Subarna Chowdhury Soma
SJSU ID: 014549587

1. **[a] [2 points]**

Find out if Google is doing (a) Case Folding (b) Stemming (c) Removing stop words (d) Lemmatizing. How about Bing and DuckDuckGo?

Answer:

	Google	Bing	DuckDuckGo
Case Folding	**Yes**	**Yes**	**Yes**
Stemming	**Yes**	**No**	**Yes**
Removing stop words	**Yes**	**Yes**	**Yes**
Lemmatizing	**Yes**	**No**	**No**

[b] [2 points]

Generate a word cloud for your favorite public document at:

https://www.wordclouds.com/

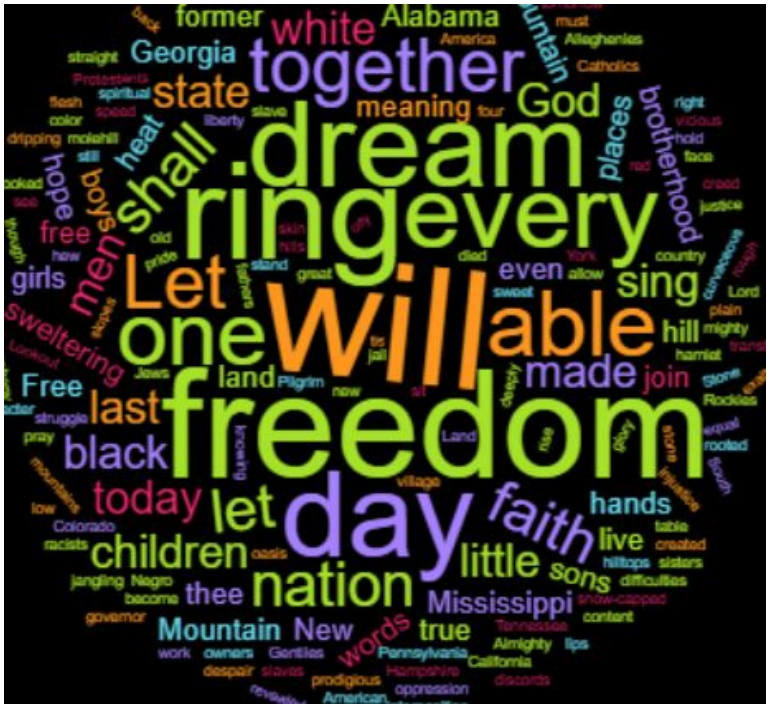
Describe some statistical properties that you observe from the word cloud so generated.

Answer:

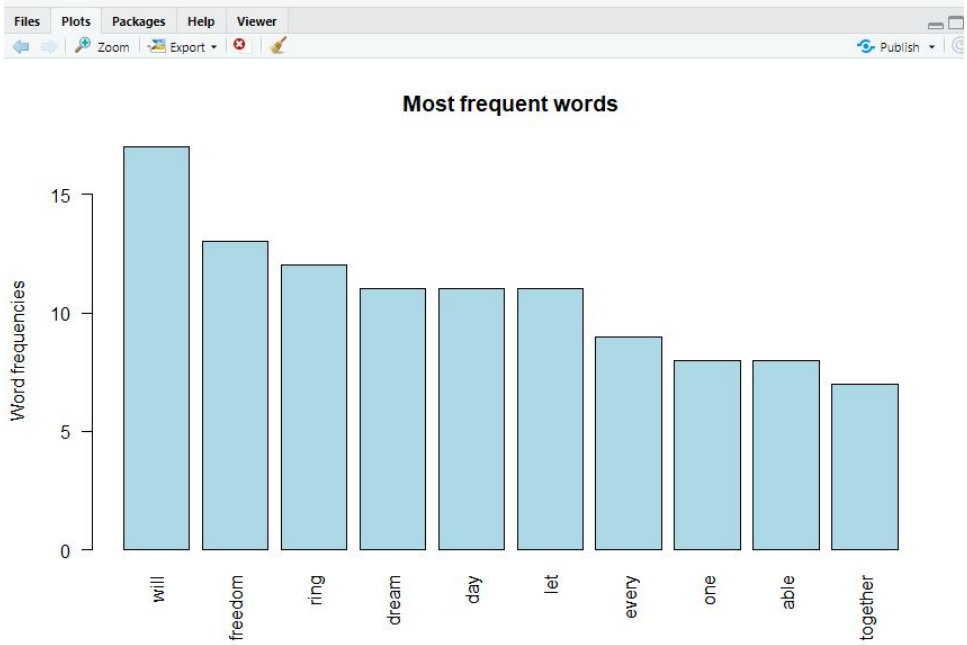
Doc:
http://www.sthda.com/sthda/RDoc/example-files/martin-luther-king-i-have-a-dream-speech.txt

I have used Martin Luther King's speech to generate word clouds.

Output:



Observation: Looking at the results we can see that “Will”, “freedom”, “dream”, “day” and “together” are the five ’ are the most emphasised words. From the statistical and practical point of view also it is observed that these five words are the most important words from Martin Luther King’s famous speech “I have a dream” . Statistical orientation of the most frequent words are given below:



[c] [2 points]

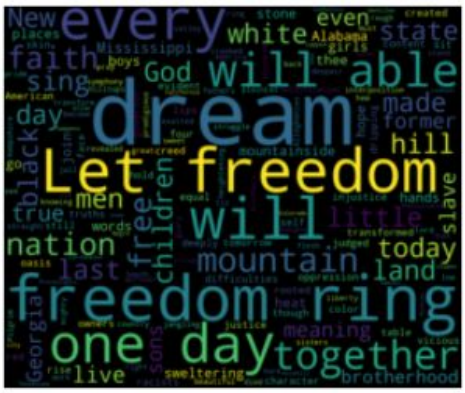
Now generate the same word cloud for the same document using Python.

Answer:

File: hwExtra_WordCloud_MartinLuthar_Python

Output:

```
In [26]: plt.imshow(wordcloud, interpolation='bilinear')
         plt.axis("off")
         plt.show()
```

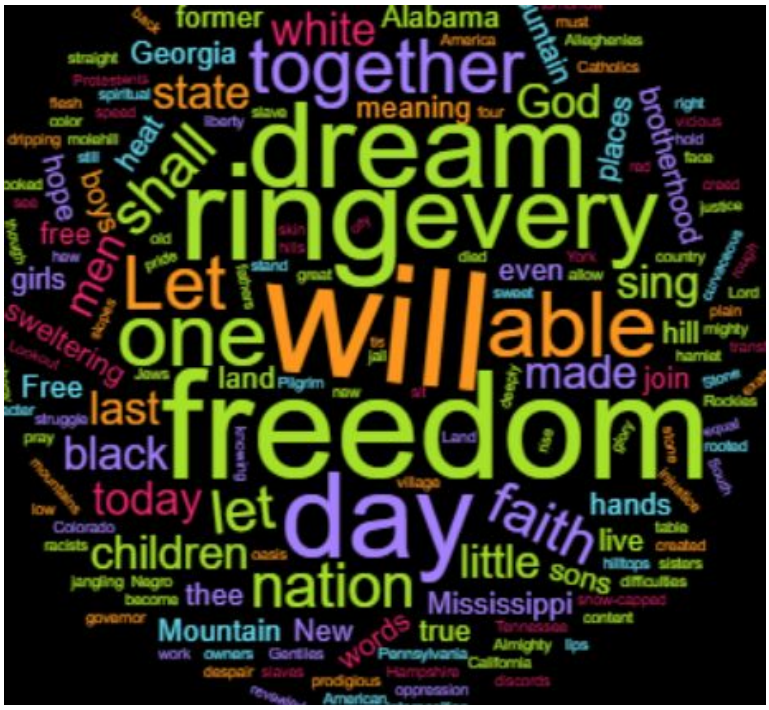


[d] [2 points]

Generate the same word cloud for the same document using R.

File: hwExtra_WordCloud_MartinLuthar_R

Output:



[e] [2 points]

Why do we use only Cosine for measuring similarity but not other trigonometric ratios?

Answer: Cosine similarity is used as a distance metric to measure to find how similar two documents are. Formula of cosine similarity is, S(x,y) = x.y/ (||x|| ||y||), where the numerator is the inner product. So it is clear that the ‘cosine’ here is coming from the formula of inner product which uses ‘cosine’ to measure the angle between two vectors.

That's why we use cosine for measuring similarity instead of other trigonometric. Moreover cosine is a decreasing function. Two documents are similar at the increasing rate of cosine value or the decreasing rate of angle.

2. [10 points]

Refer to online tutorials on regularization such as

https://medium.com/coinmonks/regularization-of-linear-models-with-sklearn-f88633a93a2

and

https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b

Apply the techniques from the above tutorial to the student dataset at https://archive.ics.uci.edu/ml/datasets/student+performance

. Does regularization help improve the accuracy of predicting the final Math grade of the students?

Answer:

File: hwExtra_StudentMathData_Regularization;

student-mat.csv

Output and Observation: Student database file has 33 columns,among them G3 contains final grade, which is the target column. First I have used linear regression to predict the final math grade of students. Then I have applied different regularization techniques of polynomial degree 2. I have observed from the output that not all regularization techniques improve the prediction accuracy, only L1 Regularization or Lasso Regression improves the accuracy.

With normal linear regression, test accuracy is 79%. But after computing plain Polynomial features, test accuracy drops to 66% due to overfitting issue. Then I have applied L2 Regularization or Ridge Regression with varying alpha value. For L2 Regularization or Ridge Regression-alpha value =.01, test accuracy is still 66%, but for alpha= 10 and 100, test accuracy increases to 71% and 73% consecutively. Finally with L1 Regularization or Lasso Regression achieved test accuracy is 80%. So it can be said

that, yes, regularization helps to improve the accuracy of predicting the final math grade of the students.

Student-math data:

]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6	5	6	6
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4	5	5	6
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10	7	8	10
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2	15	14	15
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	4	6	10	10

5 rows × 33 columns

a. Linear regression

```
#linear regression

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

print('Training score%: ',lr_model.score(X_train, y_train) *100, '%')
print('Test score: ',lr_model.score(X_test, y_test)*100, '%')

y_pred = lr_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)

print('RMSE: {}'.format(rmse))
```

```
Training score%:  84.94776543596151 %
Test score:  80.49063890373247 %
RMSE: 2.0708839001184685
```

b. Polynomial features:

```
#creating polynomial features

steps = [
    ('scalar', StandardScaler()),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', LinearRegression())
]

pipeline = Pipeline(steps)
pipeline.fit(X_train, y_train)

print('Training score%: ',pipeline.score(X_train, y_train) *100, '%')
print('Test score: ',pipeline.score(X_test, y_test)*100, '%')
# Testing score has decreased due to overfitting
```

```
Training score%:  93.10073473326045 %
Test score:  66.30630737051591 %
```

c. L2 Regularization or Ridge Regression-alpha value =.01

```
#l2 Regularization or Ridge Regression-alpha value =.01

steps = [
    ('scalar', StandardScaler()),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', Ridge(alpha=.01, fit_intercept=True))
]

ridge_pipe = Pipeline(steps)
ridge_pipe.fit(X_train, y_train)


print('Training score%: ',ridge_pipe.score(X_train, y_train) *100, '%')
print('Test score: ',ridge_pipe.score(X_test, y_test)*100, '%')

mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)

print('RMSE: {}'.format(rmse))
```

```
Training score%:  93.10072970999369 %
Test score:  66.32703779920287 %
RMSE: 2.0708839001184685
```

d. L2 Regularization or Ridge Regression-alpha value =10

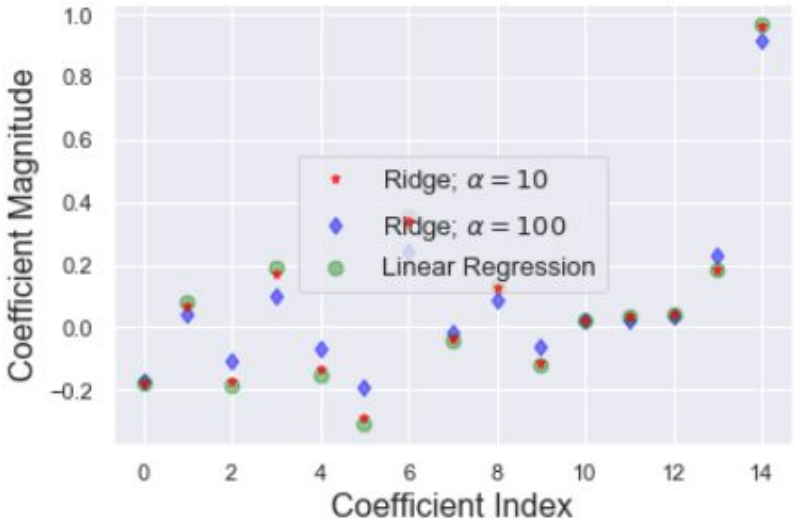
```
#l2 Regularization or Ridge Regression-alpha value =10

steps = [
    ('scalar', StandardScaler()),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', Ridge(alpha=10, fit_intercept=True))
]

ridge_pipe = Pipeline(steps)
ridge_pipe.fit(X_train, y_train)


print('Training score%: ',ridge_pipe.score(X_train, y_train) *100, '%')
print('Test score: ',ridge_pipe.score(X_test, y_test)*100, '%')

mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)

print('RMSE: {}'.format(rmse))
```

```
Training score%:  92.31426420378129 %
Test score:  71.80494844150981 %
RMSE: 2.0708839001184685
```

e. L2 Regularization or Ridge Regression-alpha value =100

```
#l2 Regularization or Ridge Regression-alpha value =100

steps = [
    ('scalar', StandardScaler()),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', Ridge(alpha=100, fit_intercept=True))
]

ridge_pipe = Pipeline(steps)
ridge_pipe.fit(X_train, y_train)


print('Training score%: ',ridge_pipe.score(X_train, y_train) *100, '%')
print('Test score: ',ridge_pipe.score(X_test, y_test)*100, '%')

mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)

print('RMSE: {}'.format(rmse))
```

```
Training score%:  86.81847871793414 %
Test score:  73.68383324073095 %
RMSE: 2.0708839001184685
```



f. L1 Regularization or Lasso Regression

```
# l1 Regularization or Lasso Regression

steps = [
    ('scalar', StandardScaler()),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', Lasso(alpha=0.3, fit_intercept=True))
]

lasso_pipe = Pipeline(steps)
lasso_pipe.fit(X_train, y_train)


print('Training score%: ',lasso_pipe.score(X_train, y_train) *100, '%')
print('Test score: ',lasso_pipe.score(X_test, y_test)*100, '%')

mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)

print('RMSE: {}'.format(rmse))
```

```
Training score%:  82.76858766039767 %
Test score:  80.00079885402924 %
RMSE: 2.0708839001184685
```

3. [a] [5 points]

In Malay (Bahasa Melayu and Bahasa Indonesia among others), plurals are sometimes formed by duplicating a word (cat = \kucing", cats = \kucing-kucing"; book = \buku", books = \buku-buku"). Discuss how this affects tf if the duplication is treated as two separate words, and whether such an outcome is desirable or not.

Answer:

If the duplication is treated as two separate words, the term frequency will increase a lot for the documents which contain the plural form of the word than the documents which contain the singular form of the word.

For example, let us consider two documents written in English language, Ae and Be. Now, Ae document contains the word “cats” (in the plural form) 10 times and Be document contains the word “cat” (in the singular form) 15 times. Therefore, the tf of “cat” in document Be will be 1.5 times of the tf of “cat” in document Ae. Now, if we translate both these documents in Malay language (let us call them Am and Bm), then the tf of “cat” in Am will be 20, which will be greater than the tf of “cat” in Bm. So, we may conclude from the comparison of tf between two documents in Malay language that document Am contains more information about cat, which is completely wrong. Therefore, if the duplication is treated as two separate words, the outcome is not desirable.

[b] [5 points]

In Sanskrit and to some extent in some Indian languages, multiple words are written as a single word. For instance, kaviḿpurāṇamanuśāsitāramaṇoraṇīyāḿsamanusmaredyaḥ is a single word that can be broken down into:

kavihi purāṇam anuśāsitāram aṇoh aṇīyāḿsam anusmared yaḥ

Discuss the treatment and effect of such unions ("sandhis") on TF-IDF calculations and Text Mining in general.

Answer: First, the union of multiple words into a single word will create too many different terms in a document and it will not be good in terms of term frequency.

For example, let us consider a document written in English, Ae, which contains the following sentences: "Cat eats. Cat jumps. Cat runs. Cat likes fish. Cat hunts rat. Cat likes to eat milk." Now, if we compute the tf of the words after stemming, we will get the following stats: "Cat": 6, "eat": 2, "like": 2, "jump": 1, "run": 1, "hunt": 1, "fish": 1, "rat": 1, "milk": 1. So, we can easily understand by looking at the tf of the words that the document mostly talks about "Cat". Now, if the same document is translated into Sanskrit with union of words for every sentence, then we will have different words for all sentences and the analysis of tf will not provide us any insight about what the document mostly talks about. So, the effect of union of words on tf is not desirable.

Similar argument is true for idf. If words are joined, then it is unlikely that we will find the same word in multiple documents. For example, two documents may contain two different sentences: "dog eats meat" and "cat eats fish". Now due to the union of words, we will fail to find the word "eat" in both documents. Therefore, the idf for all the words will be the same: log (N/2), if we use the formula as log (N / (number of documents containing the word + 1)). This again fails to provide any insight about how commonly or rarely a word is found in the documents.

Therefore, TF-IDF is badly affected by the union of words. The more number of words are joined together to form one word, the worse TF-IDF of the words become.

4. [a] [5 points]

Examine a sample implementation of the PageRank algorithm in different languages at:

https://github.com/louridas/pagerank

Assume that each of the web pages already has a trustworthiness score. Write pseudo-code for the revised page rank algorithm that weighs-in this score when computing the rank.

Answer: For the revised page rank algorithm with trustworthiness score, I am assuming that values are stored in an array, T[] of n elements, when n is the number of pages. Consider that I am computing page rank for page Pi and Pj is pointing to Pi. In the revised pagerank algorithm, while calculating the page rank of page Pi, instead of dividing by the out-link count of Pj, now I shall consider trustworthiness score. So the denominator will be the sum of (1 - T(Pk)) where T(Pk) is the trustworthiness score of each out-link Pk from Pj. This way pages with trustworthy out-links will have more impact on computing page rank of Pi. For numerator, with sum of page rank of pages pointing to Pi, trustworthiness score of the pages will be considered also. There for final numerator consists of the summation of {page rank * (1+ trustworthiness score) } of the pages pointing to Pi. I am using (1+ trustworthiness score) to minimize the risk of overfitting with trustworthiness score. Another assumption I shall make that the trustworthiness score will range between [-1, 1]. For pages with trustworthiness score value <=0 , I shall not consider those pages for computing page rank to reduce the effect of using risky pages and to make the page ranks more trustworthy.

Pseudocode for PageRank (G):

Input: Let G represents set of nodes or web pages and T is an array of trustworthiness score of n elements/pages

Output: An n-element array of PR with represents PageRank for each web page

```
1. For i← 0 to (n-1) do
2.     Let A be an array of n elements
3.     A[i]← 1/n
4.     Let T is an array of trustworthiness score
5.     T[i]← trustworthiness score of each pages // -1 <= T[i] <= 1
6. d← 0.8 [or any value between (0,1)]
7. Repeat until converges
8.     For i← 0 to (n-1) do
9.         Let PR be a n-element array
10.        PR[i]← (1-d)/N
11.        For all pages Q such that Q links to PR[i] do
12.            Let T[k] be the trustworthiness score of a page k pointing to Q
13.            V ← sum of (1- T[k]) for all pages k pointing to Q
14.            if (T[Q] <= 0)
15.                continue
16.            else
17.                PR[i]← PR[i] + d * A[Q] * (1+T[Q]) / V
18.    For i← 0 to (n-1) do
19.        A[i] ← PR[i]
```

[b] [5 points]

Modify the python code at https://github.com/ashkonf/PageRank/blob/master/pagerank.py

to incorporate the logic for weighing in the trustworthiness score [No need to run it]. Remember to clearly mark and document your changes in the code.

Answer:

Above algorithm has published function 'powerIteration' for which implements the PageRank algorithm's main function to compute steady probabilities. This function considers 'transitionWeights' for transition metric only.Here I have added 'trustScore' also to consider transition and combined it with 'transitionWeights'. Therefore 'transitionProbabilities ' will be now a combination of 'transitionWeights' and 'trustScore'. Code changes are listed below as well as in the file: **hwExtra_PageRank_Revised_withTrustScore,** but I have not executed the file as per requirement.

1st modification in code:

*def powerIteration(transitionWeights, **trustScore**, rsp=0.15, epsilon=0.00001, maxIterations=1000):*

```
    # Clerical work:
    transitionWeights = pandas.DataFrame(transitionWeights)
    trustScore = pandas.DataFrame(trustScore)
    nodes = __extractNodes(transitionWeights, trustScore)
    transitionWeights = __makeSquare(transitionWeights, trustScore, nodes, default=0.0)
    transitionWeights = __ensureRowsPositive(transitionWeights, trustScore)
    # Setup:
    state = __startState(nodes)
    transitionProbabilities = __normalizeRows(transitionWeights, trustScore)
```

```
    transitionProbabilities = __integrateRandomSurfer(nodes, transitionProbabilities, rsp)
    # Power iteration:
    for iteration in range(maxIterations):
        oldState = state.copy()
        state = state.dot(transitionProbabilities)
        delta = state - oldState
        if __euclideanNorm(delta) < epsilon:
            break
    return state
```

2nd modification:

```
def __makeSquare(matrix, trustScore, keys, default=0.0):
    matrix = matrix.copy()
    def insertMissingColumns(matrix):
        for key in keys:
            if not key in matrix:
                matrix[key] = pandas.Series(default, index=matrix.index) +
pandas.Series(default, index=trustScore.index)
        return matrix
    matrix = insertMissingColumns(matrix) # insert missing columns
    matrix = insertMissingColumns(matrix.T).T # insert missing rows
    return matrix.fillna(default)
```

3rd Modification:

```
def __ensureRowsPositive(matrix,trustScore):
    matrix = matrix.T
    for colKey in matrix:
```

if matrix[colKey].sum() == 0.0:

*matrix[colKey] = pandas.Series(numpy.ones(len(matrix[colKey])), index=matrix.index) + (1- pandas.Series(numpy.ones(len(**trustScore**[colKey])))*

return matrix.T

4th Modification:

*def __normalizeRows(matrix,**trustScore**):*

*return matrix.div(matrix.sum(axis=1) * **trustScore**.sum(axis=1), axis=0)*

5. [a] [5 points]

Given that most of the world population today speaks and writes multiple languages today, do you think that Zipf law applies to the combined corpus of the languages that you can speak? Why?

Answer: Zipf's law is about the statistical distribution of words in linguistic corpus. According to Zipf's law, frequency of occurrence of words is inversely proportional to the rank in this frequency of occurrence. It is a power law. Usually we use Zipf's law for english corpora or monolingual corpora, but I think that, Zipf's law is applicable to combined multilingual corpus also.

On relevant point, it can be mentioned that there are a variety of corpora, e.g. monolingual, parallel and multilingual. Monolingual corpora deals with only a single language of any type. Parallel corpora is used when we need 'same' texts in more than one language say for translation, which is kind of multilingual also. Finally multilingual is combined corpus of multiple languages.

As Zipf's law tells us that natural language tokens(words) in a corpus is in inverse relation with their rank. So Zipf's law should work for monolingual -non-english, parallel and multilingual corpora. Some recent studies show that two parallel corpora have common characteristics that frequency-rank distribution follows zipf's law and values are very close to each other in both the documents. Same for multilingual corpora also. This law implies that rank-frequency distribution will fall for inverse relation and its not language specific. For example, one famous multilingual corpus is Aarthus, which is a corpus of Danish, French and English contract law . A recent analysis has been published on this showing that words follow statistical distribution of Zipf's law for

n-gram word phrases as well as for single words for this corpora. Zipf's law curve is valid and almost identical for bilingual corpus also.

[b] [5 points]

The letter, 'V' has the greatest number of words associated with Big Data. Can you think of any other letter that comes close and associates with the next largest number of Big Data terms? List as many of those terms starting with this letter and give a brief description of each.

Answer: I have listed below a few Big Data related terms and their description starting with 'C' :

1. Connection/Addition: Connection or addition deals with the complexity of Big data. Here complexity means the capability to accept the complexity and inaccuracy of big data and making a connection with it so that data reveals the rule.

2. Correctness: One important property of Big data is correctness. Correctness measures the accuracy and veracity of big data. The more accurate data, the more accurate insight- that's the main mantra.

3. Coverage: Coverage represents the volume of data. Big data should have sufficient coverage to be used for an application. Different cross platform data is covered under the umbrella of Big data.

4. Currency: Big data applications needs up to date data received at required time. Currency represents that property. Some applications need data soon enough to respond and some applications may wait.

5. Capture: Among Big data properties, capture plays a significant role. It means that capture data that is scalable. We get huge volume of Big data. Sometimes we use it for visualization. We have to capture in a way that the data is scalable and feasible to visualize 'Capture' property adds an extra layer to that.

6. Clean: I have already discussed about the volume of available Big data. With the increased amount of data, quality assurance is also important. We need clean data. Hense 'clean' is associated with Big data. Cleaningness property of big data is very useful for visualization.

7. Combine: We often combine large data sources to form big data source. Source of Big data is increasing exponentially which is a big challenge for big data combination and integration into a common repository. The hardest one is to make a combination of cross -platform big data.

8. Calculate: Big data is often used for reporting system which requires lots of calculation and computation. Similar to clean data visualization, calculation also done in the first place to get a pivotal solution. Prior to data visualization, centralized calculation is also done on clean processed big data in various reporting applications.

9. Control: We gain the control on Big data by putting everything together. Control is not just a simple word or property of Big data, it reflects ideas of decision making process on Big data to make data driven application. We need more control over big volume of data to get ultimate goal in a data hungry application.