EE559 final Project

Data Set: Hand Postures

Wenhao Cui, wenhaocu@usc.edu

05/07/2020.

1. Abstract

In this project, I try to deal with the hand postures data set from UCI, specifically, it has 12 volunteers to show their hand postures and we set these postures from posture 1 to posture 5 as class 1 to class 5. From this information, I try to know if these postures can be figured out by our machine. And if the different posture will show the same similar accuracy in each classification. To figure out, I first preprocess the data by removing the unlabeled data, then feature extract the data as 13 features, rescaling, dimensionality adjustment (using PCA), use cross-validation (leave-one-user-out). I choose perceptron, SVM, Naïve Bayes, KNN, and neural network(MLP) as my classifiers, and compare to all of these, I get the best performance from polynomial SVM with C:0.10404983103657856 and its test score is 0.921312753105729.

2. Introduction

2.1. Problem Statement and Goals

In this project, I use 5 types of hand postures from 12 users that were recorded using unlabeled markers on the fingers of a glove in a motion capture environment. My goal is, after preprocessing data, I need to find the optimal classifier, then try to improve our classifier performance when we need to predict new unlabeled data.

2.2. Literature Review

When I preprocess the data, I Normalized and Standardized data. Normalization typically means rescales the values into a range of [0,1]. Standardization typically means rescales data to have a mean of 0 and a standard deviation of 1 (unit variance). Then I use PCA and K Best to reduce the dimensionalities. PCA (Principal component analysis) is used to decompose a multivariate dataset in a set of successive orthogonal components that explain a maximum amount of the variance. K Best is to Select features according to the k highest scores. For choosing classifiers, I tried perceptron, SVM, Naïve Bayes, KNN, and neural network (MLP).

Perceptron is a simple classification algorithm suitable for large scale learning especially in linear. Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression, and outlier detection. Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption. K neighbors-based classification is a type of instance-based learning or non-generalizing learning. Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function f() from $R^n \rightarrow R^0$ with multiple layers.

3. Approach and Implementation

3.1. Preprocessing (wash nan and data extraction)

First, I downloaded the raw data and find many data are unlabeled. Therefore, I need to clean the raw data without nan first. So I loaded data as a dataframe using pandas and assigned the column as 'number', 'x_mean', 'y_mean', 'z_mean', 'x_std', 'y_std', 'z_std', 'x_max', 'y_max', 'z_max', 'x_min', 'y_min', 'z_min. At last I also added the columns class and feature for my next cross validation.

In this step, there are some maker numbers (0 in some column) that have no significance (unlabeled). When I calculated these features, I use np.nanmean, np.nanstd, np.nanmax, np.nanmin to prevent the influence of these blank positions. Also, I use each row size minus the blank positions n each row to get the number parameters to consider the influence of unlabeled data points as 1 parameter.

Meaning of each feature:
For each data point: number of recorded markers, mean x of marker locations, mean y of marker locations, mean z of marker locations; the standard deviation of x of marker locations (similarly for y and z); maximum x of marker locations (similarly for y and z), and minimum x of marker locations (similarly for y and z)

3.2. Preprocessing (rescaling)

After extracting the feature, I tried preprocessing the data with Standardization and Normalization. In these two methods, I both used train set to fit and transform the training set and testing set.

The reason why I choose these two preprocessing methods is that they both can preprocess the data, which means that the values fall into a unified range

of values, so that in the process of modeling, each feature is not treated differently.

For normalization, it can eliminate the dimension and speed up the convergence. Different features often have different dimensional units, which will affect the results of data analysis. To eliminate the dimensional impact between indicators, it is necessary to carry out data normalization processing to solve the comparability between data indicators. After the normalization of the original data, each index is in the decimal number between [0, 1], which is suitable for comprehensive comparative evaluation. But the disadvantage is that the abnormal data are very sensitive, and the outliers in the data will disappear after processing.

For standardization, turns the data into a standard normal distribution, even if it turns out to be some strange distribution (has abnormal data), is known by the central limit theorem, the amount of data is large enough to become normal.

I tried both data in Naïve Bayes, perceptron:

Using standardization in Naïve Bayes:
Its avg score is: 0.7824444444444444
Its avg test score is: 0.7879362371044443

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual 1	4328.222222	47.555556	58.666667	13.444444	18.111111
Actual 2	91.888889	3965.222222	3.666667	46.777778	294.444444
Actual 3	403.777778	58.444444	2153.333333	461.555556	1701.888889
Actual 4	0.000000	909.333333	109.888889	2829.111111	65.666667
Actual 5	0.000000	119.333333	23.888889	46.000000	3348.777778

Its avg f1 score is:
[0.93212487 0.83141642 0.60169083 0.77530752 0.74859927]

Using normalization in Naïve Bayes:
Its avg score is: 0.6843703703703703
Its avg test score is: 0.6758350843378569

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual 1	3200.888889	48.000000	146.222222	826.555556	244.333333

Actual 2	213.555556	4078.555556	0.000000	4.888889	105.000000
Actual 3	384.222222	582.333333	1002.222222	1557.888889	1252.333333
Actual 4	0.888889	1052.777778	32.555556	2820.777778	7.000000
Actual 5	308.666667	14.111111	27.555556	30.666667	3157.000000

Its avg f1 score is:
[0.74772973 0.80127398 0.33177281 0.61731469 0.76155014]

Using standardization in Perceptron:
Its avg score is: 0.8345185185185185
Its avg test score is: 0.9244855967078188

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual 1	2727.666667	0.000000	16.777778	0.111111	39.444444
Actual 2	3.111111	2526.444444	9.555556	22.777778	20.111111
Actual 3	14.555556	2.000000	2560.111111	64.777778	7.555556
Actual 4	36.111111	35.666667	205.666667	2057.555556	141.000000
Actual 5	19.666667	331.111111	17.666667	31.777778	2608.777778

Its avg f1 score is:
[0.97677538 0.92270914 0.93909618 0.88186541 0.89562948]

Using normalization in Perceptron:
Its avg score is: 0.708962962962963
Its avg test score is: 0.6225413526707427

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual 1	4152.555556	48.444444	47.333333	102.222222	115.444444
Actual 2	27.222222	4348.333333	8.111111	18.333333	0.000000
Actual 3	68.444444	880.000000	3097.555556	639.333333	93.666667
Actual 4	53.666667	2808.333333	101.444444	863.666667	86.888889
Actual 5	594.888889	2156.333333	92.666667	21.22222	672.888889

Its avg f1 score is:
[0.88769525 0.59690582 0.74095702 0.27169162 0.25202181]

(This result is from vscode. There are some difference when I use perceptron in vscode rather than I run it in jupyter before. In vscode, the terminal show: FutureWarning: max_iter and tol parameters have been added in <class 'sklearn.linear_model.perceptron.Perceptron'> in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3. "and default tol will be 1e-3." % type(self), FutureWarning.)

It is obvious to see that in both classifiers we tested, the data after standardization is better than normalization, especially in non-linear classifiers. From the confusion matrix and f1 score we know that there are some confusing data in class 3 and 4 which makes normalization unsuitable.

3.3. Feature dimensionality adjustment

From adjustment dimensionality, I tried PCA and K Best to reduce the dimensionalities. (“thanks to Weizhongjin / APS-Failure-at-Scania-Trucks-Dataset for the following code for function Feature_selection () [https://github.com/Weizhongjin/APS-Failure-at-Scania-Trucks-Dataset]”(knowing the dimensionality reduction except for PCA))The principle of PCA is that to reduce the data from n-dimension to k-dimension, we need to find k vectors to project the original data, which minimizes the projection error (projection distance). In this project, I choose the k value according to the formula so that the error is less than 0.01 (99% of the information is retained), so I set n_components = 0.99. K Best is easy to understand as choose kth Best feature. I use default k =10. In my assumption, PCA can get closer data from the original data and K BEST will ignore some important features.

I tried both data in Naïve Bayes:

Using PCA in Naïve Bayes:
Its avg score is: 0.748
Its avg test score is: 0.7069318714420377

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual 1	4003.888889	119.111111	294.000000	30.222222	18.777778

Actual 2	30.333333	4253.888889	7.111111	100.333333	10.333333
Actual 3	576.444444	213.111111	2429.555556	323.222222	1236.666667
Actual 4	0.000000	1047.666667	1617.222222	984.111111	265.000000
Actual 5	0.000000	196.555556	13.111111	84.222222	3244.111111

Its avg f1 score is:
[0.88254262 0.83282075 0.53083579 0.35319867 0.78233251]

Using K Best in Naïve Bayes:
Its avg score is: 0.7757037037037038
Its avg test score is: 0.7542642884602218

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual 1	4330.888889	47.444444	64.666667	9.777778	13.222222
Actual 2	28.222222	3195.666667	2.777778	16.222222	1159.111111
Actual 3	383.666667	16.333333	2128.111111	669.666667	1581.222222
Actual 4	0.000000	691.333333	28.444444	2828.888889	365.333333
Actual 5	0.000000	25.666667	27.222222	54.444444	3430.666667

Its avg f1 score is:
[0.94083529 0.76076764 0.60348531 0.75646759 0.68240837]

Although we can see that both data are similar to the original data(3.2 Using standardization in Naïve Bayes). From my point, the PCA does not ignore any important data after we have already the data first and I think any of these 13 features are important, so I just choose the PCA in the next steps.)

3.4. Dataset Usage

The data I use are all from the Hand Postures (from motion capture) Datasets. I use “D_train” as the training data set and use “D_test” as the testing dataset. Due to the similarity of each class, I keep the origin data at first.

“D_train” contains 13500 data points and “D_test" contain 21099 points. Because the data will not work well in its raw form, since the number of features varies from data point to data point. I firstly use calculate the

number of recorded markers, mean x of marker locations, mean y of marker locations, mean z of marker locations; the standard deviation of x of marker locations (similarly for y and z); maximum x of marker locations (similarly for y and z), and minimum x of marker locations (similarly for y and z) to extract 13 features both in "D_train" and "D_test". Then I use standardization to preprocess both "D_train" and "D_test" data sets. (Fit with "D_train" and transform both). Then use PCA to reduce both data sets' dimensionalities. Before the training, I split the "D_train" as a training set and validation set using leave-one-user-out cross validation before training each classifier, which is each validation set has all the data from just 1 user. I do leave-one-user-out 9 times to ensure each user in the training set can be the validation set once to prevent any overfitting for some particular splitting. Each time I have 12000 training data points, 1500 validation points.

After choosing each optimal parameter in each classifier, I use 21099 testing points in "D_test" to see the performance of each classifier.

3.5. Training and Classification

3.5.1 Perceptron:

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. In perceptron, we use the weight vector of the input data to several decision boundaries. Input data to adjust with certain weights until they satisfied for every decision boundaries in one entire iteration and halt at that time.

I set the random_state equals to np.random to shuffle the data each time. I iterate 50 times to get a high-performance score in both training accuracy and testing accuracy. Due to the shuffling, each time we run the perceptron function will get different best training accuracy and testing accuracy. In this time I get the result as:

Best perceptron test score is: 0.8139722261718565
Best perceptron validation score is: 0.8230370370370371

In the default setting of the perceptron, we get that:

Its avg score is: 0.8049629629629629
Its avg test score is: 0.8187170534675157

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual 1	4275.777778	48.000000	65.000000	0.666667	76.555556

Actual 2	77.111111	3942.888889	20.666667	23.444444	337.888889
Actual 3	127.555556	57.888889	3863.222222	495.111111	235.222222
Actual 4	0.000000	940.111111	479.333333	2164.888889	329.666667
Actual 5	2.555556	287.666667	44.888889	175.555556	3027.333333

Its avg f1 score is:
[0.95569407 0.81591097 0.8332878 0.6259346 0.80811542]

(This result is from vscode. There are some difference when I use perceptron in vscode rather than I run it in jupyter before. In vscode, the terminal show: FutureWarning: max_iter and tol parameters have been added in <class 'sklearn.linear_model.perceptron.Perceptron'> in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3. "and default tol will be 1e-3." % type(self), FutureWarning.)

3.5.2 Naïve Bayes:

Gaussian Naive Bayes algorithm for classification is a simple (“naïve”) classification method based on Bayes rule. The likelihood of the features is assumed to be Gaussian:

$$P(X_i|Y) = \frac{1}{\sqrt{2\pi\sigma_y^2}}\exp\left(-\frac{\left(x_i-\mu_y\right)^2}{2\sigma_y^2}\right)$$

The parameters σ_y and μ_y are estimated using maximum likelihood.
Because it is just a simple statistical classification using as a baseline. I just use the default parameter to train the classifier and get the results as:

Its avg score is: 0.7824444444444444

Its avg test score is: 0.7879362371044443

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual 1	4328.222222	47.555556	58.666667	13.444444	18.111111
Actual 2	91.888889	3965.222222	3.666667	46.777778	294.444444
Actual 3	403.777778	58.444444	2153.333333	461.555556	1701.888889
Actual 4	0.000000	909.333333	109.888889	2829.111111	65.666667

Actual 5	0.000000	119.333333	23.888889	46.000000	3348.777778

Its avg f1 score is:
[0.93212487 0.83141642 0.60169083 0.77530752 0.74859927]

3.5.3 SVM (Support Vector machine):

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression, and outlier detection. Using SVM is to move the hyperplane to both sides as far as possible. It is useful in high dimensions. To get the best result, I need to find the best C in each SVM (linear, poly, RBF (radial basis function kernel), sigmoid). C is the regularization parameter. The strength of the regularization is inversely proportional to C. I tried to figure out which kernel and C is most suitable for our data points.

I divide C ∈ $[10^{-3}, 10^{3}]$ in 50 numbers in linear and sigmoid kernel and I divide C ∈ $[10^{-1.5}, 10^{1.5}]$ in 30 numbers in poly and RBF kernel by using function logspace() (change this logspace() function by hand). I set a 50*1 array or 30*1 array for each kernel to store each C I generated. Then use amax() function to find the maximum validation accuracy and use argmax() to find its representative index. Then use C[max_index] to find its C.

Using the linear kernel (with 50 iteration in logspace(-3,3,50)):

Best acc of linear SVM in validation is 0.876074074074074 with C: 0.012648552168552958

Its avg test score is: 0.8307766034198567

	Predict1	Predict3	Predict3	Predict4	Predict5
Actual 1	4338.777778	48.000000	49.111111	0.333333	29.777778
Actual 2	42.555556	4040.222222	88.555556	9.111111	221.555556
Actual 3	202.222222	6.777778	3758.111111	318.333333	493.555556
Actual 4	0.000000	244.222222	911.000000	2058.777778	700.000000
Actual 5	1.444444	111.222222	16.222222	76.444444	3332.666667

Its avg f1 score is:
[0.95889658 0.91255211 0.78210943 0.64062279 0.8046329]

Using the polynomial kernel (with 30 iteration in logspace(-1.5,1.5,30)):

Best acc of poly SVM in validation is 0.9103703703703704 with C: 0.10404983103657856

Its avg test score is: 0.921312753105729

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual 1	4340.777778	48.000000	3.777778	0.222222	73.222222
Actual 2	45.777778	3963.111111	74.555556	0.000000	318.555556
Actual 3	283.111111	0.111111	4361.000000	107.222222	27.555556
Actual 4	0.000000	85.000000	230.111111	3582.333333	16.555556
Actual 5	21.777778	221.333333	23.111111	80.222222	3191.555556

Its avg f1 score is:

[0.94852045 0.90845508 0.92076824 0.92815583 0.8924078]

Using the RBF kernel (with 30 iteration in logspace(-1.5,1.5,30)):

Best acc of RBF SVM in validation is 0.8989629629629629 with C: 0.13203517797162953

Its avg test score is: 0.8854711387058891

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual 1	4084.222222	48.000000	212.000000	49.555556	72.222222
Actual 2	52.666667	3739.444444	19.777778	13.777778	576.333333
Actual 3	67.777778	18.000000	4449.111111	111.555556	132.555556
Actual 4	0.000000	233.444444	16.000000	3008.111111	656.444444
Actual 5	0.111111	61.666667	10.222222	64.333333	3401.666667

Its avg f1 score is:

[0.94207177 0.8796894 0.93719243 0.83994193 0.81403303]

Using the sigmoid kernel (with 50 iteration in logspace(-3,3,50)):

Best acc of sigmoid SVM in validation is 0.7781481481481483 with C: 0.009540954763499945

Its avg test score is: 0.7052730250512136

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual 1	4252.888889	48.000000	24.777778	0.111111	140.222222

Actual 2	31.000000	4075.666667	201.666667	61.666667	32.000000
Actual 3	1670.666667	1165.111111	858.666667	232.111111	852.444444
Actual 4	0.000000	1099.777778	241.666667	2570.000000	2.555556
Actual 5	2.888889	296.000000	65.666667	50.111111	3123.333333

Its avg f1 score is:
[0.81929611 0.73618415 0.2665526 0.74394718 0.81269262]

3.5.4 KNN (K nearest neighbors):

In the KNN algorithm, we need to use k nearest neighbors to classified the output label. It is easy to see that if we use a very little k to classified the trained accuracy will be extremely high and the classifier may be overfitted. It is obvious that if we do not use cross validation, the training accuracy will be 1.0 if the k is 1 (just find themselves).

Thanks for using leave-one-user out cross validation, I can get a suitable k.

Best KNN validation score is: 0.8064444444444444 with 5 parameters.
Its avg test score is: 0.7739387332732989

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual 1	4269.666667	42.777778	58.333333	8.555556	86.666667
Actual 2	91.666667	3797.333333	5.666667	0.000000	507.333333
Actual 3	44.666667	13.555556	3084.333333	1168.444444	468.000000
Actual 4	0.000000	182.111111	1747.666667	1762.222222	222.000000
Actual 5	0.888889	56.222222	20.333333	44.777778	3415.777778

Its avg f1 score is:
[0.96261665 0.89376698 0.63228154 0.51021219 0.83292835]

3.5.5 Neural Network (Multilayer Perceptron classifier):

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function f() from $R^n \rightarrow R^0$ with multiple layers. It can learn nonlinear function approximators for classification or regression. Unlike Logistic regression, there can be one or more non-linear layers between the input

layer and the output layer, called hidden layers. From my data set I just set each hidden layer is 100, which means I will have about 200 layers to fit my data. I think it reasonable because it's neither too overfitted nor too close to linear (because the data is not good in one perceptron I mentioned before). The result is :

Its avg score is: 0.9199259259259259

Its avg test score is: 0.8291124908500139

	Predict1	Predict2	Predict3	Predict4	Predict5
Actual1	4256.333333	51.222222	69.111111	79.111111	10.222222
Actual2	33.555556	3059.111111	82.777778	0.000000	1226.555556
Actual3	158.888889	3.222222	3670.111111	810.000000	136.777778
Actual4	0.000000	1.555556	1.222222	3085.444444	825.777778
Actual5	31.444444	28.555556	9.333333	46.222222	3422.444444

It's avg f1 score is:
[0.95161174 0.8027429 0.85046518 0.77481607 0.75596709]

4. Analysis: Comparison of Results, Interpretation

Accuracy for each classifier

	Perceptron	Naïve Bayes	SVM	KNN(5NN)	MLP
ACCURACY	0.823	0.787	0.921	0.773	0.829

For each method's accuracy:
SVM>MLP>perceptron>Naïve Bayes >KNN

I thought that the reason of KNN is the worst is because of overfitted (cause k just ch oose 5). So I tried the whole k in range 50. Then I got the best test performance is th at accuracy equals to 0.8462907668083268 (k=43).

So the final performance for each classifier is SVM>43NN>MLP>perceptron>Naïve Bayes>5NN due to accuracy. For a reasonable parameter, all the classifier is greater than Naïve Bayes (Setting as baseline).

Also, we can notice that from the previous data, both nonlinear kernel SVM and MLP can perform better than the linear SVM and perceptron. It means that these data set is more suitable for nonlinear classifiers.

F1 scores for each classifier

	Class1	Class2	Class3	Class4	Class5
Perceptron	0. 88886466	0. 79022257	0. 76234961	0. 61880032	0. 68504888
Naïve Bayes	0.93212487	0.83141642	0.60169083	0.77530752	0.74859927
Linear SVM	0.95889658	0.91255211	0.78210943	0.64062279	0.8046329
Poly SVM	0.94852045	0.908455 08	0.92076824	0.92815583	0.8924078
KNN	0.962616 65	0.89376698	0.63228154	0.51021219	0.832928 35
MLF	0.951611 74	0.8027429	0.85046518	0.774816 07	0.75596 709

From each confusion matrix and f1 score in 3.5, we can notice that the class 3 and class 4 have a low f1 score and confused in the confusion matrix. Class 1, 2, and 5 have a good performance in each either linear or nonlinear classifiers. In the above table, we can see that poly SVM can deal with each class very well, which makes it become the best classifier.

I make a random classifier and its accuracy is 0.20749798568652542, which is similar to 1/5. Obviously, any of my classifiers can perform much better than the random classifier. From this, I know that my classifier worked and perform very well.

5. Summary and conclusions

In this project, I know how to deal with a huge number of data points and how to make data reasonable from the raw data.
From preprocessing, I understand how to deal with the unlabeled (no significance data). From cleaning this unlabeled data, I extract the data number as a parameter (Before rescaling do this may not get the best data, But I think I need to deal with the blank part first.) I know we rescale the data is for that the multi-dimensional features will have similar scales, which will help the gradient descent algorithm converge faster.
For dimensionality adjustment, I know how to extract the data and make each new feature reasonable, how to reduce the dimensionalities.

In classifier selection, I learned to use cross validation to minimize the overfitting problem. Comparing the accuracy, I know how to compare with each classifier, and the confusion matrix helped me to find a more detailed problem. From f1 score I know how to process the data to make the accuracy better.
I also met some problems, e.g. the sequence of the data processing as mentioned blow. I may tried do scaling first in the future work. Furthermore, I need to try the more parameter in each classifier. Because this data set points recorded for a given user are likely highly correlated. So I do not use K-Fold cross validation but use leave-one-user out cross validation. And I find a RandomizedSearchCV() function in skelearn.model_selection which can help me to find the best parameter in random N iteration. In future work, I try to understand how to use this function in leave-one-user out or another cross validation rather than K-Fold cross validation.

References

[1] Hoffmann, Heiko. "Kernel PCA for novelty detection." *Pattern recognition* 40.3 (2007): 863-874.

[2] Schuldt, Christian, Ivan Laptev, and Barbara Caputo. "Recognizing human actions: a local SVM approach." *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004*. Vol. 3. IEEE, 2004.

[3] Bezdek, James C., Siew K. Chuah, and David Leep. "Generalized k-nearest neighbor rules." *Fuzzy Sets and Systems* 18.3 (1986): 237-256.

[4] Chiddarwar, S. S., & Babu, N. R. (2010). Comparison of RBF and MLP neural networks to solve an inverse kinematic problem for 6R serial robot by a fusion approach. *Engineering applications of artificial intelligence*, *23*(7), 1083-1092.