**Increasing Subscription Revenue by Providing Faster and More Accurate Recommendations to Customers**

Project:

Chandra Mouli Marrivada (801026021)

Contents

[Business Summary and Challenge 3](#_Toc532410369)

[Analytics Project Plan 3](#_Toc532410370)

[Discovery and Data Preparation 4](#_Toc532410371)

[Model Planning and Building 9](#_Toc532410372)

[Results and Implications 10](#_Toc532410373)

[Appendix 1 – Code 11](#_Toc532410374)

[Libraries 11](#_Toc532410375)

[Cleaning and Exploring the Data 11](#_Toc532410376)

[Word cloud 11](#_Toc532410377)

[Term-document matrix 11](#_Toc532410378)

[Topic modeling 11](#_Toc532410379)

[Classifiers 11](#_Toc532410380)

[Appendix 2 – Intermediate Data Results 12](#_Toc532410381)

# Business Summary and Challenge

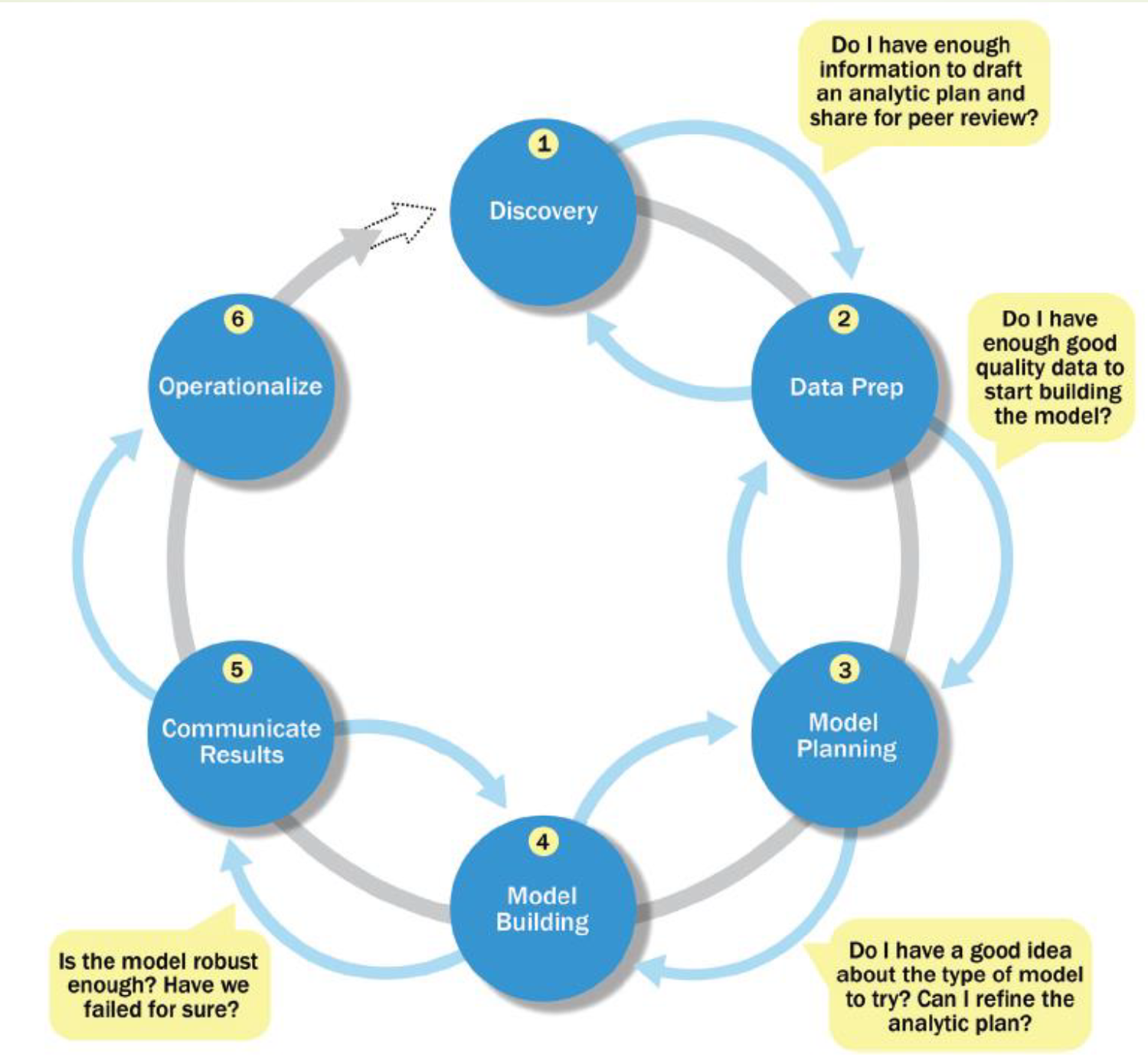
We are a paid database that provides subscribed users (from both corporate and research fields) with access to the articles that they can’t get anywhere for free and some fully exclusive ones. Our business model relies on users being happy with: 1) suggested documents matching their topic search and 2) other related documents populated in “Recommendations”. That way we encourage them to keep the subscriptions active and incentivize them to progress to the “Premium” tier with more exclusive materials.

Our operational process involves receiving the documents from our partners, categorizing & indexing the articles, and then releasing them to the customers based on their requests (they can also browse by categories without searching).

In the past, we relied on primarily manual efforts to go through the materials and assign tags. However, the growth of the company has been steady, and we are approaching the volumes that we can’t efficiently process without automation.

Therefore, the project goal is to develop a classifier that would improve the search (indexing) and recommendation efficiency while keeping the accuracy close to the current level. That should in turn lead to increase in subscription revenue and reduction in operational costs.

# Analytics Project Plan



*Diagram 1. Phases in the Data Analytics Life Cycle (from class presentation)*

We selected a set of existing documents. Our project plan includes the following steps:

1. Clean the data and explore it with the help of visualizations

2. Create term-document matrices

3. Run a Topic Modeling algorithm to determine the dependent variable for the classifiers

4. Build a model for classifying new documents – we aim to build 1 Bayes and 1 logistic regression classifier and compare their performance.

# Discovery and Data Preparation

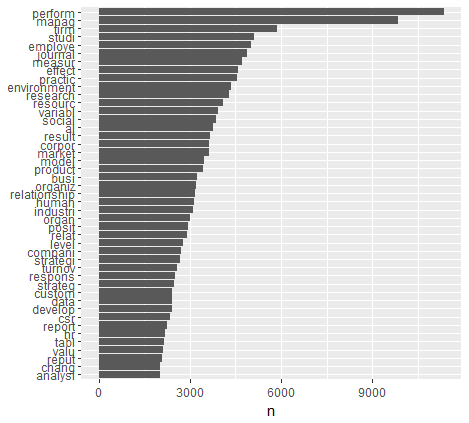
Once we explored several of the documents, we confirmed the need for extensive data cleaning.

The extracted data presented rather common data mining challenges: special characters, symbols from different encodings, incorrect line and word separations, etc. Therefore, we applied the following techniques[[1]](#footnote-1):

* 1. **Tokenization** - breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements called tokens. In the process of tokenization, some characters like punctuation marks were discarded.
  2. **Data conversion** - converting the data streams into similar objects so that they can be compared or processed together. Here we converted all characters to the same encoding and lower case.
  3. **Stripping** - removing all the special characters (such as punctuation marks and non-alphabetic values).
  4. **Stop words filtering** - removing commonly used words (such as "the") to increase the efficiency of indexing and retrieval.
  5. **Stemming** - reducing inflected (or sometimes derived) words to their base form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. In our case as well, we performed stemming to group different tenses and forms of the same root word.

The NLTK library was instrumental in performing these natural language processing functions, along with String and Glob. Four documents returned empty and were excluded from the analysis.

Next, we decided to get a sense of the content in the selected documents with the help of visualizations. After merging the articles, we took a look at the frequency distribution of words and a word cloud to identify potential topics.



*Chart 1. Frequency of the tokens that appeared at least 2000 times*



*Chart 2. Word cloud for the corpus of documents*

Despite using preemptive stemming, we can still understand the words intuitively. This set of documents could be especially interesting to our corporate clients and researchers focused on HR management practices. The materials seem to cover corporate social responsibility and practices which influence the performance of the employees in the firm. Furthermore, the occurrences of the words “journal”, “academi”, and “study” help us infer that at least some of these documents are coming from the high-value, respected sources. We would expect these insights to be reflected in the topic/label assignments and classification later in the project.

At the same time, it’s important to keep in mind the assumptions and simplifications made so far. The removal of numeric values would make prioritizing the timeline relevance difficult. The next limitation was that some of the documents still had remaining formatting issues (lack of any spaces between words), which could influence the corpus-wide distribution. Nevertheless, after multiple iterations of the previous steps, it was reasonable to move forward, reach the final stage of the project, and then review the full learnings.

That’s why, as the next step, we converted the articles to a term-document matrix. As we calculated the frequency of all words in the merged document, we selected only 50 most popular ones, and used those as column headers for document-by-document row counts. This limited the influence of unique words and potentially overgeneralized the label assignment. However, we viewed it as an appropriate foundation for the first round of analysis. Later in the project, we could revisit this step and refine the selection using customized regular expressions (given that we would know more about the specifics of the texts).

In transition to the next stage, we performed Topic Modeling using different values of K to refine our understanding of the corpus and develop the target variable for the classifiers. For example, performing LDA[[2]](#footnote-2) on all the rows in the data for 10 topics resulted in the following breakdown:



*Chart 3. LDA Topic Modeling in R with K=10*

At the same time, the python algorithm version with K=4 provided us with the topics below (given hardware limitations, the number of iterations was set to 25):

*Chart 4. LDA Topic Modeling in python with K=4*

These outputs along with better conceptual understanding of the collection of the documents led us to believe that it may be more efficient to ultimately have multiple layers of classifiers (at the final stage of the overall project). That would help with the fact that most documents have a lot of common words and coming up with distinct topic names for K=10 or K=4 proved to be complicated. In other words, lowering the K value could help us focus on the high-level content of document with greater accuracy. Additionally, it would help to start building classifiers on a binary target instead of a multilevel one.

This finding also reinforces the potential necessity to further group words using regular expressions to identify more explicit differences between documents. Also, we noticed that certain stemming actually complicates the interpretation (e.g. “employe” can be the employer or the employee). In the second iteration of the data cleaning and preparation, we would address these issues.

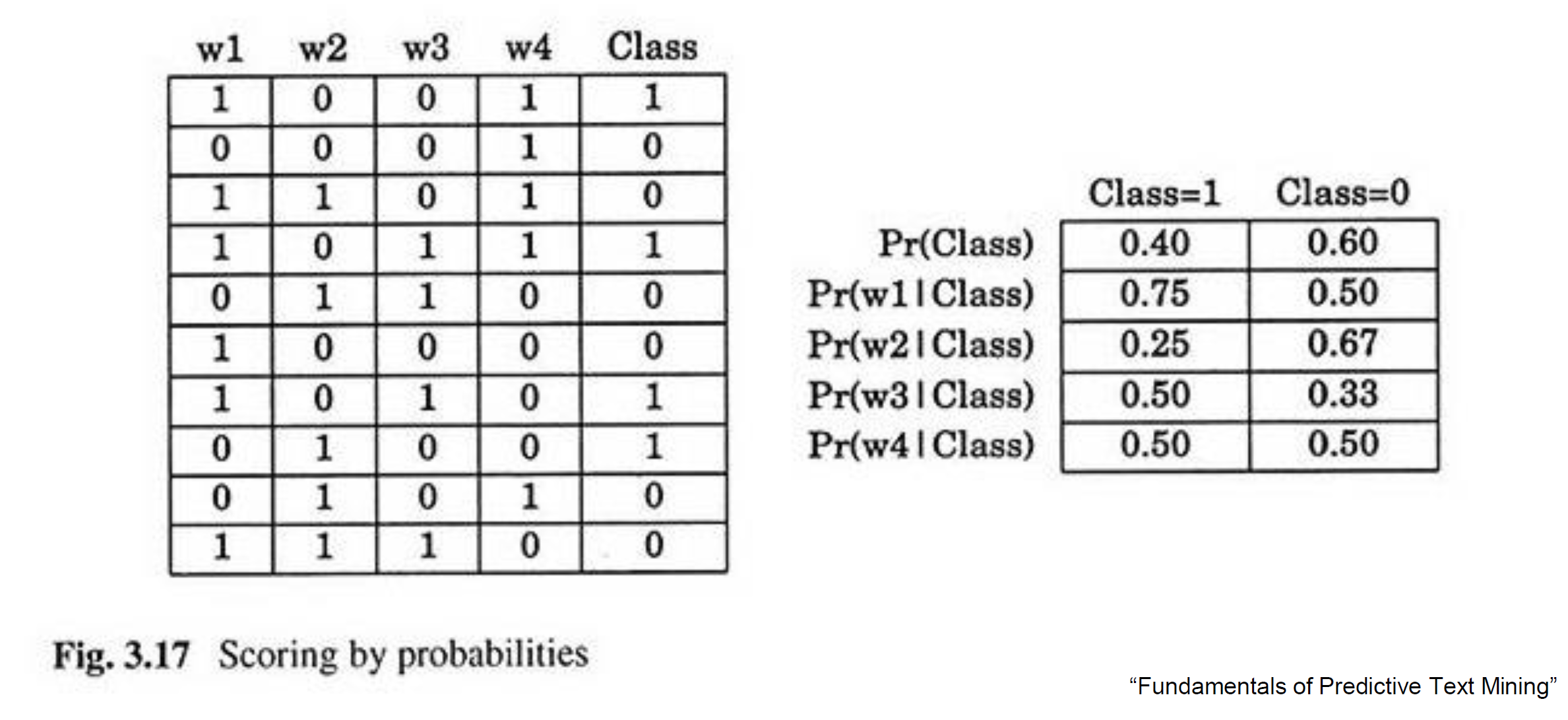
For two topics, the breakdown of most common words is the table below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | perform | 6153 |  |  | 1 | perform | 5892 |
| 0 | manag | 5250 |  |  | 1 | manag | 4777 |
| 0 | employe | 4849 |  |  | 1 | environment | 4701 |
| 0 | practic | 4266 |  |  | 1 | firm | 4200 |
| 0 | human | 3162 |  |  | 1 | corpor | 3714 |
| 0 | resourc | 3146 |  |  | 1 | social | 3282 |
| 0 | effect | 2932 |  |  | 1 | market | 3244 |
| 0 | use | 2896 |  |  | 1 | journal | 2858 |
| 0 | organiz | 2762 |  |  | 1 | studi | 2575 |
| 0 | measur | 2623 |  |  | 1 | use | 2538 |
| 0 | turnov | 2621 |  |  | 1 | csr | 2360 |
| 0 | work | 2578 |  |  | 1 | respons | 2231 |
| 0 | studi | 2556 |  |  | 1 | measur | 2208 |
| 0 | organ | 2358 |  |  | 1 | reput | 2195 |
| 0 | research | 2201 |  |  | 1 | compani | 2164 |
| 0 | journal | 2073 |  |  | 1 | research | 2146 |
| 0 | variabl | 2005 |  |  | 1 | busi | 2053 |
| 0 | result | 1960 |  |  | 1 | variabl | 2003 |
| 0 | model | 1954 |  |  | 1 | industri | 1986 |
| 0 | custom | 1936 |  |  | 1 | product | 1918 |

The overlap is still pretty noticeable. Nevertheless, we assigned the first topic value “0” for the classifier and labeled it “Result-oriented” (the group has higher values for “perform”, “effect”, “practic”, “measure”, “work”). Similarly, the other topic corresponds to “1” and “Socially Responsible” (the group has higher values for “environment”, “csr”,“social”).

The corresponding values were then added to the term-document matrix as the “Class” column. For mixed membership, the topic with the higher score was selected.

# Model Planning and Building



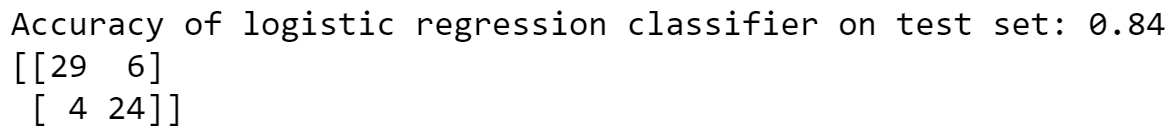
*Diagram 2. Template form of the document collection for Bayes classifier (from class presentation)*

After preparing the term-document matrix to look similar to the template on Diagram 2, we moved on to building the models. Our Topic Modeling could already be considered a descriptive model, but to progress further, we chose to focus on classifiers.

As we would need the “gold benchmark to evaluate performance” and we lack the domain knowledge for that, we came up with the idea of just selecting one of two models, based on the test set accuracy. Given the simplified assumptions of data preparation and topic modeling, we view this step as the final in the first end-to-end iteration of the project.

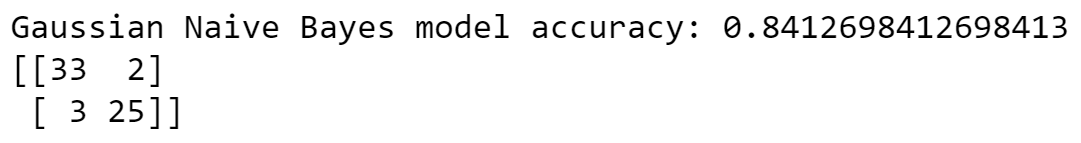
We chose to compare a Bayes classifier with a logistic regression due to their interpretable results and manageable complexity. The complete set has 76 records of class “1” and 81 record of class “0”, so it is pretty balanced. We randomly selected 60% of the documents for the training set, with the remaining serving the validation purpose.

Starting with logistic regression[[3]](#footnote-3), for consistency, we kept all the variables (words) in the model. The modeling produced the following results:



As you can see from the confusion matrix, the model correctly predicted 53 documents from the test set.

Moving on to the Bayes classifier model, we researched multiple methods and chose Multinomial Naive Bayes. It is applicable to feature vectors representing the frequencies with which certain events have been generated by a multinomial distribution. This is the event model typically used for document classification. The performance of this model is below:



The accuracy metric is displayed as almost equal between 2 models, however the confusion matrix for Multinomial Bayes shows better results. Therefore, Bayes will be our primary model of choice going forward.

# Results and Implications

We have completed the first end-to-end iteration towards the project goal of improving the search (indexing) and recommendation efficiency. Based on the collection of text documents, we:

* used techniques of natural language processing and text mining in order to extract the data,
* developed the descriptive model in the form of Topic Modeling,
* built the predictive model in the form of Multinomial Bayes classifier.

The progress so far will allow our company to **reduce the costs** of categorizing the documents we receive from our partners. Furthermore, it will **improve the user satisfaction** by delivering the search results and recommendations faster.

At the same time, the conclusions along the way clearly demonstrated opportunities for future improvements. In the next iteration of the project, we would focus on:

* advanced data cleaning that would handle the documents without any whitespaces;
* more robust word transformation that would allow more precise content understanding;
* more nuanced evaluation of the rare words;
* implementation of other layers for the classifier model or exploring the class variable with multiple levels.

# Appendix 1 – Code



## Libraries

import glob

import string

import nltk

import os

from os import listdir

import re

import codecs

import numpy as np

import matplotlib.pyplot as plt

from wordcloud import WordCloud

import csv

import math, random

from collections import defaultdict, Counter

from bs4 import BeautifulSoup

import requests

import pandas as pd

from sklearn import preprocessing

plt.rc("font", size=14)

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

import seaborn as sns

sns.set(style="white")

sns.set(style="whitegrid", color\_codes=True)

import statsmodels.api as sm

from sklearn import metrics

from sklearn.metrics import confusion\_matrix

from sklearn.naive\_bayes import MultinomialNB

## Cleaning and Exploring the Data

path = "Z:\\UNCC Ganesh\\Assignments\\Final Project\\absTextsNov07\\Text\\\*.txt"

files=glob.glob(path)

for file in files:

    f = open(file, encoding="ascii", errors="surrogateescape")

    text = f.read()

    print(" Cleaning the file: "  + os.path.basename([f.name](http://f.name/)))

*#remove awkward spacing around special characters - there are splits like "de- scribe"*

text = re.sub(r'[-]\s\n', '', text)

text = re.sub(r'[-]\n', '', text)

*# split into words*

    from nltk.tokenize import word\_tokenize

    tokens = word\_tokenize(text)

*# convert to lower case*

    tokens = [w.lower() for w in tokens]

*# remove punctuation from each word*

    import string

    table = str.maketrans('', '', string.punctuation)

    stripped = [w.translate(table) for w in tokens]

*# remove remaining tokens that are not alphabetic*

    words = [word for word in stripped if word.isalpha()]

*# filter out stop words*

    from nltk.corpus import stopwords

    stop\_words = set(stopwords.words('english'))

    words = [w for w in words if not w in stop\_words]

*# stemming of words*

    from nltk.stem.porter import PorterStemmer

    porter = PorterStemmer()

    stemmed = [porter.stem(word) for word in words]

*# writing the cleaned data to clean data directory*

    cleanedFilePath = "Z:\\UNCC Ganesh\\Assignments\\Final Project\\absTextsNov07\\cleanedText\\" + os.path.basename([f.name](http://f.name/))

    cleanedFile = open(cleanedFilePath, 'w')

    cleanedFile.writelines(["%s\n" % word  for word in stemmed])

    cleanedFile.close()

print('Clean up completed - Refer to the cleanedText directory for data')

*#Merging files*

filenames = [fname for fname in os.listdir('D:\\uncc2018\\cleanedtext\\')]

with open('D:\\uncc2018\\cleanedtext\\WordCloudBase.txt','w') as outfile:

for fname in filenames:

with open('D:\\uncc2018\\cleanedtext\\' + fname) as infile:

lines = infile.readlines()

mystr = ' '.join([line.strip() for line in lines if len(line) > 3])

outfile.write(mystr + '\n')

*#Executing wordcloud*

figure = plt.figure(figsize = (8,7))

with codecs.open('D:\\uncc2018\\cleanedtext\\WordCloudBase.txt', "r", encoding='utf-8') as abs:

text = abs.read()

wc = WordCloud(background\_color="white")

wc.generate(text)

plt.imshow(wc, aspect='auto')

plt.axis("off")

plt.show()

## Word cloud

#Merging files

filenames = [fname for fname in os.listdir('D:\\uncc2018\\cleanedtext\\')]

with open('D:\\uncc2018\\output\\WordCloudBase.txt','w') as outfile:

for fname in filenames:

with open('D:\\uncc2018\\cleanedtext\\' + fname) as infile:

lines = infile.readlines()

mystr = ' '.join([line.strip() for line in lines if len(line) > 3])

outfile.write(mystr + '\n')

#Executing wordcloud

figure = plt.figure(figsize = (10,10))

with codecs.open('D:\\uncc2018\\output\\WordCloudBase.txt', "r", encoding='utf-8') as abs:

text = abs.read()

wc = WordCloud(background\_color="white")

wc.generate(text)

plt.imshow(wc, aspect='auto')

plt.axis("off")

plt.show()

## Term-document matrix

from collections import Counter

csvfile = 'D:\\uncc2018\\output\\term\_document\_matrix.csv'

header\_count = 1

f = open(csvfile, 'w+').close()

with codecs.open('D:\\uncc2018\\output\\WordCloudBase.txt', "r", encoding='utf-8') as abs:

read\_abs = abs.read()

words = read\_abs.split()

Counter = Counter(words)

most\_occur = Counter.most\_common(50)

word\_list= [i[0] for i in most\_occur]

header = ['Document Index'] + word\_list

doc = re.split(r'\n', read\_abs)

i = 1

word\_counts = []

for para in doc:

int\_count = []

word\_count\_list =[]

for k in range(0,(len(word\_list))):

int\_count = sum(1 for match in re.finditer(word\_list[k], para))

word\_count\_list = word\_count\_list + [int\_count]

word\_counts = [i] + word\_count\_list

with open(csvfile, 'w', newline='') as myfile:

wr = csv.writer(myfile)

if header\_count == 1:

wr.writerow(header)

header\_count = 0

wr.writerow(word\_counts)

i = i + 1

## Topic modeling

def roll\_a\_die():

return random.choice([1,2,3,4,5,6])

def direct\_sample():

d1 = roll\_a\_die()

d2 = roll\_a\_die()

return d1, d1 + d2

def random\_y\_given\_x(x):

"""equally likely to be x + 1, x + 2, ... , x + 6"""

return x + roll\_a\_die()

def random\_x\_given\_y(y):

if y <= 7:

# if the total is 7 or less, the first die is equally likely to be

# 1, 2, ..., (total - 1)

return random.randrange(1, y)

else:

# if the total is 7 or more, the first die is equally likely to be

# (total - 6), (total - 5), ..., 6

return random.randrange(y - 6, 7)

def gibbs\_sample(num\_iters=100):

x, y = 1, 2 # doesn't really matter

for \_ in range(num\_iters):

x = random\_x\_given\_y(y)

y = random\_y\_given\_x(x)

return x, y

def compare\_distributions(num\_samples=1000):

counts = defaultdict(lambda: [0, 0])

for \_ in range(num\_samples):

counts[gibbs\_sample()][0] += 1

counts[direct\_sample()][1] += 1

return counts

def sample\_from(weights):

total = sum(weights)

rnd = total \* random.random() # uniform between 0 and total

for i, w in enumerate(weights):

rnd -= w # return the smallest i such that

if rnd <= 0: return i # sum(weights[:(i+1)]) >= rnd

def p\_topic\_given\_document(topic, d, alpha=0.1):

"""the fraction of words in document \_d\_

that are assigned to \_topic\_ (plus some smoothing)"""

return ((document\_topic\_counts[d][topic] + alpha) /

(document\_lengths[d] + K \* alpha))

def p\_word\_given\_topic(word, topic, beta=0.1):

"""the fraction of words assigned to \_topic\_

that equal \_word\_ (plus some smoothing)"""

return ((topic\_word\_counts[topic][word] + beta) /

(topic\_counts[topic] + W \* beta))

def topic\_weight(d, word, k):

"""given a document and a word in that document,

return the weight for the k-th topic"""

return p\_word\_given\_topic(word, k) \* p\_topic\_given\_document(k, d)

def choose\_new\_topic(d, word):

return sample\_from([topic\_weight(d, word, k)

for k in range(K)])

from collections import defaultdict, Counter

abs = codecs.open('D:\\uncc2018\\output\\WordCloudBase.txt', "r", encoding='utf-8')

docs = abs.readlines()

documents = [line.split() for line in docs]

K = 2

document\_topic\_counts = [Counter() for \_ in documents]

topic\_word\_counts = [Counter() for \_ in range(K)]

topic\_counts = [0 for \_ in range(K)]

document\_lengths = [len(d) for d in documents]

distinct\_words = set(word for document in documents for word in document)

W = len(distinct\_words)

print(distinct\_words)

D = len(documents)

print(D)

random.seed(0)

document\_topics = [[random.randrange(K) for word in document]

for document in documents]

for d in range(D):

for word, topic in zip(documents[d], document\_topics[d]):

document\_topic\_counts[d][topic] += 1

topic\_word\_counts[topic][word] += 1

topic\_counts[topic] += 1

for iter in range(25):

for d in range(D):

for i, (word, topic) in enumerate(zip(documents[d],

document\_topics[d])):

# remove this word / topic from the counts

# so that it doesn't influence the weights

document\_topic\_counts[d][topic] -= 1

topic\_word\_counts[topic][word] -= 1

topic\_counts[topic] -= 1

document\_lengths[d] -= 1

# choose a new topic based on the weights

new\_topic = choose\_new\_topic(d, word)

document\_topics[d][i] = new\_topic

# and now add it back to the counts

document\_topic\_counts[d][new\_topic] += 1

topic\_word\_counts[new\_topic][word] += 1

topic\_counts[new\_topic] += 1

document\_lengths[d] += 1

csvfile2 = 'D:\\uncc2018\\output\\TMK2\_full.csv'

for k, word\_counts in enumerate(topic\_word\_counts):

for word, count in word\_counts.most\_common():

with open(csvfile2, 'a+', newline='') as myfile2:

wr = csv.writer(myfile2)

if count > 800:

wr.writerow([k, word, count])

topic\_names = ["Result-oriented","Socially Responsible"]

csvfile3 = 'D:\\uncc2018\\output\\TMwDocs.csv'

for document, topic\_counts in zip(documents, document\_topic\_counts):

with open(csvfile3, 'a+', newline='') as myfile3:

wr = csv.writer(myfile3)

for topic, count in topic\_counts.most\_common():

if count > 0:

wr.writerow([topic\_names[topic], count])

wr.writerow('')

## Classifiers

from sklearn.metrics import confusion\_matrix

regdata = pd.read\_csv('D:\\uncc2018\\output\\modelinput.csv')

target=['class']

predict=['perform','manag','firm','use','employe','studi','journal','measur','environment','practic','effect','research','resourc','variabl','social','corpor','result','market','product','model','organiz','relationship','busi','industri','may','human','organ','posit','relat','compani','level','work','strategi','turnov','respons','also','strateg','custom','develop','differ','data','csr','report','new','reput','tabl','valu','inform','analysi', 'control']

x=regdata[predict]

y=regdata[target]

logreg = LogisticRegression(solver='lbfgs', max\_iter=500)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.4, random\_state=0)

logreg.fit(x\_train, y\_train.values.ravel())

y\_pred = logreg.predict(x\_test)

print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(x\_test, y\_test)))

confusion\_matrix = confusion\_matrix(y\_test, y\_pred)

print(confusion\_matrix)

print()

from sklearn.metrics import confusion\_matrix

model = MultinomialNB().fit(x\_train, y\_train.values.ravel())

predicted = model.predict(x\_test)

print("Gaussian Naive Bayes model accuracy:", metrics.accuracy\_score(y\_test, y\_pred))

print(confusion\_matrix(y\_test, predicted))

# Appendix 2 – Intermediate Data Results

**

**

**

**

**

**

1. We leveraged the class discussions about the data cleaning and used the abs1.txt as a good example of the final form for the collection of documents. The code and the full library list are in Appendix 1. [↑](#footnote-ref-1)
2. The concept of Topic Modeling and LDA in python has been covered in class. We also utilized this technique in R. [↑](#footnote-ref-2)
3. Multiple python libraries have been used including sklearn and numpy. [↑](#footnote-ref-3)