform better than linear classifier.

- Standard variance on cross validation set:

$$SVM(rbf) < SVM(linear) < GNB < KNN < MSE < Perceptron$$

Perceptron is a random classifier with iterations, which means in different run time on the same data set will results in different trained classifier with different weight vector, thus causes a big standard variance. So generally, in non linearly seperable problem like here, constant classifiers will have smaller standard variance than random classifiers.

(2) **training set:**
- Accuracy on training set:

$$KNN > SVM(rbf) > GNB > SVM(linear) > Perceptron > MSE$$

Still, non linear classifiers have better performence than linear classifiers. Here the reason why KNN and SVM(rbf) perform better than GNB may be: GNB has already estimated parameters on cross validation, the extra number on training set does little change on the parameters, thus will not change the discriminant rules too much, which leads to small changes among cross validation and training set. While the decision boundaries for KNN and SVM(rbf) will have more change with new datas.

(3) **test set:**
- Accuracy on test set:

$$KNN > SVM(rbf) > GNB > SVM(linear) > Percceptron > MSE$$

This result is almost same as training data, which means our models have learned useful information to classify test data.

(4) **Overall:**
KNN has the best performance and MSE has the worst performance in comapraring with other methods we used here.

- **Confidence measure:**
- To improve the result in another way: set constraints of the probability of the predict label, if the probability estimated is lower than a threshold, the data will be regared as rejected data and will not be countered.
- Get *.predict_proba* parameters and set *probability = True* in KNN, SVM and GNB models. Set threshold and filter the data with smaller predict probability.
- With threshold more close to 1.0, our final accuracy will quickly reach 1.0, but meantime with more data being rejected. So this method is just a selection of unsatisfied data after training.
- When we set threshold = 0.9, KNN can reach 1.0 accuracy on training set and 0.9973 accuracy on test set. For other models need more strict threshold (greater than 0.9) and with more data be dropped.

4 Summary and conclusions

- **Key findings:**
(1) Wifi Signal Strength dataset is non linearly seperable. I get the best accuracy result of 0.9825 by using KNN with 4 nearest neighbors.
(2) In non linearly seperable problem, generally a non linear classifier will perform better than a linear classifier. So if we want to improve the accuracy of non linearly seperable problem, we can :
- Do most of our work on feature adjustment or feature transformation before training linear classifier. More powerful way to do this may be to use ANN with more than one layers to extract deeper features.
- Use a non linear classifier, choose proper feature adjustment and transformation method, find parameters as best as possible.
- Add more new datas.
(3) For constant and random classifier, the latter will have a greater standard variance than the former in non linearly seperable cases.

- **Intersting or useful things to do as follow-on work:**
(1) To find if we can find a feature space that can be linearly seperable. Theoretically this is feasible, because in real world, these 4 rooms must be linearly seperable.
(2) Find more detailed information about how distance, medium and devices will affect strengh of wifi signal. According to this, we may find ideas of how to extract features.
(3) In this problem KNN works really well with the easiest algorithm and withou feature

extraction, want to find out how it works on other non linearly seperable classes.
(4) Try to work this problem by ANN or other structrue of deep learning, find out how do they work.

- **Tips:**
(1) Before starting the program, we need to familiar with the dataset, like if it is balanced, sparced or well spread and etc.
(2) Choose a baseline model to find more information of the dataset. Generally, we can choose Naive Bayes model.
(3) To improve the baseline model, we may first do the preprocessing and the feature adjustment correspending to current model.
(4) In every situation for comparing and finding the best result, don't forget to use the cross validation method.
(5) To further improve our result, we can choose more models with proper preprocessing and feature adjustment to train the data.
(6) For higher dimension feature space, it may be difficult to find the 2D plot of the datas and also difficult to find relationship between datas. Here we can use confusion matrix to find the confusion region we are interested in.

References

[1] *User Localization in an Indoor Environment Using Fuzzy Hybrid of Particle Swarm Optimization and Gravitational Search Algorithm with Neural Networks*, available at `https://www.researchgate.net/publication/313954230_User_Localization_in_an_Indoor_Environment_Using_Fuzzy_Hybrid_of_Particle_Swarm_Optimization_Gravitational_Search_Algorithm_with_Neural_Networks`

[2] *Classifier comparison*, available at `https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py`