Final Project Report

Course: COGS 109

Team member: Xinmeng Xu, Sijie Liu, Jiayan Ma, Sizhe Fan

Abstract

In this project, we use machine learning and data analysis methods including Principal Component Analysis, Classification and Clustering to predict whether the lungs shown in the Chest X-Rays are pneumonia or healthy. By training the models, the machine can tell if patients are infected with pneumonia with high accuracies. By analysing the model performances and results, we can numerically and visually reveal the significance of PCA, and we also have an insight about the stages of infection of pneumonia.

Introduction/Background

Motivation & background information:

We were interested in studying how intelligent diagnosis could help with disease diagnosis made by experienced individuals. Doctors usually undergo extensive pressure and workload each day so that the lack of rest could sometimes result in mistaken diagnosis. Thus, an AI doctor with the capability of double checking the proposed diagnosis would help reduce the number of mistakes doctors would make so that patients could get the most optimal treatment.

Dataset:

Considering that the COVID-19 pandemic has drawn people's attention to lung health, we decided to focus on developing AI techniques in diagnosing the health condition of lungs. We found a dataset from kaggle.com that was available for studying the differences between healthy lungs and lungs with Pneumonia. Here is a link for it: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia. The dataset is organized into 3 folders (train, test, val) and each of these three folders contains two subfolders, Pneumonia and Normal, representing two general categories of the health state of the lungs. In total, the dataset contains 5,863 X-Ray images (in JPEG form) categorized into two sets, normal (negative/healthy) and Pneumonia (positive/infected). These images were collected from retrospective cohorts of pediatric patients aged between one to five years from Guangzhou Women and Children's Medical Center, China. We picked a subset of normal samples (n=1000) and Pneumonia samples (n=1000) for later analysis. Each sample is an X-Ray image in the form of a JPEG photo. Samples are of slightly different sizes, which makes it necessary for us to resize the images into equal size for the convenience of further manipulation. We finally decided to resize the images to 100 x 100 pixels so that each sample contains 10000 dimensions. Samples are categorized as either healthy (negative) or Pneumonia (positive) without further distinguishing the Pneumonia population into bacterial infection or viral infection. Thus, the label is 0 for normal (negative) and 1 for Pneumonia (positive). The diagnoses were provided by two experienced physicians and a third expert evaluated their diagnoses to reduce the error rate.

Methods

Data Transformation:

Because images of lungs are not all scaled at the same center, and some images are of different sizes, we first resized all images to 100x100 pixels. Then, considering the limited computing power of the computer, we randomly selected 1000 images from the healthy lung and infected lung folders respectively, which made 2000 in total, and converted them to a csv file of 2000 rows where each row represented an image. There were 10001 columns in the csv file. The first column was either 0 or 1, representing normal/healthy condition and pneumonia condition respectively. With all images processed, the data was successfully transformed from image form to an array of 10001 dimensions and was ready for analysis.

PCA:

PCA is used to reduce the dimensionality of quantitative data to a more manageable set of variables and the reduced set can then be input to reveal underlying patterns in the data or as inputs into a model (regression or clustering, etc).

We used PCA with the intention to speed up the training with data of reduced dimensionality while keeping most information intact. First, we calculated the eigenvalues and eigenvectors of the 2000*10001 matrix and ordered the resulting eigenvalues from the smallest to the largest. We then calculated the total information by summing up each component's covered information and plotted to find out the most optimal number of components to use.

Logistic regression (Classification):

As we intended to train a model to predict whether a patient's lung was healthy to make full use of the given labels that were binary (0 for normal/healthy, 1 for pneumonia), we conquered this binary classification problem using Logistic Regression. We used the sklearn (sci-kit learn) package in Python and first trained the model using 70% of the shuffled data. After the model was done training, we tested the prediction accuracy using the rest 30% of the data. Cross validation was also implemented to validate the predictions because this resampling procedure could further prove the validity of our model especially when a high-dimensional dataset was used. Besides the predictions, we also want to analyze the effect of our PCA dimensionality reduction on the results by fitting the model with reduced dimensions and full dimensions separately. To prove that PCA indeed helped with speeding up the analysis, we also examined the runtime of data with reduced dimensionality and data with full dimensionality.

K-means clustering:

k-means clustering is a method of vector quantization used in unsupervised data in machine learning. With a belief that some data points in the given dataset share some similarities, it aims to partition n observations (sample points) into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centroid).

In our case, it is true that the original data has already been categorized into two big categories, healthy and diseased, and that our classification analysis corresponded to this division, there is no more information about which type of Pneumonia the lung should be diagnosed with, e.g. viral Pneumonia or bacterial Pneumonia. Thus, the k-means clustering was chosen because we wanted to further explore if there were subcategories of diseased lungs. We implemented the k-means clustering upon data points classified as "diseased" by defining a function "k_mean_plot_final_dis_total" that took three input: k, the number of clusters, it, the number of iterations, and C, the randomly generated matrix representing the centroids of k clusters. It utilizes the method we learned in class by calculating each sample point's distances to the centroids and grouping the corresponding sample points to the point where they were the closest to. After a certain number of iterations (in our case it was 100), we calculated the total distance by summing distances between each datapoint and its centroid. We used a for-loop to iterate through the k value from 1 to 10 and appended each iteration's output to an array of total distances. We then generated a plot to find out the most possible number of clusters residing in the dataset.

Results

PCA:

The original dataset has 100*100 pixels, which led to 100000 variables for each image. By plotting the information represented in dimension reduced figures, we can tell that when the final variable is 2000, the scatter plot is getting very close to 1. So we concluded that 2000 variables would be enough to provide enough information to distinguish the difference of figures.

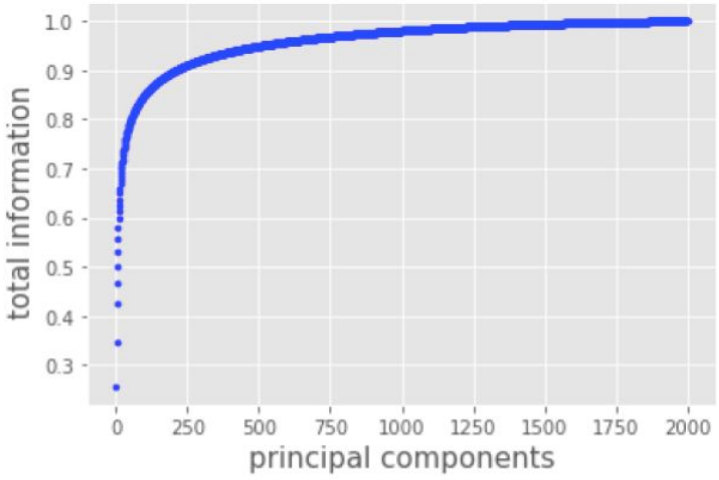


Figure 1: A scatter plot of the total information contributed by the components.

Logistic Regression:

The accuracy of one-time prediction based on full dimensions is around 0.945 (loss = 0.360). Because of the size of full-dimension data is 2,000 * 10,000, which is relatively small in size and large in dimension, we validate our result by a 10-folds cross validation that gives scores of min 0.914, max 0.957, mean 0.936 (figure 4 left). Thus, our model works well so far, and the results should be substantial.

To be clear on the performance of the model, the left figure of figure 2 shows the distribution label probability of each prediction, for example, if a prediction is {[pneumonia, 0.7],[health, 0.3]}, then the point (0.7, 0.3) will be plotted, meaning that the model was 70% confident that its prediction was correct. As we can see, points are denser near corners and sparser around the middle part, which means this model is sure about predictions. Numerically speaking, there are 98.1% predictions with certainty over 0.8, and 96.8% over 0.9 (figure 3 left).

Similarly, LR model fitted by 2000*2000 principal-dimensions dataset, the accuracy is around 0.953 (loss = 0.199). 10-folds cross validation gives scores of min 0.929, max 0.957, mean 0.942 (the accuracy is similar but a little bit larger than the previous model) (figure 4 left). The right figure of figure 2 shows the label probability distribution of this model. The prediction confidence is 96.5% over 0.8, and 94.7% over 0.9 (figure 3 left).

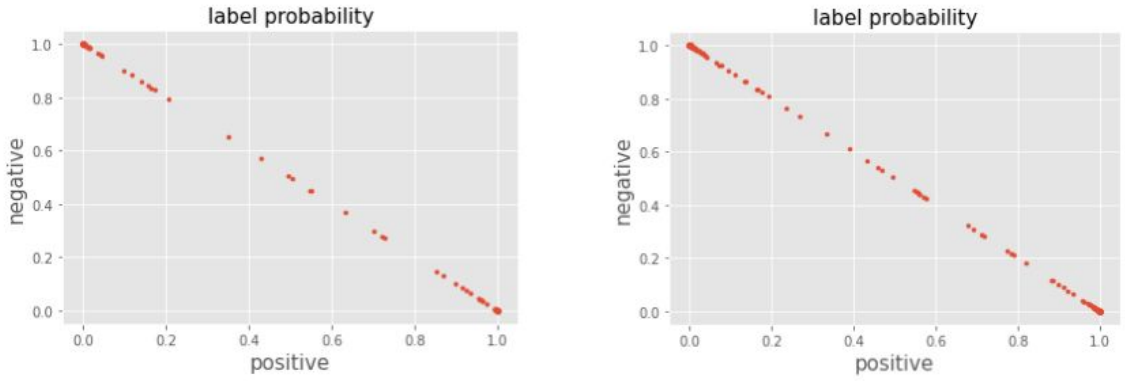


Figure 2: These two scatter plots demonstrated the level of confidence of the prediction. Left: Prediction confidence of LR on data with reduced dimensions. Right: Prediction confidence of LR on data with full dimensions.

Comparing loss and certainty of two models (figure 3), it is clear that the full-dimensional model is more certain about the predictions, but it has much more loss than the principal-dimensional model. We can understand this result as the full-dimensional model is sure about many false predictions, and it has more this kind of "false positive" result than the model based on principal components. By this analysis, we can conclude that the PCA is a good way to clean the data and wipe out the anonymous interferences, so that the predictions are more accurate.

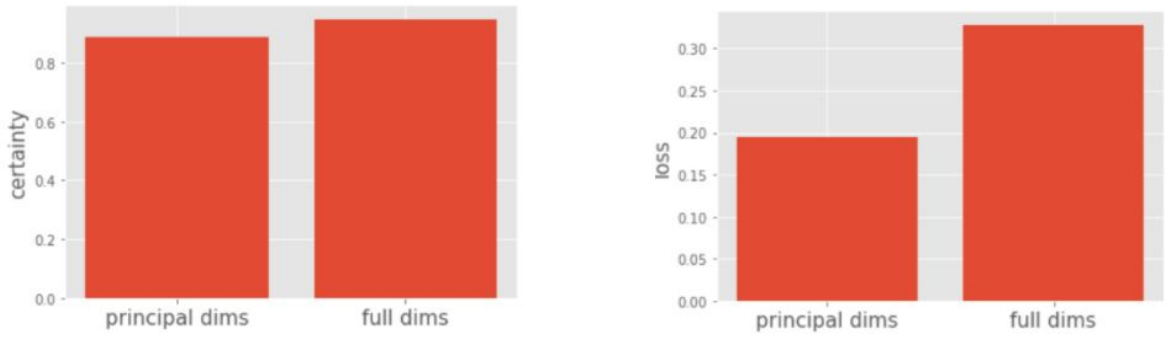


Figure 3: These two bar plots demonstrated the level of confidence of the prediction. Left: Prediction confidence of LR on data with reduced vs. full dimensions. Right: Loss of LR on data with reduced vs. full dimensions.

As we have a large dataset, running time should be one of the most significant indicators of model performance. Along with training the model, we also record the running time. Model based on principal-components takes 1.6 seconds, in contrast, the full-dimensional model takes 13.1 seconds (figure 4 right).

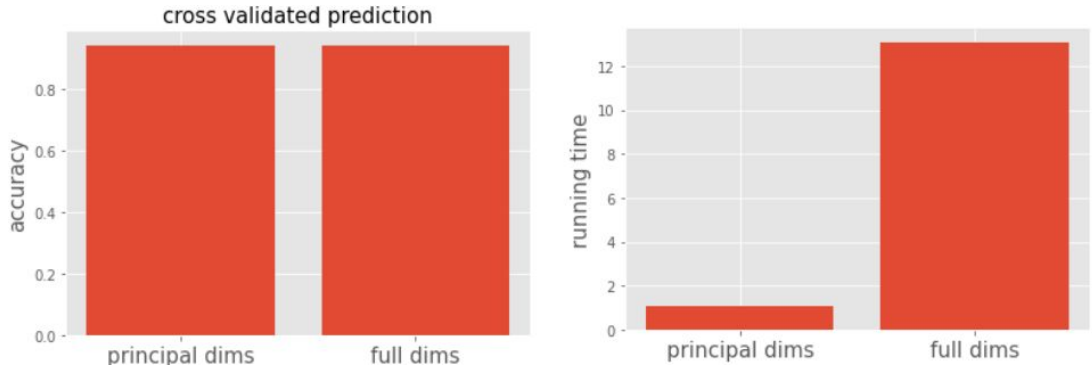


Figure 4: These two bar plots summarized about post-model-training examination. Left: Accuracy score of the cross validations of data with reduced vs. full dimensionality. Right: run time of model training using data with reduced vs. full dimensions.

Assembling the above analysis, it is reasonable to conclude that the model trained by principal components performed much better because although its validated accuracy is similar to the full-dimensional model (0.942 vs 0.936), it only use 8% of the time taken by another model to train, only about 50% of the another model's loss to predict.

Revealed by our LR analysis above, significance of PCA is so high that the performance of a complex model with large size high-dimensional dataset can be substantially promoted.

K-Means clustering:

We found that there were likely 2-3 clusterings within the data points classified as "diseased" because the reduction of total distances slowed down when k got larger than 3. The

total distance when k equals 1 is 97780.26, 33877.72 when k=2, and 21965.65 when k=3 (figure 5). Total distances for k = 1-10 are detailed in Table1.

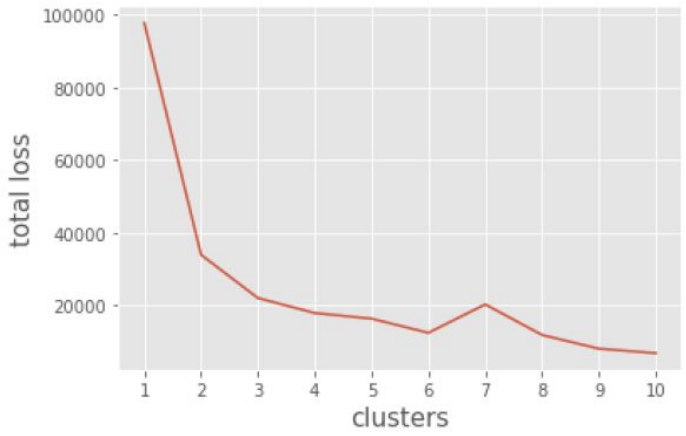


Figure 5: A line plot of the total distance of data points when number of clusters, k, ranges from 1 to 10.

k	**1**	**2**	**3**	**4**	**5**
Total distance	97780.26	33877.72	21965.65	17837.14	16285.88
k	**6**	**7**	**8**	**9**	**10**
Total distance	12374.82	20227.99	11791.27	8007.47	6788.62

Table 1: The total distances of data points when number of clusters, k, ranges from 1 to 10.

Discussion

In a nutshell, our research question about whether trained models could help diagnose health states of lungs was answered by our analysis. We concluded that the logistic regression model was able to classify an image of lungs into either healthy or Pneumonia with 94.5% of accuracy. Furthermore, it corresponds to our hypothesis that Pneumonia could be further categorized based on the causes or other distinguishing factors. Based on the correspondence between our hypothesis and the conclusion, we could conclude that the analysis techniques used were appropriate for the dataset.

Classification:

The overall performance of Linear regression was high, with accuracy of one-time prediction based on full dimensions is around 0.945 and accuracy of one-time prediction based on principal-dimensions is around 0.953. The result is pretty close, but accuracy of one-time prediction based on principal-dimensions is slightly higher compared to one based on full dimensions. Our assumption to this result is that with dimension-reduced variables, “noises” caused by unimportant information are removed by dimension reduction. The speed of classification was increased a lot with dimension reduced figures while the accuracy was increased. The accuracy of detecting if the lung is infected by Pneumonia is pretty high, which means our model works well as a diagnostic tool.

K-Means Clustering:

This corresponds to our assumption that there are different types of infection to the lung that induces Pneumonia. We were enlightened with this idea when viewing the kaggle website of this dataset because it demonstrated three images of different types of lungs: healthy, viral Pneumonia, and bacterial Pneumonia. Though the dataset itself didn’t make a clear distinction among different subtypes of lungs, we were hinted to delve deeper into this possibility that diseased lungs caused by different types of infections could demonstrate different features. Our analysis supported this hypothesis because we observed that the total distance of all sample points to their corresponding centroids decreased by a huge amount as the number of clusters increased. This result corresponded to the number of possible subcategories of diseased lungs as we already knew there were at least two types of Pneumonia, viral-infected and bacterial-infected.

Future Works:

The dataset we chose did not include pictures of lungs infected by the coronavirus. With a general interest in more accurate diagnosis, the next step to work on could be to include more observations of lungs influenced by COVID. By feeding the trained model with a novel type of lung with potentially distinguishable features, we could potentially train the AI to distinguish coronavirus infection from viral or bacterial infection. Another possible direction is to optimize the algorithm so it would take less time. Currently, we are able to run 2000 images (resized to have 10000 dimensions) for 100 iterations. If k-means clustering could be optimized or a more efficient clustering algorithm could be used, we could input more images or run more iterations on the data to generate more accurate and generalizable predictions.

Works Cited

Paul Mooney. (2017). Chest X-Ray Images (pneumonia). Kaggle. Retrieved from: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia.

Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830.

Project Analysis

December 17, 2020

[42]:
```
import cv2
import glob
import numpy as np
import csv
import pandas as pd
from matplotlib import pyplot as plt
```

[6]:
```
# save images to a csv file, one row per image
# @param imgDir: directory name of the images
# @param csvpath: name of the csv file
# @param number: number of images to save
# @param label: label of the images, 0 for normal, 1 for pneumonia
def imgToCsv(imgDir, csvPath, number, label):
    with open(csvPath, mode='a+') as file:
        writer = csv.writer(file)
        count = 0
        for currImg in glob.glob(imgDir+'/*.jpeg'):
            img = cv2.imread(currImg)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            img = cv2.resize(img, (100, 100))
            img = img.reshape(1, -1)
            imgList = [label]
            for i in range(len(img[0])):
                imgList.append(img[0][i])
            writer.writerow(imgList)
            count += 1
            if count == number:
                break
```

[7]:
```
# clear a csv file
# @param csvPath: name of csv file
def clearCsv(csvPath):
    f = open(csvPath, 'r+')
    f.truncate(0)
    f.close()
```

1 Convert Images to CSV

[16]:
```
imgToCsv('normal', 'xray_images.csv', 1000, 0)
```

[17]:
```
imgToCsv('pneumonia', 'xray_images.csv', 1000, 1)
```

[18]:
```
images = pd.read_csv('xray_images.csv', header=None)
data = images
data.shape
```

[18]:
```
(2000, 10001)
```

2 PCA

[182]:
```
from sklearn.preprocessing import StandardScaler
features = data.loc[:,1:data.shape[1]] #all features
labels = data.loc[:,0] #all labels
features = StandardScaler().fit_transform(features) #standarlize all the features
```

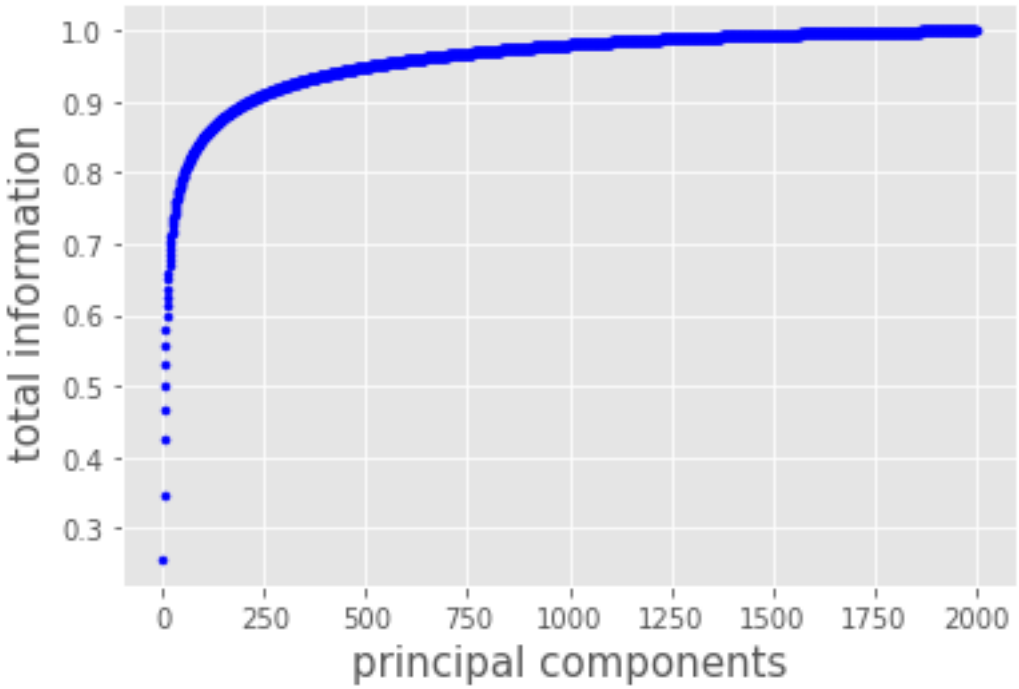
[183]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=features.shape[0])
principalComponents = pca.fit_transform(features)
principalDf = pd.DataFrame(data = principalComponents) #principal components
```

[184]:
```
finalDf = pd.concat([labels, principalDf], axis = 1) #add in the all labels
finalDf.shape
```

[184]:
```
(2000, 2001)
```

[185]:
```
ratio = pca.explained_variance_ratio_ #information(variance) of each of the␣
 ↪principal components
variance = 0
count = 0
for value in ratio: #sum the each component's covered information one by one
    variance += value
    count += 1
    plt.scatter(count, variance, c = 'b', marker = '.') #show the trends of␣
 ↪total information
plt.xlabel('principal components', fontsize = 15)
plt.ylabel(' total information', fontsize = 15)
```

[185]:
```
Text(0, 0.5, ' total information')
```



3 Logistic Regression

[300]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.metrics import log_loss
import timeit
from sklearn.model_selection import cross_val_score
```

3.1 LR on Principal Dimensions

[410]:
```
#70% train, 30% test
x_train, x_test, y_train, y_test = train_test_split(finalDf.iloc[:,1:], finalDf.
↪iloc[:,0], test_size = 0.3, shuffle = True)
```

[411]:
```
start = timeit.default_timer()
LR = LogisticRegression(max_iter=600)
LR.fit(x_train, y_train)
prediction = LR.predict(x_test) #predicton
accuracy1 = LR.score(x_test, y_test) #score of prediction
end = timeit.default_timer()
```

```
duration1 = end - start #running time
print('Accuracy: ', accuracy1)
```

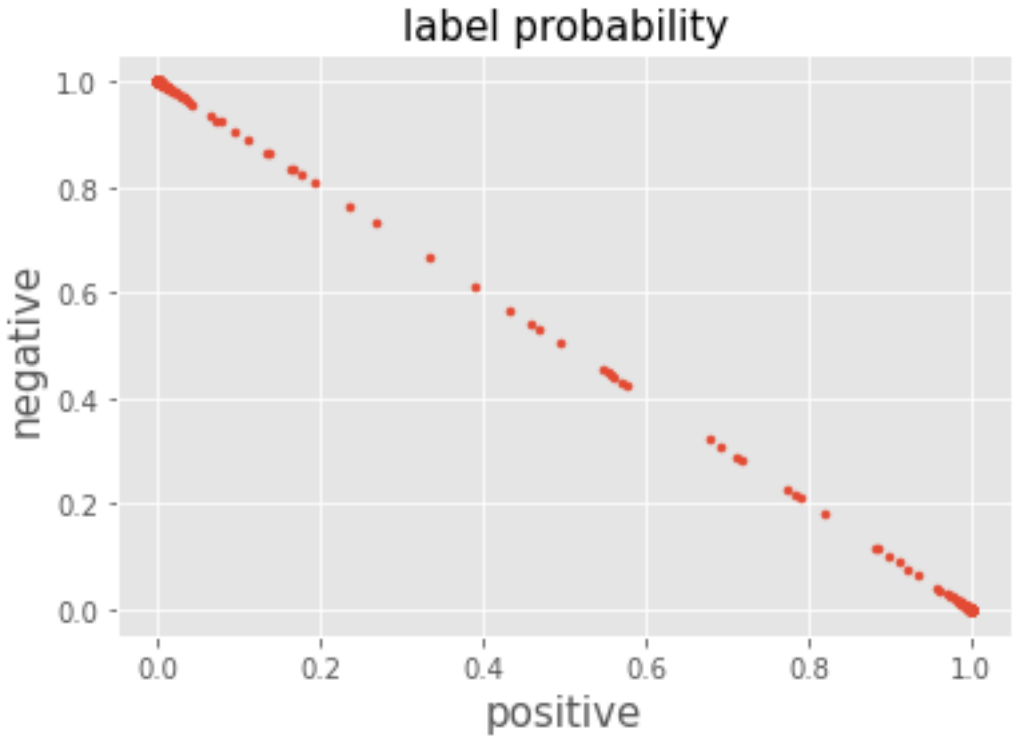
```
Accuracy:  0.9466666666666667
```

[412]:
```
prob = LR.predict_proba(x_test) #probabilities of lable 1 for each prediction
loss1 = log_loss(y_test, prob) #loss of the prediction
print(loss1)
```

```
0.2389197666264811
```

[393]:
```
plt.scatter(prob[:,0], prob[:,1], marker='.')
plt.xlabel('positive', fontsize = 15)
plt.ylabel('negative', fontsize = 15)
plt.title('label probability', fontsize = 15)
```

[393]:
```
Text(0.5, 1.0, 'label probability')
```



[398]:
```
count = 0
for i in range(len(prob)):
    if (prob[i,0] > 0.9) or (prob [i,1] > 0.9):
        count+=1
certainty1 = count/len(prob)
```

```
print('Certainty Rate:', certainty1)
```

```
Certainty Rate: 0.9466666666666667
```

[395]:
```
#10-folds cross validation
scores = cross_val_score(LR, x_train, y_train, cv=10)
scores = pd.Series(scores)
cross_score1 = scores.mean()
print("10-folds Cross Validation Score:", cross_score1)
```

```
10-folds Cross Validation Score: 0.9421428571428571
```

[397]:
```
scores.max()
```

[397]:
```
0.9571428571428572
```

3.2 LR on Full Dimensions

[407]:
```
#70% train, 30% test
x_train, x_test, y_train, y_test = train_test_split(data.iloc[:,1:], data.iloc[:
 ↪,0], test_size = 0.3, shuffle = True)
```

[408]:
```
start = timeit.default_timer()
LR = LogisticRegression(max_iter=600)
LR.fit(x_train, y_train)
prediction = LR.predict(x_test) #prediction
accuracy2 = LR.score(x_test, y_test) #model score
end = timeit.default_timer()
duration2 = end - start #running time
print('Accuracy: ', accuracy2)
```

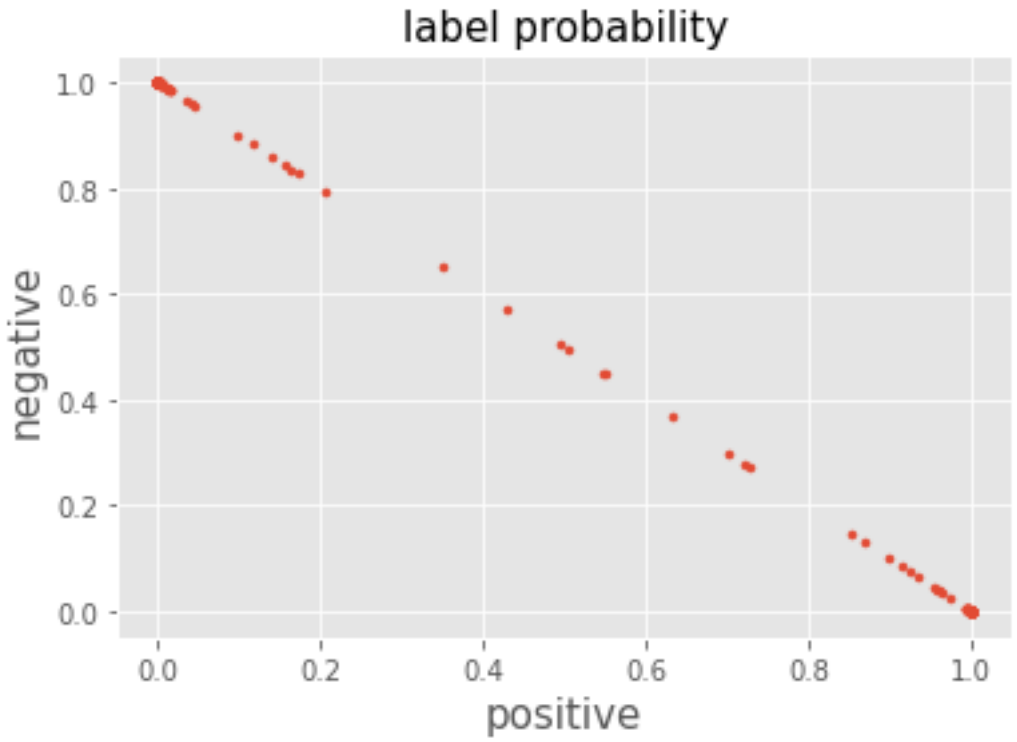
```
Accuracy:  0.955
```

[409]:
```
prob = LR.predict_proba(x_test) #probabilities of lable 1 for each prediction
loss2 = log_loss(y_test, prob) #loss of the prediction
print(loss2)
```

```
0.31411028597081464
```

[385]:
```
plt.scatter(prob[:,0], prob[:,1], marker='.')
plt.xlabel('positive', fontsize = 15)
plt.ylabel('negative', fontsize = 15)
plt.title('label probability', fontsize = 15)
```

[385]:
```
Text(0.5, 1.0, 'label probability')
```



[389]:
```
count = 0
for i in range(len(prob)):
    if (prob[i,0] > 0.9) or (prob [i,1] > 0.9):
        count+=1
certainty2 = count/len(prob)
print('Certainty Rate:', certainty2)
```

```
Certainty Rate: 0.9683333333333334
```

[375]:
```
#10-folds cross validation
scores = cross_val_score(LR, x_train, y_train, cv=10)
scores = pd.Series(scores)
cross_score2 = scores.mean()
print("10-folds Cross Validation Score:", cross_score2)
```

```
10-folds Cross Validation Score: 0.9357142857142857
```

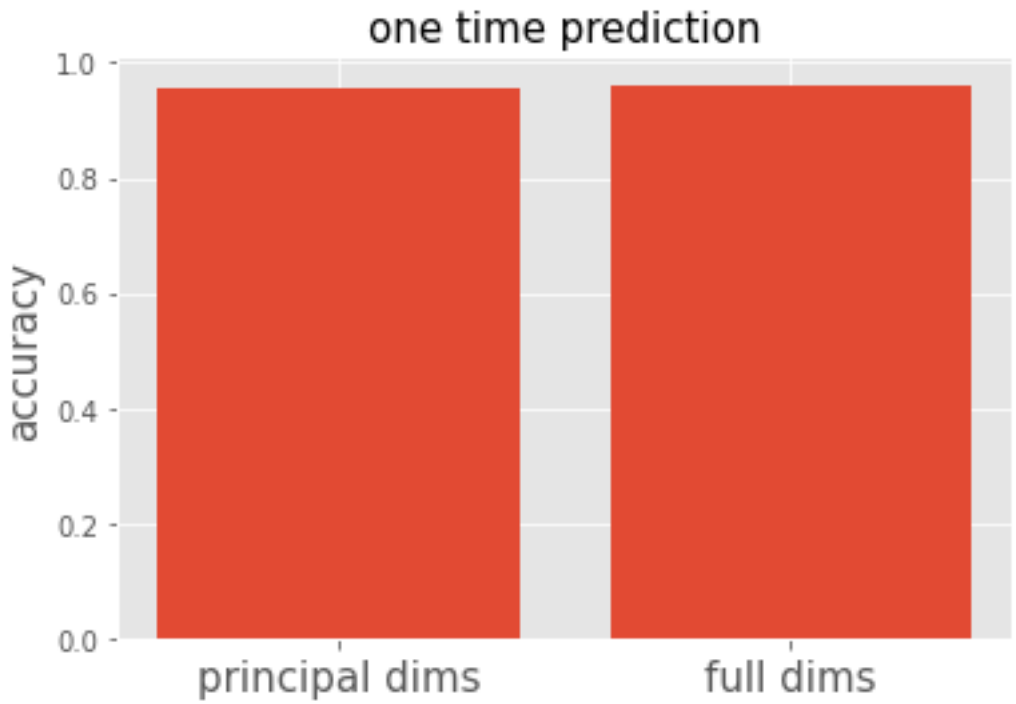
[379]:
```
scores.max()
```

[379]:
```
0.9571428571428572
```

3.3 Comparisons

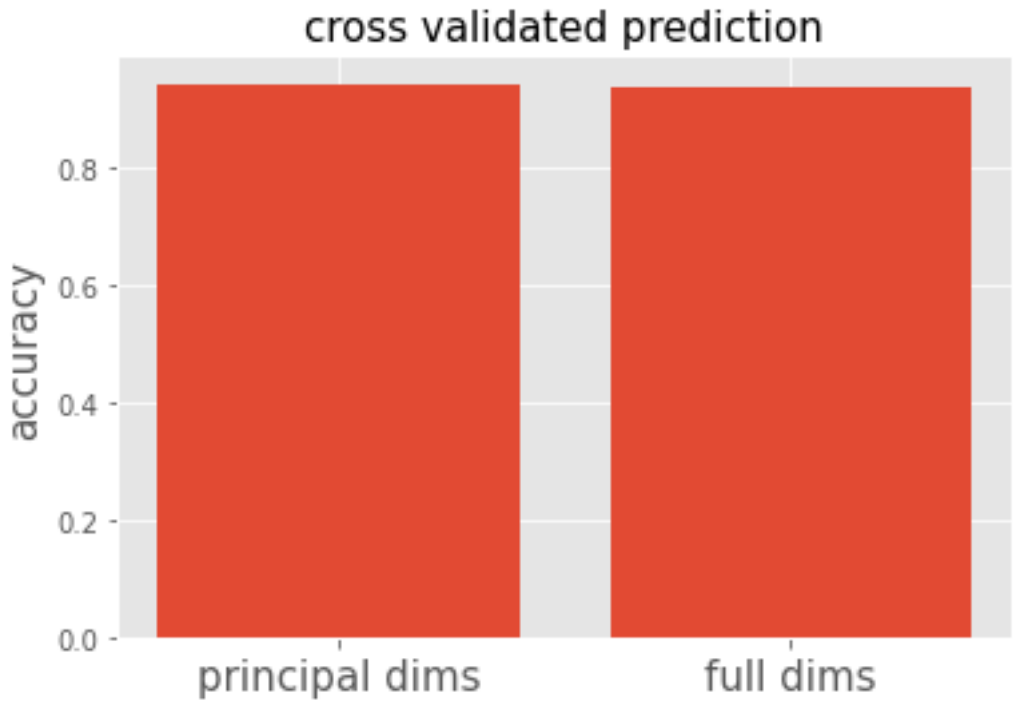
[368]:
```
#show accuracies of two modeles
x = [1,2]
plt.bar(x, height=[accuracy1, accuracy2])
plt.xticks(x, ['principal dims','full dims'], fontsize = 15)
plt.ylabel('accuracy', fontsize = 15)
plt.title('one time prediction', fontsize=15)
```

[368]:
```
Text(0.5, 1.0, 'one time prediction')
```



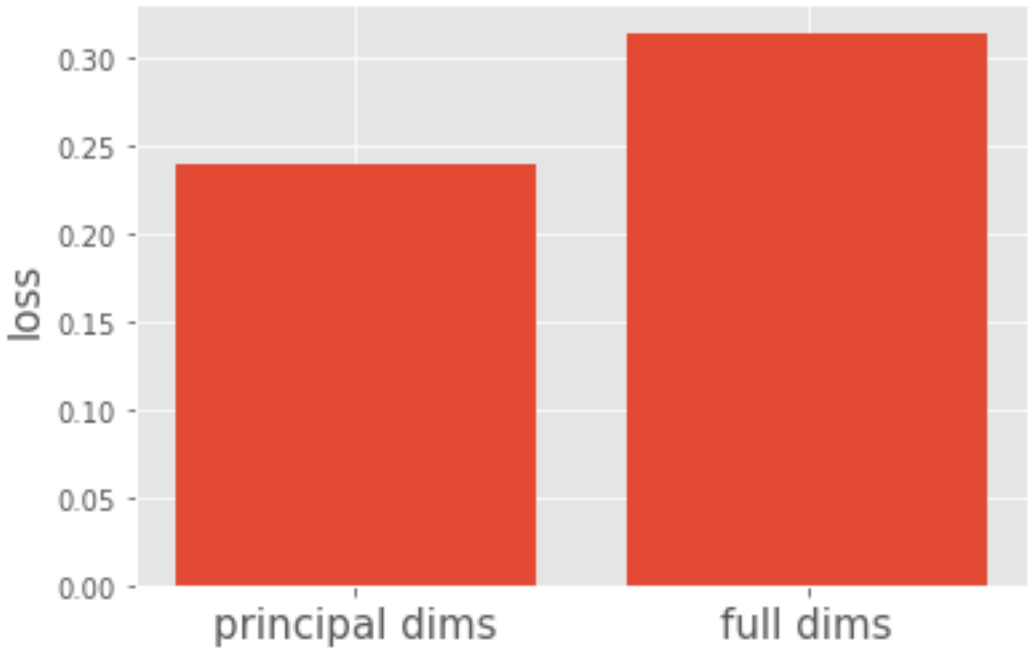
[416]:
```
#show validated accuracies of two modeles
x = [1,2]
plt.bar(x, height=[cross_score1, cross_score2])
plt.xticks(x, ['principal dims','full dims'], fontsize = 15)
plt.ylabel('accuracy', fontsize = 15)
plt.title('cross validated prediction', fontsize = 15)
```

[416]:
```
Text(0.5, 1.0, 'cross validated prediction')
```



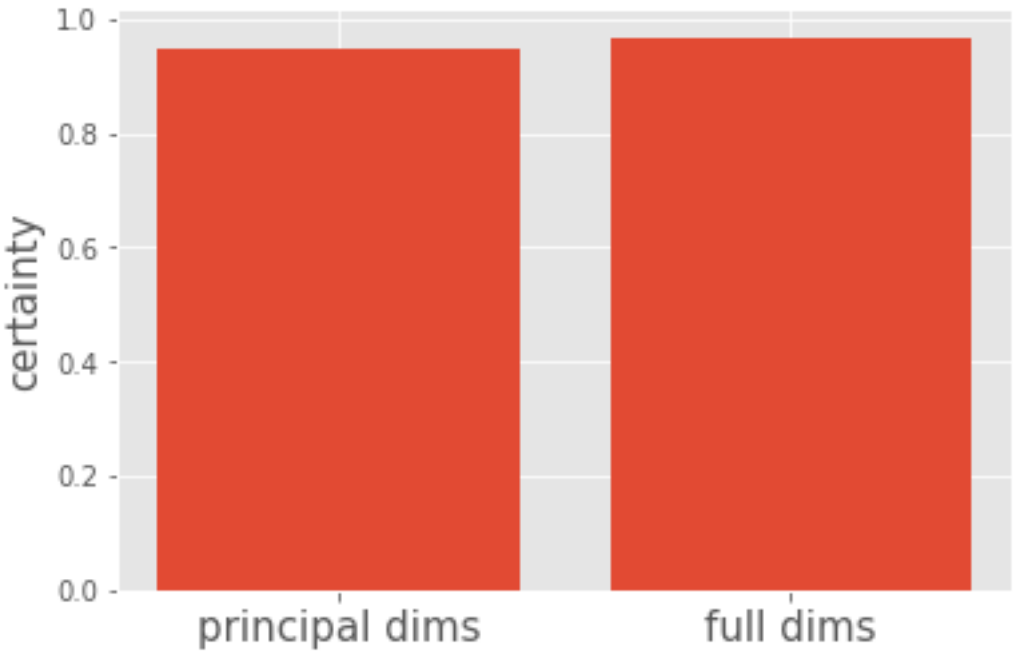
[413]:
```
#show loss of two modeles
x = [1,2]
plt.bar(x, height=[loss1,loss2])
plt.xticks(x, ['principal dims','full dims'], fontsize = 15)
plt.ylabel('loss', fontsize = 15)
```

[413]: Text(0, 0.5, 'loss')



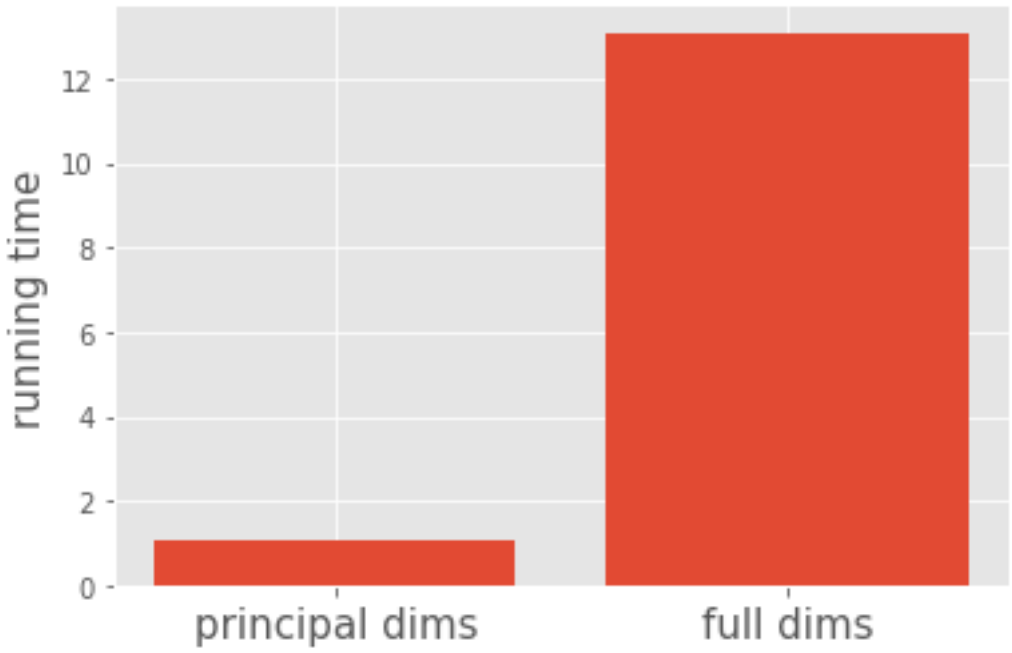
[414]:
```
#show certainty of two modeles
x = [1,2]
plt.bar(x, height=[certainty1,certainty2])
plt.xticks(x, ['principal dims','full dims'], fontsize = 15)
plt.ylabel('certainty', fontsize = 15)

#explaination: certainty does not mean accuracy, false positive happen on full␣
 ↪dims
#(certain on incorrect predictions)
```

[414]:
```
Text(0, 0.5, 'certainty')
```



[418]:
```
#show running time of two modeles
x = [1,2]
plt.bar(x, height=[duration1, duration2])
plt.xticks(x, ['principal dims','full dims'], fontsize = 15)
plt.ylabel('running time', fontsize = 15)
print('Running time on pricipals: ', duration1)
print('Running time on full dimentions: ', duration2)
print('Ratio: ', duration1/duration2)
```

```
Running time on pricipals:  1.0572169160004705
Running time on full dimentions:  13.088632651022635
Ratio:  0.08077367164230624
```



4 K-Means

[196]:
```
%matplotlib inline
import numpy as np
from scipy.io import loadmat
from matplotlib import pyplot as plt
import random

plt.style.use('ggplot')
```

[216]:
```
X = principalDf.loc[1000:2000, :]
# print(X.shape)
C = X.iloc[np.random.randint(X.shape[0], size=2)]
C.shape
clus = np.sqrt(sum((X.iloc[1]-C.iloc[0])**2))
clus
len(X.index)
```

[216]: 1000

[217]:
```
def k_mean_total_dis(k,it):
    C = X.iloc[np.random.randint(X.shape[0], size=k)] #random centers
    for itr in range(it):
```

```
        cluster_ind = np.zeros(len(X.index))
        distance = np.zeros((len(X.index), k))
        # find distance of every point to each centroid, and cluster membership
        for k_pos in range(k):
            for i in range(len(X.index)):
                clus = np.sqrt(sum((X.iloc[i]-C.iloc[k_pos])**2))
                distance[i,k_pos] = clus
        cluster_ind = np.argmin(distance,axis=1)
        for k_pos in range(k):
            ind = []
            val = []
            summ = []
            dist = []
            alldist = []
            for i in range(len(X.index)):
                if (cluster_ind[i]==k_pos):
                    ind.append(i)
                    val.append(X.iloc[i])
                    dist.append(np.sqrt(sum((X.iloc[i]-C.iloc[k_pos])**2)))
            alldist.append(sum(dist))
            if (len(ind)!=0):
                summ = sum(val)
                mean = summ/len(ind)
                C.iloc[k_pos] = mean
    return alldist
```

[238]:
```
totalDist = []
for k in range(1,11):
    dist = k_mean_total_dis(k,100)
    totalDist.append(dist)
```

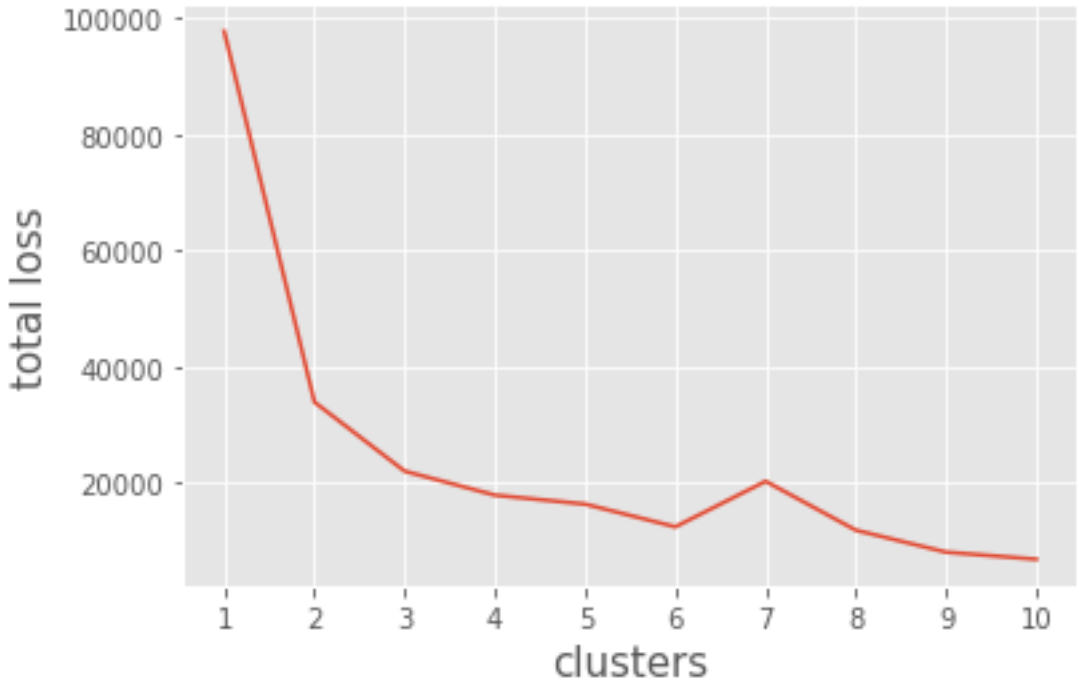
```
/Users/Versace/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/indexing.py:671: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_with_indexer(indexer, value)
<ipython-input-217-1b6696683be1>:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  C.iloc[k_pos] = mean
```

[239]:
```
totalDist
```

[239]:
```
[[97780.2643317914],
 [33877.72215992445],
 [21965.64507227402],
 [17837.139405829228],
 [16285.884768418224],
 [12374.82096652662],
 [20227.99248440403],
 [11791.262577937614],
 [8007.468390321226],
 [6788.617367098232]]
```

[330]:
```
k = np.arange(1, 11)
plt.plot(k,totalDist)
plt.xticks(np.arange(1,11))
plt.xlabel('clusters', fontsize=15)
plt.ylabel('total loss', fontsize=15)
```

[330]:
```
Text(0, 0.5, 'total loss')
```



[ ]: