School of Engineering and Applied Science (SEAS), Ahmedabad University

B.Tech (CSE Semester VI)/M.Tech/PhD:
Machine Learning (CSE 523)

Project Submission #3: Principal Component Analysis

Submission Deadline: April 05, 2020 (11:59 PM)

- **Group No.: B_NLP5**
- **Project Area: Natural Language Processing**
- **Project Title: Harnessing Twitter 'Big Data' for Automatic Emotion Identification**
- **Name of the group members :**
  (a) Jay Patel(AU1741018)
  (b) Manav Shah(AU1741042)
  (c) Prima Sanghvi(AU1741045)
  (d) Priyanshi Deliwala(AU1741047)

PRINCIPAL COMPONENT ANALYSIS:

Case 1:
Dimensionality of the dataset is smaller than the number of samples.

Our unprocessed Twitter datasets contains 0.5 Million tweets text extracted directly from twitter API and exactly as posted on twitter.The dataset is supervised, the tweets have been annotated(1=Joy, 2=Fear, 3= Anger, 4= Surprised, 5=Love, 6=Sadness) which is used to detect the emotion.
After importing and preprocessing the respective dataset we cleaned our dataset by removing,

- Punctuation
- Preposition
- Conjunction
- Stop Words

And then after tokenizing and lemmatizing the dataset, we extracted the feature matrix using **TF-IDF Transform** and **CountVectorizer**. We obtained in total 300 features for 12000 rows. Hence, the dimentionality of our feature matrix would be 12000X300.After the feature matrix is extracted we normalize the feature matrix and calculate the covariance matrix as,

$S = \frac{1}{N}\sum_{n=1}^{N} x_n x_n^T$

The Covariance matrix helps us to understand the variance among the data and the diagonal elements represents the variance whereas the non-diagonal elements represents the correlation of the samples with each other having dimensions 300x300. We found eigen value and its corresponding eigen vectors using linalg.eig function and sorted the eigen-values in decreasing order.

From analysis we know that,

$$V_m = \lambda_m$$

The variance of the data, when projected onto an M-dimensional subspace, equals the sum of the eigenvalues associated with corresponding eigen vectors.

Next, to find the projection matrix we assign a matrix B with top eigenvectors corresponding to M greatest eigen values. The projection matrix is as follows:

$$P = B.B^T$$

The dimension of projection matrix is 300x300.

Now the reconstructed matrix obtained by multiplying X with projection matrix is:

$$\hat{X} = BB^T X$$

The dimension of the projected reconstructed matrix $\hat{X}$ is

Now, if the eigen values were unsorted the top eigen values won't have maximum variance hence the first principal component won't be same as top principal component with highest variance.The results for the same have been attached below.

Case 2:

In order to do Principal Component Analysis, we need to compute the data covariance matrix. In D dimensions, the data covariance matrix is a $D \times D$ matrix. Computing the eigenvalues and eigenvectors of this matrix is computationally expensive as it scales cubically in D. Therefore, PCA, as we discussed earlier, will be infeasible in very high dimensions. For example, if our x_n are images with 10,000 pixels (e.g., 100×100 pixel images), we would need to compute the eigen-decomposition of a $10,000 \times 10,000$ covariance matrix. In the following, we provide a solution to this problem for the case that we have substantially fewer data points than dimensions, i.e., $N \ll D$.

Assume we have a centered dataset $x_1, x_2, x_3, ..., x_n \in \mathbb{R}^{D \times D}$. Then the data covariance matrix is given as

$$S = \frac{1}{N} X X^T \in \mathbb{R}^{D \times D}$$

where $X = [x_1, x_2, ..., x_n]$ is a $D \times N$ matrix whose coloumns are data points.

As we have assumed that $N \ll D$, i.e the number of data points is smaller than the dimensionality of the data. If there are no duplicate points, the rank of the covariance matrix S is N, so it has D - N + 1 many eigenvalues that are 0. This results into a sparse matrix.

1. **Implementation code:**

```
#Required Libraries
import nltk
import pandas as pd
import numpy as np
import re
from gensim.models import Word2Vec
from io import StringIO
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer, TfidfTransf
from sklearn.feature_selection import chi2
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
import seaborn as sns
from sklearn.svm import LinearSVC
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_val_score

#""" Importing Required Packages"""

import nltk
nltk.download('wordnet')

import nltk
nltk.download('punkt')

import nltk
nltk.download('averaged_perceptron_tagger')

import nltk
nltk.download('stopwords')

from google.colab import drive
drive.mount('/content/drive')

#""" Data Loading and Preprocessing
1.Removed the words other than A-Z and a-z and space

---

2.Tokenize the words

---

3.Convert the tokenized words into a list
```

```
# importing the stop words
stop_words = set(nltk.corpus.stopwords.words('english'))

lemmatizer = nltk.stem.WordNetLemmatizer()

dt = []

# importing the sample dataset from Emotion_Phrases.csv
an = open('/content/drive/My Drive/Dataset/ANGER_Phrases.txt','r')
anger = an.readlines()
an.close()
for i in anger:
    dt.append([i, 'anger'])

fe = open('/content/drive/My Drive/Dataset/FEAR_Phrases.txt', 'r')
fear = fe.readlines()
fe.close()
for i in fear:
    dt.append([i, 'fear'])

jo = open('/content/drive/My Drive/Dataset/JOY_Phrases.txt', 'r')
joy = jo.readlines()
jo.close()
for i in joy:
    dt.append([i, 'joy'])

lo = open('/content/drive/My
Drive/Dataset/LOVE_Phrases.txt','r')

love = lo.readlines()
lo.close()

for i in love:
    dt.append([i, 'love'])

sa = open('/content/drive/My Drive/Dataset/SADNESS_Phrases.txt','r')
sad = sa.readlines()
sa.close()
for i in sad:
    dt.append([i, 'sad'])

su = open('/content/drive/My Drive/Dataset/SURPRISE_Phrases.txt','r')
surprise = su.readlines()
su.close()
for i in surprise:
    dt.append([i, 'surprise'])

print(len(dt))
```

```
data = pd.DataFrame(dt)

# adding the coloumns names
data.columns = ['text', 'emotions']

# getting the informations of data
data.info()

# converting the emotions to the ids
data['emo_id'] = data['emotions'].factorize()[0]

# print(data.head(10))
# print(data['emo_id'])
#data.iloc[np.random.permutation(len(data))]
#data.reindex(np.random.permutation(data.index))

data = data.sample(frac=1)

#print(data)



# dropped the duplicates and sorted it
emo_id_df = data[['emotions', 'emo_id']].drop_duplicates().sort_values('emo_id')

# converted it into a dictionary
emo_to_id = dict(emo_id_df.values)

# representing the dataset with a plot


# converted the data into the dictionary
data_dict = data.to_dict()

sw_rem = []
txt = []
tagged = []

# preprocessed the database:
# * Removed the words other than A-Z and a-z and space
# * Tokenize the words
# * Convert the tokenized words into a list
tagged = []
print(list(data_dict['text'].values())[:5])
for k in range(len(list(data_dict['text'])[:12000])):
    data_dict['text'][k] = re.sub('([^A-Za-z ])', '', data_dict['text'][k])
    txt.append(nltk.tokenize.word_tokenize(data_dict['text'][k]))
    tagged.append(nltk.pos_tag(txt[k]))
    for j in range(len(txt[k])):
        if (txt[k][j] not in stop_words) and (tagged[k][j][1] not in ['CC', 'CD', 'IN', '
           sw_rem.append(lemmatizer.lemmatize(txt[k][j]))
    data_dict['text'][k] = " ".join(sw_rem)
    sw_rem = []
print(list(data_dict['text'].values())[:5])
data = pd.DataFrame.from_dict(data_dict)
#print(data)

"""##Representation of dataset with respective Emotions"""

fig = plt.figure()
data.groupby('emotions').text.count().plot.bar(ylim=0)
plt.show()

"""##Extracting features
1.   CountVectorizer:   We segment each text file into words. count no. of times each wor
2.   TF:   CountVectorizer will give more weightage to longer documents than shorter docu
3. TF-IDF:    Finally, we can even reduce the weightage of more common words like (the, i

count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(data.text[:12000])
#print(X_train_counts)
#print(X_train_counts.shape)


tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
from scipy import sparse
X_train_tfidf = sparse.lil_matrix(sparse.csr_matrix(X_train_tfidf))
#print(X_train_tfidf)
print(X_train_tfidf.shape)

"""##Converting Sparse Matrix to Dense Matrix"""

labels = data.emo_id
#print(X_train_tfidf)
#print(features.get_shape())
features_d = X_train_tfidf.toarray()
#print(features_d)
rows = len(features_d)
columns = len(features_d[0])
print(rows)
print(columns)
############
X_train, X_test, y_train, y_test, indices_train, indices_test = train_test_split(X_train_

"""##PCA Analysis
```

```
1. Normalize Function"""

def normalize(X):
 mu = np.mean(X, axis=0)
 std = np.std(X, axis=0)
 a = np.subtract(X,mu)
 new_std = np.where(std>0,std,1)
 Xbar = np.divide(a,new_std)
 # std_filled = std.copy()
 # std_filled[std==0] = 1.
 return Xbar, mu, std

"""##2. Eig Function
It will compute :
1.   Eigen Values
2.   Eigen Vector """

def eig(S):
 eigvals, eigvecs = np.linalg.eig(S)
 idx = eigvals.argsort()[::-1]
 eigvals = eigvals[idx]
 eigvecs = eigvecs[:,idx]
 return (eigvals, eigvecs)

"""##3. Projection_Matrix Function
Computes the projection matrix onto the space spanned by `B`
"""

def projection_matrix(B):
 P = np.matmul(B,B.T)
 #return np.eye(B.shape[0]) # <-- EDIT THIS to compute the projection matrix
 return P

"""##4. PCA Function
*   Input
      1.   X: ndarray of size (N, D), where D is the dimension of the data, and N is the
      2.   num_components: the number of principal components to use.
*   Returns
      1.   X_reconstruct: ndarray of the reconstruction
  of X from the first `num_components` principal components.
      2.   sum_value: sum of eigen values from the first `num_components` principal compo
"""

def PCA(X, num_components):
 Xbar, mu, std = normalize(X)
 covariance = np.dot(Xbar.T,Xbar)
 S = covariance
 eigvals, eigvecs = eig(S)
 # eigen_trans = eigvecs.T
```

```
  # B = np.stack(eigen_trans[:,:num_components])
  sum_value = sum(eigvals[:num_components])
  B = np.stack(eigvecs[:,:num_components])
  P = np.matmul(B,B.T)
  X_reconstruct = np.matmul(P,X.T)
  X_reconstruct = X_reconstruct.T
  return X_reconstruct, sum_value

"""#5. Normalizing Input Matrix
Normalization is necessary to make every variable in proportion with each other
"""

NUM_DATAPOINTS = 12000
data1 = (features_d [:NUM_DATAPOINTS]) / 255.
#print(data1)
Xbar, mu, std = normalize(data1)
print(Xbar.shape)
print(mu.shape)
print(std.shape)
#print(np.trace(Xbar))

"""## 6. Computing Covariance Matrix"""

covariance = np.matmul(Xbar.T,Xbar)
print(covariance.shape)
print(np.trace(covariance))
S = covariance
eigvals, eigvecs = eig(S)
print(eigvals[:20])
print(eigvecs)
print(eigvecs.shape[0])

num_components = 150

"""## 7. Computing Projection Matrix"""

B = np.stack(eigvecs[:,:num_components])
print('Shape of matrix B: ', B.shape)
print('Trace of matrix B: ', np.trace(B))

P = projection_matrix(B)  # finding projection matrix P
print('Shape of Projection matrix : ', P.shape)

X_reconstruct ,sum_value = PCA(Xbar, num_components)            # finding reconstructed X
print('Shape of X_reconstruct matrix : ', X_reconstruct.shape)  # printing shape of X_rec
print('Shape of normalized X matrix : ', Xbar.shape)            # printing shape of norma

print()
print('X reconstruced : \n', X_reconstruct)
```

```
"""## 8. Our PCA function Vs sklearn library"""

from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler(with_mean=True,with_std=True)
standardized_data = standardized_data.fit_transform(Xbar)

print('Shape of Normalized input matrix computed using library : ' , standardized_data.sh
print('Shape of Normalized input matrix computed using user functions : ' , Xbar.shape)


sample_data = standardized_data
covar_matrix = np.matmul(sample_data.T, sample_data)

print("Shape of covariance matrix computed using Library = ", covar_matrix.shape)
print("Shape of covariance matrix computed by user functions = ", S.shape)

print('The covariance matrix is : \n' ,covar_matrix)
print('The trace of covariance matrix : ', np.trace(covar_matrix))

"""## 9. Analysis of number of principal component"""

for num_component in range(1, 5):
 from sklearn.decomposition import PCA as SKPCA
# We can compute a standard solution given by scikit-learn's implementation of PCA
 pca = SKPCA(n_components=num_component, svd_solver='full')
 sklearn_reconst = pca.inverse_transform(pca.fit_transform(Xbar))

 reconst ,sum_value = PCA(Xbar, num_component)
 np.testing.assert_almost_equal(reconst, sklearn_reconst)
 print(np.square(reconst - sklearn_reconst).sum())

"""## 10. Computing loss and variance for every number of principal components"""

def mse(predict, actual):
 return np.square(predict - actual).sum(axis=1).mean()

loss = []
reconstructions = []
variance_values = []
print(sum_value)
# iterate over different numbers of principal components, and compute the MSE
for num_component in range(1, 140):
    reconst,sum_value = PCA(Xbar, num_component)
    error = mse(reconst, Xbar)
    reconst = reconst*std + mu
    reconstructions.append(reconst)
    # print('n = {:d}, reconstruction_error = {:f}'.format(num_component, error))
    loss.append((num_component, error))
    variance_values.append((num_component, sum_value))
reconstructions = np.asarray(reconstructions)
reconstructions = reconstructions * std + mu # "unnormalize" the reconstructed image
#print(reconstructions)
print(reconstructions.shape)
loss = np.asarray(loss)

import pandas as pd
# create a table showing the number of principal components and MSE
pd.DataFrame(loss).head(100)

"""## 11. Plotting MSE Vs Principal Components"""

fig, ax = plt.subplots()
ax.plot(loss[:,0], loss[:,1]);
ax.axhline(105, linestyle='--', color='r', linewidth=2);
ax.xaxis.set_ticks(np.arange(1, 130, 5));
ax.yaxis.set_ticks(np.arange(100, 300, 30));
ax.set(xlabel='num_components', ylabel='MSE', title='MSE vs number of principal component

"""## 12. Plotting Explained Variance ratio Vs Principal Components"""

tot = sum(eigvals)
var_exp = [(i / tot)*100 for i in eigvals] # computing variance

plt.figure(figsize=(6, 4))#plotting results

plt.bar(range(len(var_exp)), var_exp, alpha=1 , align='center', label='individual explain
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.xlim(0,100)
plt.legend(loc='best')
plt.tight_layout()

# taking random value of M for analysis
print('Eigen values sorted in decreasing order:')
for i in eigvals:
    print(i)
print('Top M (where M=2) Eigenvalues : ', eigvals[:num_components])              # printin
print()
print('M eigenvectors correspondong to top M eigenvalues: \n', eigvecs[:,:num_components]

print()
print('Shape of one eigenvector : ', eigvecs.shape[0])

"""## 13. Plotting Eigen Values vs index (decreasing order)"""

# Plotting sorted eigen values (in decreasing order) v/s its index
eigvals = np.asarray(eigvals)      # converting to array
```

```
eig_df = pd.DataFrame(eigvals)     # making a dataframe with index and eigenvalues
eig_df.rename(columns={0 : 'eigenvalues'},  inplace = True)  #renaming the column of data
print(eig_df.head())        # printing all the eigenvalues ( 1 to D (total number of feat

fig, ax = plt.subplots()
ax.plot(eigvals);
ax.xaxis.set_ticks(np.arange(1, 5, 5));
ax.set(xlabel='index', ylabel='Eigen values', title='Eigen Values vs index (decreasing or
plt.xlim(0,220)

"""## 14. Plotting Variance on num_components Vs Number of Principal Components"""

variance_values = np.asarray(variance_values) # converting variance into variance
var_df = pd.DataFrame(variance_values) #convrting variance into a data frame
var_df.rename(columns={0 : 'num_components', 1: 'Variance'},  inplace = True)  # renaming
print(var_df.head(150))

sns.set()
fig, ax = plt.subplots()
ax.plot(variance_values[:,0], variance_values[:,1]);
ax.xaxis.set_ticks(np.arange(1, 5, 5));
ax.set(xlabel='num_components', ylabel='Variance', title='Variance on Number of Principal

for l in ax.lines:
    plt.setp(l,linewidth=4)

"""## 15. Analysis of MSE Error vs No. of principal components with non sorted eigenvalue

def PCA_unsorted(X, num_components):
  Xbar, mu, std = normalize(X) #normalizing our input matrix
  covariance = np.dot(Xbar.T,Xbar) # covariance
  S = covariance # s is covariance
  eigvals, eigvecs = np.linalg.eig(S) # calculating eigen values and eigen vectors
  for i in eigvals:
    print(i)
  sum_value = sum(eigvals[:num_components])#sum of all components, we have NOT TAKEN SORT
  B = np.stack(eigvecs[:,:num_components]) # calculating matrix B by taking summation of
  P = np.matmul(B,B.T) #P = B * Btranspose
  X_reconstruct = np.matmul(P,X.T) #reconstructing matrix X
  X_reconstruct = X_reconstruct.T
  return X_reconstruct, sum_value

loss_unsorted = []
reconstructions_unsorted = []
variance_values_unsorted = []

data2 = (features_d [:NUM_DATAPOINTS]) / 255.
#print(data1)
Xbar1, mu1, std1 = normalize(data2)
```

```
# iterate over different numbers of principal components, and compute the MSE
for num_component in range(1, 130):
  reconst, sum_value = PCA_unsorted(Xbar1, num_component)  #reconst contains reconstructe
  error = mse(reconst, Xbar1)
  reconst = reconst*std1 + mu1 #reconst is un-normalized data
  reconstructions_unsorted.append(reconst) # appending un-normalized data
  print('M = {:d}, reconstruction_error = {:f}'.format(num_component, error))
  loss_unsorted.append((num_component, error)) # loss for unsorted un normalized data
  variance_values_unsorted.append((num_component, sum_value)) # variance for unsorted un

"""## Plotting MSE vs No. of Principal components(Unsorted)"""

loss_unsorted = np.asarray(loss_unsorted) # loss unsorted as an array
loss_df = pd.DataFrame(loss_unsorted) # as an dataframe
loss_df.rename(columns={0 : 'num_components', 1: 'MSE'},  inplace = True)  #renaming the
print(loss_df)

fig, ax = plt.subplots()
ax.plot(loss_unsorted[:,0], loss_unsorted[:,1]);
ax.xaxis.set_ticks(np.arange(1, 5, 5));
ax.set(xlabel='num_components', ylabel='MSE', title='MSE vs no. of principal components(U

for l in ax.lines:
    plt.setp(l,linewidth=4)
```

2. **URL links:** Datasets and Code links
   click here

3. **Inference:**
Using the Principal Component analysis tool we have reduced a large set of variables to a small set of variables which still retain most of the information of the original dataset. In feature selection all the specified tags are may not be useful.If all the tags are considered than the feature space increases exponentially, as a result the number of tweets required increases. Considering the more frequently occurred words as features. After applying PCA(Principal Component Analysis) the feature space no longer contains all the words but instead contains frequently occurred sentiment carrying words.Hence, as a result reduces the size of the dataset without losing the information.

Our task is to identify the emotions from a tweet, so the data besides the text is irrelevant.The collected data was labelled dataset.

Feature generation of the Tweet:
Feature generation process includes processing the tweet text.

Text Preprocessing:

- Convert to Lowercase: Convert all characters from the content to lowercase.
- Remove Redundancies: Remove singe reiterations(for example heeey =¿ hey).
- Remove Stop words: Remove regular stop words.
- Remove punctuation or Tokenization: Remove the common punctuation marks such as "." or ","

After the text preprocessing, we obtained the feature matrix using Bag of Words and Tf-Idf model.
Bag of Words Model: A vector space model is essentially a mathematical model to represent unstructured information as numeric vectors, with the end goal that each element of the vector is a particular feature. The bag of words model which keeps a count of the total occurrences of most frequently used words.
Tf-Idf Model: Tf-Idf term stands for term frequency-inverse document frequency. Frequency indicated the number of time the word w is occurred in dataset d. We used to represent the vector using bag of words model because the order of words is not important. Thenafter, we computed the inverse document frequency which is the frequency of dataset d of the word w.
After implementing the model we obtain the feature matrix which we further normalize for mathemtical implementation of PCA(Principal Component Analysis)
Computation of Principal Component Analysis in steps:

- Import Data
- Standardize each column
- Compute Covariance Matrix
- Compute Eigen values and Eigen Vectors
- Derive Principal Component Features by taking dot product of eigen vector and standardized columns
- Reconstruction of the matrix.

Representation of Samples of Dataset as different Emotions :

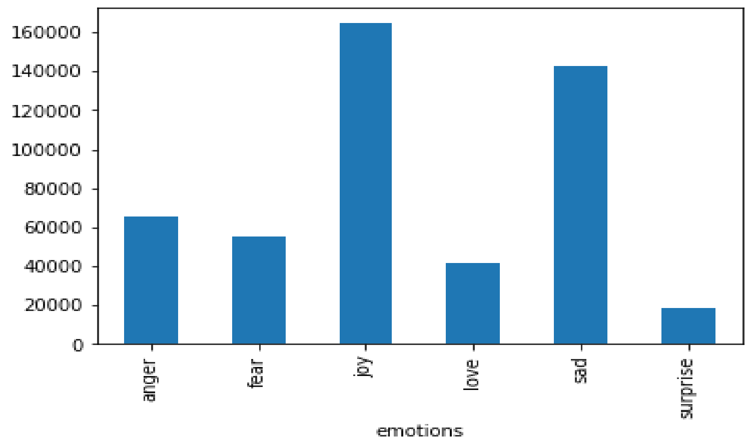

Figure 1: Samples Vs Emotions

Results :

- **Sorted Eigenvalues in decreasing order:**

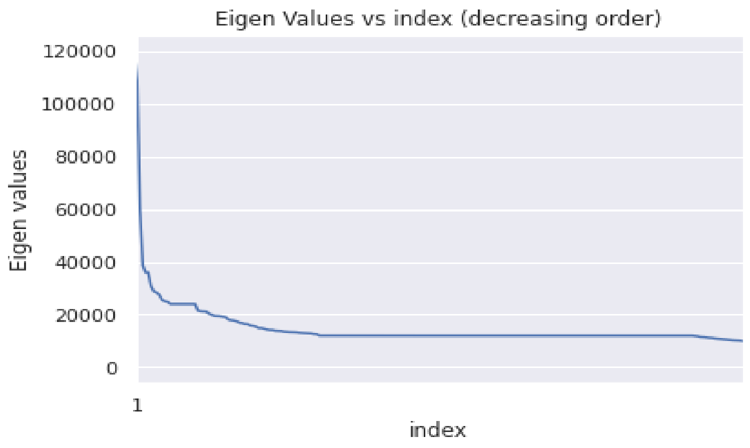

Figure 2: eigen values vs index

The above graph shows the sorted eigenvalues in decreasing order. We have 300 features, and corresponding to that we have 300 eigenvalues and 300 corresponding eigenvectors found from the covariance matrix S. We sort the eigenvalues in decreasing order and eigenvectors correspondingly.Hence, higher the eigenvalue; higher will be the variance presented by the eigenvector.

- **Maximum Captured Variance:** The above graph shows the maximum amount of variance with respect to principal components. Maximum captured variance is characterized by the aggregate of the eigenvalues of the principal components, it means as we go to the right, number of principal components increases and hence, change would likewise

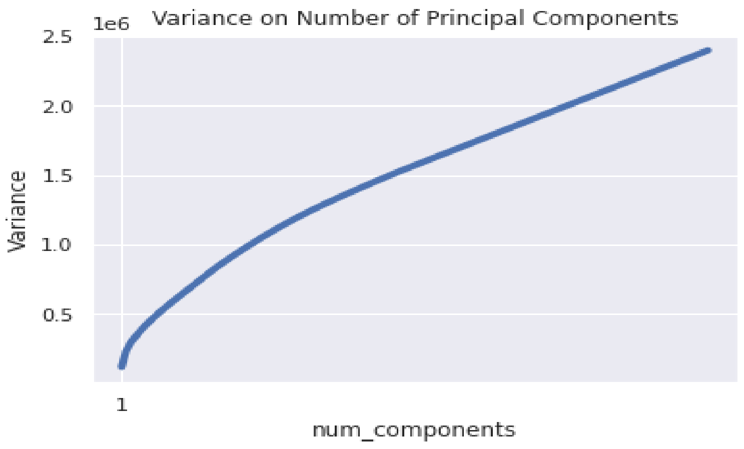

Figure 3: Variance

increment as the quantity of variance would also increase as the number of eigenvalues in the addition is increasing. But since, eigenvalues are sorted in decreasing order, at first there is a rapid increase in the captured variance due to high eigenvalues, whereas as we go to the right, the eigenvalues that are adding have less value, henceforth it is then expanding yet at a more slow rate.

- **MSE vs Number of PCA(Principal component Analysis) for sorted eigenvalues:** The above graph shows that the mean sqaured error with respect to the number

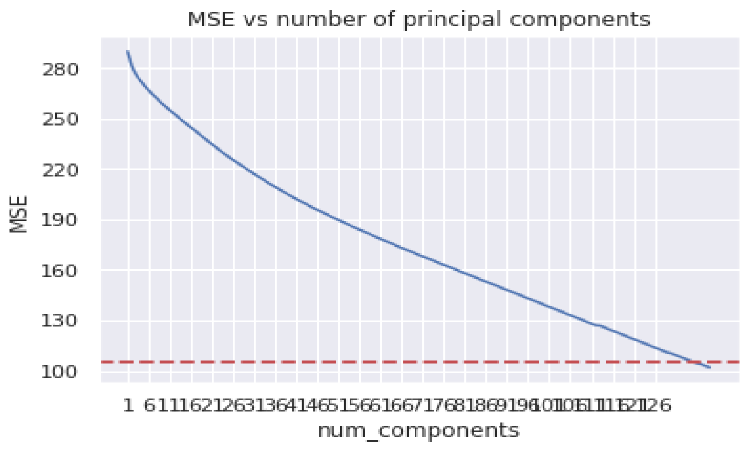

Figure 4: MSE vs Principal Components(sorted)

of principal component. The explanation is that as we take increasingly more principal components, there is less loss in the dimensions and hence, there is less misfortune in the data. Therefore, its natural that error will decrease as we take more principal components.

- **MSE vs Number of PCA(Principal Component Analysis) with unsorted eigenvalues:** The above graph shows the mean squared error with respect to the number of principal components. The MSE error is computed between projected data matrix and the original data. The eigen values are non-ordered and not sorted before applying PCA to compute the projected data matrix and computing the MSE error. Since, there are 300 features the features are significantly large and the eigenvalues is

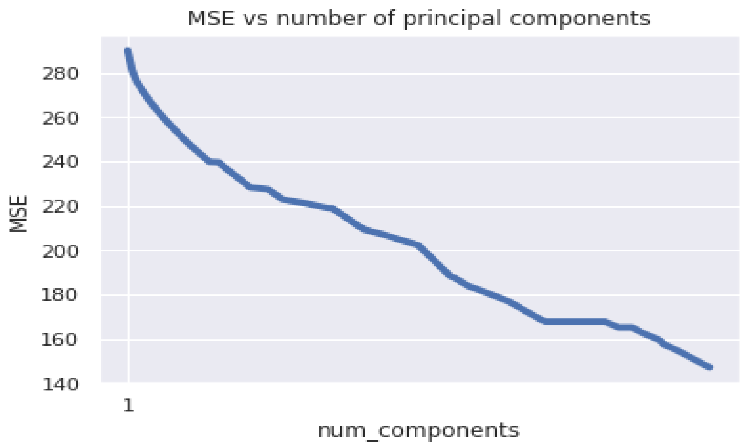

Figure 5: MSE vs Principal Components(unsorted)

unordered, the MSE error first decreases then for some values becomes constant and then again decreases.

- **Variance Ratio:**

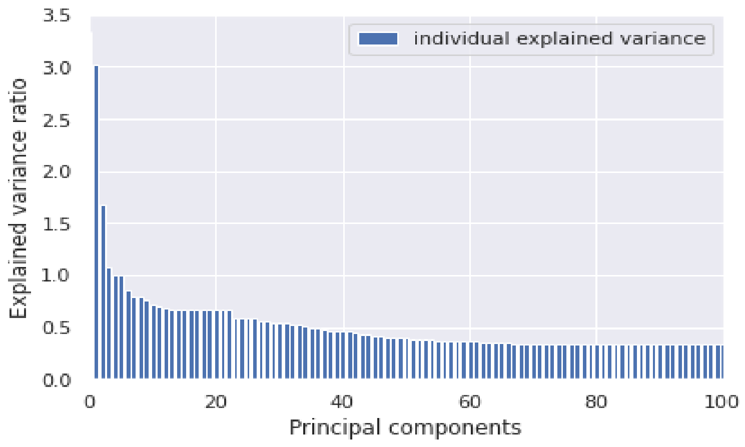

Figure 6: Explained variance ratio

After sorting the eigen pairs, the above plotted graph is the result of variance to each of the principal components. It can be inferred from the graph that the first two principal component contribute to maximum variance. Hence we choose first two components as low dimensional space for further analysis.

References

[1] Shaikh, Javed .*"Machine Learning, NLP: Text Classification Using Scikit-Learn, Python and NLTK." Medium, Towards Data Science, 30 Oct. 2017, towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a.*

[2] Elderfield, James. *"Feature Generation from Tweets." Medium, Verifa, 13 Aug. 2018, medium.com/verifa/feature-generation-from-tweets-9af0565ad6e6./.*

[3] Usman Malik *Implementing PCA In Python With Scikit-Learn". Stack Abuse, 2020, https://stackabuse.com/implementing-pca-in-python-with-scikit-learn/.*

[4] Kim, Ricky. *"Another Twitter Sentiment Analysis with Python - Part 8 (Dimensionality Reduction: Chi2, PCA)." Medium, Towards Data Science, 25 Jan. 2018, towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-8-dimensionality-reduction-chi2-pca-c6d06fb3fcf3./.*

[5] Jeevanandam Jotheeswaran1, Loganathan R, Madhu Sudhanan *Ijcst.Org, 2020, https://www.ijcst.org/Volume3/Issue5/p20*$_{35}$*.pdf./.*

[6] Deisenroth, Marc Peter, *Mathematics for Machine Learning. Cambridge University Press, 2020.*