Yuhao Chen
MB654712

COIS716 Mid-term Report

Machine Learning Project on Mushroom Classification

Introduction

In this project, I implemented a machine learning model on the mushroom dataset to figure would if the mushroom is poison or edible, this is a classification work. Through this project, I use 7 classification algorithms like logistic regression, random forest classifier and supporter vector machine (SVM), etc. to build up a classifier on the non-linear dataset, I also did some feature engineering, i.e. label-binarizer, one-hot encoder, and PCA decomposition algorithm to deal with the discrete features and a high dimension dataset.

Dataset

I chose one dataset named "Mushroom Classification" from UCI Machine Learning group on Kaggle(https://www.kaggle.com/uciml/mushroom-classification).

In this dataset, there are 8124 rows of data(instances), and each instance has 23 features with the start index as their class with is poison or edible, other 22 features are:

Mushroom Dataset

cap-shape	bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
cap-surface	fibrous=f, grooves=g, scaly=y, smooth=s
cap-color	brown=n, buff=b, cinnamon=c, grey=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, bruises: bruises=t, no=f
odor	almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s
gill-attachment	attached=a, descending=d, free=f, notched=n
gill-spacing	close=c, crowded=w, distant=d
gill-size	broad=b, narrow=n
gill-color	black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y

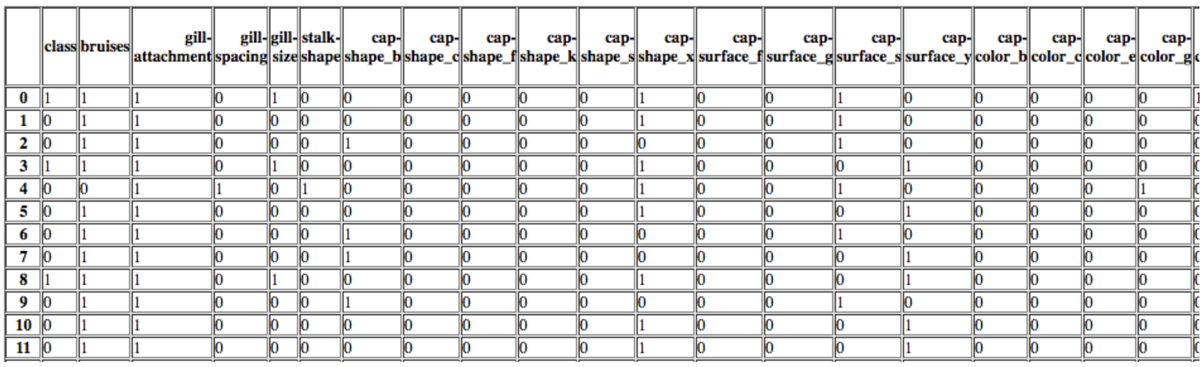
Mushroom Dataset	
stalk-shape	enlarging=e, tapering=t
stalk-root	bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
stalk-surface-above-ring	fibrous=f, scaly=y, silky=k, smooth=s
stalk-surface-below-ring	fibrous=f, scaly=y, silky=k, smooth=s
stalk-color-above-ring	brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
stalk-color-below-ring	brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
veil-type	partial=p, universal=u
veil-color	brown=n, orange=o, white=w, yellow=y
ring-number	none=n, one=o, two=t
ring-type	cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
spore-print-color	black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
population	abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
habitat	grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

Methodology

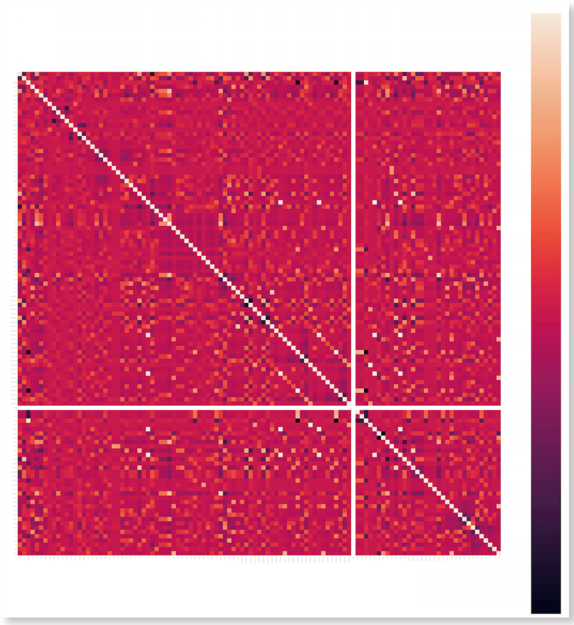
I used python and its data science(pandas, scikit-learn, matplotlib, seaborn) packages to build up the models.

Feature Engineering

Since the features of the dataset are all discrete, by using those supervise machine learning algorithm, we should first do some feature engineering work. I used two different method to make those features numeric, for the feature with only two labels, i,e, potion or edible of the mushroom class, a label binarizer would solve the problem, it would turn twovalued feature into 1 and 0. For other features which has lots of classes, I observed that the number of classes of a feature is less than 10. I decided to use one-hot encoder which would call the get_dummies() function to append the new array or data frame into the current feature data frame. After feature engineering process, 23 columns of features are transformed into 113 columns of new feature with 1 and 0. The figure shows one simple glance at the new feature table.



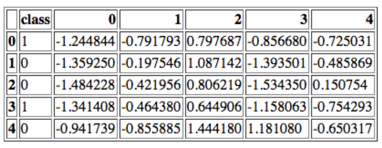
	class	bruises	gill-attachment	gill-spacing	gill-size	stalk-shape	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f	cap-surface_g	cap-surface_s	cap-surface_y	cap-color_b	cap-color_c	cap-color_e	cap-color_g
0	1	1	1	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
2	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
3	1	1	1	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
4	0	0	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	1
5	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
6	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
7	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
8	1	1	1	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
9	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
10	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
11	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0



This would cause a problem which is, the features are too sparse to analysis, I projected a data from 23 dimensions to 113 dimensions, and I drew a heat map based on the current table, and we can see that this is too large as a table I want to dig in.

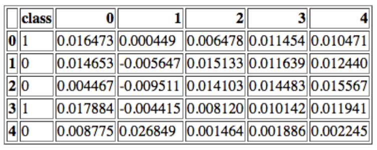
In order to squeeze the table I have right now into a narrow one. I used some dimensionality reduction algorithm. In "sklearn.decomposition" package, there are three mainstream decompositor to reduct the dimensions, PCA, FastICA and FactorAnalysis. Here are some comparisons between the data decomposed by different kinds of decompositors. By using the dimensionality reduction, the less dimension you squeeze the table into, the more detail would be lost. In the later section, I would compare the result with multiple dimension choices.

In the following chart, we can see the different results come from the different dimension reduction algorithms.



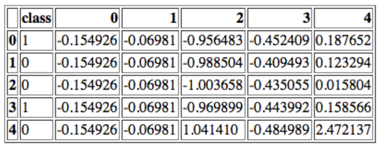
	class	0	1	2	3	4
0	1	-1.244844	-0.791793	0.797687	-0.856680	-0.725031
1	0	-1.359250	-0.197546	1.087142	-1.393501	-0.485869
2	0	-1.484228	-0.421956	0.806219	-1.534350	0.150754
3	1	-1.341408	-0.464380	0.644906	-1.158063	-0.754293
4	0	-0.941739	-0.855885	1.444180	1.181080	-0.650317

PCA



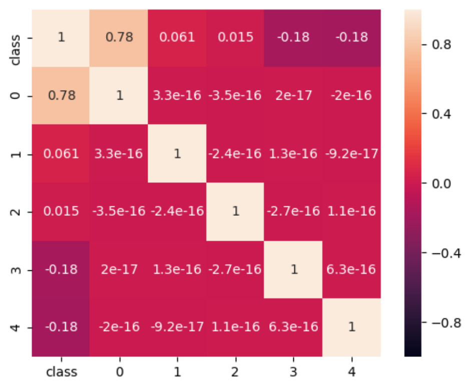
	class	0	1	2	3	4
0	1	0.016473	0.000449	0.006478	0.011454	0.010471
1	0	0.014653	-0.005647	0.015133	0.011639	0.012440
2	0	0.004467	-0.009511	0.014103	0.014483	0.015567
3	1	0.017884	-0.004415	0.008120	0.010142	0.011941
4	0	0.008775	0.026849	0.001464	0.001886	0.002245

FastICA



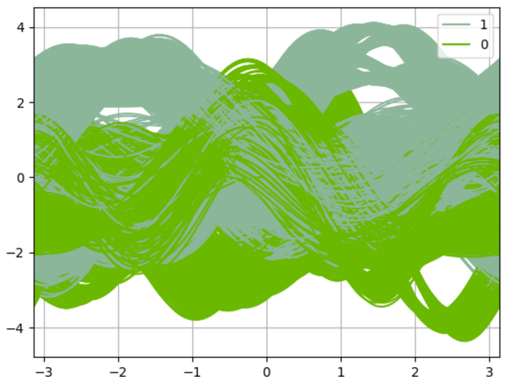
	class	0	1	2	3	4
0	1	-0.154926	-0.06981	-0.956483	-0.452409	0.187652
1	0	-0.154926	-0.06981	-0.988504	-0.409493	0.123294
2	0	-0.154926	-0.06981	-1.003658	-0.435055	0.015804
3	1	-0.154926	-0.06981	-0.969899	-0.443992	0.158566
4	0	-0.154926	-0.06981	1.041410	-0.484989	2.472137

FactorAnalaysis



In the data analysis with linear dataset, we can clearly see which feature with concrete label would be important, so we can keep the important feature. But in the discrete dataset, since we have squeezed the data, we don't want to continue losing the detail of the dataset, so I decided to still keep all dimensions.

I also wanted to see the distribute situation of the dataset on their high dimensions, I used andrew_curves function to figure out the situation with two different classes of mushrooms. This visualisation is based on PCA decomposition algorithm we can see that there are some minor difference between two different classes of mushrooms.

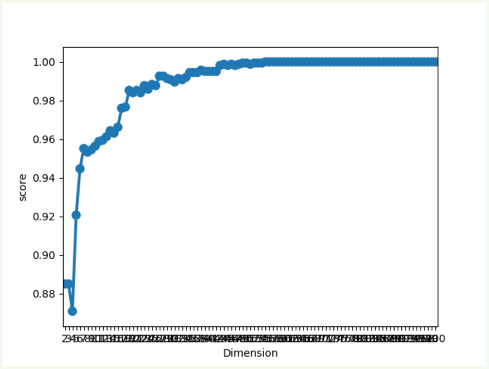
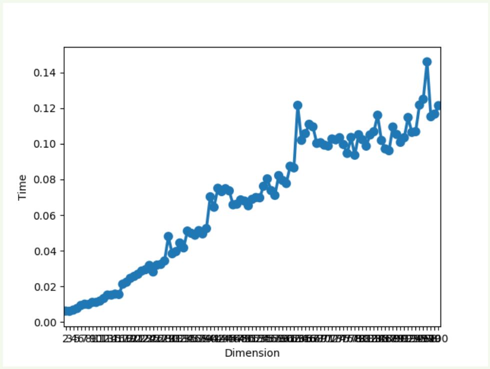
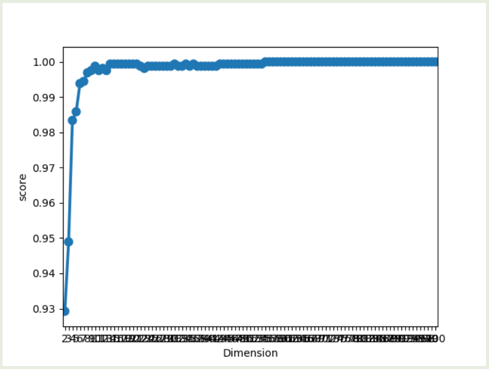
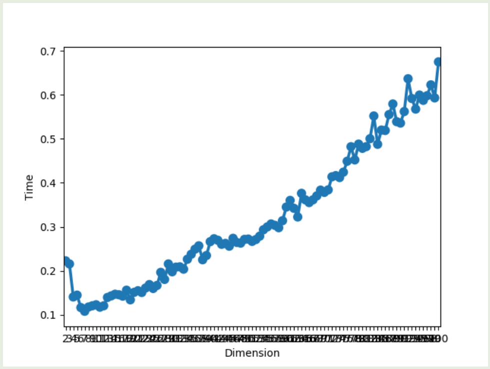
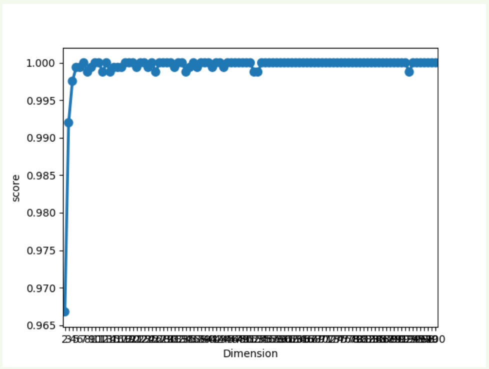
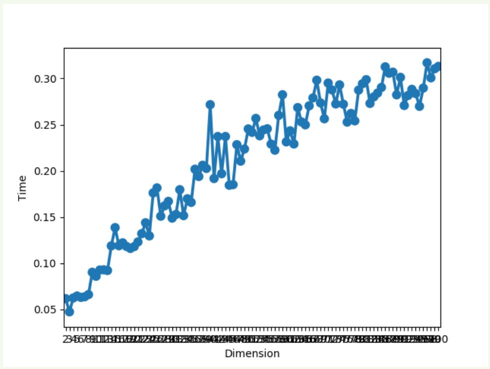


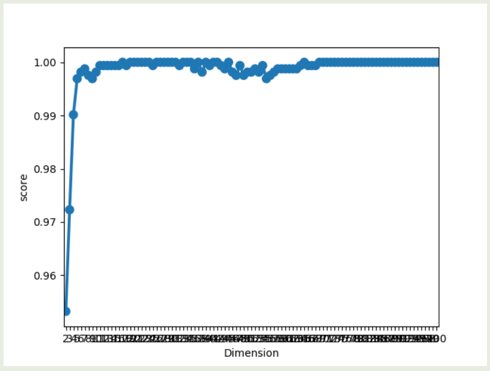
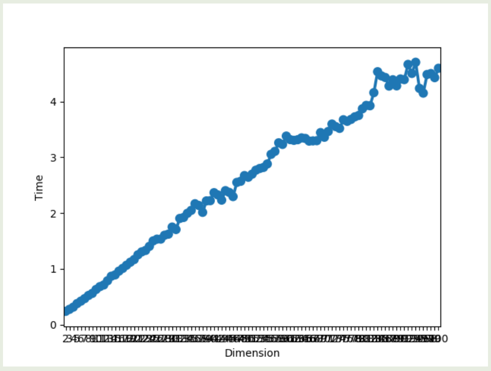
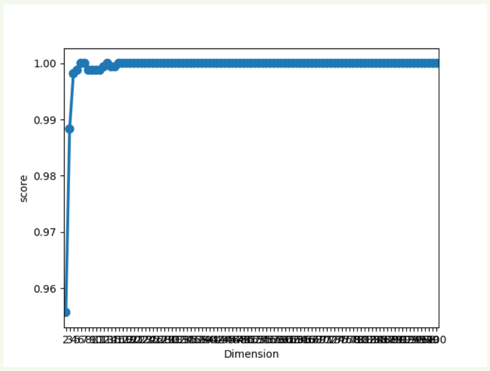
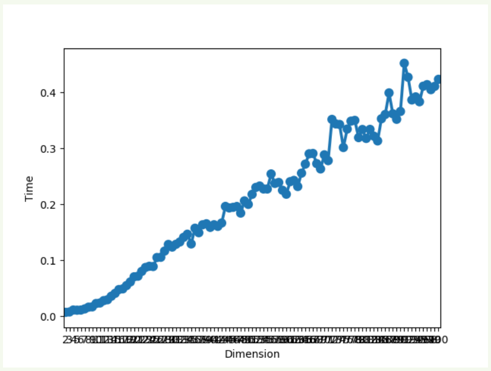
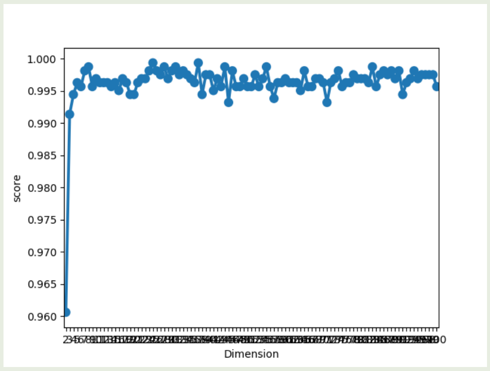
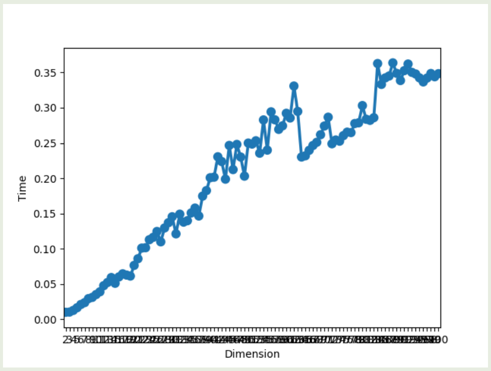
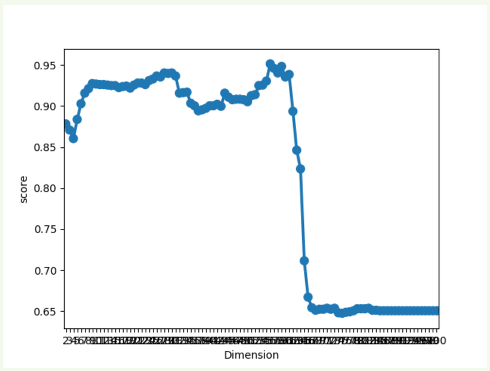
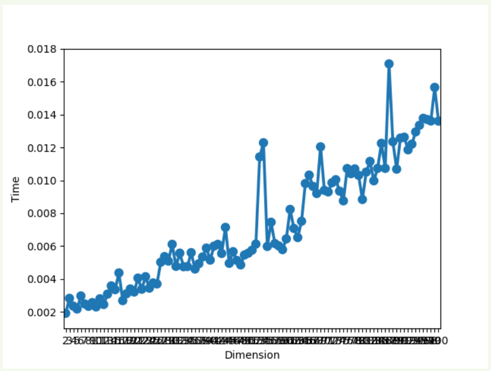
Training and Testing

By using the train_test_split package, we split the dataset into four sub-dataset, and the rate for splitting the whole dataset is 0.2% as the testing dataset. In the chart, the 5 means the current dimension number, we also have one column of the label we want to classify.

Training and Testing	
X-train	6499*5
y-train	6499*1
X-test	1625*5
y-test	1625*1

In this project, I use 7 classification algorithms to test our result. The algorithms are: logistic regression, SVM, random forest, GBDT, KNN, decision tree and Naive Bayes Classifier. I am also curious about relationship between the score, time and the number of dimension. We use decomposition algorithm to set the dimension of the dataset in the range of 2 to 100. After coding and testing, I made a chart to show the results of each model.

Algorithm/ Training & Testing	Score	Time
Logistic Regression		
SVM		
Random Forest		

Algorithm/ Training & Testing	Score	Time
GBDT		
KNN		
Decision Tree		
Naive Bayes		

Conclusion

According to the result, I found that most of the algorithms I chose have a really good accuracy on this dataset, and most of the time/dimension ratio of each model is linear, but it takes a lot of time for GBDT to evaluate the result, and the accuracy of Naive Bayes Classifier drop down very fast when it comes to a high-dimension. I would choose KNN and SVM in this case of dataset because the idea of the these two algorithms are pretty clear and direct to the target, and the performance(time/dimension ratio) is also pretty charming comparing to other models.

Appendix

Here is the code of the whole program.

```
import numpy as np
import pandas as pd
from pandas.plotting import andrews_curves, parallel_coordinates, radviz
from sklearn import preprocessing, svm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.decomposition import PCA, FastICA, FactorAnalysis
from sklearn.ensemble import RandomForestClassifier
from matplotlib import pyplot as plt
import seaborn as sns
import time

model_dict = {
    'Logistic_Regression': LogisticRegression(),
    'SVM': svm.SVC(),
    "Random_Forest": RandomForestClassifier(),
    "GBDT": GradientBoostingClassifier(),
    "KNN": KNeighborsClassifier(),
        "Decision_Tree": DecisionTreeClassifier(),
        "Naive_Bayes": GaussianNB()
    }
    data_frame = pd.read_csv('./mushroom.csv')
    # feature engineering
    for column in data_frame:
        if len(data_frame[column].unique()) == 2:
            data_frame[column] =
preprocessing.LabelBinarizer().fit_transform(data_frame[column])
        else:
            data_frame = pd.get_dummies(data_frame, columns=[column])

    score_list = []
    time_list = []
    dimension_range_list = list(range(2, 101))

    for key, value in model_dict.items():
        print('==================')
        print('This is {0} model.'.format(key))
        print('==================')
        for new_dimension in dimension_range_list:
            decompositor = PCA(n_components=new_dimension)
            # decompositor = FastICA(n_components=new_dimension, max_iter=200)
            # decompositor = FactorAnalysis(n_components=new_dimension)
            decompositor_result =
decompositor.fit_transform(data_frame[data_frame.columns[1::]])
            X = np.array(decompositor_result)
            y = np.array(data_frame[data_frame.columns[0]])
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)

            clf = value
            start_flag = time.time()
            clf.fit(X_train, y_train)
            score = clf.score(X_test, y_test)
            end_flag = time.time() - start_flag
            print("The score of this classifier is: {0}, and the time is: {1} 
seconds.".format(score, end_flag))
            score_list.append(score)
            time_list.append(end_flag)

        score_dataframe = pd.DataFrame(score_list)
        score_dataframe.columns = ['score']
        dimension_dataframe = pd.DataFrame(dimension_range_list)
        dimension_dataframe.columns = ['Dimension']
        time_dataframe = pd.DataFrame(time_list)
        time_dataframe.columns = ['Time']
        dimension_score_dataframe = pd.concat([dimension_dataframe, score_dataframe], 
axis=1)
        dimension_time_dataframe = pd.concat([dimension_dataframe, time_dataframe], 
axis=1)

        del (score_list[:])
        del (time_list[:])

        plt.figure()
        sns.pointplot(x='Dimension', y='score', data=dimension_score_dataframe)
        plt.savefig('./results/{0}_dimension_score.png'.format(key))
        plt.show()

        plt.figure()
        sns.pointplot(x='Dimension', y='Time', data=dimension_time_dataframe)
        plt.savefig('./results/{0}_dimension_time.png'.format(key))
        plt.show()
```