NATURAL LANGUAGE PROCESSING

Natural Language Processing (NLP) is a collection of techniques that allows working with and analyzing strings of words. Some most common applications are summarizing large volumes of text (e.g., computing word frequencies for different authors) and categorizing sentences (e.g., classifying news headlines as negative/neutral/positive, or classifying customer complaints by issues addressed). Below we consider two examples.

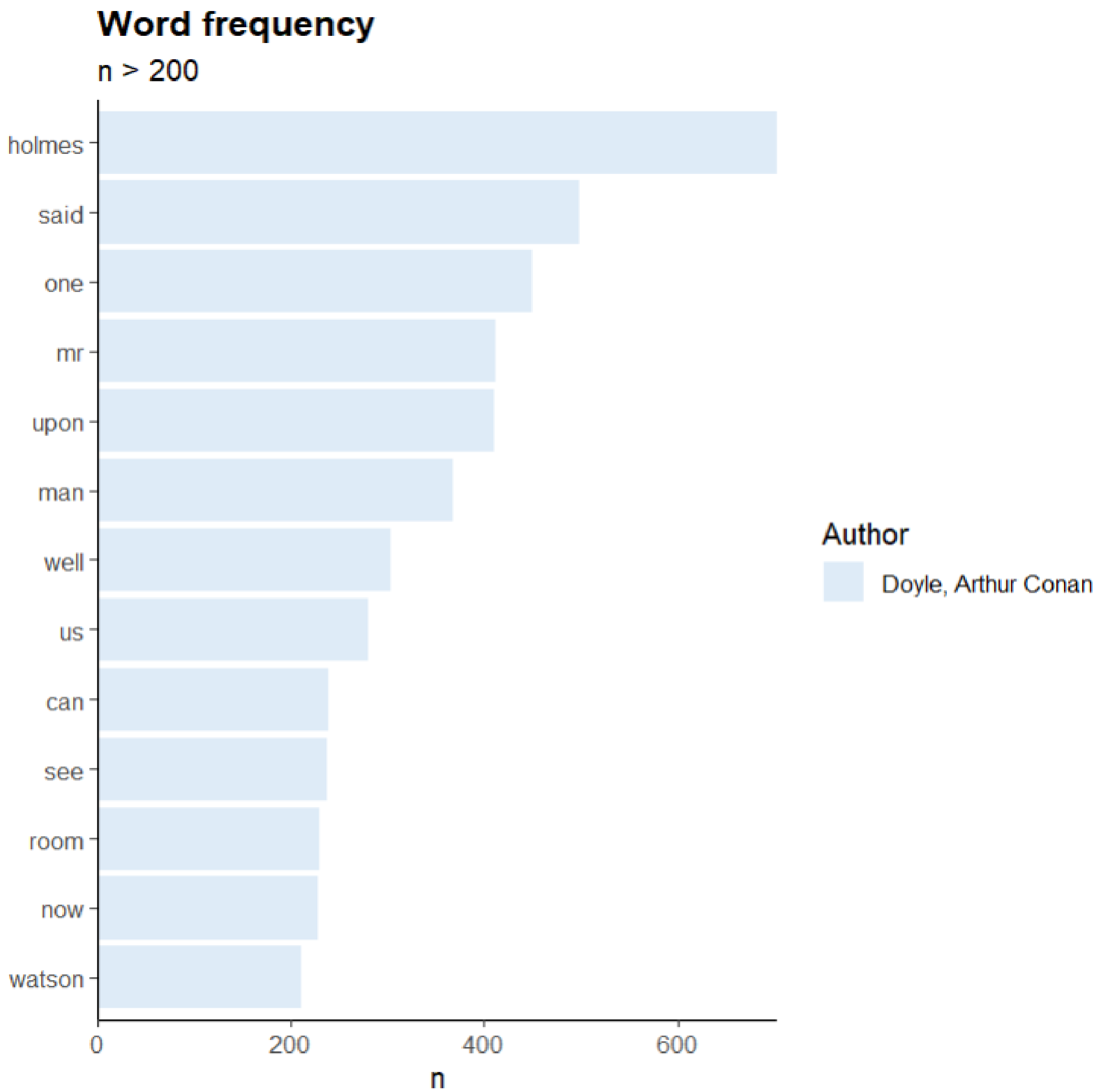
Example. The R code below downloads a digital book from the Project Gutenberg collection, divides the text into words, removing all articles, prepositions, etc. (called "stopwords"), computes frequency distribution of words, and visualizes the results by plotting bar graph and word cloud.

```
#install.packages(c("gutenbergr", "stringr", "dplyr", "tidytext", "stopwords", "tibble", "ggraph",
"wordcloud"))

library(gutenbergr)
library(stringr)
library(dplyr)
library(tidytext)
library(stopwords)
library(tibble)
library(ggplot2)
library(wordcloud)

book<- gutenberg_download(108, meta_fields="author")

#puts text into tibble format
book<- as_tibble(book) %>% mutate(document=row_number())
%>% select(-gutenberg_id)

#creates tokens (words)
#tokenization is the process of splitting text into tokens
tidy_book <- book %>% unnest_tokens(word, text) %>%
group_by(word) %>% filter(n() > 10) %>% ungroup()

#identifying and removing stopwords (prepositions, articles)
stopword<- as_tibble(stopwords::stopwords("en"))
stopword<- rename(stopword, word=value)
tb <- anti_join(tidy_book, stopword, by="word")

#calculating word frequency
word_count<- count(tb, word, sort=TRUE)
```

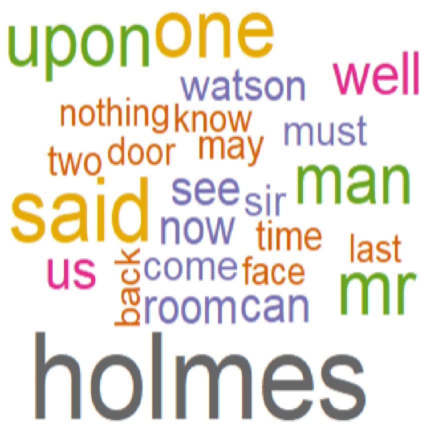
```
print(word_count, n=15)
```

```
   word       n
 1 holmes   703
 2 said     499
 3 one      449
 4 mr       412
 5 upon     411
 6 man      367
 7 well     303
 8 us       279
 9 can      238
10 see      237
11 room     229
12 now      227
13 watson   210
14 come     189
15 sir      188
```

```
#plotting bar graph
tb %>% count(author, word, sort=TRUE) %>%
filter(n > 200) %>% mutate(word=reorder(word, n)) %>%
ggplot(aes(word, n)) + geom_col(aes(fill=author)) + xlab(NULL)
+ scale_y_continuous(expand=c(0, 0)) + coord_flip() +
theme_classic(base_size = 12) + labs(fill="Author", title="Word frequency",
subtitle="n > 200") + theme(plot.title=element_text(lineheight=.8, face="bold")) + scale_fill_brewer()
```




```
#plotting word cloud
tb %>% count(word) %>% with(wordcloud(word, n, max.words=25, colors=brewer.pal(8, "Dark2")))
#brewer.pal(n,name) = color palette, n=# of colors, name=c("Accent", "Dark2", "Paired"
#"Pastel1", "Pastel2", "Set1", "Set2", "Set3")
```



Example. The data set "FinancialNewsHeadlines.csv" contains the sentiments for financial news headlines from the perspective of an investor. It was downloaded from Kaggle (https://www.kaggle.com/datasets/ankurzing/sentiment-analysis-for-financial-news). The data set contains two columns, "Sentiment" and "News Headline". The sentiment can be negative, neutral, or positive. We conduct a **sentiment analysis** on these data by training a **Bidirectional Encoder Representations from Transformers (BERT)** model. This methodology was introduced in 2018 by researchers at Google. BERT learns information from a text from the left and right side of each word during training and consequently gains a deeper understanding of the context. We compute the accuracy of prediction and test the model with a few sentences of our own.

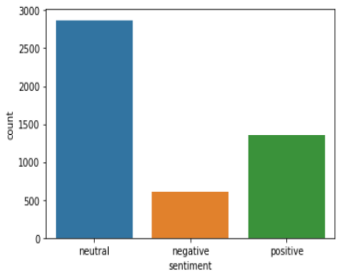
```
#!pip install wordcloud
import pandas
import numpy
import seaborn
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from wordcloud import WordCloud

data=pandas.read_csv("./FinancialNewsHeadlines.csv",encoding='ISO-8859-1')
data=data.rename(columns={'neutral':'sentiment','According to Gran , the company has no plans to move all production to Russ
data.drop_duplicates(subset=['statement'],keep='first',inplace=True)
text = " ".join([x for x in data.statement])

#plotting wordclouds for all news
wordcloud = WordCloud(background_color='white').generate(text)

plt.figure(figsize=(8,6))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.show()
```

```
#plotting wordclouds for neutral news

text = " ".join([x for x in data.statement[data.sentiment=='neutral']])

wordcloud = WordCloud(background_color='white').generate(text)

plt.figure(figsize=(8,6))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.show()
```


```
#plotting wordclouds for positive news

text = " ".join([x for x in data.statement[data.sentiment=='positive']])

wordcloud = WordCloud(background_color='white').generate(text)

plt.figure(figsize=(8,6))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.show()
```


```python
#plotting wordclouds for negative news

text = " ".join([x for x in data.statement[data.sentiment=='negative']])

wordcloud = WordCloud(background_color='white').generate(text)

plt.figure(figsize=(8,6))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.show()
```


```python
#plotting bar graph for sentiments
seaborn.countplot(data.sentiment)
```



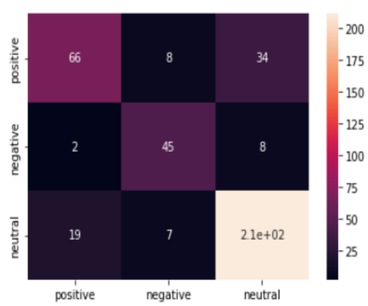

```
#displaying frequency by sentiment
data['sentiment'].value_counts()
```

```
neutral     2871
positive    1362
negative     604
```

```
#training model
numpy.random.seed(5677934)
train, test = train_test_split(data,test_size = 0.2)

#!pip install simpletransformers
#!pip install torch

from simpletransformers.classification import ClassificationModel

# Create a TransformerModel
model = ClassificationModel('bert', 'bert-base-cased', num_labels=3,
args={'reprocess_input_data': True, 'overwrite_output_dir': True},use_cuda=False)

def making_label(st):
    if(st=='positive'):
        return 0
    elif(st=='neutral'):
        return 2
    else:
        return 1

train['label']=train['sentiment'].apply(making_label)
test['label']=test['sentiment'].apply(making_label)

train_df = pandas.DataFrame({
    'text': train['statement'][:1500].replace(r'\n', ' ', regex=True),
    'label': train['label'][:1500]
})

eval_df = pandas.DataFrame({
    'text': test['statement'][-400:].replace(r'\n', ' ', regex=True),
    'label': test['label'][-400:]
})

model.train_model(train_df)
```

```python
#computing predicted sentiments for testing set
result, model_outputs, wrong_predictions = model.eval_model(eval_df)

lst = []
for arr in model_outputs:
    lst.append(numpy.argmax(arr))

true = eval_df['label'].tolist()
predicted = lst

#displaying confusion matrix (positive/negative/neutral)
import sklearn
confmatrix = sklearn.metrics.confusion_matrix(true, predicted)
print(confmatrix)

#displaying heatmap for confusion matrix
df_cm = pandas.DataFrame(confmatrix, ['positive','negative','neutral'], ['positive','negative','neutral'])

seaborn.heatmap(df_cm, annot=True)
plt.show()
```

```
[[ 66   8  34]
 [  2  45   8]
 [ 19   7 211]]
```




```python
#displaying performance metrics
sklearn.metrics.classification_report(true,predicted,target_names=['positive','negative','neutral'])
```

```
'              precision    recall  f1-score   support\n\n    positive       0.76      0.61      0.68       108\n    negative
0.75      0.82      0.78        55\n     neutral       0.83      0.89      0.86       237\n\n    accuracy
0.81       400\n   macro avg       0.78      0.77      0.77       400\nweighted avg       0.80      0.81      0.80       400\n'
```

```python
#computing predicted accuracy
sklearn.metrics.accuracy_score(true,predicted)
```

0.805

```python
#using the trained model to classify user-defined sentences
def classify(statement):
        result = model.predict([statement])
        pred_class = numpy.where(result[1][0] == numpy.amax(result[1][0]))
        pred_class = int(pred_class[0])
        sentiment_dict = {0:'positive',1:'negative',2:'neutral'}
        print(sentiment_dict[pred_class])
        return

classify('People keep money in a bank.')
classify('S&P rose 1000 points in one day.')
classify('Inflation is going down now.')
```

```
neutral
positive
negative
```

Note that the trained BERT model has 80.5% accuracy and that last statement is misclassified. □

Further Reading

1. Generative Adversarial Networks (GANs)
   - https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/
   - https://realpython.com/generative-adversarial-networks/

2. Self Organizing Maps (SOMs)
   - https://davis.wpi.edu/~matt/courses/soms/#Introduction

3. Restricted Boltzmann Machines (RBMs)
   - https://en.wikipedia.org/wiki/Restricted_Boltzmann_machine
   - https://wiki.pathmind.com/restricted-boltzmann-machine

4. Deep Belief Networks (DBNs)
   - https://en.wikipedia.org/wiki/Deep_belief_network
   - https://www.analyticsvidhya.com/blog/2022/03/an-overview-of-deep-belief-network-dbn-in-deep-learning/

5. AutoEncoders
   - https://towardsdatascience.com/introduction-to-autoencoders-7a47cf4ef14b

6. Learning Vector Quantization (LVQ)
   - https://machinelearningmastery.com/learning-vector-quantization-for-machine-learning/