Pattern Recognition Analysis

Data Set: Hand Postures

Tieming Sun, tiemings@usc.edu

May 8th, 2020

1. Abstract

This project makes a thorough procedures for pattern recognition system in a hand posture dataset. After data preprocessing, feature recombination, cross-validation and model training, a support vector machine (SVM) based algorithm with optimal parameter is trained, leading to f1-score of 0.96 in testing dataset. Meanwhile, feature reduction and nested cross validation are fully analyzed in order to improve the accuracy for better model training, especially for k-nearest neighbors.

2. Introduction

2.1. Problem Statement and Goals

In this project, the Hand Posture Dataset is processed. [1] This data set contains information of 5 different hand postures monitored by 12 markers on the glove, which means the data set has class of 5. Useful labels are:

'Class': integer range from 1 to 5, meaning there are five classes of postures.

'User': ID of users, in training dataset, there are totally 9 users, and testing dataset exists 2 other users.

'Xi', 'Yi', 'Zi': the x-y-z coordinate of i-th marker's positions, totally there are 12 markers.

Then a counting method is used to show how many data belongs to different class, in order to determine this dataset is balancing or not. The following histograms shows how data is distributed in classes on training dataset and testing dataset. We could find the number of data in different classes could be seen as equally distributed.

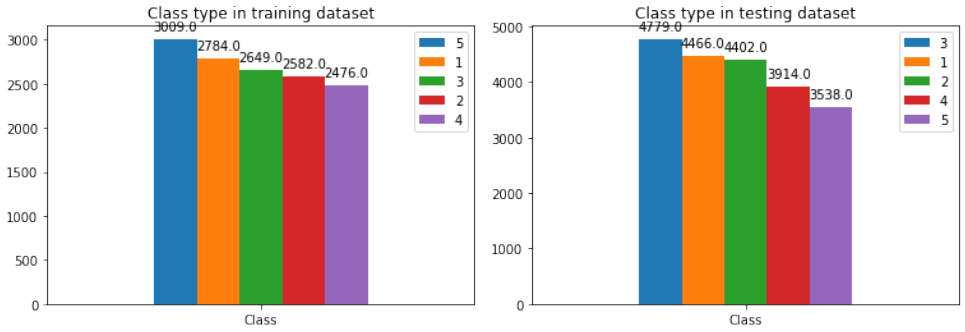

Figure 1. Class types in training and testing dataset

The desired achievement of this project is to use pattern recognition procedures, including preprocessing, feature-design, cross-validation, model training, testing, to optimize and compare model's metrics (accuracy score, f1-score).

3. Approach and Implementation

3.1. Feature engineering

It's common to deal with missing data in the real-world dataset, however, in this hand posture dataset, some markers might be hidden in some classes or postures. Thus, valid coordinates vary from 9 to 36 among different postures, the difference of valid data is so notable that we have to redesign the features.

To convert the data into a useful form, I use the following 13 features: number of recorded markers, mean x of marker location, mean y of marker location, mean z of marker location, standard deviation x of marker locations, standard deviation y of marker locations, standard deviation z of marker locations, maximum x of marker locations(similar to y, z), and minimum x of marker locations(similar to y, z).

3.2. Preprocessing

Data preprocessing is a common technique to change and transform the raw data into the standardized data. This technique is useful because some models like Multilayer Perceptron are sensitive to the input data range. Another preprocessing method is to normalize the raw data, that is, getting the maximum and minimum number in one feature, then scaling them to a real number ranging from 0 to 1.

For the hand posture data set, I use standardizing to set data into zero mean, one standard deviation to all real-number features using function "StandardScaler" from library sklearn.preprocessing.

3.3. Feature dimensionality adjustment

Common feature dimensionality adjustment contains choose k-best features and PCA. PCA, as abbreviation of principal component analysis, is firstly normalized the original data, and tries to find a best fitting line to minimize MSE to this projected line. After feature engineering and preprocessing, we could make an exploration by applying PCA. The figure below shows how 5-class distributes has reduced features into 2, using function "PCA" from library sklearn.decomposition.

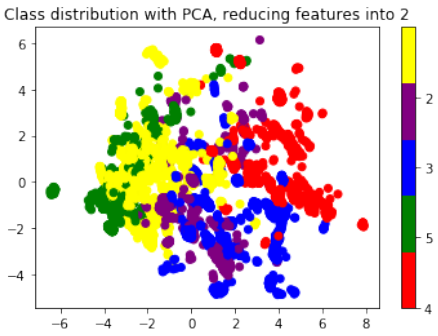

Figure 2. Class distribution with PCA, reducing features into 2

From this figure, we could notice that as PCA loses useful information by reducing data's effective dimensions, PCA with 2 dimensions doesn't work well, but giving us an intuition about class distribution.

Secondly, as for degree of freedoms (d.o.f), among all models we use, some model (such as KNN) is sensitive to numbers of features. For Bayes minimum-error estimators with density estimation, if parameter "k_neighbors" is low, then

$$d.o.f = M^D,$$

where M is number of bins, which is inversely proportional to "k_neighbors"; D is number of features

if $d.o.f < (3-10)N_C$, then we are facing the under-fitting problem, which is the so-called curse of dimensionality.

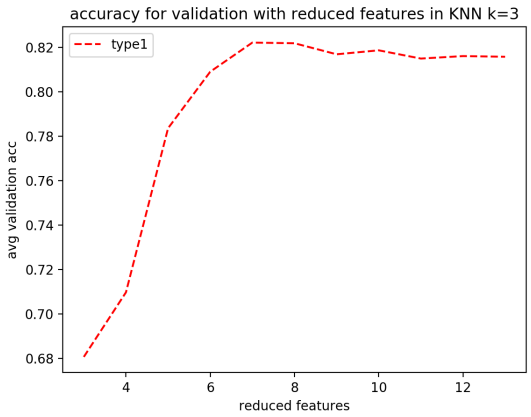

Figure 3. Accuracy for validation with reduced features, in KNN, k=3

The following figure shows how numbers of features affects result in validation dataset. And we could find that the curse of dimensionality isn't affect much. Moreover, if k_neighbors increases, d.o.f. will further decreases. So, in the following of project, all 13 features is still chosen to train the dataset.

3.4. Dataset Usage

3.4.1. Cross validation

This project uses leave-one-user-out technique. Due to the relatively small number of users, and the high correlations of data points given by a single user, we should always keep track of 'User ID' label, to set training dataset and validation dataset. Thus, we should use the so-called leave-one-user-out technique to split training and validation set. This method will give a result in validating parameters using data that the training model haven't seen before, leading to a better way to search on hyper parameters.

Furthermore, based on this single layer cross-validation, nested cross validation is also tried. After applying the leave-one-out method, I use a stratified method (set 5 folds) to separate the training dataset into smaller-training and validation parts. In order to accelerate speed, a random parameter choosing function is used in the stratified layer. In sklearn, it is called "RandomizedSearchCV". To test the effect of the nested cross validation, this method is tried on K-nearest neighbors (choose n_neighbors) to show the effects. After the nested cross validation, I generate a histogram to generate the best parameter. In this method, n_neighbors = 3 is chosen, the average for validation set is 0.816, and the single layer cross validation is

0.820. The nested cross validation might be useful for refining parameter, but the effect of nested cross validation might be overestimated.

3.4.2. Overall Procedures

In this project, firstly when dealing with the raw data to get the 13 new features, then based on these features, features are standardized except 'count', because 'count' data is the shown as integer. Test dataset is dealt with the same procedure, and it is standardized using the mean and variance calculated in training dataset. Therefore, I get a training dataset, with 5 classes, 13 features, and 13500 data points, for the testing dataset, there are 21098 data points.

Then, the leave-one-out technique is applied for the cross-validation part, as training dataset has 9 'User ID', each time I choose one user as the validation set, and the rest is the remaining dataset. So, there are 9 folds in a single 'for loop' in the cross validation. In this loop, I take down the average and standard deviation of accuracy for a model with parameter. Then choose the highest average and lowest standard deviation of accuracy as the best hyper parameter, which is good candidates.

Once finishing the cross-validation part, test dataset is used to evaluate model with parameters. The test dataset contains different 'User' to training dataset's, so it's safe to use the dataset. To fully check the effect of the trained model, confusion matrix and evaluation report are used to see precision, recall and f1-score for each class. Based on these evaluations, a better model with trained parameter is chosen.

3.5. Training and Classification

3.5.1. Naïve Bayes

Naïve Bayes method is a supervised learning based on Bayes theorem. This method is based on a simple assumption, each features of the dataset is (conditional) independent. As a baseline system, Gaussian Naïve Bayes is implemented, and the likelihood of features is assumed to be Gaussian. The function can be called as 'GaussianNB' in sklearn.naive_bayes and be used with cross validation method. The average validation accuracy is 0.78.

3.5.2. K Nearest Neighbors

K Nearest Neighbors(KNN) is also a part of statistical method classifier, which is still based on Bayes theorem. To estimate $P(x|S_i)$, we assume x_i are i.i.d. in class S_i. Number of neighbors is set so that KNN set different number of bins to estimate $P(x|S_i)$. Implementation is completed by using function "KNeighborsClassifier" in sklearn.neighbors library. Other than k_neighbors, Power parameter for the Minkowski metric is another parameter that matters. One can choose to use Euclidean distance and Manhattan distance to form the tree distance (used to compute nearest neighbors).

To get the best parameter, grid search is used to compute the best validation accuracy in cross validation. The final answer is K=14, p=1, with validation accuracy=0.817, validation standard deviation=0.113.

Feature reduction and nested cross validation are also applied to KNN algorithm, however, with a tiny improvement of validation accuracy, the standard deviation also gets higher, thus, these two methods aren't work very well in this dataset. To see detail, please see Section 3.3 and 3.4.1.

3.5.3. Support Vector Machine

Support Vector Machine (SVM) is a supervised learning algorithm, by maximizing the margin of different classes and choosing support vector to find the boundaries. It is effectively used on high dimension space, and memorial efficient because it only uses a small portion of data to set boundary. This implementation is completed by function SVC in sklearn.svm library. And SVM contains multiple kernel functions to choose.

This project uses two non-linear kernel functions: 'rbf' for radial basis function and 'poly' for boundaries are polynomial nature. In order to find the optimal parameter 'C' and 'gamma', grid search is a common choice. However, the number of datapoints in Hand Posture dataset is relatively high, so that grid search for different 'C' and 'gamma' is very time-consuming.

Thus, a rough grid search is firstly implemented with a bigger region of the two parameters, then 'gamma' is set to be 1/n_features = 1/13, to search base on 'C'. The following figure shows the rough grid search for kernel='rbf'.

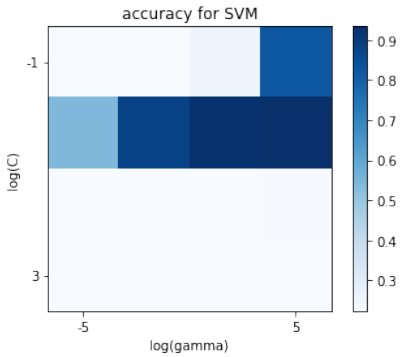

Figure 4. Accuracy of SVM using grid search technique

Finally, for radial basis function kernel, C= 0.78804628, gamma = 0.0769; for polynomial boundary kernel, C = 0.0067, gamma = 0.0769.

3.5.4. Multilayer Perceptron Classifier

Multilayer Perceptron Classifier (MLP) is a neural network classifier training a function from C dimensions into 1 dimension. For multi-class labels, it uses OVR method to compute the result. And it contains activation functions and one or more hidden layers to solve Perceptron algorithm. Finally, it outputs a single number, showing its class. This is implemented by function MLPClassifier.

The crucial parameters for MLP is hidden layers, and number of neurons & number of layers could be defined by user. For activation, Relu is most commonly used.

With a grid search on number of neurons and number of layers, the best parameter for cross validation is one layer with 200 neurons. The average accuracy is 0.86.

3.5.5. Perceptron

Perceptron is very common linear models. Here we use OVR methods to deal with multiclass dataset.

Apply max iteration time as 1000 and set balanced class weight. By cross validation, the average accuracy for perceptron is 0.762 and 0.74.

3.5.6. Decision Tree

“Decision Tree is a tree like graph model. It is a map of possible outcomes a series of related choices. Statistical methods are used for ordering the attribute as root node or internal nodes. Calculating the probability of a

given record belongs to each of category. “[2] It is used from “DecisionTree Classifier” from sklearn.tree.

By cross validation, its average validation accuracy is 0.74.

3.5.7. Random Forest

“Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model’s prediction. “[3]

By choosing the max_depth as the jump out condition in cross validation, its average validation accuracy is 0.85 with max_depth=600.

3.5.8. Random Classifier

Generate a totally random classifier, average validation accuracy is 0.20. Call function from sklearn.dummy. Accuracy is about 0.20.

4. Analysis: Comparison of Results, Interpretation

4.1. Accuracy Metrics

4.1.1. Confusion Matrix

Confusion Matrix is a table that shows (two classes case):

	Class 1 predicted	Class 2 predicted
Class 1 actual	TP	FN
Class 2 actual	FP	TN

4.1.2. Classification Report

Classification report is a useful function, which will show a set of report for each class.

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Recall = TP / (TP + FN)

Precision = TP /(TP + FP)

F1-score = 2* Recall * Precision / (Recall + Precision)

4.2. Model Report

4.2.1. Data reports

There are a list of tables for confusion matrix in different model, choosing the best parameter in the cross validation part.

Table 1 . Confusion matrix for naïve Bayes and KNN

Naïve	Class 1	Class 2	Class 3	Class 4	Class 5	KNN	Class 1	Class 2	Class 3	Class 4	Class 5
Class 1	4330	48	73	15	0	Class 1	4268	48	66	12	72
Class 2	116	4173	0	0	113	Class 2	115	3764	0	0	523
Class 3	455	61	2167	338	1758	Class 3	102	15	3102	1498	62
Class 4	0	915	66	2887	46	Class 4	0	43	242	3592	37
Class 5	0	105	19	34	3380	Class 5	0	55	60	242	3181

Table 2. Confusion matrix for SVM_rbf / poly

rbf	Class 1	Class 2	Class 3	Class 4	Class 5	poly	Class 1	Class 2	Class 3	Class 4	Class 5
Class 1	4350	48	3	2	63	Class 1	4315	48	103	0	0
Class 2	38	3960	32	0	372	Class 2	31	4232	77	0	62
Class 3	61	0	4255	461	2	Class 3	0	0	4657	93	29
Class 4	0	1	0	3913	0	Class 4	0	0	277	3545	92
Class 5	0	74	1	32	3431	Class 5	0	18	39	57	3424

Table 3. Confusion matrix for MLP / Perceptron

MLP	Class 1	Class 2	Class 3	Class 4	Class 5	Percept	Class 1	Class 2	Class 3	Class 4	Class 5
Class 1	4395	48	7	15	1	Class 1	4068	48	199	0	151
Class 2	42	2321	514	0	1525	Class 2	83	4265	54	0	0
Class 3	157	0	3699	238	685	Class 3	24	61	4571	123	0
Class 4	0	125	34	2904	851	Class 4	0	2681	46	1187	0
Class 5	0	26	7	21	3484	Class 5	304	1875	393	217	749

Table 4. Confusion matrix for Decision Tree / Random Forest

Dec T	Class 1	Class 2	Class 3	Class 4	Class 5	Rand F	Class 1	Class 2	Class 3	Class 4	Class 5
Class 1	4329	15	61	2	59	Class 1	4346	16	4	0	100
Class 2	7	1783	0	196	2416	Class 2	0	1838	1	4	2559
Class 3	0	1	4024	339	415	Class 3	0	0	3802	35	942
Class 4	0	0	1342	1646	926	Class 4	0	68	1002	1752	1092
Class 5	0	5	449	11	3073	Class 5	0	18	57	19	3444

Table 5. Report for naïve Bayes and KNN

NaiveBayes	precision	recall	f1-score	support	KNN	precision	recall	f1-score	support
1	0.88	0.97	0.92	4466.00		0.95	0.96	0.95	4466.00
2	0.79	0.95	0.86	4402.00		0.96	0.86	0.90	4402.00
3	0.93	0.45	0.61	4779.00		0.89	0.65	0.75	4779.00
4	0.88	0.74	0.80	3914.00		0.67	0.92	0.78	3914.00
5	0.64	0.96	0.77	3538.00		0.82	0.90	0.86	3538.00
accuracy	0.80	0.80	0.80	0.80		0.85	0.85	0.85	0.85
macro avg	0.82	0.81	0.79	21099.00		0.86	0.86	0.85	21099.00
weighted avg	0.83	0.80	0.79	21099.00		0.87	0.85	0.85	21099.00

Table 6. Report for SVM_rbf / poly

SVC_rbf	precision	recall	f1-score	support	SVC_poly	precision	recall	f1-score	support
1	0.98	0.97	0.98	4466.00		0.99	0.97	0.98	4466.00
2	0.97	0.90	0.93	4402.00		0.98	0.96	0.97	4402.00
3	0.99	0.89	0.94	4779.00		0.90	0.97	0.94	4779.00
4	0.89	1.00	0.94	3914.00		0.96	0.91	0.93	3914.00
5	0.89	0.97	0.93	3538.00		0.95	0.97	0.96	3538.00
accuracy	0.94	0.94	0.94	0.94		0.96	0.96	0.96	0.96
macro avg	0.94	0.95	0.94	21099.00		0.96	0.96	0.96	21099.00
weighted avg	0.95	0.94	0.94	21099.00		0.96	0.96	0.96	21099.00

Table 7. Confusion matrix for MLP / Perceptron

MLP	precision	recall	f1-score	support	Perceptron	precision	recall	f1-score	support
1	0.96	0.98	0.97	4466.00		0.91	0.91	0.91	4466.00
2	0.92	0.53	0.67	4402.00		0.48	0.97	0.64	4402.00
3	0.87	0.77	0.82	4779.00		0.87	0.96	0.91	4779.00
4	0.91	0.74	0.82	3914.00		0.78	0.30	0.44	3914.00
5	0.53	0.98	0.69	3538.00		0.83	0.21	0.34	3538.00
accuracy	0.80	0.80	0.80	0.80		0.70	0.70	0.70	0.70
macro avg	0.84	0.80	0.79	21099.00		0.77	0.67	0.65	21099.00
weighted avg	0.85	0.80	0.80	21099.00		0.77	0.70	0.67	21099.00

Table 8. Report for Decision Tree/ Random Forest

Dec Tree	precision	recall	f1-score	support	Rand Forest	precision	recall	f1-score	support
1	1.00	0.97	0.98	4466.00		1.00	0.97	0.99	4466.00
2	0.99	0.41	0.57	4402.00		0.95	0.41	0.57	4402.00
3	0.68	0.84	0.76	4779.00		0.85	0.81	0.83	4779.00
4	0.75	0.42	0.54	3914.00		0.92	0.44	0.60	3914.00
5	0.45	0.87	0.59	3538.00		0.41	0.98	0.58	3538.00
accuracy	0.70	0.70	0.70	0.70		0.72	0.72	0.72	0.72
macro avg	0.77	0.70	0.69	21099.00		0.83	0.72	0.71	21099.00
weighted avg	0.79	0.70	0.70	21099.00		0.84	0.72	0.73	21099.00

4.2.2. Result analysis

In this report, 8 different kinds of supervised/unsupervised learning algorithm are implemented.

As a baseline, Naïve Bayes works as f1-score of 0.80, which is not a very bad performance. There is much difference of accuracy between class 1,2,4 and 3,5. In this dataset, naïve Bayes works badly on posture 3 and 5. As naïve Bayes assume on the independence of features, posture 3 and 5 may share more common features in natural.

For KNN, it works as f1-score of 0.85, 5% higher than Naïve Bayes, it shows that the density estimation of $P(x|S_i)$ is more "real-world" than Naïve Bayes' assumption, $P(x|S_i)$ is Gaussian. But this performance isn't work very well. Another notice on KNN is that, during the cross-validation part, if k neighbors ranges from 5 to 50, the average accuracy and standard deviation for validation doesn't change much. In other words, the choice of number of neighbors doesn't affect the result too much.

Also, for Perceptron and multi-layer perceptron algorithm. Perceptron is a linear classification, so the relatively low f1-score is normal. Multi-layer Perceptron is a non-linear neuron network classification, for every neuron, it uses a perceptron algorithm to train data. In this report, the f1-score 0.80, which is almost the same as Naïve Bayes. However, it doesn't work stable in this report as it randomized much, and different random state leads to different solution, which is the reason why I don't choose it.

For decision tree and random forest algorithm, they have a f1-score of 0.70 and 0.72, which are much lower than Naïve Bayes. The reason why they don't work well is both algorithms work on a subset of data and tend to be overfit. That's why random forest has a validation accuracy much higher than test data accuracy.

Lastly, for SVM with radial basis function and polynomial kernel, both of them work very good in both validation accuracy and test accuracy. For f1-score, SVM_rbf = 0.94, and SVM_poly = 0.96, both of which are much higher than 0.80, Naïve Bayes' score. For different classes, SVM_poly has the best average f1-score and accuracy of 0.96, while SVM_rbf has more balanced f1-score in different classes. Also, for the SVM_poly, totally it uses 720 data points as the support vectors, which has the proportion of 0.05 in training dataset. This proportion is relatively "healthy", without overfitting problems. The reason why they work so excellent is mainly because they maximum the boundary margin in different classes. And as the dataset has higher dimension and not too imbalanced, find this margin is much easier.

Overall, considering all of the algorithms, SVM with radial basis kernel and polynomial kernel are the best ones in the hand posture dataset.

5. Summary and conclusions

To maximize the performance, it is crucial to process data with projects' procedures, which is preprocessing, feature engineering, feature analysis, cross validation, training parameter and test model with trained parameter.

Among 8 different supervised / unsupervised learning algorithm, SVM with radial basis kernel and polynomial kernel has the best performance, the f1-score in testing dataset are 0.94 and 0.96, with cross-validation accuracy of 0.91, 0.93.

It's necessary to apply feature analysis (extraction) in KNN algorithm, as KNN may face "the curse of dimensionality", which means when 'n_neighbors' is low, PCA for the existing feature might be helpful.

When testing multi-class dataset, a fully examination of all classes' confusion matrix is crucial. Some algorithm works excellent in a few classes but not all, which leads to the drop of overall accuracy.

References

[1] "Motion Capture Hand Postures Dataset " [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Motion+Capture+Hand+Postures

[2] "Introduction to decision trees" 25 October 2018. [Online]. Available: https://medi-um.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb.

[3] "Understanding Random forest" 12 January 2019. [Online]. Available: https://towardsdatascienc- e.com/understanding-random-forest-58381e0602d2