IN4080 – 2023
Exercise set 2

Friday 1 September 2023

Part 1: Probability distributions

In this exercise, we explore the relationship between word length and grammatical categories (parts of speech).

NLTK provides the Brown corpus of English together with its part-of-speech tags. You will first have to download the corpus:

```
import nltk
nltk.download("brown")
```

You can then load the corpus and the list of word-tag pairs with the following command:

```
from nltk.corpus import brown
wordlist = brown.tagged_words()
wordlist[:20]
```

You will see that there are composite tags containing – or +, such as NP-TL or DTS+BEZ. To reduce the amount of tags, we will only consider the first part of the tag, i.e. NP or DTS.

Create a dictionary that contains, for each word length in characters and for each (simplified) part of speech tag, the number of occurrences of words in the corpus. For further processing, it is easiest to create a dictionary of dictionaries, with the tags and lengths as keys. The resulting dictionary might look as follows:

```
{'AT': {3: 69969, 2: 5546, 1: 23070, 5: 492},
 'NP': {6: 7697, 7: 5816, 17: 18, 3: 3070, 4: 5825, …},
 'NN': {6: 25413, 4: 31089, 13: 1607, 7: 20324, …},
 …}
```

You can convert this dictionary easily into a Pandas dataframe with the following command:

```
import pandas as pd
df = pd.DataFrame.from_dict(d)
df
```

You will see that missing values will be displayed as NaN. You can replace them with 0s using the following command:

```
df = df.fillna(0)
```

You should get a table that looks approximately as follows:

	AT	NP	NN	JJ	VBD	NR	IN	NP$
3	69969.0	3070.0	11513	4666.0	1603.0	0.0	10145.0	3.0
2	5546.0	1648.0	1495	14.0	1.0	0.0	85567.0	0.0
1	23070.0	431.0	519	0.0	0.0	0.0	113.0	0.0
5	492.0	6163.0	27659	10572.0	3208.0	603.0	3793.0	122.0
6	0.0	7697.0	25413	7525.0	5163.0	232.0	3226.0	386.0
7	0.0	5816.0	20324	8305.0	3960.0	96.0	3202.0	481.0
17	0.0	18.0	103	95.0	1.0	0.0	0.0	0.0
4	0.0	5825.0	31089	8771.0	6679.0	599.0	15861.0	6.0
8	0.0	3786.0	15720	9022.0	2919.0	205.0	46.0	671.0
10	0.0	1446.0	9717	4858.0	620.0	0.0	208.0	272.0
9	0.0	2037.0	11831	6922.0	1665.0	160.0	402.0	493.0
11	0.0	452.0	5860	3292.0	252.0	0.0	24.0	156.0
13	0.0	171.0	1607	1281.0	12.0	0.0	0.0	34.0

You can sort the rows according to length with df.sort_index(). You can switch rows and columns with df.transpose().

Which are the five most frequent part-of-speech tags? If you are not familiar with the labels, you find the list here: http://korpus.uib.no/icame/manuals/BROWN/INDEX.HTM#bc6

Which are the five most frequent word lengths?

Create another dataframe that represents the **joint probability distribution** of the two random variables tag and length, i.e. $P(\text{tag} = t, \text{length} = l)$. Make sure that the resulting values correspond to your expectations. The joint distribution answers the question: what is the probability that a randomly selected word form in the corpus has tag t and length l?

Create two additional dataframes with the **conditional probability distributions**. One of them should represent $P(\text{tag} = t \mid \text{length} = l)$ and the other one $P(\text{length} = l \mid \text{tag} = t)$.

What is the probability that a common noun (NN) has length 5?
What is the probability that a word of length 1 is an article (AT)?
What is the average length of adjectives (JJ)?

Part 2: Vectors, matrices, linear algebra basics

Exercise 1: Let us first define two vectors: $\boldsymbol{u} = [1, 2, 3, 5]$ and $\boldsymbol{v} = [-1, 0, 3, -1]$.

Compute the following expressions, both by hand and using NumPy:

- $\boldsymbol{u} + \boldsymbol{v}$
- $2 \cdot \boldsymbol{v}$
- $\boldsymbol{u} \cdot \boldsymbol{v}$ (the dot product)
- $\|\boldsymbol{u}\|$ (the norm of $\boldsymbol{u}$)
- $\text{sim}(\boldsymbol{u}, \boldsymbol{v})$ (the cosine similarity between the two vectors)

Exercise 2: What is the product $\boldsymbol{AB}$ of the two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ defined below?

$$\boldsymbol{A} = \begin{bmatrix} 3 & -2 & 1 \\ -2 & 1 & -1 \\ 3 & 1 & -2 \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} 2 & -1 \\ -2 & 0 \\ 1 & 1 \end{bmatrix}$$

Again, first compute the result by hand and then check it with NumPy.

Exercise 3: You are given a vector of word length counts computed from a short text:

```
l = [0, 11, 42, 53, 52, 60, 77, 55, 45, 33, 10, 2, 1, 0, 1]
```

There are $l[i]$ words of length i in the text. For example, there are 52 words of length 4. (Note that $l[0] = 0$ because words are always at least one character long.) Use NumPy functions to figure out:

- What is the length of the text in words?
- Which word length is the most frequent?
- What percentage of words has length 10?
- What is the total character count of the words of the text? (Hint: You can use the dot product.)

Part 3: Getting started with Naïve Bayes

For our first Naïve Bayes classifier, we will se the Movie Reviews corpus that comes with NLTK:

```
from nltk.corpus import movie_reviews
```

The following commands allow us to determine the available labels, the files with the documents of a particular label, and the raw text of one file:

```
movie_reviews.categories()
movie_reviews.fileids('pos')
movie_reviews.raw('pos/cv000_29590.txt')
```

The following command builds a list of text-label pairs:

```
movie_docs = [(movie_reviews.raw(fileid), label)
                for label in movie_reviews.categories()
                        for fileid in movie_reviews.fileids(label)
                ]
```

We transform this list into a Pandas dataframe for easier manipulation:

```
df = pd.DataFrame(movie_docs , columns=['text', 'label'])
```

Next, we shuffle the data and split it into training (80%), validation (10%) and test (10%) set. Scikit-Learn contains a method to split a dataset into two parts:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(df['text'], df['label'], test_size=0.1)
```

You'll have to apply this function twice to get a three-way split.

For basic tokenization and bag-of-words feature extraction, we can use the CountVectorizer class from scikit-learn:

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
x_train_bow = cv.fit_transform(x_train)
```

The following commands may help you inspect the features:

```
cv.vocabulary_
cv.vocabulary_['boring']
```

Now, we train a multinomial Naïve Bayes classifier, using again existing libraries from scikit-learn:

```
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
nb.fit(x_train_bow, y_train)
```

Let us evaluate this model on the validation set. We first need to transform it to the bag-of-words representation, using the same CountVectorizer already used for training:

```
x_val_bow = cv.transform(x_val)
predicted_y_val = nb.predict(x_val_bow)
```

Now, we can compare the predictions with the gold labels of the validation set and compute accuracy:

```
from sklearn import metrics
acc = metrics.accuracy_score(y_val, predicted_y_val)
```

Scikit-learn also provides a "classification report" that shows precision and recall for all classes:

```
print(metrics.classification_report(y_val, predicted_y_val))
```

You can also display the raw confusion matrix, but note that scikit-learn displays the gold labels as the rows and the predicted labels as columns:

```
print(metrics.confusion_matrix(y_val, predicted_y_val))
```

Create two other classifiers by changing some parameters either in the data extraction (*CountVectorizer*) or the training (*MultinomialNB*) step. Select the best of the three models based on the development set performance and use that model for predicting and evaluating the test set (don't forget to vectorize the test set with the appropriate CountVectorizer!).

A few suggestions for modifying the parameters:

- The corpus is already lowercased and tokenized, so there are limited options in that respect, but you may still try to improve tokenization.
- The *CountVectorizer* documentation is available here: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
  In particular, you could experiment with the *stop_words* and *binary* parameters.
- The *MultinomialNB* documentation is here: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
  The *alpha* parameter specifies the amount of smoothing and can be changed.