EE559 Final Project

Data Set(s): Hand Postures

Jingquan Yan, jingquan@usc.edu, ID:1071912676

5/8/2020

1. Abstract

We consider a data set that contains coordinate captures from different people with different motions. Our goal is to explore and find the pattern between those collected coordinates and hand motion class.

To construct the machine learning pipeline, we are going to find the best components and parameters by trying all different pairs of preprocessing, dimensional adjustment and classifiers.

In the end, we will do a comparison on the result and make our comment on some sapient results.

2. Introduction

2.1. Problem Statement and Goals

There are 5 classes of hand postures from 12 different users were recorded using unlabeled markers. Our goal is to explore and find the pattern between those collected coordinates and hand motion class.

2.2. Literature Review (Optional)

No literature reference / prior knowledge to this dataset.

3. Approach and Implementation

There are three components in our machine learning pipeline: preprocessing, feature extraction and dimensional adjustment. We will going to discuss all the possible options for each step and determine what we are going to work with.

3.1. Preprocessing

Preprocessing is always helpful to enhance the classification performance and sometimes even crucial to the classifier. In this part, we have two options: standardization and normalization. The reason why we are not going to "Impute Missing Data" is that interpolation methods handle the data in columns and since the raw data's columns are corrupted (randomly shuffled), it is meaningless to interpolate the missing values in the raw data.

Since we decide to employ 4 different classifiers, which are K-NN, Naïve Bayes, SVM and Decision Tree, we must evaluate how would the two preprocessing methods affect the classifiers above.

a) SVM requires standardization because this will accelerate the convergence when solving the optimization problem with Stochastic Gradient Decent method.

b) K-NN requires standardization because this model need to find the nearest neighbors base on distance. Distance in all dimensions should be standardized to avoid the larger scale features to dominant the smaller ones.

c) Naïve Bayes don't necessarily require standardization because it only consider the conditional and the prior probability which have nothing to do with the scale of data.

d) Decision Tree separate the feature space independently and greedily. Since the partition in each dimension is independent, the decision tree does not necessarily need standardization.

Here I take K-NN as an example to illustrate how data scaling difference may affect some classifiers. First, we use GridsearchCV to find the best value of k and it turns out to be 5 among k=1-7. Then we have the comparison between using the standardization or not.

K-NN Accuracy	With standardization	Without standardization
K=5	82.99%	72.71%

In addition, we have to consider how centralization may affect PCA process. As we know, the original covariance matrix and the covariance matrix after standardization are the same, in other word, same eigenvector decomposition. We will figure out if whether standardization will affect PCA results. To illustrate this, given the training dataset, we first calculate the first 6 largest PCA components without standardization:

```
PCA first 6 largest components without standardization
[0.43066192 0.25323894 0.09839378 0.08989567 0.06628123 0.03661315]
The first 6 components adds up to: 0.9750846922268127
```

Then, we do PCA after standardization on the training set, the result is as follow:

```
PCA first 6 largest components with standardization
[0.34190993 0.2646643  0.11043225 0.09605038 0.06165411 0.05276476]
The first 6 components adds up to: 0.9274757298930398
```

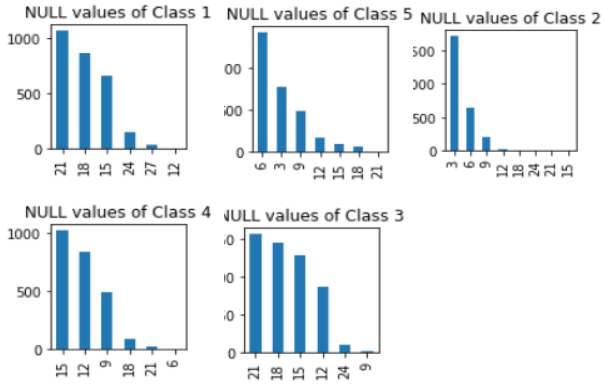
Now we can say, standardization may be helpful to make the PCA variance ratio more even, that is, not concentrating in some individual components and the larger components are less likely to dominant the smaller ones. So it would be slightly better to employ standardization together with PCA. However, this is only one specific case and we have to test the performance of PCA with standardization or not in other situations.

3.2. Feature engineering (if applicable)

Since the original data are unlabeled, we have to extract and construct new features for further classification.

As normal, we calculate the mean value, standard deviation, maximum value and minimum value of X, Y and Z axis respectively, which adds up to 12 features.

In addition, noticing there are some values left blank due to the overlapping when capturing the motions, I assume that the number of NULL values in the sample could be somehow related to the motion classes. So, I plotted the histogram of NULL values in each class and see if there do exist some correlations:



There do exist! For instance, we can notice that the NULL values in class 2 are much less than class 1 and 3. Also, the NULL values in class 3 are more evenly distributed than other classes. So I extract the number of NULL values in a given sample as a feature and add it to the constructed feature space.

Now we have 13 features (ignore the feature 'User') as listed:

Feature 1	X_mean	Feature 2	Y_mean
Feature 3	Z_mean	Feature 4	X_std
Feature 5	Y_std	Feature 6	Z_std
Feature 7	X_min	Feature 8	Y_min

Feature 9	Z_min	Feature 10	X_max
Feature 11	Y_max	Feature 12	Z_max
Feature 13	Num_NULL		

3.3. Feature dimensionality adjustment

We have three options to adjust the feature dimensionality according to what we have learned this semester: feature's correlation, PCA and LDA. This time I will try all these three methods elaborately and compare how these options are going to affect on the results with standardization preprocessing and SVM classifier at the end of this chapter. We will take both accuracy and average fitting time into account as the "benchmark".

a) Baseline

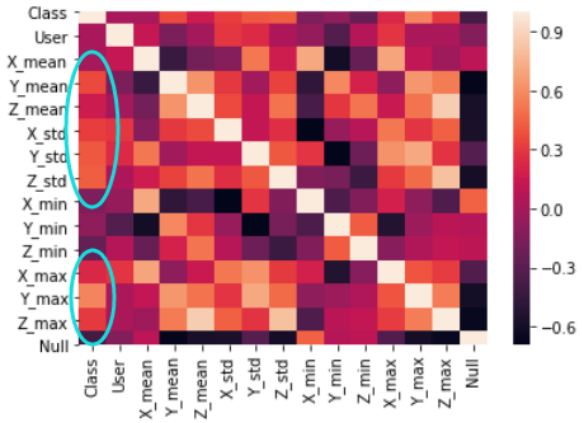
We are only going to use standardization and SVM (rbf kernel) without any dimensional adjustment as the baseline.

```
Best accuracy rate: 0.9247407407407408
Best parameter set: {'svm__C': 1, 'svm__gamma': 0.01}
Average time to fit: 3.456 seconds
```

We can see the accuracy is quite good without any preprocessing, nevertheless the average fitting time is a little long.

b) Correlation Criteria

In correlation criteria, we are going to brutally choose the several features that are most relevant to the given class. First, let's see the covariance matrix between each columns:



There indeed exists several features that are somehow relevant to the "class" feature and now we extract the most relevant 6 features:

```
res.corr()['Class'].abs().sort_values(ascending=False)[:7]
```

```
Class     1.000000
Y_max     0.567533
Z_std     0.428338
Y_std     0.408427
Null      0.389059
Y_mean    0.356043
X_std     0.316704
Name: Class, dtype: float64
```

Then, we extract those most relevant features, construct a new data frame and feed them into the pipeline which includes standardization and SVM (rbf kernel) parts. The best accuracy and fitting time is:

```
Best accuracy rate: 0.8918518518518519
Best parameter set: {'svm__C': 10, 'svm__gamma': 0.005}
Average time to fit: 1.906 seconds
```

We notice that the accuracy is acceptable when we abandoned several irrelevant redundant features. In addition, with the reduction of the dimension, the time it takes to fit the model is much faster than the original feature space.

c) PCA (Principal Components Analysis)

PCA's main idea is to project multiple correlated features to a less correlated coordinate and the new projected, less relevant features are called the principal components. In other words, PCA recognizes the latent pattern in the dataset to construct new features, maximize the data deviation.

```
pca = PCA()
pca.fit(res.iloc[:,2:])
print("The 6 components ratio that adds up to:", pca.explained_variance_ratio_[:6].sum().round(4))
```

```
The 6 components ratio that adds up to: 0.9775
```

We find out that only 6 dimensions after PCA can cover more than 95% of the original dataset's information. Then, lets plug this new dimension into the pipeline and the classifier benchmark is:

```
Best Accuracy: 0.8412592592592593
Best Parameters: {'preprocessing__pca__n_components': 6, 'svm__C': 50, 'svm__gamma': 0.001}
Average Time to Fit (s):2.556
Average Time to Score (s):0.209
```

The fitting time gets faster but the accuracy seems didn't live up to expectation. The accuracy is even worse than classification without any preprocessing. Here I explored potential reason why lead to this result and I highly suspect that the highly non-linearity of the dataset could lead to this negative affect. Because PCA (or SVD) is linear transformation and can be represented as shifting and rotation. Only when the dataset has a strong linearity would the PCA work well and if it's non-linear, the transformation has to increase the dimension (using kernel) instead of reducing the dimension.

d) LDA (Linear Discriminant Analysis)

Compared with PCA, instead of concentrating on the feature covariance matrix, LDA optimizes the low dimension space to get the best separability. LDA is helpful to prevent the Curse of Dimensionality from happening and reduce the computation costs.

The classifier benchmark with components LDA, standardization and SVM (rbf kernel) is as follows:

```
Best Accuracy: 0.8917777777777778
Best Parameters: {'preprocessing__lda__n_components': 4, 'svm__C': 50, 'svm__gamma': 0.01}
Average Time to Fit (s):2.255
Average Time to Score (s):0.145
```

Not a bad performance with much less fitting time. This give me a inspiration to concatenate PCA with LDA together and we'll see how is the performance.

e) PCA + LDA

```
Best Accuracy: 0.8843703703703703
Best Parameters: {'preprocessing__lda__n_components': 4, 'preprocessing__pca__n_components': 6, 'svm__C': 100, 'svm
__gamma': 0.001}
Average Time to Fit (s):2.246
Average Time to Score (s):0.202
```

The result is between using PCA and LDA individually.

f) Conclusion

	Without preprocessing	Covariance	PCA	LDA	PCA+LDA
Accuracy	92.47%	89.19%	84.12%	89.18%	88.44%
Fit Time	3.456 sec	1.906 sec	2.09 sec	2.255 sec	2.246 sec

All these preprocessing have a positive effect on reducing the model fitting time at the cost of slight decreasing in accuracy. In my opinion, the reason why these dimensional processes don't improve the accuracy is the non-linearity of the dataset. So, I decide not to use any of these dimensional adjustments in later classification.

3.4. Dataset Usage

a) Preprocessing

From last chapter, we have discussed whether to use the standardization for different classifiers and compared the performance of using three various dimensional adjustments as well. In light of the LDA may increase the fitting efficiency at the cost of a slight accuracy drop, I decide to try standardization with LDA or not:

K-NN	standardization	Standardization + LDA

Naïve Bayes	standardization	Standardization + LDA
SVM	standardization	Standardization + LDA
Decision Tree	None (not sensible to distance)	Only LDA

For convenience and formalization, I'm going to use Pipeline to encapsulate all this procedures into one pipeline and assemble several steps that can be cross-validated together while setting different parameters.

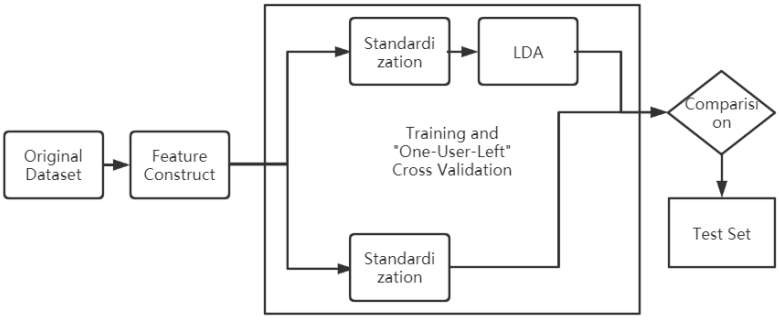
Sequentially apply a list of transforms and a final estimator. Intermediate steps of the pipeline must be 'transforms', that is, they must implement fit and transform methods. The final estimator only needs to implement fit.

Also, for LDA and SVM parameter selection, we import GridSearchCV from sklearn. GridsearchCV function can traverse all possible pairs of given parameters and use cross-validation to figure out the best performance one. In addition, GridSearchCV support customizing cross-validation sets so we can implement a cross-validation method to left one user data at each validation process.

b) Data Usage

I will use the "D_train.csv" as training set and further test on the "D_test.csv".

The "D_train.csv" dataset includes 13500 rows. I pick all of them and constructed each row to 13 features introduced in the last chapter. Then use 4 different classifier to predict with "one-user-left"cross-validation and find the best model. The whole procedure can be modeled as this flow chart:



The "D_test.csv" dataset includes 21100 samples from exactly different users from the "D_train.csv" thus I'm going to use this "D_test.csv" to test the generalization ability of my classifier.

3.5. Training and Classification

a) Classifiers

i. K-NN

K-NN is a basic method for classification and regression. For classification case, the inputs are vectors of instances / points in feature space and the outputs are their classes. Basically, K-NN algorithm separates the feature space according to the training data and classify each point by its k nearest neighbors.

Normally, sklearn K-NN chooses the data structure either KD-tree or Ball-tree to find the k neighbors. Take KD-tree as an instance, first sort the value in one dimension and usually picks the median as the root. Then, recursively separate the dimension space into two subspaces by the median of that dimension until there's no node left in the subspaces. All the algorithms and formulas I use are given in the EE599 lecture.

ii. Naïve Bayes

Naïve Bayes learning process is as follows:

First, given training data, learn the joint distribution of both input and output based on the conditional independent assumption. Then, given an input **x**, use Bayes Theorem to calculate the **y** with the maximum posterior probability:

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

All the algorithms and formulas I use are included in the EE599 lecture.

iii. SVM

The SVM is a model that classify with the largest margin and it is also equivalent to the minimization of the regularized hinge loss function problem. By introducing the kernel function and soft margin, one can use SVM to classify those non-linearly separable data. All the algorithms and formulas I use are given in the EE599 lecture.

iv. Decision Tree

Decision tree is a tree method for classification and regression. It can be considered as a set of "if-then", or as a conditional distribution based on the feature space.The criteria of decision tree is entropy or gini index.

Decision tree recursively generate decision trees until there's no more separable features. These kinds of trees are always precision for training data classification but performs worse in test data, in other words, overfits.

So decision tree need to regularization, add the depth of tree as the penalty to loss function. This is the regularized loss function of decision tree:

$$C_\alpha(T) = \sum_{i=1}^{|T|} N_t H_t(T) + \alpha|T|$$

We will use GridSearchCV to brutally find the best value of max_depth of decision tree.

b) Parameters

Since we are using GridSearchCV to find best parameters with cross-validation, we have to feed in the choices of all possible parameters and let the GridSearch to find the best pairs of choices.

StandardScaler():

Parameter	Possible Values	Note
with_std	True, False	Devide std or note
with_mean	True, False	Minus mean or not

K-NN:

Parameter	Possible Values	Note
n_neighbors	1, 2, 3, 4, 5, 6, 7	Number of neighbors

SVM:

Parameter	Possible Values	Note
kernel	linear, rbf	Kernel function
C	0.1,0.5,1,10,50,100,500,1000	C value
gamma	0.001,0.005,0.01,0.05,0.1,0.5,1,5,10,50	Gamma value

Decision Tree:

Parameter	Possible Values	Note
Max_depth	None, 1, 3, 5, 7	None = No limitation

c) Cross-Validation

In GridSearchCV, we can customize the cross-validation method by importing the PredefinedSplit() function. PredefinedSplit() provides train/test indices to split data into train/test sets using a predefined scheme specified by the user with the test_fold parameter.

For instance, [0, 0, 1, 2, -1] means that data with index 0 and 1 are used as validation set in the first round. Data with index 2 will be used as validation

set in the second round and index 3 used for third round. -1 stands for never be chosen for validation.

Since there are 9 unique users and according to "One-User-Left", we will have 9 iterations and then calculate the mean accuracy and mean fitting time by calling the attribute best_score_ of GridSearchCV and cv_results_.

d) Classification Results

The accuracy on training set (the screenshots are in zip with raw code):

	With/Without LDA	Accuracy	Fitting Time
K-NN	With	93.24%	0.67 sec
	Without	89.39%	0.38 sec
Naïve Bayes	With	92.30%	0.58 sec
	Without	84.50%	0.26 sec
SVM	With	95.57%	2.271 sec
	Without	98.87%	3.543 sec
Decision Tree	With	88.73%	0.99 sec
	Without	86.26%	1.41 sec

We figure out that the best model on training set is SVM without LDA. However its fitting time is the worst. Now we will plug in the parameters and use SVM without LDA to make a classification on the test set, The result is:

```
The confusion matrix is:
 [[4281   48    8    0  129]
 [  96 4252    8    0   46]
 [ 681   61 3782  236   19]
 [   0 1060    0 2851    3]
 [   0   69   10   23 3436]]
The accuracy on the test set is: 0.8816531589174842
```

The result is pretty good because as given that the motions are related to the users, the test set and the training set have definitely different users, and this result indicates that our model is not getting overfitting and have pretty good generalization ability.

4. Analysis: Comparison of Results, Interpretation

4.1. Reasoning for approach

As explained in chapter 3, the reason for choosing preprocessing depends on the individual property of the classifiers. For all 4 classifier options, SVM, K-NN are sensitive to distance and on contrary, Naïve Bayes and decision tree are not sensitive. So we would apply standardization on all models to be fair.

Also as explained in chapter 3.3, I have 4 options for dimensional adjustment. I brutally examined the performance of all options on a SVM without any parameter optimization and obtain the result as follows:

	Without feature adjust	Covariance	PCA	LDA	PCA+LDA
Accuracy	92.47%	89.19%	84.12%	89.18%	88.44%
Fit Time	3.456 sec	1.906 sec	2.09 sec	2.255 sec	2.246 sec

It seems that without feature adjustment has the most accuracy and since there's no dimension reduction, it has the longest fitting time as well. So I decided to add LDA as an preprocessing option for classification.

4.2. Result analysis and comparison

As shown in last chapter, we have the accuracy, fitting time with all possible classifiers on the training set cross-validation:

	With/Without LDA	Accuracy	Fitting Time
K-NN	With	93.24%	0.67 sec
	Without	89.39%	0.38 sec
Naïve Bayes	With	92.30%	0.58 sec
	Without	84.50%	0.26 sec
SVM	With	95.57%	2.271 sec
	Without	98.87%	3.543 sec
Decision Tree	With	88.73%	0.99 sec
	Without	86.26%	1.41 sec

Also, we have the confusion matrix and accuracy on the test data:

```
The confusion matrix is:
 [[4281   48    8    0  129]
 [  96 4252    8    0   46]
 [ 681   61 3782  236   19]
 [   0 1060    0 2851    3]
 [   0   69   10   23 3436]]
The accuracy on the test set is: 0.8816531589174842
```

i. Overall, the SVM performs the best, but at the cost of long fitting time. K-NN performs above average and runs faster than SVM. Naïve Bayes and Decision Trees' accuracy doesn't live up to expectation but both run relatively fast than SVM.

ii. In addition, we can observer how LDA affect on these models: LDA helps Decision Tree, Naïve Bayes and K-NN to get higher accuracy, and helps SVM, Decision Tree to get faster runtime. So it is quite empirical to decide whether using an dimensional adjustment or not because you can't know how the

linearity would be in advance, especially when the feature space is in high dimension.

iii. Normally compared with PCA, instead of concentrating on the feature covariance matrix, LDA optimizes the low dimension space to get the best separability and this should have been helpful for SVM. However, the result tells that LDA has negative effect on SVM classifier and has positive effect on the rest classifiers. This is somehow confusing to me.

4.3. Comments based on result

The reason why SVM takes such long time may be answered by its documentation[1]:

"The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using sklearn.svm.LinearSVC or sklearn.linear_model. SGDClassifier instead, possibly after a sklearn.kernel_approximation.Nystroem transformer."

As I switched to SGDClassifier, the result is as follows and the fitting speed really shortened instead of using SVC classifier:

SVM	With LDA	95.18%	2.037 sec
	Without LDA	97.91%	3.124 sec

Also, the bad performance of classification when using PCA recalls me that the data in this project could be nonlinear or lack in correlation. I suppose that employ Neural Networks to train these nonlinear data sets may have better performance due to the "Universal Approximation Theorem".

5. Contributions of each team member

I'm "one-man army".

6. Summary and conclusions

For this particular hand motion dataset, we find that the "pure" SVM works best with the training dataset among K-NN, Naïve Bayes and decision tree using cross-validation. The accuracy on training dataset is over 98% and 89% on the test set.

We also notice that PCA works bad with this dataset and LDA's performance depends on the classifier you choose.

References

[1] " sklearn.svm.SVC" [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html