EE559 Project

Pattern Recognition System Development About Online News Popularity

Wang, Xin
USC ID : 8005661207

April 28, 2018

wang631@usc.edu

Abstract

The goal of this project is to develop a pattern recognition system using a collected dataset from the Mashable website to predict online news popularity (Fernandes, Vinagre, & Cortez, 2015).

In the dataset, there have totally nearly 5000 articles with 58 features. In preprocessing step, I first clean data through drop NaN, duplicated, missing data, invalid features and etc., and then classify raw data into 2 classes based on the value of "shares" column. If an article is shared more than 1300 times, it's labeled class 1, that is, popular; otherwise, it's unpopular. Finally, I split total data to training set and test set, which proportion is 70%(3467)/30%(1487), and then standardize them by robust scaler for outliers less influencing data.

Secondly, in the feature selection part, I mainly use PCA to extract features. First, I check scree plot while counting the number of singular value(S) >= 1 items, which has 51 items, and I keep them as a PCA's input parameter, number components; after PCA fitting by training data has finished, test data are transformed in the new feature space. In contrast, I also try feature extraction by LDA and TSNE, because feature dimension is so high, TSNE is not expected, so I end up with PCA and view the percentage of variance explained by each of the selected components, 34 features contribute above 99%; since training data are large enough, I continue to keep 51 features for the following experiment so that there have much more information preserved.

Next, in model selection and evaluation, I first select NaiveBays as baseline because it's simple, and begin with distribution-free classifiers like linear SVM, RBF SVM and Perceptron; and also try some statistical classifiers such as KNeighbors, RandForest and NaiveBayes with initial parameters based on previous experience; here cross validation is performed with KFold = 5 cross validation to validate and evaluate performance among these classifiers. The grid search cross validation is used to find the best hyper-parameters, mainly including SVM's C and gamma trade off; number of neighbors of KNN; and number of trees of Random Forest. During the search, a random 5-Fold splitter divides training data into training set (80%=2772) and validation set (20%=695). Besides, I especially emphasize on RBF SVM and first use validation curve to check training validation score and cross validation(CV) score changing trend along with different gamma value, because gamma has a larger influence about RBF SVM performance, through the step, I have a better way to get a reasonable value to avoid overfitting, meanwhile, combined with validation curve result, I can give more elaborate range to the following grid service estimator. From parameter estimation result, gamma=0.01 and C=100 are the best choice to RBF SVM; C=100 to linear SVM, number of trees to random forest is 500, number of neighbors to KNN is 20 are the best parameters to these models on the training data set.

In the end, the models are fit to all training data once the best parameters are selected, and then the models predict remained 30% test set, and from AUC, I get the best model is linear SVM classifier.

Key Words

PCA CV KNN SVM AUC K-Fold RBF

Introduction

The project target is to predict if an article is popular, and the original dataset provides several attributes published by Mashable (Fernandes, Vinagre, & Cortez, 2015), The labels (popular/ not popular) are based on the number of shares that article received. Original link (for information about the data): https://archive.ics.uci.edu/ml/datasets/online+news+popularity#. For a 2-class problem, I define "popular" as shares > 1600; "not popular" if shares ≤ 1600 in order to get a roughly balanced dataset.

"The overarching goal and approach in pattern classification is to hypothesize the class of models, process the sensed data to eliminate noise (not due to the models), and for any sensed pattern choose the model that corresponds best. (Duda, Hart, & Stock, 2000)". Following the overarching goad, the project mainly consists of four stages: data cleaning to eliminate inference; feature selection; model selection and final prediction on test set.

The next report goes forward based on the order one by one. At first, I introduce the pre-processing step including cleaning ineligible data, distinguishing features and target, labeling data to 2 classes and splitting training set and test set for the next requirement. Then, I mainly present feature extraction to simplify it while keeping much enough information, and map test set into the new fitted feature space. The third part occupies the largest proportion of the report, which consists of model comparison and parameter estimation, in each part, I separately implement several major distributed free classifiers and statistical classifiers, and give the details of each classifier with the experimental results. Lastly, a comparative analysis is given based on predicted results of test set using different classifiers with the best parameters.

Language, Toolbox and Libs

The whole project is implemented using Python 3.6 and sklearn pattern recognition toolboxes, pandas and numpy package to operating data as well as matplotlib to plot diagrams.

sklearn :

 # pre-processing and feature extraction
from sklearn.preprocessing import StandardScaler : standardize data set
from sklearn.preprocessing import RobustScaler : standardize data set and make data as much linearly independent as possible
from sklearn.decomposition import PCA : feature extraction
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.manifold import TSNE
# model selection and parameter estimation
from sklearn.learning_curve import validation_curve : plotting validation curve to analyze gamma of SVM classifier
from sklearn.model_selection import GridSearchCV : grid search cross validation method
from sklearn.model_selection import train_test_split : split train set, validation set

and test set
from sklearn.model_selection import cross_val_score : cross validation score
from sklearn.model_selection import StratifiedKFold : k-fold for cross validation
 # classifiers
from sklearn.neighbors import KNeighborsClassifier : KNN
from sklearn.naive_bayes import GaussianNB : Naïve Bayes
from sklearn.svm import SVC : two SVC separately linear and rbf
from sklearn.linear_model import Perceptron : Perceptron
from sklearn.ensemble import RandomForestClassifier : Random Forest
 # performance statistics
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report

pandas and numpy :

import numpy as np
import pandas as pd

matplotlib :

import matplotlib

Dataset Pre-processing

The step is to choose a proper dataset to perform the classification. In order to do it, pandas is a good choice to handle csv parsing. I first select load Online New Popularity data set and check feature types, only url is a categorical feature, which is like the index excluding any available effect to the next process, therefore, the column is dropped. next I carefully check each column, and find some features like weekday, which has already been divided to multiply columns and expressed by binary value. Therefore, at the end of the step, only numerical features are left. Furthermore, "shared" column is as target data, and then I traverse each column to check if there have invalid features, e.g. all elements are the same value such as "kw_min_min", and directly remove these features. After the step, totally 57 features are left.

Secondly, I need to check each row if there exist any missing data, duplicated data and drop these items because data size is large enough and most of them look like normal, actually when I finish the above step, the number of samples does not decrease, which identify the data set is clean before and totally 4954 data samples.

In the third step, labelling the data to two class, it's a popular article if shared value is equal or large than 1300times, otherwise, it's unpopular. This split keeps two classes are almost balanced in the whole samples,

Balanced :
1 2581 52%
0 2373 48%

Finally, I decide to normalize the samples in order to reflect the real influence of each feature because the ranges of different features vary so much, besides, outliers influence also would be reduced through standardization. First samples are **randomly** split

to two parts, and train_test_split() function can guarantee balanced selection from two classes. Training data occupy 70% and test data are 30% based on experience.

Name: shares, class distribution:

train:		test:	
class	#	class	#
1	1789	1	792
0	1678	0	695
total	3467	total	1487

Then I choose robust scaler instead of common standard scaler because it is not influenced by a few number of very large marginal outliers (Compare the effect of different scalers on data with outliers), and fit it with training data, and at last transform the whole samples.

Feature Extraction

In the project, only feature extraction is used, and feature expansion is not necessary, because we normally prefer to learn in the over-determined realm and in the condition, the performance is commonly fine.

Before selecting a method to reduce feature space, I begin with SVD decomposition to determine how many components I decide to maintain is a reasonable choice, figure 1 is the result in scree plot, when dimensions increase to 30, eigenvalues almost decrease to zero. Then I select 30 as the next number of components.

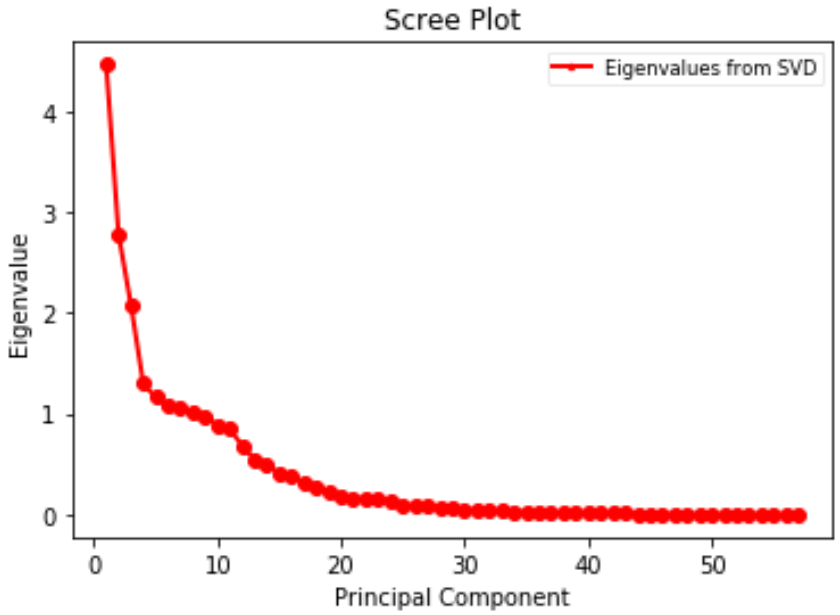

Figure 1: Eigenvalue and number trend

Here I researched PCA, Fisher's-LDA, and TSNE. Principal Component Analysis(PCA) can further make some assumption on the linear independence of the features, because data are normalized by the robust scatter, PCA is likely to be a feasible choice, and to PCA with whiten=True, I can further remove the linear correlation across features. LDA tries to find an axis that projecting thereon should maximize the value J(w) [2]. TSNE reflects better, but it only accepts a few dimensions. In this project, I implement PCA and LDA

Both PCA and FLD was implemented using the Classification toolbox functions.

the parameter of both of them is nComponents, just as mentioned before, I select 30, the final accuracy scores are little different, from figure 2, I check variance_ratio sum, and notice that over 98% information explained on 30 features, I also use cross-validation and PCA to analyze the classifier performance for higher dimensions, and finally I decide to select PCA with S>1(#51) dimensions for the following steps because our data sets are large enough compared with feature number, and 51 features can keep much more information and bring the higher performance, which also follows #d.o.f (#samples > 3~10 D)=51+1, #constraints is about #train set=3467, #constraints is absolutely much larger than #d.o.f, which is a reasonable choice except for SVM, actually to SVM, #support vectors is 57(the details are in SVM section), and also over-determined.

```
feature numbers: 30
explained_variance_ratio_:
[0.14988101 0.12019424 0.0711981  0.06327241 0.05998209 0.05737455
 0.05684666 0.05437004 0.05162772 0.04622721 0.03792312 0.03003253
 0.02640645 0.02279385 0.0206452  0.01724197 0.01492899 0.01199177
 0.01077486 0.00883023 0.00872689 0.00787057 0.0072154  0.00509573
 0.004897   0.00415628 0.00368193 0.00345397 0.00277173 0.00257215]
explained_variance_ratio_.cumsum:
[0.14988101 0.27007525 0.34127335 0.40454577 0.46452786 0.5219024
 0.57874906 0.6331191  0.68474682 0.73097403 0.76889716 0.79892969
 0.82533613 0.84812999 0.86877518 0.88601715 0.90094614 0.91293791
 0.92371277 0.93254301 0.9412699  0.94914046 0.95635586 0.96145159
 0.96634859 0.97050487 0.97418681 0.97764077 0.98041251 0.98298465]
```

Figure 2 : PCA explained variance_ratio

Besides, the number of features is large enough, then it's unnecessary to expand feature space into a higher dimension. Hence, I only implement the feature extraction.

Cross Validation for Parameter Estimation

In the project, I totally compare three distribution-free classifiers, linear SVM (support vector machine), radial basis function SVM, and perceptron and three statistical classifiers, K neighbors (KNN) classifier, random forest (RF) classifier and naïve bayes (NB) classifier. Especially KNN and RF belong to non-parametric classifiers, and NB is a parametric classifier. In order to get best parameters, model selection for prediction I mainly refer to the following diagram's suggestion, total samples < 100K.

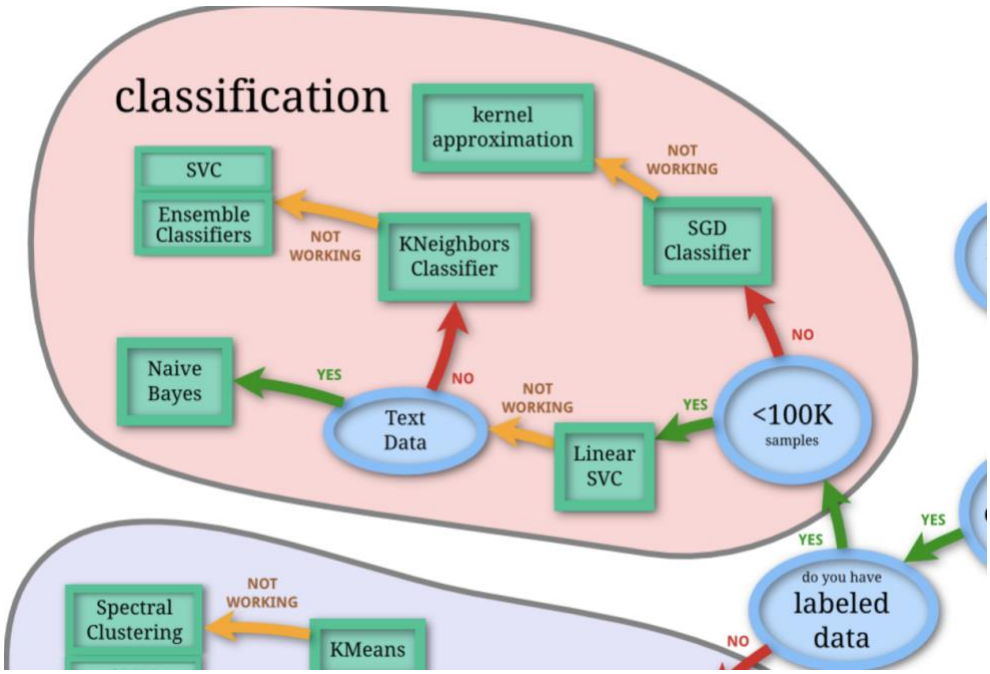

Figure 3 : Derived from "scikit learn algorithm cheat sheet" (Choosing the right estimator, n.d.)

Then I use validation curve and grid search cross validation methods on these classifiers. I always fill in K-Fold=5, that means total training set is #3467 samples, each time has #2773 samples as training, and validation set is #694 for validation, both split based on 5:1, which split is different with train-test split before, here once iteration I extract a contiguous indexed segment from training set as validation set, and next iteration the next segment will be as a new validation set.

In the next report, I report different classifiers in detail.

Distribution Free Classifier

The distribution free classifiers assign some class to a sample without known statistical distribution and is based on the criterion function.

Support Vector Machine

Linear SVM is the first considered classifier, which is effective in high dimensional space, which classifies data by creating a hyper plane in a high dimensional space. RBF SVM only has a different kernel in compare to linear SVM, we know that kernel is used to map sample space to the high dimensional space, RBF uses normal curves around the samples. During my previous pre-processing, linear SVM may be a better choice, however, I try two of them.

First, the validation curve method is a good way to help me first search a reasonable range about gamma parameter for RBF SVM. The parameter defines how far the influence of a single training example reaches (RBF SVM parameters, n.d.), which affect the classifier performance a lot. The figure 4 is an accuracy score with K-Folder = 5 cross validation curve, when gamma equals 0.01, accuracy score on validation set reaches the peak, and then overfitting occurs, training set accuracy continues to increase while validation accuracy start to decrease, therefore, next I train RBF SVM by grid search CV mainly consider around the value and trade off with C parameter.

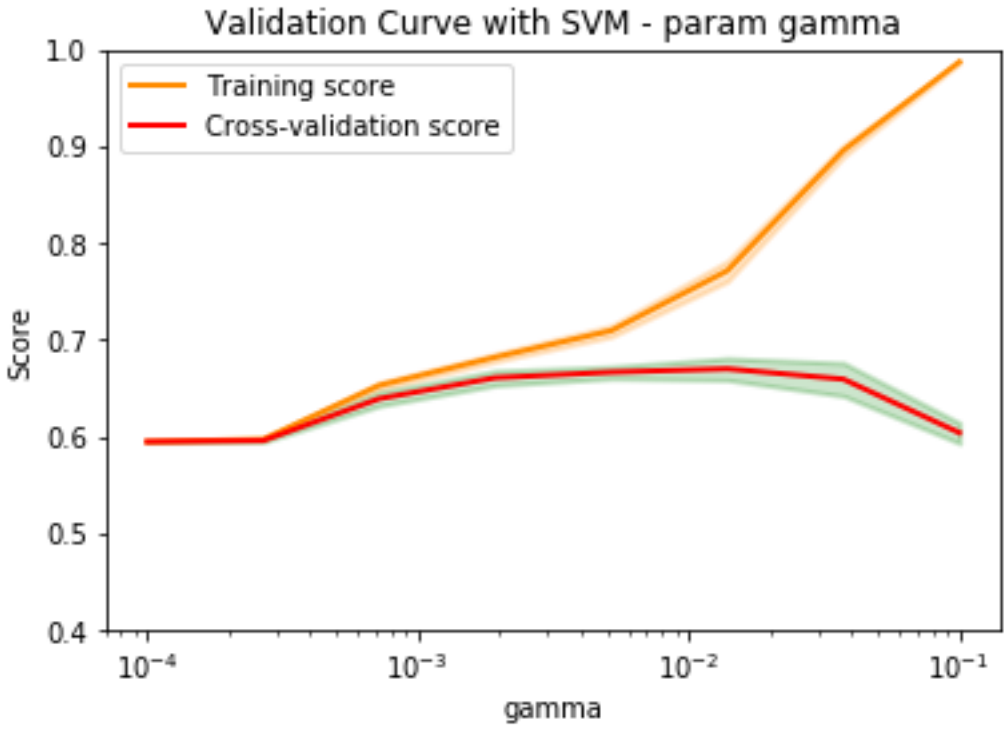

Figure 4 : validation curve with SVM – param gamma

Next, I select grid search cross validation to evaluate C-gamma parameter because the method uses brute force to find the best parameter and is also a popular way to estimate parameters. I select tuned parameters = [{'kernel': ['linear'], 'C': [1, 10, 100, 500, 1000]}, {'kernel': ['rbf'], 'gamma': [1e-4, 6e-3, 1e-2],'C': [1, 10, 100, 500, 1000]}], K-Fold is 5 and scores through precision and recall performance, the result is as follows, I cut out part of them:

SVM Best parameters set found on development set:
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
Grid scores on development set:
0.297 (+/-0.043) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.574 (+/-0.015) for {'C': 10, 'gamma': 0.006, 'kernel': 'rbf'}
0.632 (+/-0.013) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.344 (+/-0.000) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.651 (+/-0.026) for {'C': 100, 'gamma': 0.006, 'kernel': 'rbf'}
0.654 (+/-0.015) for {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
0.648 (+/-0.027) for {'C': 1, 'kernel': 'linear'}
0.647 (+/-0.032) for {'C': 10, 'kernel': 'linear'}
0.650 (+/-0.012) for {'C': 10, 'kernel': 'linear'}

therefore, I'm ready to select C=100 for linear SVM, and select C=100, gamma=0.01 for RBF SVM.

Besides, support vectors to linear SVM are as follows, and total support_vectors_ #SV=51

support_vectors: [[0.1200956 -0.08312018 -0.53424138 ... 0.06818182 -0.90909091 0.09191919]
[0.18014339 -0.03513717 0.10005311 ... -0.04393939 -0.7030303 0.04343434]
[0. -0.02285805 0.03425147 ... 0. 0. -0.04444444]
...
[0.0600478 -0.02928097 0.00700989 ... -0.00479798 -0.7888888 0.03484848]
[-0.18014339 0.03721517 -0.0266854 ... 0. 0. -0.04444444]

[0. 0.07273016 -0.05175565 ... -0.1 -0.8 0.15555556]]

Perceptron classifier

Perceptron is a supervised classification, and an input is to be one of several possible non-binary outputs. It is a type of linear classifier. In the project, I do not adjust its parameters, all default values are used for next fitting.

Statistical Classifier

The statistical classifier is based on the statistical or probabilistic inference to find the best class for a given sample. Statistical classification consists of two types, parametric and non-parametric as introduced before.

K-Nearest Neighbors

KNN classifier is a non-parametric statistical classifier, which calculate the K-nearest samples conditions to determine the class of a sample,

I still use grid search cross validation with the same configuration above, and tuning parameter is [{'n_neighbors': [2, 3, 5, 6, 8, 10, 20, 30, 40, 50]}], the following is the part of result, and n_neighbors=20 is selected for further use.

```
# Tuning hyper-parameters for precision
KNN Best parameters set found on development set:
{'n_neighbors': 20}
Grid scores on development set:
0.624 (+/-0.032) for {'n_neighbors': 8}
0.625 (+/-0.040) for {'n_neighbors': 10}
0.634 (+/-0.033) for {'n_neighbors': 20}
0.626 (+/-0.030) for {'n_neighbors': 30}
0.628 (+/-0.030) for {'n_neighbors': 40}
0.629 (+/-0.031) for {'n_neighbors': 50}
# Tuning hyper-parameters for recall
KNN Best parameters set found on development set:
{'n_neighbors': 20}
Grid scores on development set:
0.589 (+/-0.027) for {'n_neighbors': 8}
0.592 (+/-0.040) for {'n_neighbors': 10}
0.604 (+/-0.026) for {'n_neighbors': 20}
0.597 (+/-0.023) for {'n_neighbors': 30}
0.597 (+/-0.017) for {'n_neighbors': 40}
0.595 (+/-0.020) for {'n_neighbors': 50}
```

Random Forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting (Random Forest Classifier, n.d.). Its major tuning parameters = [{'n_estimators': [50, 100, 200, 300, 400, 500]}], and the output is as follows, and the final choice is 500 number of trees.

```
# Tuning hyper-parameters for precision
RF Best parameters set found on development set:
{'n_estimators': 500}
Grid scores on development set:
0.657 (+/-0.025) for {'n_estimators': 50}
0.654 (+/-0.032) for {'n_estimators': 100}
```

0.655 (+/-0.052) for {'n_estimators': 200}
0.654 (+/-0.037) for {'n_estimators': 300}
0.651 (+/-0.035) for {'n_estimators': 400}
0.660 (+/-0.045) for {'n_estimators': 500}
Tuning hyper-parameters for recall
RF Best parameters set found on development set:
{'n_estimators': 500}
Grid scores on development set:
0.635 (+/-0.046) for {'n_estimators': 50}
0.634 (+/-0.038) for {'n_estimators': 100}
0.637 (+/-0.026) for {'n_estimators': 200}
0.637 (+/-0.033) for {'n_estimators': 300}
0.639 (+/-0.024) for {'n_estimators': 400}
0.640 (+/-0.024) for {'n_estimators': 500}

Naive Bayes

Naive Bayes classifier is an approximation to Bayes classifier, in which we assume the features are conditionally independent given the class. I use default parameters.

Training and Classification

It's the last step, I use training set with the above estimated parameters to fit each model, classifiers = [

```
models.append(('Linear SVM', SVC(kernel="linear", C=100)))
models.append(('RBF SVM', SVC(gamma=1e-2, C=100)))
models.append(('Perceptron', Perceptron()))
models.append(('KNN', KNeighborsClassifier(20)))
models.append(('RandForest', RandomForestClassifier(n_estimators=500)))
models.append(('NaiveBayes', GaussianNB()))
```

and also use K-Fold=5 to split training set and validation set for cross validation. When fitting procedure finishes, 30% test data split in the dataset pre-processing step are predicted on each classifier and I statistics their performance.

Detailed classification report:

The model is trained on the full training set, the prediction is computed on the test set.

		precision	recall	f1-score	support
Naive Bayes: AUC = 0.60	avg / total	0.63	0.62	0.58	1487
Linear SVM: AUC = 0.66	avg / total	0.66	0.66	0.66	1487
RBF SVM: AUC = 0.64	avg / total	0.65	0.65	0.63	1487
Perceptron: AUC = 0.54	avg / total	0.60	0.60	0.52	1487
KNN: AUC = 0.60	avg / total	0.63	0.63	0.61	1487
Rand Forest: AUC = 0.56	avg / total	0.61	0.61	0.55	1487

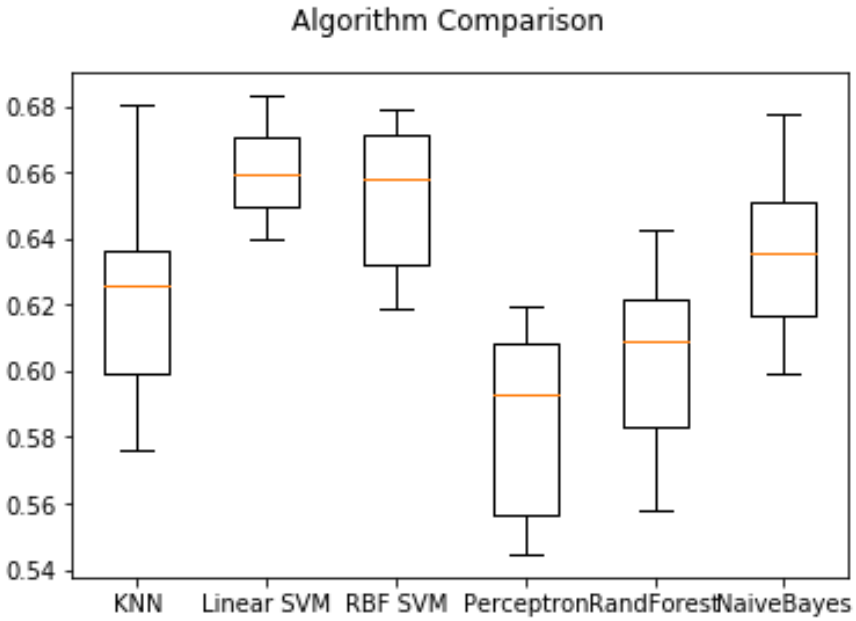

Figure 5 : Boxplot algorithm comparison after PCA reduction

From the above result, linear SVM has the best performance, the box plot is drawn with accuracy score, the result also reflects linear SVM is the best one, which accuracy score is 66%, f1-score is 65%, and AUC is 63%. The result is corresponding to my previous prediction before. First, it's a two class system with enough high dimensional space and samples are less than 10,000, I first speculate they are likely to be distinguish by a hyperplane in the high dimensional space; and in feature extraction part, I select PCA with whiten=True, which further remove the linear correlation across features and make linear SVM more suitable.

Actually, if I do not execute feature extraction, and directly use original data because VC dimension meets and samples of training data are large enough, meanwhile, support vector=#57 in the case,

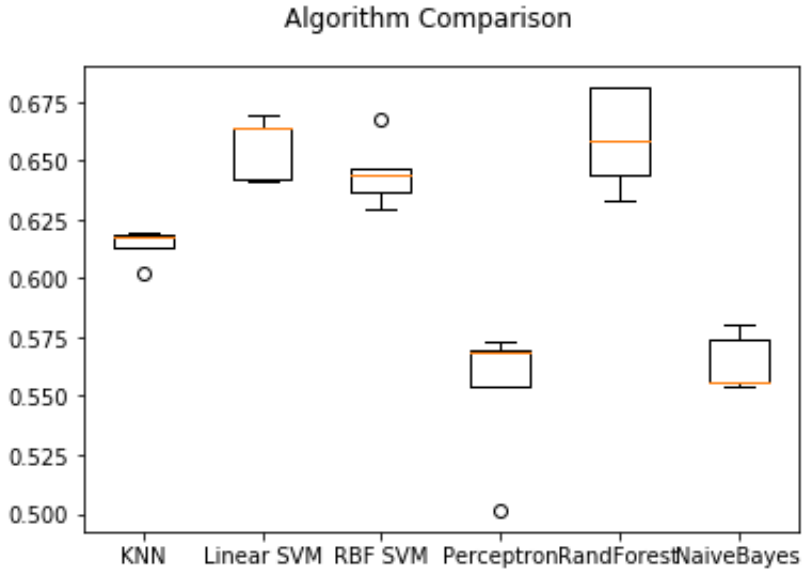

Figure 6 : Boxplot algorithm comparison in original sample space

precision recall f1-score support

Naive Bayes: AUC = 0.56	avg / total	0.61	0.54	0.49	1487
Linear SVM: AUC = 0.66	avg / total	0.66	0.66	0.66	1487
RBF SVM: AUC = 0.66	avg / total	0.66	0.66	0.66	1487
Perceptron: AUC = 0.55	avg / total	0.56	0.56	0.54	1487
KNN: AUC = 0.63	avg / total	0.63	0.62	0.62	1487
Rand Forest: AUC = 0.66	avg / total	0.67	0.67	0.67	1487

Comparison with feature extraction version, although samples scatter with mean, random forest accuracy enhances a lot even better than linear SVM(here #SV=57), and Naïve Bayes performance decrease a little and it's expected because feature dependency increases. By the way, if samples distribution is not uniformed, random forest is a better estimator.

Conclusion:

This report summarizes my research and results obtained, and analyze them to each classifier. Finally, I observe that linear SVM is the best classification for the samples, and the accuracy is 66%. From analyzing the performance of SVM, I have studied the effect of gamma parameter and C parameter to different kernel.

Besides, comparing feature dimension reduction results, I see that PCA always give better classification in all the classifiers in compared to Fisher LDA.

Reference:

Choosing the right estimator. (n.d.). Retrieved from sk-learn: http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

Compare the effect of different scalers on data with outliers. (n.d.). Retrieved from http://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py

Duda, R. O., Hart, P. E., & Stock, D. G. (2000). *Pattern Classification.* New York: Wiley- Interscience.

Fernandes, K., Vinagre, P., & Cortez, a. P. (2015). *A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News.* Portugal: INESC TEC Porto.

Random Forest Classifier. (n.d.). Retrieved from http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

RBF SVM parameters. (n.d.). Retrieved from sklearn: http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html#sphx-glr-auto-examples-svm-plot-rbf-parameters-py