Genetic Data PCA implementation

Introduction

This analysis is carried out in order to uncover if it is possible to visualize
on statistical information obtained from Gene data. Feature engineering using principal component analysis is applied on Genetic data to reduce the features to two dimensions to visualize the data.

Data description

In this analysis I used Genetic dataset, which contains 22411 features and 30 samples.
Files used in this analysis
Gene_data.csv.
Meta_data.csv.

Principal component analysis

The main idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. The same is done by transforming the variables to a new set of variables, which are known as the principal components (or simply, the PCs) and are orthogonal, ordered such that the retention of variation present in the original variables decreases as we move down in the order. So, in this way, the 1st principal component retains maximum variation that was present in the original components. The principal components are the eigenvectors of a covariance matrix, and hence they are orthogonal.

Methods

Exploratory analysis

This kind of analysis was conducted in order to identify the quality of data detect missing values, outliers and choose observations for training and test sets. To make statistical modeling more convenient, we slightly renamed the variables.
Exploratory analysis was performed by examining tables and plots of the obtained data.

Statistical Modeling

For statistical modeling I decided to choose two two supervised learning models

These two models are:

1.PCA

Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.It uses the LAPACK implementation of the full SVD or a randomized truncated SVD by the method of Halko et al. 2009, depending on the shape of the input data and the number of components to extract.

2. Random forest

An ensemble learning method for classification that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees. This model was chosen due to its robustness and resistance to overfitting.

Step by Step approach how the data interpreted

Importing necessary modules

```
from pca_gene import *
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
```

Loading dataset

```
dataset=pd.read_csv('gene_data.csv')
metaset=pd.read_csv('meta.csv')
```

Class designed specially for PCA on genetic data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
sns.set()

class Error(Exception):
   """Base class for other exceptions"""
   pass

class GenePca(object):
    """docstring for ClassName"""

    def __init__(self, x,y):
        try:
            if x.shape[0]==y.shape[0]:
                self.x=x
                self.y=y
                print('x shape:',x.shape)
                print('y shape:',y.shape)
                nan_list=self.x.columns[self.x.isna().any()].tolist()
                print('x dataset contains nan values in columns
:',nan_list)
            else:
                raise Error
        except Error:
            print('x rows:',x.shape[0])
            print('y rows:',y.shape[0])
            print('GP Error: dataframes shape mismatch')
    def standardize(self):
        self.sc_x=StandardScaler()
        self.x=self.sc_x.fit_transform(self.x)
    def set_components(self,components=2):
        self.pca=PCA(n_components=components)

    def get_reduced_set(self,std_scale=0):
        try:
            if std_scale!=0 and std_scale!=1:
                raise Error
            elif std_scale==1:
                self.standardize()
                self.red_x=self.pca.fit_transform(self.x)
                self.red_x=pd.DataFrame(self.red_x)
```

```
            else:
                self.red_x=self.pca.fit_transform(self.x)
                self.red_x=pd.DataFrame(self.red_x)
            return self.red_x
        except Error:
            print("Error:wrong value for scale")

    def visualize(self):
        try:
            if self.red_x.shape[1]==2:
                classes=set(self.y)
                cmap = plt.get_cmap('Set1', len(classes))
                cmap.set_under('gray')
                fig, ax = plt.subplots()
                cax = ax.scatter(self.red_x.iloc[:, 0], self.red_x.iloc[:,
1],
                                 c=self.y ,s=100, cmap=cmap, vmin=self.y.min(),
vmax=self.y.max())
                fig.colorbar(cax, extend='min')
                plt.xlabel('PC1')
                plt.ylabel('PC2')
                plt.show()
            else:
                raise Error
        except Error:
            print("GP Error:contains more than two dimensions")
    def get_evr(self):
        try:
            if self.pca==None:
                raise Error
            return self.pca.explained_variance_ratio_
        except Error:
            print("GP Error: PCA object Not found")
    def get_ev(self):
         try:
            if self.pca==None:
                raise Error
            return self.pca.explained_variance_
         except Error:
            print("GP Error: PCA object Not found")
    def fill_nans(self,measure='mean'):
        nan_list=self.x.columns[self.x.isna().any()].tolist()
```

```
        self.x=self.x.apply(pd.to_numeric)
        try:
            for i in nan_list:
                if measure=='mean':
                    self.x[i].fillna(self.x[i].mean(),inplace=True)
                elif measure=='median':
                    self.x[i].fillna(self.x[i].median(),inplace=True)
                else:
                    raise Error
                print("removed nan values in column:",i)
        except Error:
            print("GP Error:unknow measure choose only median or mean")

    def getpca(self):
        return self.pca
```

Documentation for the above class

```
GenePca is class used to work on PCA of gene datasets

GenePca(x_dataframe,y_dataframe)

standardize()
    -standardize the given x_dataframe

set_components(components=2)
    -will reduce x_dataframe into given components features

get_reduced_set(std_scale=0)
    -will return reduced set after PCA on it
    -if std_scale=1 x_dataframe will get standardized before PCA

getpca()
    -return pca object

get_evr()
    -return explained_variance_ratio

get_ev()
    -return explained_variance
```

```
fill_nans(measure='mean')
    -replace the Nan values in column with column mean/median

visualize()
    -visualize the datasets
    -applicable only for two dimensions
```

Data preprocessing

```
dataset=dataset.transpose()
dataset.drop(['Unnamed: 0'],inplace=True)
dataset.drop(['symbol'],inplace=True)
```

Applying PCA on dataset

```
#importing GenePca class
genepca=GenePca(dataset,metaset['Cluster'])
#filling nan values with mean
genepca.fill_nans(measure='mean')
#setting dimension to 2
genepca.set_components(2)
```

Code to get reduced output

```
#reduced dataset
dataset=genepca.get_reduced_set(std_scale=1)
pca=genepca.getpca()

print('Explained variance:',genepca.get_ev())
print('Explained variance ratio:',genepca.get_evr())
```

Trainig and testing model on dataset

```
x_train,x_test,y_train,y_test=train_test_split(dataset,metaset['Cluster'],t
est_size=0.3,random_state=20)
'''
Random forest classifier has good accuracy when compared to
other classification
algorithms

'''
```

```
#random forest
from sklearn.ensemble import RandomForestClassifier
classifier=RandomForestClassifier(n_estimators=10,criterion='entropy',rando
m_state=0)
'''
#LogisticRegression
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression(random_state=0)

#naive bayes
from sklearn.naive_bayes import GaussianNB
classifier=GaussianNB()

#random forest
from sklearn.ensemble import RandomForestClassifier
classifier=RandomForestClassifier(n_estimators=10,criterion='entropy',rando
m_state=0)

#kmeans
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)

'''
#training dataset
classifier.fit(x_train,y_train)
#testing dataset
y_pred=classifier.predict(x_test)
print('Actaul--Predicted')
for i,j in zip(y_test.values,y_pred):
      print(i,'-',j)
```

Visualizing the output

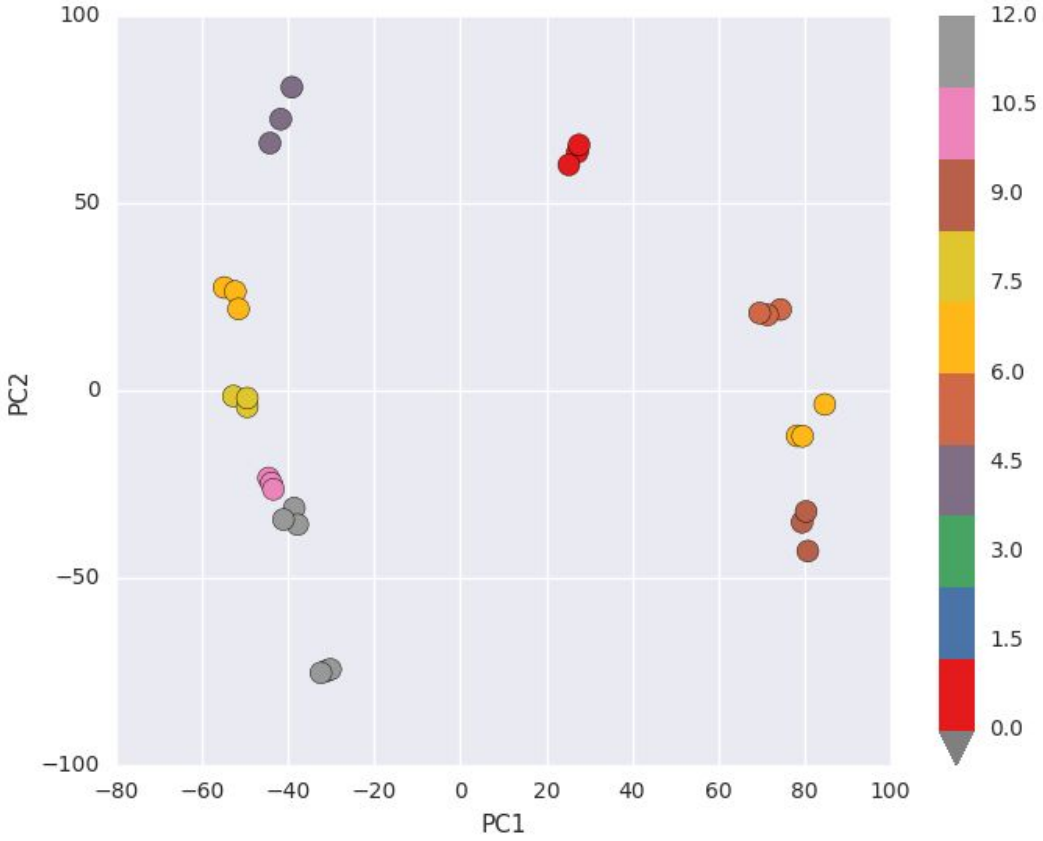
```
#visualizing the dataset
genepca.visualize()
```

I used python module matplotlib for the visualizing the reduced dataset

Results

x shape: (30, 22411)
y shape: (30,)
x dataset contains nan values in columns : [12076]
removed nan values in column: 12076
Explained variance: [3145.89499222 1995.86160249]
Explained variance ratio: [0.20147289 0.12782118]
x shape: (30,2)

Scatter plot of PC1 and PC2



Model used for predicting values: RandomForestClassifier

Output of the model accuracy: 77.79%

Actual-cluster	Predicted cluster
4	4
7	7
9	12
0	0
11	11
4	4
8	8
11	11
12	12

Conclusion

Dataset before PCA contains 22411 features and it is reduced to two dimension/features which has more contribution to dataset variance.PC1 and PC2 are contributing more than 30% of variance of dataset. Remaining 22409 dimension are contributing remaining 60% of variance of dataset. After feature extraction using PCA.We visualized the dataset of containing PC1,PC2 and target variable still genes of similar groups formed clusters and from this we can say that whole dataset can be analysed with PC1 and PC2