IN4080 – 2023

Mandatory assignment 1 - Alessia Sanfelici

In [ ]:
```
import nltk
import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk.tokenize as tok
import string
import collections
import seaborn as sns
from nltk.corpus import brown
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings('ignore')
```

PART 1 – Conditional frequency distributions (35 pts)

a. Conduct a similar experiment as the one mentioned above with the genres: news, religion, government, fiction, romance as conditions, and occurrences of the words: he, she, her, him, as events. Make a table of the conditional frequencies and deliver code and table.

In [ ]:
```
cfd = nltk.ConditionalFreqDist((genre, word.lower()) for genre in brown.categories()
                               for word in brown.words(categories = genre))
genres = ["news", "religion", "government", "fiction", "romance"]
pronouns = ["he", "him", "she", "her"]

# Table of conditional frequencies
cfd.tabulate(conditions = genres, samples = pronouns)
```

```
             he  him  she  her
     news   642   93   77  121
  religion  206   94   12    8
government  169   26    1    3
   fiction 1308  382  425  413
   romance 1068  340  728  680
```

In [ ]:
```
print("Occurence of masculine pronouns in Romance:")
print(cfd["romance"]["he"]+cfd["romance"]["him"])

print("-----------------------------------------")

print("Occurence of feminine pronouns in Romance:")
print(cfd["romance"]["she"]+cfd["romance"]["her"])
```

```
Occurence of masculine pronouns in Romance:
1408
-----------------------------------------
Occurence of feminine pronouns in Romance:
1408
```

b. Describe your results. How does gender vary across genres?

By looking at the data, we can notice that the masculine pronouns are more frequent than the feminine ones across all genres. In particular, both religion and government have just a few cases of feminine pronouns, especially for the government sector. Focusing on news, the masculine pronouns still prevails, but the difference between feminine and masculine is lower than in the previous cases. We can also notice that, in fiction and romance, the feminine pronouns finally increase their frequency, reaching high levels, especially in romance. In Fiction, the masculine pronuons still prevales, but with a smaller difference than for the other genres. On the contrary, in Romance, the masculine and feminine pronouns occur the same number of times (1408, as shown above).

Hypothesis: The relative frequency of the object form, her, of the feminine personal pronoun (she or her) is higher than the relative frequency of the object form, him, of the masculine personal pronoun, (he or him).

c. First, consider the complete Brown corpus. Construct a conditional frequency distribution, which uses gender as condition, and for each gender counts the occurrences of nominative forms (he, she) and object forms (him, her). Report the results in a two-by-two table.

In [ ]:
```
nom = ["he", "she"]
obj = ["him", "her"]
text = brown.words()

cdf = nltk.ConditionalFreqDist()

# Count the frequency of nominative and objective form according to gender
for word in text:
    if word.lower() == nom[0]:
        cdf["Male"]["Nominative form"] += 1
    elif word.lower() == nom[1]:
        cdf["Female"]["Nominative form"] += 1
    elif word.lower() == obj[0]:
        cdf["Male"]["Objective form"] += 1
    elif word.lower() == obj[1]:
        cdf["Female"]["Objective form"] += 1

# Table of conditional frequencies
cdf.tabulate()
```

```
        Nominative form  Objective form
Female             2860            3036
  Male             9548            2619
```

Then calculate the relative frequency of her from she or her and compare to the relative frequency of him from he or him. Report the numbers. Submit table, numbers and code you used.

In [ ]:
```
relative_her = cdf["Female"]["Objective form"]/(cdf["Female"]["Objective form"] +
                                                cdf["Female"]["Nominative form"])
print("Relative frequency of \"her\" from \"she\" or \"her\":", relative_her)

print("----------------------------------------------------------------------")

relative_him = cdf["Male"]["Objective form"]/(cdf["Male"]["Objective form"] +
                                              cdf["Male"]["Nominative form"])
print("Relative frequency of \"him\" from \"he\" or \"him\":", relative_him)
```

```
Relative frequency of "her" from "she" or "her": 0.5149253731343284
----------------------------------------------------------------------
Relative frequency of "him" from "he" or "him": 0.21525437659242214
```

d. Use the tagged Brown corpus to count the occurrences of she, he, her, him as personal pronouns and her, his, hers as possessive pronouns. Report the results in a two-ways table.

In [ ]:
```
wordlist = brown.tagged_words()
personal = ["she", "he", "her", "him"]
possessive = ["her", "his", "hers"]
d = {"Personal pronouns": {"she": 0, "he": 0, "her" : 0, "him" : 0},
     "Possessive pronouns": {"her" : 0, "his" : 0, "hers" : 0}}

# Update the values in the dictionary according to the frequency of the pronouns
for word, tag in wordlist:
    if tag == "PPS" or tag == "PPO":
        if word.lower() in personal:
            d["Personal pronouns"][word.lower()] += 1
    if tag == "PP$$" or tag == "PP$":
        if word.lower() in possessive:
            d["Possessive pronouns"][word.lower()] += 1

# Transform in dataframe format
df_pronouns = pd.DataFrame.from_dict(d)
df_pronouns = df_pronouns.fillna(0)
df_pronouns
```

Out[ ]:

	Personal pronouns	Possessive pronouns
she	2857.0	0.0
he	9541.0	0.0
her	1107.0	1925.0
him	2614.0	0.0
his	0.0	6983.0
hers	0.0	16.0

In [ ]:
```
pers_fem = ["she", "her"]
pers_mas = ["he", "him"]
poss_fem = ["her", "hers"]
poss_mas = ["his"]

d = {"Personal pronouns": {"Feminine": 0, "Masculine": 0},
     "Possessive pronouns": {"Feminine": 0, "Masculine": 0}}

# Update the values in the dictionary according to the frequency of the pronouns
for word, tag in wordlist:
    if tag == "PPS" or tag == "PPO":
        if word.lower() in pers_fem:
            d["Personal pronouns"]["Feminine"] += 1
        if word.lower() in pers_mas:
            d["Personal pronouns"]["Masculine"] += 1
    if tag == "PP$$" or tag == "PP$":
        if word.lower() in poss_fem:
            d["Possessive pronouns"]["Feminine"] += 1
        if word.lower() in poss_mas:
            d["Possessive pronouns"]["Masculine"] += 1

# Transform in dataframe format
df_pronouns_gender = pd.DataFrame.from_dict(d)
df_pronouns_gender
```

Out[ ]:

	Personal pronouns	Possessive pronouns
Feminine	3964	1941
Masculine	12155	6983

e. We can now correct the numbers from point (b) above. What percentage of the feminine personal pronoun occurs in nominative form and in object form? What are the respective percentages for the masculine personal pronoun?

In [ ]:
```
pers_nom = df_pronouns.loc["she", "Personal pronouns"]*100/(
    df_pronouns_gender.loc["Feminine", "Personal pronouns"])
print("Percentage of the feminine personal pronoun in nominative form:", pers_nom, "%.")

print("-------------------------")

pers_obj = df_pronouns.loc["her", "Personal pronouns"]*100/(
    df_pronouns_gender.loc["Feminine", "Personal pronouns"])
print("Percentage of the feminine personal pronoun in object form:", pers_obj, "%.")
```

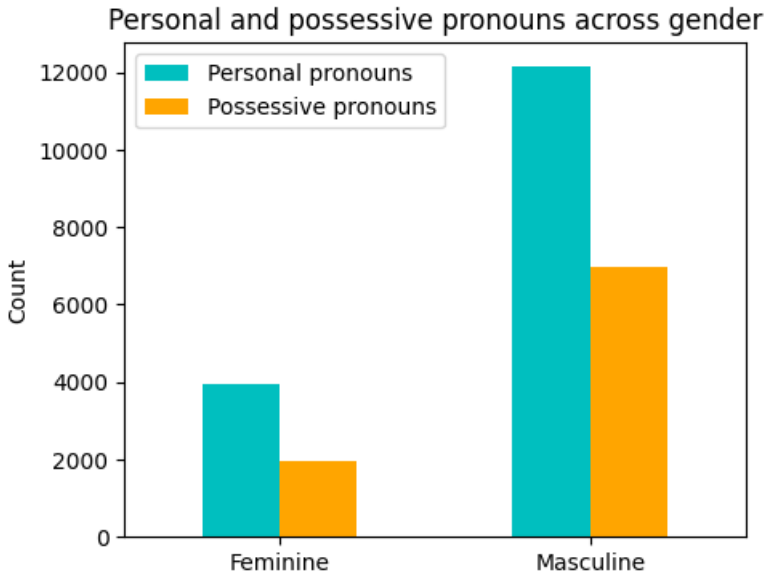
```
Percentage of the feminine personal pronoun in nominative form: 72.0736629667003 %.
-------------------------
Percentage of the feminine personal pronoun in object form: 27.926337033299696 %.
```

In [ ]:
```
pers_nom = df_pronouns.loc["he", "Personal pronouns"]*100/(
    df_pronouns_gender.loc["Masculine", "Personal pronouns"])
print("Percentage of the masculine personal pronoun in nominative form:", pers_nom, "%.")

print("-------------------------")

pers_obj = df_pronouns.loc["him", "Personal pronouns"]*100/(
    df_pronouns_gender.loc["Masculine", "Personal pronouns"])
print("Percentage of the masculine personal pronoun in object form:", pers_obj, "%.")
```

```
Percentage of the masculine personal pronoun in nominative form: 78.49444672974084 %.
-------------------------
Percentage of the masculine personal pronoun in object form: 21.505553270259153 %.
```

f. Illustrate the numbers from (d) with a bar chart.

In [ ]:
```
df_pronouns_gender.plot(kind = "bar", color = ["c", "orange"], figsize = (5, 4))
plt.title("Personal and possessive pronouns across gender")
plt.xticks(rotation = 0)
plt.ylabel("Count")
plt.legend(loc = "upper left")
plt.show()
```



g. Write a short text (200-300 words) and discuss the consequences of your findings. Why do you think the masculine pronoun is more frequent than the feminine pronoun? If you find that the there is a different distribution between nominative and object forms for the masculine and the feminine pronouns, why do you think that is the case? Do you see any consequences for the development of language technology in general, and for language technology derived from example texts in particular? Make use of what you know about the Brown corpus.

I think that the first reason why the masculine pronoun is more frequent than the feminine pronoun is a factor of society and common belief. This gap in frequency surely reflects gender biases, which have been reinforced over time by the traditional patriarchal historical context. Another factor that can have a high influence on this bias is the fact that, when dealing with english language, if we want to talk about a person in a generic way (without any specification about gender), we generally use the masculine pronouns (and not the feminine ones). Thus, the frequency of the masculine pronouns increases over the frequency of feminine pronouns.

For both masculine and feminine personal pronouns, the nominative form is the most prevalent (probably due to the fact that the nominative form is generally used for representing subjects, which tend to be mentioned more often in texts). In particular, this diffence is higher in masculine gender (78.5% - 21.5%) than in feminine gender (72% - 28%).

For sure, the difference in frequency of pronouns according to gender brings consequences for the development of language technology. This influence is given by the fact that, when developing models, we train them with a bias, where masculine pronouns dominate over feminine pronouns. As a consequence, even the models will be affected by this bias, thus creating a vicious circle and renforcing stereotypes.

Part 2 – Zipf's law of abbreviation (30 pts)

In [ ]:
```
# Open the book file
f = open("book.txt", "r", encoding="utf-8-sig")
t = f.read()
```

a. You will see that the downloaded text contains a preamble and a long appendix about the Gutenberg project and copyrights. Load the file with Python and remove these parts so that you only use the book text.

In [ ]:
```
# Remove the preamble
t = t.split("""*** START OF THE PROJECT GUTENBERG EBOOK THE ADVENTURES OF TOM SAWYER,
            COMPLETE ***""")[1]

# Remove the appendix
t = t.split("""*** END OF THE PROJECT GUTENBERG EBOOK THE ADVENTURES OF TOM SAWYER,
            COMPLETE ***""")[0]
```

b. Tokenize the text, remove the punctuation marks, and produce a frequency distribution of the word lengths. Format this distribution as a Pandas dataframe.

In [ ]:
```
# Tokenize the text
tokens = tok.word_tokenize(t, language = "english")

# Remove punctuation marks and put everything in lower case
tokens_new = [i.lower() for i in tokens if i.isalpha()]

# Produce frequency distribution of the word lenghts
d = {}
for word in tokens_new:
    word_length = len(word)
    if word_length in d:
        d[word_length] += 1
    else:
        d[word_length] = 1
print(d)

# Format as a dataframe
df = pd.DataFrame(d.items(), columns=['Length', 'Frequency'])
df
```

```
{3: 18723, 10: 810, 2: 11281, 6: 5213, 4: 14154, 5: 7542, 9: 1531, 7: 3843, 8: 2359, 1: 4508, 13: 73, 11: 458, 12: 200, 15: 7, 14: 33, 16: 1}
```

Out[ ]:

	Length	Frequency
0	3	18723
1	10	810
2	2	11281
3	6	5213
4	4	14154
5	5	7542
6	9	1531
7	7	3843
8	8	2359
9	1	4508
10	13	73
11	11	458
12	12	200
13	15	7
14	14	33
15	16	1

c. What are the five most frequent word lengths? How long are the longest words of the text?

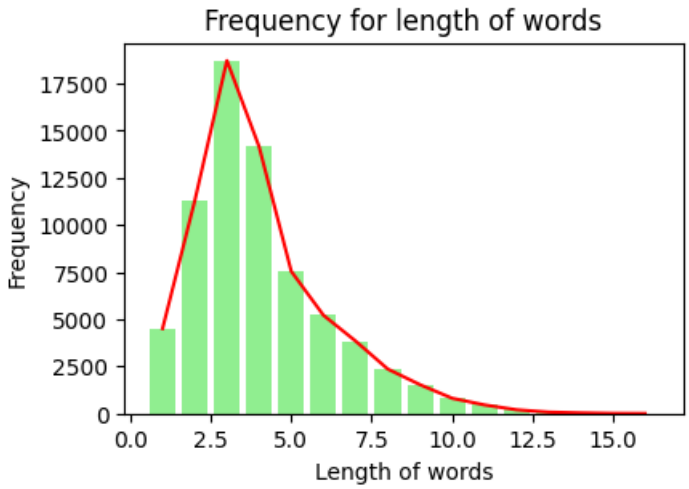
In [ ]:
```
# Sort the values by frequence and take the 5 highest values
print("The five most frequent word lengths are:")
print(df.sort_values("Frequency", ascending = False)[:5])

print("-----------------------------------------------")

# Sort the values by length of the words and then take the maximum value
```

```
print("The length of the longest words of the text is:")
print(df.sort_values("Length").max()[0])
print("The number of words with this length is:")
print(df.sort_values("Length", ascending = False).iloc[0][1])
```

```
The five most frequent word lengths are:
   Length  Frequency
0       3      18723
4       4      14154
2       2      11281
5       5       7542
3       6       5213
-----------------------------------------------
The length of the longest words of the text is:
16
The number of words with this length is:
1
```

d. Order the table by word length and produce a plot that shows the frequencies.

In [ ]:
```
# Sort the values by length
df_length = df.sort_values("Length")

# Create the plot
plt.figure(figsize = (4.5, 3))
plt.plot(df_length.Length, df_length.Frequency, color = "r")
plt.bar(df_length.Length, df_length.Frequency, color = "lightgreen")
plt.xlabel("Length of words")
plt.ylabel("Frequency")
plt.title("Frequency for length of words")
plt.show()
```

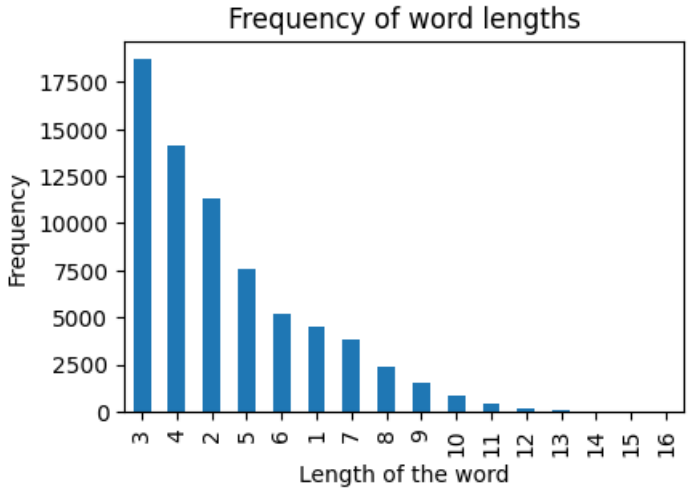


e. Also produce a visualization with the data ordered by decreasing frequency.

In [ ]:
```
# Sort the values by decreasing frequency
df_freq = df.sort_values("Frequency", ascending = False)
df_freq.index = df_freq.Length
print(df_freq)

df_freq.Frequency.plot(kind = "bar", figsize = (4.5, 3))
plt.xlabel("Length of the word")
plt.ylabel("Frequency")
plt.title("Frequency of word lengths")
plt.plot()
```

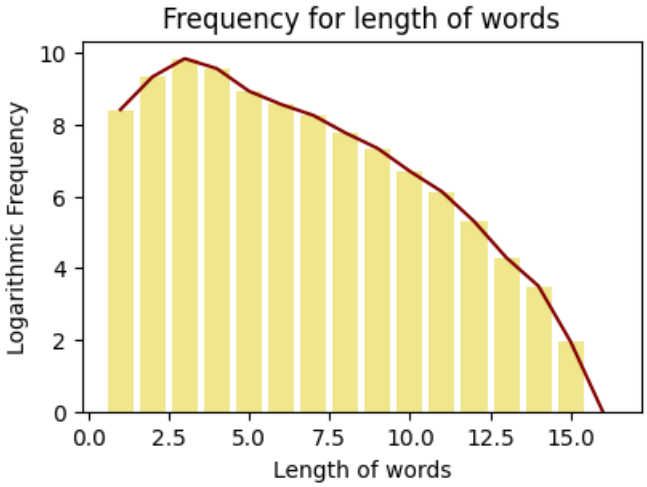
Length	Length	Frequency
3	3	18723
4	4	14154
2	2	11281
5	5	7542
6	6	5213
1	1	4508
7	7	3843
8	8	2359
9	9	1531
10	10	810
11	11	458
12	12	200
13	13	73
14	14	33
15	15	7
16	16	1

Out[ ]: []



f. When dealing with word frequency data, it is often recommended to plot frequencies on a logarithmic scale. How does this change the plots?

In [ ]:
```
# Create the plot
plt.figure(figsize = (4.5, 3))
plt.plot(df_length.Length, np.log(df_length.Frequency), color = "maroon")
plt.bar(df_length.Length, np.log(df_length.Frequency), color = "khaki")
plt.xlabel("Length of words")
plt.ylabel("Logarithmic Frequency")
plt.title("Frequency for length of words")
plt.show()
```



The result is different from the previous one: the Logarithmic Frequency of the words tends to follow a more stable behaviour than the Frequency. It goes from 8 to 10 for the shortest words (from length 1 to length 3), and then it

starts decreasing in a more constant way. On the contrary, for the previous plot, the decrease of the curve was very sharp, more or less exponential.

g. How well does this dataset match Zipf's law of abbreviation? In your opinion, which plot is most suitable to prove or disprove the law of abbreviation?

Zip's law of abbreviation states that the length of a word is inversely correlated with its frequency. It seems that this behaviour can be seen also in this case, in particular for the words of length higher than or equal to 3.

In my opinion, the best plot to prove the law of abbreviation is the first one I created (without the logarithmic scale). That is because with this graph I am able to understand well the relationship between length of words and frequency, proving that somehow the frequence decrease as the lenght increase.

h. Select one word length and investigate in detail which words of this word length occur in the text. Are these words specific to the Tom Sawyer text, or would you expect them to occur similarly frequently in other English texts? Or is there any evidence of preprocessing (e.g. tokenization) errors?

I want to investigate more about the words of length 3, as their frequency is very high with respect to the others.

In [ ]:
```
# Create a dictionary with all the words of length 3 and their frequencies
d = {}
for word in tokens_new:
    if len(word) == 3:
        if word not in d:
            d[word] = 1
        else:
            d[word] += 1

# Transform in a dataset and print it ordered by descending frequency
df_three = pd.DataFrame(d.items(), columns = ["Word", "Frequency"])
df_three = df_three.sort_values("Frequency", ascending = False)
print(df_three[0:25])
print("------------------")
print("Total frequence:")
print(sum(df_three.Frequency))
```

```
   Word  Frequency
0   the       3747
4   and       3005
57  was       1162
51  you        833
7   his        819
1   tom        777
24  but        545
19  for        510
71  had        510
65  him        411
52  she        394
44  all        318
25  not        282
2   her        278
38  out        267
77  now        255
60  don        213
41  one        184
40  got        171
8   joe        164
56  did        138
59  see        132
16  boy        129
28  ain        119
64  can        118
------------------
Total frequence:
18723
```

From this analysis, we can see that the most frequent words in this category are very common words in English (the, and, was, you..). So, I suppose that this behavior is normal, and I expect to see a very high frequence for words of length 3 also in other English texts. Surely, the fact that the protagonist of the novel is called Tom increases the number of times words of this length appear in the text. However, the frequency for the word Tom is not so high

(777), which is just around the 4% of the total words of length three (which are 18723). So, the word Tom does not give a high contribution to this behavious, that is just a normal behaviour due to English language.

In [ ]:
```
# See the percentage of word "Tom" with respect to all the words of length 3 in the text
print("""Percentage of \"Tom\" in the text,
      with respect to all the words of length 3:""")
print(df_three.iloc[5][1]/sum(df_three.Frequency)*100, "%.")
```

```
Percentage of "Tom" in the text,
      with respect to all the words of length 3:
4.149975965390162 %.
```

Part 3 – Identifying dialogue act types in chat messages (35 pts)

In [ ]:
```
# Load the chat data into a Pandas dataframe
nltk.download("nps_chat")
from nltk.corpus import nps_chat
data = []
for f in nps_chat.fileids():
    posts = nps_chat.xml_posts(f)
    for p in posts:
        data.append((p.get('class'), p.text))
df_chat = pd.DataFrame(data, columns=['label', 'text'])
print(df_chat.head(20))
```

```
[nltk_data] Downloading package nps_chat to
[nltk_data]     C:\Users\sanfe\AppData\Roaming\nltk_data...
[nltk_data]   Package nps_chat is already up-to-date!
```

	label	text
0	Statement	now im left with this gay name
1	Emotion	:P
2	System	PART
3	Greet	hey everyone
4	Statement	ah well
5	System	NICK :10-19-20sUser7
6	Accept	10-19-20sUser7 is a gay name.
7	System	.ACTION gives 10-19-20sUser121 a golf clap.
8	Emotion	:)
9	System	JOIN
10	Greet	hi 10-19-20sUser59
11	Statement	26/ m/ ky women that are nice please pm me
12	System	JOIN
13	System	PART
14	Statement	there ya go 10-19-20sUser7
15	Reject	don't golf clap me.
16	Reject	fuck you 10-19-20sUser121:@
17	whQuestion	whats everyone up to?
18	System	PART
19	System	PART

a. Answer the following questions:

• How many distinct labels are there, and how many instances per label?

In [ ]:
```
# Number of distinct labels
print("Number of distinct labels:")
print(len(df_chat.label.unique()))

print("-----------------------------")

# Number of instances per label
print("Number of instances per label:")
print(df_chat.label.value_counts())
```

```
Number of distinct labels:
15
------------------------------
Number of instances per label:
Statement     3185
System        2632
Greet         1363
Emotion       1106
ynQuestion     550
whQuestion     533
Accept         233
Bye            195
Emphasis       190
Continuer      168
Reject         159
yAnswer        108
nAnswer         72
Clarify         38
Other           35
Name: label, dtype: int64
```

• Try to understand what the labels mean, looking at some examples if necessary.

Understanding of the labels:

- Statement: sentences where something is stated;
- System: system messages (they are mostly "PART" and "JOIN");
- Greet: sentences with the aim of greeting someone;
- Emotion: expressions used to represent and reflect emotions;
- ynQuestion: questions which requires Yes or No as answers;
- whQuestion: questions starting with who, what, where, when or why;
- Accept: affermative answers or statments;
- Bye: sentences or expressions aimed at saying bye;
- Emphasis: sentences with exclamation marks, used to give emphasis to the message;
- Continuer: sentences used to continue other unfinished sentences (e. g. sentences containing and, or...);
- Reject: sentences with the aim or rejecting something or someone or saying no;
- yAnswer: short sentences (maybe answers) with affirmative meaning;
- nAnswer: short sentences (maybe answers) with negative meaning;
- Clarify: adverbs, words or expressions used to clarify or correct something previously written;
- Other: numbers, symbols and other unclear expressions.

• What is the average message length (in characters)?

In [ ]:
```
# Average message length (in characters),
# taking into account also the spaces between words
l = []
for i in df_chat.text:
    l.append(len(i))
np.mean(l)
```

Out[ ]:
```
21.910570644459167
```

- How do these numbers compare to the Movie Reviews corpus used in the last exercise set?

In the Movie Reviews corpus used in the last exercise set, the average length of reviews was much more higher than this case. I think it is mostly due to the fact that here we are talking about online chat conversations, where it is very common to write very short messages. On the contrary, for movies reviews, it is more natural to write a longer text. Moreover, in the Movie Review oìcorpus, we had just two distinct labels, while here we have 15 labels.

• Explain in a few sentences how these differences may impact the text classification performance. Which text classification methods or parameter settings do you expect to be better adapted to the NPS Chat problem?

In this case, we don't have a binary problem, but the labels to predict are 15. As a consequence, we need to apply a multi-class classification model. Both Naive Bayes and Logistic Regression can be suitable options in this context. I will compare both methods (even with some varaitions), in order to find which is the best method in this case.

• Split the data into training and test data. To simplify things, we will just use a two-way split of 90% training data and 10% test data in this first experiment.

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(df_chat["text"],
                                                    df_chat["label"],
                                                    test_size = 0.1,
                                                    random_state = 42)
```

• Instantiate a CountVectorizer with default parameters and fit it to the training data.

In [ ]:
```
cv = CountVectorizer()
x_train_bow = cv.fit_transform(x_train)
```

• Instantiate a MultinomialNB model and fit it to the training data.

In [ ]:
```
nb = MultinomialNB()
nb.fit(x_train_bow, y_train)
```

Out[ ]:
```
MultinomialNB()
```

• Predict the test set labels using the trained model and display the classification report.

In [ ]:
```
x_test_bow = cv.transform(x_test)
# Prediction
predicted_y = nb.predict(x_test_bow)

# Classification report
print(metrics.classification_report(y_test, predicted_y))
```

```
              precision    recall  f1-score   support

      Accept       0.00      0.00      0.00        23
         Bye       0.00      0.00      0.00        14
     Clarify       0.00      0.00      0.00         3
   Continuer       0.00      0.00      0.00        16
     Emotion       0.92      0.61      0.74       116
    Emphasis       0.00      0.00      0.00        12
       Greet       0.79      0.93      0.85       135
       Other       0.00      0.00      0.00         2
      Reject       0.00      0.00      0.00        21
   Statement       0.63      0.75      0.69       315
      System       0.65      0.98      0.78       292
     nAnswer       0.00      0.00      0.00         7
  whQuestion       0.83      0.12      0.21        41
     yAnswer       0.00      0.00      0.00         8
  ynQuestion       0.80      0.08      0.14        52

    accuracy                           0.69      1057
   macro avg       0.31      0.23      0.23      1057
weighted avg       0.64      0.69      0.62      1057
```

b. Write a few lines about your observations: Which classes are the easiest / most difficult to predict? Are there classes where precision differs drastically from recall, and if so, what does this mean?

The easiest class to predict is System, with a Precision of 0.94 and a Recall of 0.97. The model works well also in predicting Emotion, Greet and Statement (with high values for both precision and recall). On the contrary, the most difficult classes to predict are Clarify, Continuer, nAnswer, Emphasis, Reject, yAnswer and Other with both precision and recall equal to 0.

There are some classes where precision and recall are drastically different. These classes are: Accept and Bye, especially the first one. In particular, for these classes, precision is very high with respect to recall. This means that the model has high probability of being right when predicting a positive value, while it has low probability of identifying correctly all the positives.

c. Produce the cross-validation scores with the initial Naïve Bayes model.

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(df_chat["text"],
                                                    df_chat["label"],
                                                    test_size = 0.1,
                                                    random_state = 42)

cv = CountVectorizer()
x_train_bow = cv.fit_transform(x_train)

nb = MultinomialNB()
nb.fit(x_train_bow, y_train)

x_test_bow = cv.transform(x_test)
predicted_y = nb.predict(x_test_bow)

# Five-fold cross-validation
scores = cross_val_score(nb, x_train_bow, y_train, cv=5)
print("{:.3f} accuracy with a standard deviation of {:.3f}".format(scores.mean(),
                                                                   scores.std()))
```

```
0.701 accuracy with a standard deviation of 0.008
```

d. By default, the CountVectorizer lowercases all input and uses a simple whitespace-based tokenizer. Check if other settings provide better results.

In [ ]:
```
from nltk.tokenize import TweetTokenizer

tw = TweetTokenizer()

def tokenizer_tweet(text):
    return tw.tokenize(text)

cv = CountVectorizer(tokenizer = tokenizer_tweet, token_pattern = None)
x_train_bow = cv.fit_transform(x_train)

nb = MultinomialNB()
nb.fit(x_train_bow, y_train)

x_test_bow = cv.transform(x_test)
predicted_y = nb.predict(x_test_bow)

# Five-fold cross-validation
scores = cross_val_score(nb, x_train_bow, y_train, cv=5)
print("{:.3f} accuracy with a standard deviation of {:.3f}".format(scores.mean(),
                                                                   scores.std()))
```

```
0.750 accuracy with a standard deviation of 0.006
```

There seems to be an improvement in the accuracy of the model.

e. By default (and as its name implies), the CountVectorizer produces frequency counts. Evaluate the impact of binary feature values (presence or absence of a word in an instance).

In [ ]:
```
cv = CountVectorizer(binary = True, tokenizer = tokenizer_tweet, token_pattern = None)
x_train_bow = cv.fit_transform(x_train)

nb = MultinomialNB()
nb.fit(x_train_bow, y_train)

x_test_bow = cv.transform(x_test)
predicted_y = nb.predict(x_test_bow)

# Five-fold cross-validation
scores = cross_val_score(nb, x_train_bow, y_train, cv=5)
print("{:.3f} accuracy with a standard deviation of {:.3f}".format(scores.mean(),
                                                                   scores.std()))
```

```
0.746 accuracy with a standard deviation of 0.007
```

It seems that the binary feature makes the model worse.

What is the difference between the binary multinomial model and the Bernoulli model (both in terms of scores and theoretically)?

Theoretically, the difference between Multinomial and Bernoulli NB is that Multinomial NB uses count-based bags of words (you count how many times a word is present in a text), while Bernoulli NB uses binary counts (you register only whether a term is present or not in a text: 1 if it present, 0 if it is not present). As a consequence, Multinomial NB considers only the presents terms, ignoring absent terms. On the contrary, Bernoulli NB considers both the present and the absent terms.

In [ ]:
```
cv = CountVectorizer(tokenizer = tokenizer_tweet, token_pattern = None)
x_train_bow = cv.fit_transform(x_train)

nb = BernoulliNB()
nb.fit(x_train_bow, y_train)

x_test_bow = cv.transform(x_test)
predicted_y = nb.predict(x_test_bow)

# Five-fold cross-validation
scores = cross_val_score(nb, x_train_bow, y_train, cv=5)
print("{:.3f} accuracy with a standard deviation of {:.3f}".format(scores.mean(),
                                                                   scores.std()))
```

```
0.654 accuracy with a standard deviation of 0.005
```

Bernoulli NB has a very low accuracy with respect to multinomial NB.

f. We know that Logistic Regression may produce better results than Naive Bayes. We will see what happens if we use Logistic Regression instead of Naive Bayes on this task. Keep the same CountVectorizer as before and replace the Naïve Bayes classifier by a Logistic Regression one with default parameters.

In [ ]:
```
cv = CountVectorizer(tokenizer = tokenizer_tweet, token_pattern = None)
x_train_bow = cv.fit_transform(x_train)

lr = LogisticRegression(max_iter = 200)
lr.fit(x_train_bow, y_train)

x_test_bow = cv.transform(x_test)
predicted_y = lr.predict(x_test_bow)

# Five-fold cross-validation
scores = cross_val_score(lr, x_train_bow, y_train, cv=5)
print("{:.3f} accuracy with a standard deviation of {:.3f}".format(scores.mean(),
                                                                   scores.std()))
```

```
0.823 accuracy with a standard deviation of 0.005
```

As expected, the accuracy is higher with Logistic Regression.

g. Evaluate different regularization settings (L2, L1, different values of C). The supported types of regularization depend on the Solver used for gradient descent. The documentation provides more details about the different available solvers: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

In [ ]:
```
cv = CountVectorizer(tokenizer = tokenizer_tweet, token_pattern = None)
x_train_bow = cv.fit_transform(x_train)

lr = LogisticRegression(solver = "liblinear")
lr.fit(x_train_bow, y_train)

x_test_bow = cv.transform(x_test)
predicted_y = lr.predict(x_test_bow)

# Five-fold cross-validation
scores = cross_val_score(lr, x_train_bow, y_train, cv=5)
print("{:.3f} accuracy with a standard deviation of {:.3f}".format(scores.mean(),
                                                                   scores.std()))
```

```
0.818 accuracy with a standard deviation of 0.004
```

In [ ]:
```
cv = CountVectorizer(tokenizer = tokenizer_tweet, token_pattern = None)
x_train_bow = cv.fit_transform(x_train)

lr = LogisticRegression(penalty = "l1",solver = "liblinear")
lr.fit(x_train_bow, y_train)
```

```
x_test_bow = cv.transform(x_test)
predicted_y = lr.predict(x_test_bow)

# Five-fold cross-validation
scores = cross_val_score(lr, x_train_bow, y_train, cv=5)
print("{:.3f} accuracy with a standard deviation of {:.3f}".format(scores.mean(),
                                                                   scores.std()))
```

```
0.821 accuracy with a standard deviation of 0.007
```

In [ ]:
```
cv = CountVectorizer(tokenizer = tokenizer_tweet, token_pattern = None)
x_train_bow = cv.fit_transform(x_train)

lr = LogisticRegression(solver = "liblinear", C = 4.0)
lr.fit(x_train_bow, y_train)

x_test_bow = cv.transform(x_test)
predicted_y = lr.predict(x_test_bow)

# Five-fold cross-validation
scores = cross_val_score(lr, x_train_bow, y_train, cv=5)
print("{:.3f} accuracy with a standard deviation of {:.3f}".format(scores.mean(),
                                                                   scores.std()))
```

```
0.825 accuracy with a standard deviation of 0.005
```

The last model seems to be the one with the highest accuracy.

h. Retrain the best model on the entire training set and evaluate it on the test set. Report the detailed evaluation scores and compare them with the initial Naïve Bayes scores. On which classes did your model improve the most? Which classes are still difficult to predict?

In [ ]:
```
cv = CountVectorizer(tokenizer = tokenizer_tweet, token_pattern = None)
x_train_bow = cv.fit_transform(x_train)

lr = LogisticRegression(solver = "liblinear", C = 4.0)
lr.fit(x_train_bow, y_train)

x_test_bow = cv.transform(x_test)
predicted_y = lr.predict(x_test_bow)

# Five-fold cross-validation
scores = cross_val_score(lr, x_train_bow, y_train, cv=5)
print("{:.3f} accuracy with a standard deviation of {:.3f}".format(scores.mean(),
                                                                   scores.std()))
print("---------------------------------")

# Classification report
print(metrics.classification_report(y_test, predicted_y))
```

```
0.825 accuracy with a standard deviation of 0.005
---------------------------------
              precision    recall  f1-score   support

      Accept       0.59      0.43      0.50        23
         Bye       0.92      0.79      0.85        14
     Clarify       0.00      0.00      0.00         3
   Continuer       0.36      0.25      0.30        16
     Emotion       0.87      0.77      0.82       116
    Emphasis       0.55      0.50      0.52        12
       Greet       0.92      0.96      0.94       135
       Other       1.00      1.00      1.00         2
      Reject       0.67      0.10      0.17        21
   Statement       0.76      0.90      0.82       315
      System       1.00      0.97      0.99       292
     nAnswer       0.71      0.71      0.71         7
  whQuestion       0.84      0.88      0.86        41
     yAnswer       0.57      0.50      0.53         8
  ynQuestion       0.80      0.69      0.74        52

    accuracy                           0.85      1057
   macro avg       0.70      0.63      0.65      1057
weighted avg       0.85      0.85      0.84      1057
```

In this case, the scores seem to be way better than the initial Naïve Bayes scores. In particular, we can see that the classes Continuer, Emphasis, Other, Reject, yAnswer and nAnswer have now higher values than in the previous case (where their precision and recall were both 0). Moreover, higher values are also present in other classes.

There are some classes that are still difficult to predict, but less than before: now we have only Clarify with zero values in both precision and recall. Some classes also have very high values of both recall and precision: Greet, Other and System are predicted almost perfectly.