All the weights associated with the topics from the sentence seem almost similar. You can perform this on huge data to extract significant topics. The whole idea to implement this on sample data is to make you familiar with it, and you can use the same code snippet to perform on the huge data for significant results and insights.

Recipe 4-6. Classifying Text

Text classification – The aim of text classification is to automatically classify the text documents based on pretrained categories.

Applications:

- Sentiment Analysis
- Document classification
- Spam – ham mail classification
- Resume shortlisting
- Document summarization

Problem

Spam - ham classification using machine learning.

Solution

If you observe, your Gmail has a folder called “Spam.” It will basically classify your emails into spam and ham so that you don’t have to read unnecessary emails.

How It Works

Let's follow the step-by-step method to build the classifier.

Step 6-1 Data collection and understanding

Please download data from the below link and save it in your working directory:

https://www.kaggle.com/uciml/sms-spam-collection-dataset#spam.csv

```
#Read the data
Email_Data = pd.read_csv("spam.csv",encoding ='latin1')

#Data undestanding
Email_Data.columns

#output
Index(['v1', 'v2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],
dtype='object')

Email_Data = Email_Data[['v1', 'v2']]
Email_Data = Email_Data.rename(columns={"v1":"Target",
"v2":"Email"})

Email_Data.head()

#output
    Target   Email
0      ham   Go until jurong point, crazy.. Available only ...
1      ham   Ok lar... Joking wif u oni...
2      spam  Free entry in 2 a wkly comp to win FA Cup fina...
3      ham   U dun say so early hor... U c already then say...
4      ham   Nah I don't think he goes to usf, he lives aro...
```

Step 6-2 Text processing and feature engineering

The code is below:

```
#import
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import string
from nltk.stem import SnowballStemmer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
import os
from textblob import TextBlob
from nltk.stem import PorterStemmer
from textblob import Word
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
import sklearn.feature_extraction.text as text
from sklearn import model_selection, preprocessing, linear_
model, naive_bayes, metrics, svm

#pre processing steps like lower case, stemming and
lemmatization

Email_Data['Email'] = Email_Data['Email'].apply(lambda x:
                      " ".join(x.lower() for x in x.split()))
stop = stopwords.words('english')
Email_Data['Email'] = Email_Data['Email'].apply(lambda x: " ".join
                      (x for x in x.split() if x not in stop))
st = PorterStemmer()
```

```
Email_Data['Email'] = Email_Data['Email'].apply(lambda x: " ".join
                      ([st.stem(word) for word in x.split()]))
Email_Data['Email'] = Email_Data['Email'].apply(lambda x: " ".join
                      ([Word(word).lemmatize() for word in
                      x.split()]))

Email_Data.head()

#output
  Target                                                 Email
0    ham  go jurong point, crazy.. avail bugi n great wo...
1    ham                      ok lar... joke wif u oni...
2    spam free entri 2 wkli comp win fa cup final tkt 21...
3    ham          u dun say earli hor... u c alreadi say...
4    ham              nah think goe usf, live around though

#Splitting data into train and validation

train_x, valid_x, train_y, valid_y = model_selection.train_
test_split(Email_Data['Email'], Email_Data['Target'])

# TFIDF feature generation for a maximum of 5000 features

encoder = preprocessing.LabelEncoder()
train_y = encoder.fit_transform(train_y)
valid_y = encoder.fit_transform(valid_y)

tfidf_vect = TfidfVectorizer(analyzer='word',
             token_pattern=r'\w{1,}', max_features=5000)
tfidf_vect.fit(Email_Data['Email'])
xtrain_tfidf =  tfidf_vect.transform(train_x)
xvalid_tfidf =  tfidf_vect.transform(valid_x)

xtrain_tfidf.data
```

```
#output
array([0.39933971, 0.36719906, 0.60411187, ..., 0.36682939,
0.30602539, 0.38290119])
```

Step 6-3 Model training

This is the generalized function for training any given model:

```
def train_model(classifier, feature_vector_train, label,
feature_vector_valid, is_neural_net=False):
    # fit the training dataset on the classifier
    classifier.fit(feature_vector_train, label)
    # predict the labels on validation dataset
    predictions = classifier.predict(feature_vector_valid)
    return metrics.accuracy_score(predictions, valid_y)

# Naive Bayes trainig
accuracy = train_model(naive_bayes.MultinomialNB(alpha=0.2),
xtrain_tfidf, train_y, xvalid_tfidf)
print ("Accuracy: ", accuracy)

#output
Accuracy:  0.985642498205

# Linear Classifier on Word Level TF IDF Vectors
accuracy = train_model(linear_model.LogisticRegression(),
           xtrain_tfidf, train_y, xvalid_tfidf)
print ("Accuracy: ", accuracy)

#output
Accuracy:  0.970567121321
```

Naive Bayes is giving better results than the linear classifier. We can try many more classifiers and then choose the best one.