

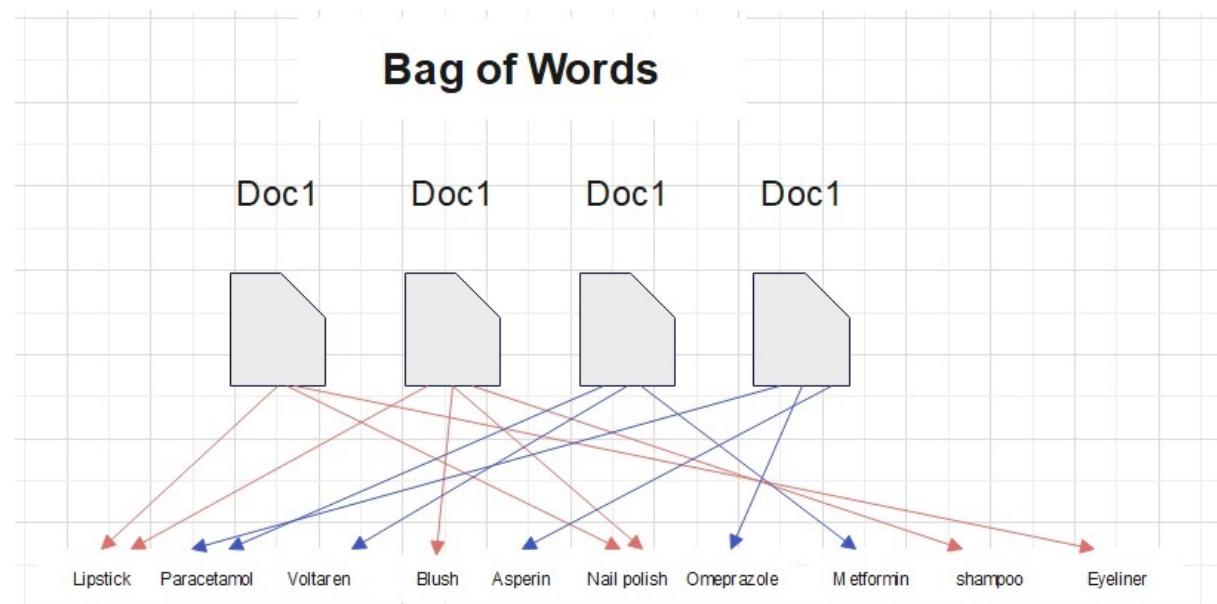
## Stimmungsanalyse

Stimmungsanalysen oder Stimmungsklassifizierungen fallen in die breite Kategorie der Textklassifizierungsaufgaben, um die Stimmung hinter einer Phrase oder einer Liste von Phrasen in positiv, negativ oder neutral zu klassifizieren. Bei Stimmungsanalyseaufgaben vermitteln nicht alle Wörter in einer Phrase die Stimmung der Phrase. Wörter wie „Ich“, „Sind“, „Bin“ usw. tragen zu keinerlei Gefühlen bei und daher ist es wichtig, die Merkmalsauswahl anzuwenden, damit nur die relevantesten Merkmale für die Klassenbezeichnungen analysiert und extrahiert werden die Stimmung des Satzes. Im Allgemeinen erweist sich das Identifizieren und Extrahieren der relevanten Merkmale als eine herausfordernde Aufgabe.

Stimmungsanalysemodelle sind in zwei Hauptkategorien unterteilt:

1. Ein Wortbeutel-Modell: Dies wird durch die Erstellung eines Wortbeutel-Vektors erreicht, der den Wortschatz angibt. Ein Wortbeutel-Vektor enthält alle eindeutigen Wörter im Korpus. Jetzt wird jeder einzelne Satz durch diesen Wortbeutelvektor codiert, der auch Merkmalsvektor genannt wird. Als Vektorisierungsmethode können wir den Counter Vectorizer verwenden, um die Worthäufigkeiten als Werte im Wortbeutelvektor zu speichern. Wenn wir zum Beispiel einen Text mit zwei Sätzen haben: „Wir hatten einen schönen Urlaub. Das Ambiente war erstaunlich“, der Wortschatzvektor für diesen Text sollte lauten: ['wir':1, 'hatten':1, 'einen':1, 'schönen':1, 'Urlaub':1, 'das':1, 'Ambiente':1, 'war':1, 'erstaunlich':1]. Der erste Satz wird also durch [1, 1, 1, 1, 0, 0, 0] dargestellt.
2. Zeitreihenansatz: Hier wird jedes Wort durch einen einzelnen Vektor dargestellt. Ein Satz wird also als Vektor von Vektoren dargestellt.

In unserem Beispiel verwenden wir das Bag-of-Words-Modell. Der erste Schritt steht in der Definition des „Bag of Words“. Dies ist ein grundlegendes Konzept im NLP, das den Wortschatz in einem Korpus durch ein Vector angibt. Nachdem wir alle Dokumente oder Sätze des Korpus in Merkmalsvektoren codiert haben, werden Merkmalsvektoren zusammen mit Klassenbezeichnungen der Dokumente als Trainingsdaten an den Sentiment-Klassifikator übergeben.



Die Stimmungsanalyse kann mit mehreren Klassifikatoren durchgeführt werden. Naive Bayes ist ein überwachter Lernalgorithmus zur Stimmungsanalyse, der zur Berechnung der Klassifizierungswahrscheinlichkeiten der Eingabetexte verwendet wird. Es handelt sich um einen reduzierten Bayes'schen Klassifikator mit einer reduzierten Anzahl von Parametern und einer linearen Zeitkomplexität im Gegensatz zu der exponentiellen Zeitkomplexität, die der Bayes'sche Klassifikator erfordert. Der Naive-Bayes-Klassifikator ergibt sich aus der folgenden Formel:

$$C_{NB} = \operatorname{argmax}_{c \in C} P(c) \prod_{x \in X} P(x|c)$$

Wobei  $\operatorname{argmax} P(c)$ , das auch als Maximum a posteriori oder MAP bekannt ist, die Klasse  $c$  ist, für die die Wahrscheinlichkeit  $P(c)$  für  $c \in C$  am größten ist; wobei  $C$  die Menge der Klassen ist. Wenn beispielsweise  $P(+ > P(-)$ , gibt  $\operatorname{argmax} P(c)$  (+) zurück, vorausgesetzt  $C = \{+, -\}$ . Für die Stimmungsanalyse ist  $P(x|c)$  die Wahrscheinlichkeit des Wortes  $x \in X$  bei gegebener Klasse  $c$ , wobei  $X$  das Vokabular des Korpus ist.

Der naive Bayes-Klassifikator verwendet Trainingsdaten, um sowohl MAP als auch  $p(x|c)$  zu bestimmen. Für MAP berechnet Naive Bayes die Klassenprioritäten wie folgt:

Prior für eine bestimmte Klasse = Anzahl der Proben in dieser Klasse/Gesamtanzahl der Proben.

Naive Bayes geht davon aus, dass  $P(x_1|c_j)$ ,  $P(x_2|c_j)$  usw. unabhängig voneinander sind. Mit dieser Annahme ist gemeint, dass nur die Häufigkeit und nicht die Wortposition in die Berechnungen einbezogen wird. Aufgrund der Annahme unabhängiger Prädiktoren weist der Naive-Bayes-Klassifikator jedoch einige Einschränkungen auf. Im wirklichen Leben ist es fast unmöglich, eine Reihe völlig unabhängiger Prädiktoren zu erhalten.

Ein weiterer Sentiment-Klassifikator ist Random Forest, eine Reihe von Entscheidungsbaum-Klassifikatoren, die für verschiedene Teilstichproben des Datensatzes verwendet werden und Mittelung verwenden, um die Vorhersagegenauigkeit zu verbessern und Überanpassungen zu kontrollieren. Die Teilstichprobengröße wird mit dem Parameter `max_samples` gesteuert, wenn `Bootstrap=True` (Standard), andernfalls wird der gesamte Datensatz zum Erstellen jedes Baums verwendet. Später in diesem Kapitel werden wir die Leistung beider Methoden vergleichen.

#### Anwendungsfälle :

Verschiedene Arten von Unternehmen können die Stimmungsanalyse nutzen, um tiefe Einblicke in die Eindrücke und Meinungen ihrer Kunden zu einer bestimmten Dienstleistung oder einem Produkt zu gewinnen. Es ist wichtig zu wissen, wo und wie man die Stimmungsanalyse einsetzt. Ein Anwendungsfall der Stimmungsklassifizierung besteht darin, die Kundenstimmung im Laufe der Zeit zu verfolgen, um ungewöhnliche Anstiege der Verbraucherstimmung zu einem Produkt oder einer Dienstleistung herauszufinden. Dies kann der Kundendienstabteilung helfen, die tatsächlichen Gründe für dieses Phänomen zu ermitteln, anstatt falsche Informationen zu erhalten. Eine Stimmungsanalyse kann auch bei der Planung von Verbesserungen in einigen Aspekten des Geschäfts hilfreich sein. Beispielsweise können wir feststellen, ob Kunden Schwierigkeiten bei der Verwendung einiger Produkte haben, indem wir die Stimmung in Verbraucherbewertungen analysieren, um Beschwerden im Zusammenhang mit der „Benutzerfreundlichkeit“ zu ermitteln. Darüber hinaus können wir die Stimmungsklassifizierung verwenden, um Kundenprobleme zu priorisieren, indem wir die Aspekte extrahieren, mit denen die Verbraucher am meisten unzufrieden sind. Dies hilft dem Dienstleister, schnell auf das negative Feedback zu reagieren.

## Stimmungsanalyse Übung

### Importieren des Datensatzes:

Wir werden den Twitter-Datensatz für die Stimmungsanalyse verwenden. Das Hauptziel besteht darin, Dateneinträge nach dem emotionalen Inhalt in positive, negative und neutrale Tweets zu klassifizieren. Der Datensatz kann [hier](#) heruntergeladen werden. Der Datensatz besteht aus zwei Spalten: „clean\_text“ und „label“ mit etwa 163.000 Zeilen.

```
In [4]: import pandas as pd
# three sentiments here: negative(-1), neutral(0), and positive(+1).
data = pd.read_csv("Twitter_Data.csv")

# rename 'category' column into 'label'
data.rename(columns = {'category':'label'}, inplace = True)
data.head(1)[['clean_text']][0]
```

Out[4]: 'when modi promised “minimum government maximum governance” expected him begin the difficult job reforming the state why does t  
ake years get justice state should and not business and should exit psus and temples'

```
In [3]: # explore the distribution of the labels
data['label'].value_counts()
```

Out[3]:

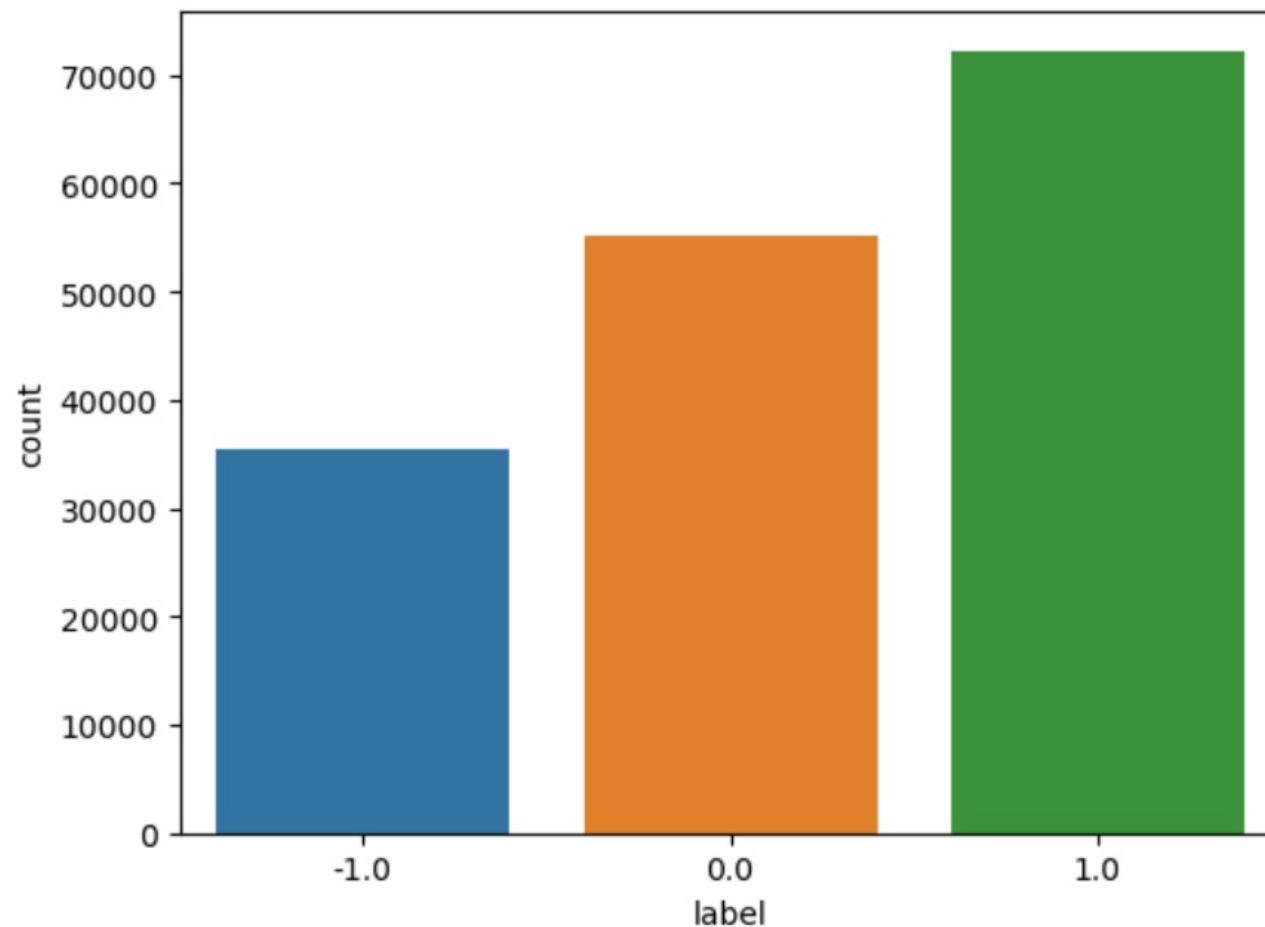
1.0	72250
0.0	55213
-1.0	35510

Name: label, dtype: int64

Werfen wir einen Blick auf die Labelverteilung der Tweets:

```
In [14]: import seaborn as sns
ax=sns.countplot(data.label)
```

Wie diese Abbildung zeigt, geben laut dieser Grafik rund 30 % der Tweets neutrale Meinungen an. Die meisten Tweets in diesem Datensatz drücken tendenziell eine positive Einstellung zum Tweet-Thema aus.

**Datensatzvorbereitung:**

Dieser Schritt umfasst die Bereinigung und Vorbereitung der Daten für das Training. Dazu gehört das Entfernen von Stopwörtern und Satzzeichen, da diese sich nicht auf bestimmte Emotionen oder Einstellungen im Kontext beziehen und somit nicht zum Lernprozess des verwendeten Algorithmus beitragen können. Gensim- und NLTK-Bibliotheken werden verwendet, um den Satz von Stopwörtern und Satzzeichen zu definieren, der für den Trainingssatz gelöscht werden soll. Bei Bedarf besteht die Möglichkeit, die eingestellten Stopwörter zu erweitern.

```
In [5]: import gensim
import nltk
from gensim.utils import simple_preprocess
nltk.download('stopwords')
from nltk.corpus import stopwords
#stop words of the English dictionary
stop_w = stopwords.words('English')
#stop_words.extend(['from', 'subject', 're', 'edu', 'use', 'of', 'as', 'by', 'uc'])
|
def process(text):

    # deacc=True removes punctuations
    no_punc = gensim.utils.simple_preprocess(str(text), deacc=True)
    return [word for word in no_punc if word not in stop_w] # for each doc in the text, remove stop words
data['clean_text'] = data['clean_text'].apply(process)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\la2022\AppData\Roaming\nltk_data...
[nltk_data]     Package stopwords is already up-to-date!
```

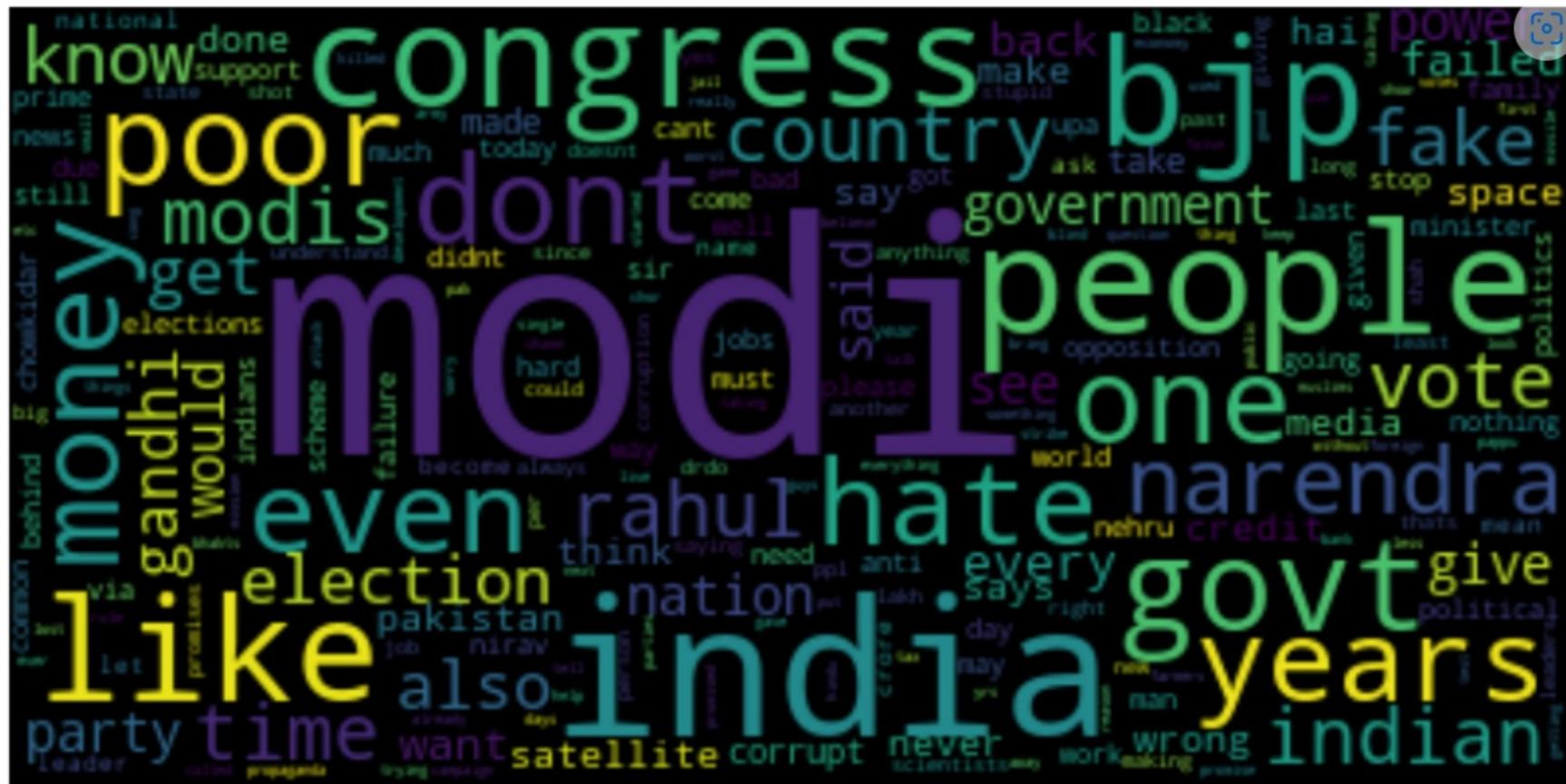
#### Datenvisualisierung:

Mithilfe der 'wordcloud' aus dem Wordcloud-Python-Modul, visualisieren wir Worthäufigkeiten in der Spalte „clean\_text“, die die Tweets darstellen. Diese Wordclouds helfen uns, Hassreden und Rassismus in den Tweets zu erkennen. Beispielsweise zeigt die nächste Wordcloud die häufigsten Wörter in negativ klassifizierten Tweets, wobei wir in diesen Tweets negative Ausdrücke und Wörter wie „Hass“, „falsch“, „korrupt“ und „falsch“ leicht erkennen können.

```
In [23]: # data visualization
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import collections
from collections import Counter
# get individual words for negative tweets
neg_rows = [r for r in data['clean_text'][data['label']== -1.0]]
neg_words = []
for twt in neg_rows :
    neg_words.extend(twt)

# Counter is a subclass for counting objects.
# It is a collection where elements are stored as dictionary keys and their counts are stored as dictionary values
# negative word frequencies
neg_freq = Counter(neg_words)

# positive words cloud
neg_cloud = WordCloud(
    background_color='black',
    max_words=2000,
    stopwords=stopwords
).generate_from_frequencies(neg_freq)
plt.figure(figsize=(10,9))
plt.imshow(neg_cloud, interpolation='bilinear') # display as an image, we could use interpolation
plt.axis('off')
plt.show()
```



## **Erstellen eines Sentiment-Analyse-Klassifikators mithilfe von Random Forest**

Als nächstes trainieren wir den Random Forest (RF)-Klassifikator von 350 Entscheidungsbäumen auf 80 % der Daten, nachdem wir mit TfIdfTransformer aus der sklearn-Klasse eine normale Transformation auf den Datensatz angewendet haben. Obwohl RF keine Normalität in den Daten voraussetzt, wird hier eine Transformation verwendet, um Textdaten in numerische Daten umzuwandeln, bevor das RF-Modell angewendet wird. 20 % des Datensatzes werden zum Testen des HF-Modells verwendet. Wir führen den Algorithmus parallel aus und nutzen dabei alle CPU-Kerne, um das Training der Zufallsstruktur zu beschleunigen.

Das Bag-of-Words-Modell ist eine vereinfachende Darstellung, die im Random-Forest-Klassifikator verwendet wird. In diesem Modell wird ein Text als Matrix von Token-Zählungen oder als Feature-Tabelle dargestellt. Die CountVectorizer-Funktion von SciKit Learn erstellt aus den Tweets die Bag of Words, die vom Random Forest-Algorithmus zum Erstellen des Klassifikators verwendet werden.

```
#The bag-of-words model is a simplifying representation used in Natural Language processing.
#In this model, a text is represented as the bag of its words (independent features) ,disregarding grammar but keeping multiplicity
# We will use SciKit Learn's CountVectorizer function which will convert a collection of
# text documents into a matrix of token counts or feature table
# we will use TfidfTransformer as a normalization method

# drop Nan values from the data
data = data.dropna()
#Split data into training and testing sets
from sklearn.model_selection import train_test_split
# join clean_txt lists into strings to apply CountVectorizer, otherwise it gives an error

x_train, x_test, y_train, y_test = train_test_split(data["clean_text"].map(' '.join),
                                                    data["label"], test_size = 0.2, random_state = 42) # 20% of the data for testing the model

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
count_vect = CountVectorizer(stop_words='english')
# normal distribution is not assumed by random forest, but we apply transformation to transform
# textual data(categorical data) into numerical.
# alternatively we can apply one-hot encoding for the same purpose.
# L2: Sum of squares of vector elements is 1 # apply sublinear Tf scaling
transformer = TfidfTransformer(norm='l2',sublinear_tf=True)
x_train_features = count_vect.fit_transform(x_train)
x_train_norm = transformer.fit_transform(x_train_features)
#print(x_train_features.shape)
print(x_train_norm.shape)
```

```
In [10]: # classification model; RandomForest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix,f1_score, accuracy_score
# the No. of trees in the model,
# n_jobs to run the algorithm in parallel(fit, predict, decision_path and apply are all parallelized over the trees),
# -1 to use all processors
# criterion: used to measure the quality of a split
RF_model = RandomForestClassifier(n_estimators=350, criterion='gini', n_jobs= -1)
RF_model.fit(x_train_norm, y_train)
predictions = RF_model.predict(x_test_norm)
```

Zur Auswertung des resultierenden RF-Modells anhand der Testdaten, wird Accuracy\_score verwendet, um den Anteil korrekt klassifizierter Stichproben des Testsatzes zurückzugeben. Ein Genauigkeitswert von etwa 83 % weist auf eine gute Klassifizierungsleistung des Modells hin. Die Verwirrungsmatrix ist eine gängige Metrik zur Bewertung von Klassifikatoren, wobei jede Zeile der Matrix die Instanzen in einer tatsächlichen Klasse darstellt, während jede Spalte die Instanzen in einer vorhergesagten Klasse darstellt. Eine weitere Bewertungsmetrik ist der F1-Score, der für jede Klasse im Datensatz wie folgt berechnet wird:

$$\text{F1-Score} = 2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall}),$$

Dabei sind Präzision die wahrhaft positiven Vorhersagen im Verhältnis zur Gesamtzahl der positiven Vorhersagen und Recall die wahrhaft positiven Vorhersagen im Verhältnis zur Gesamtzahl der tatsächlichen positiven Werte. Der F1-Score wird verwendet, um die Klassifizierungsgenauigkeit mehrerer Klassifikatoren zu vergleichen:

```
In [13]: # F-score is a measure of a test's accuracy. Used to compare the performance of two classifiers.
# If average= None, the scores for each class are returned.
f1score= f1_score(y_test,predictions, average = None)
print('f1_score:',f1score )

#Accuracy_score
acc_score = accuracy_score(y_test,predictions)*100
print('accuracy score:',acc_score)

# model evaluation using Confusion Matrix
CM = confusion_matrix(y_test,predictions)
print('confusion_matrix:\n', CM)

f1_score: [0.73619333 0.86130645 0.84133876]
accuracy score: 82.86801251764129
confusion_matrix:
 [[ 4539   858  1755]
 [   85 10166   816]
 [  555  1515 12305]]
```

Jetzt trainieren wir den Naive-Bayes-Klassifikator auf demselben Trainingssatz und treffen Vorhersagen anhand der Testdaten. Vergleichen wir die Vorhersageleistung beider Klassifikatoren:

```
In [9]: # applying Naive Bayes classification MultinomialNB
from sklearn.naive_bayes import MultinomialNB
NB_model = MultinomialNB()
#fitting NB classifier, we use monogram tokenizer; tokenizing each word as one token
NB_model.fit(x_train_features, y_train)

# NB classifier evaluation
from sklearn import metrics
predictions= NB_model.predict(x_test_features)
metrics.accuracy_score(predictions, y_test)
```

Out[9]: 0.706764841233318

```
1 {
2   "cells": [
3     {
4       "cell_type": "code",
5       "execution_count": null,
6       "id": "4a878b40",
7       "metadata": {},
8       "outputs": [],
9       "source": [
10         "import pandas as pd\n",
11         "# three sentiments here: negative(-1), neutral(0), and positive(+1).\n",
12         "data = pd.read_csv(\"Twitter_Data.csv\")\n",
13         "\n",
14         "# rename 'category' column into 'label'\n",
15         "data.rename(columns = {'category':'label'}, inplace = True)\n",
```

```
16     "data.head(1)['clean_text'][0]\n"
17   ]
18 },
19 {
20   "cell_type": "code",
21   "execution_count": 17,
22   "id": "304440fe",
23   "metadata": {},
24   "outputs": [
25     {
26       "data": {
27         "image/png": "iVBORw0KGgoAAAANSUhEUgAAAGwCAYAAC0H1ECAAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZlcnPb24zLjcuMSwgaHR0cHM6Ly9tYXRwbG90bGliLm9yZy/bCgiHAAAACXBWMA",
28         "text/plain": [
29           "<Figure size 640x480 with 1 Axes>"
30         ]
31       },
32       "metadata": {},
33       "output_type": "display_data"
34     }
35   ],
36   "source": [
37     "# plot label distribution\n",
38     "import seaborn as sns\n",
39     "ax=sns.countplot(x=data['label'], data = data)"
40   ],
41 },
42 {
43   "cell_type": "code",
44   "execution_count": 4,
45   "id": "85df708e",
46   "metadata": {},
47   "outputs": [
48     {
49       "name": "stderr",
50       "output_type": "stream",
51       "text": [
52         "[nltk_data] Downloading package stopwords to\n",
53         "[nltk_data]      C:\\\\Users\\\\la2022\\\\AppData\\\\Roaming\\\\nltk_data...\\n",
54         "[nltk_data]      Package stopwords is already up-to-date!\\n"
55       ]
56     }
57   ],
58   "source": [
59     "import gensim\\n",
60     "import nltk\\n",
61     "from gensim.utils import simple_preprocess\\n",
62     "nltk.download('stopwords')\\n",
63     "from nltk.corpus import stopwords\\n",
64     "#stop words of the English dictionary\\n",
65     "stop_w = stopwords.words('English')\\n",
66     "#stop_words.extend(['from', 'subject', 're', 'edu', 'use','of', 'as', 'by', 'uc'])\\n",
67     "\\n",
68     "def process(text):\\n",
69     "    \\n",
70     "    # deacc=True removes punctuations\\n",
71     "    no_punc = gensim.utils.simple_preprocess(str(text), deacc=True)\\n",
72     "    return [word for word in no_punc if word not in stop_w] # for each doc in the text, remove stop words\\n",
73     "data['clean_text'] = data['clean_text'].apply(process)\\n",
74     "\\n",
75     "\\n",
76     "\\n"
77   ],
78 },
79 {
```

```
80     "cell_type": "code",
81     "execution_count": 5,
82     "id": "2a83308e",
83     "metadata": {},
84     "outputs": [
85     {
86       "data": {
87         "text/plain": [
88           "0      [modi, promised, minimum, government, maximum,...\n",
89           "1      [talk, nonsense, continue, drama, vote, modi]\n",
90           "2      [say, vote, modi, welcome, bjp, told, rahul, m...",
91           "3      [asking, supporters, prefix, chowkidar, names,...\n",
92           "4      [answer, among, powerful, world, leader, today...",
93           "      ...",
94           "162975  [crores, paid, neerav, modi, recovered, congre...",
95           "162976  [dear, rss, terrorist, payal, gawar, modi, kil...",
96           "162977  [cover, interaction, forum, left]\n",
97           "162978  [big, project, came, india, modi, dream, proje...",
98           "162979  [ever, listen, like, gurukul, discipline, main...",
99           "Name: clean_text, Length: 162980, dtype: object"
100        ]
101      },
102      "execution_count": 5,
103      "metadata": {},
104      "output_type": "execute_result"
105    }
106  ],
107  "source": [
108    "data['clean_text']"
109  ],
110 },
111 {
112   "cell_type": "code",
113   "execution_count": 6,
114   "id": "66262f23",
115   "metadata": {},
116   "outputs": [
117   {
118     "data": {
119       "image/png": "iVBORw0KGgoAAAANSUhEUgAAAxoAAAGXCAYAAAA08Sz9AAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZlcnNpb24zLjcuMSwgaHR0cHM6Ly9tYXRwbG90bGliLm9yZy/bCgiHAAAACXBIXMA",
120       "text/plain": [
121         "<Figure size 1000x900 with 1 Axes>"
122       ]
123     },
124     "metadata": {},
125     "output_type": "display_data"
126   }
127 ],
128 "source": [
129   "# data visualization\n",
130   "from wordcloud import WordCloud\n",
131   "import matplotlib.pyplot as plt\n",
132   "import collections\n",
133   "from collections import Counter\n",
134   "# get individual words\n",
135   "words = []\n",
136   "for row in data['clean_text']:\n",
137     words.extend(row)\n",
138   "\n",
139   "\n",
140   "# Counter is a subclass for counting objects.\n",
141   "# It is a collection where elements are stored as dictionary keys and their counts are stored as dictionary values\n",
142   "word_freq = Counter(words)\n",
143   "\n",
```

```

144     "# plotting a Word Cloud \n",
145     "word_cloud = WordCloud(\n",
146     "    background_color='white',\n",
147     "    max_words=2000,\n",
148     "    stopwords=stopwords\n",
149     "    ).generate_from_frequencies(word_freq)\n",
150     "plt.figure(figsize=(10,9))\n",
151     "plt.imshow(word_cloud) # display as an image\n",
152     "plt.axis('off')\n",
153     "plt.show()\n"
154   ],
155 },
156 {
157   "cell_type": "code",
158   "execution_count": 7,
159   "id": "1cc09c46",
160   "metadata": {},
161   "outputs": [
162     {
163       "data": {
164         "image/png": "iVBORw0KGgoAAAANSUhEUgAAAxoAAAGXCAYAAA08SZ9AAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZlcNpb24zLjcuMSwgaHR0cHM6Ly9tYXRwbG90bGliIml9yZy/bCgiHAAAACXBIVXMA",
165         "text/plain": [
166           "<Figure size 1000x900 with 1 Axes>"
167         ]
168       },
169       "metadata": {},
170       "output_type": "display_data"
171     }
172   ],
173   "source": [
174     "# Most of the words in the cloud seem neutral. It doesn't give any idea about racist / sexist tweets.\n",
175     "# let's take a look at the positive tweets\n",
176     "positive_rows = [r for r in data['clean_text'][data['label']==1.0]]\n",
177     "pos_words =[]\n",
178     "for twt in positive_rows :\n",
179       pos_words.extend(twt)\n",
180     "#print(pos_words[:200])\n",
181     "\n",
182     "# positive word frequencies\n",
183     "pos_freq = Counter(pos_words)\n",
184     "\n",
185     "# positive words cloud\n",
186     "pos_cloud = WordCloud(\n",
187     "    background_color='white',\n",
188     "    max_words=2000,\n",
189     "    stopwords=stopwords\n",
190     "    ).generate_from_frequencies(pos_freq)\n",
191     "plt.figure(figsize=(10,9))\n",
192     "plt.imshow(pos_cloud) # display as an image\n",
193     "plt.axis('off')\n",
194     "plt.show()\n"
195   ],
196 },
197 {
198   "cell_type": "code",
199   "execution_count": 8,
200   "id": "01872845",
201   "metadata": {},
202   "outputs": [
203     {
204       "data": {
205         "image/png": "iVBORw0KGgoAAAANSUhEUgAAAxoAAAGXCAYAAA08SZ9AAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZlcNpb24zLjcuMSwgaHR0cHM6Ly9tYXRwbG90bGliIml9yZy/bCgiHAAAACXBIVXMA",
206         "text/plain": [
207           "<Figure size 1000x900 with 1 Axes>"
208         ]
209       }
210     }
211   ],
212 }
```

```
208     ]
209   },
210   "metadata": {},
211   "output_type": "display_data"
212 }
213 ],
214 "source": [
215   "# data visualization\n",
216   "from wordcloud import WordCloud\n",
217   "import matplotlib.pyplot as plt\n",
218   "import collections\n",
219   "from collections import Counter\n",
220   "# get individual words for negative tweets\n",
221   "neg_rows = [r for r in data['clean_text'][data['label']== -1.0]]\n",
222   "neg_words =[]\n",
223   "for twt in neg_rows : \n",
224     neg_words.extend(twt)\n",
225   "\n",
226   "# Counter is a subclass for counting objects.\n",
227   "# It is a collection where elements are stored as dictionary keys and their counts are stored as dictionary values\n",
228   "# negative word frequencies\n",
229   "neg_freq = Counter(neg_words)\n",
230   "\n",
231   "# positive words cloud\n",
232   "neg_cloud = WordCloud(\n",
233     background_color='black',\n",
234     max_words=2000,\n",
235     stopwords=stopwords\n",
236     ).generate_from_frequencies(neg_freq)\n",
237   "plt.figure(figsize=(10,9))\n",
238   "plt.imshow(neg_cloud, interpolation='bilinear') # display as an image, we could use interpolation \n",
239   "plt.axis('off')\n",
240   "plt.show()"
241 ]
242 },
243 {
244   "cell_type": "code",
245   "execution_count": null,
246   "id": "a5aaca3f",
247   "metadata": {},
248   "outputs": [],
249   "source": [
250     "#The bag-of-words model is a simplifying representation used in Natural language processing.\n",
251     "#In this model, a text is represented as the bag of its words (independent features) ,disregarding grammar but keeping multiplicity.\n",
252     "# We will use SciKit Learn's CountVectorizer function which will convert a collection of \n",
253     "# text documents into a matrix of token counts or feature table\n",
254     "# we will use TfidfTransformer as a normalization method\n",
255     "\n",
256     "# drop rows with Nan values from the dataset\n",
257     "data = data.dropna()\n",
258     "#Split data into training and testing sets \n",
259     "from sklearn.model_selection import train_test_split\n",
260     "# join clean_txt lists into strings to apply CountVectorizer, otherwise it gives an error\n",
261     "\n",
262     "x_train, x_test, y_train, y_test =  train_test_split(data['clean_text'].map(' '.join), \n",
263       data['label'], test_size = 0.2, random_state = 42) # 20% of the data for testing the model\n",
264     "\n",
265     "from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer\n",
266     "count_vect = CountVectorizer(stop_words='english')\n",
267     "# normal distribution is not assumed by random forest, but we apply transformation to transform \n",
268     "# textual data(categorical data) into numerical.\n",
269     "# alternatively we can apply one-hot encoding for the same purpose.\n",
270     "# 12: Sum of squares of vector elements is 1 # apply sublinear Tf scaling\n",
271     "transformer = TfidfTransformer(norm='l2', sublinear_tf=True)\n",
```

```

272 "x_train_features = count_vect.fit_transform(x_train)\n",
273 "x_train_norm = transformer.fit_transform(x_train_features)\n",
274 "#print(x_train_features.shape)\n",
275 "print(x_train_norm.shape)\n",
276 "\n",
277 "#Output :(130378, 84916) \n",
278 "x_test_features = count_vect.transform(x_test)\n",
279 "x_test_norm = transformer.transform(x_test_features)\n",
280 "print(x_test_features.shape)\n",
281 "print(x_test_norm.shape)\n",
282 "#Output :(32595, 84916)"
283 ]
284 },
285 {
286   "cell_type": "code",
287   "execution_count": null,
288   "id": "06c81e25",
289   "metadata": {},
290   "outputs": [],
291   "source": [
292     "# classification model; RandomForest\n",
293     "from sklearn.ensemble import RandomForestClassifier\n",
294     "from sklearn.metrics import confusion_matrix,f1_score, accuracy_score  \n",
295     "# the No. of trees in the model,\n",
296     "# n_jobs to run the algorithm in parallel(fit, predict, decision_path and apply are all parallelized over the trees),\n",
297     "# -1 to use all processors\n",
298     "# criterion: used to measure the quality of a split\n",
299     "RF_model = RandomForestClassifier(n_estimators=350, criterion='gini', n_jobs= -1)  \n",
300     "RF_model.fit(x_train_norm, y_train)\n",
301     "predictions = RF_model.predict(x_test_norm)\n"
302   ],
303 },
304 {
305   "cell_type": "code",
306   "execution_count": null,
307   "id": "65a5e049",
308   "metadata": {},
309   "outputs": [],
310   "source": [
311     "# F-score is a measure of a test's accuracy. Used to compare the performance of two classifiers.\n",
312     "# If average= None, the scores for each class are returned.\n",
313     "f1score= f1_score(y_test,predictions, average = None)\n",
314     "print('f1_score:',f1score )\n",
315     "\n",
316     "#Accuracy_score\n",
317     "acc_score = accuracy_score(y_test,predictions)*100\n",
318     "print('accuracy score:',acc_score)\n",
319     "\n",
320     "# model evaluation using Confusion Matrix \n",
321     "CM = confusion_matrix(y_test,predictions)\n",
322     "print('confusion_matrix:\n', CM) \n",
323     "\n"
324   ],
325 },
326 {
327   "cell_type": "code",
328   "execution_count": null,
329   "id": "c0ba84c4",
330   "metadata": {},
331   "outputs": [],
332   "source": [
333     "# applying Naive Bayes classification MultinomialNB\n",
334     "from sklearn.naive_bayes import MultinomialNB\n",
335     "NB_model = MultinomialNB()\n",

```

```

336 "#fitting NB classifier, we use monogram tokenizer; tokenizing each word as one token\n",
337 "NB_model.fit(x_train_features, y_train)\n",
338 "\n",
339 "# NB classifier evaluation\n",
340 "from sklearn import metrics\n",
341 "predictions= NB_model.predict(x_test_features)\n",
342 "metrics.accuracy_score(predictions, y_test)\n"
343 ]
344 },
345 {
346 "cell_type": "code",
347 "execution_count": null,
348 "id": "ee375da9",
349 "metadata": {},
350 "outputs": [],
351 "source": []
352 }
353 ],
354 "metadata": {
355 "kernelspec": {
356 "display_name": "Python 3 (ipykernel)",
357 "language": "python",
358 "name": "python3"
359 },
360 "language_info": {
361 "codemirror_mode": {
362 "name": "ipython",
363 "version": 3
364 },
365 "file_extension": ".py",
366 "mimetype": "text/x-python",
367 "name": "python",
368 "nbconvert_exporter": "python",
369 "pygments_lexer": "ipython3",
370 "version": "3.9.13"
371 }
372 },
373 "nbformat": 4,
374 "nbformat_minor": 5
375 }

```

## Semantische Suche

Die meisten Suchmaschinen wie Google, Bing usw. sind lexikalisch, was bedeutet, dass die Suchmaschine nach genauen Übereinstimmungen der Suchbegriffe im Inhalt sucht, ohne den Kontext der Suchanfragen zu verstehen. Die semantische Suche hingegen zielt darauf ab, die Suchgenauigkeit durch das Verständnis des Inhalts der Suchanfrage zu verbessern. Im Gegensatz zu herkömmlichen Suchmaschinen, die Dokumente nur auf der Grundlage lexikalischer Übereinstimmungen finden, kann die semantische Suche auch Synonyme finden, indem sie den Inhalt zusätzlich zu Schlüsselwörtern nach seiner Bedeutung durchsucht und so die Chancen maximiert, dass der Benutzer die gesuchten Informationen findet.

Die Idee hinter der semantischen Suche besteht darin, die Worteinbettung auf alle Einträge im Korpus anzuwenden, unabhängig davon, ob es sich um Sätze, Absätze oder Dokumente handelt. Die Worteinbettung ist eine Wortdarstellung, die es maschinellen Lernalgorithmen ermöglicht, den Kontext zu verstehen, indem sie Wörter in Vektoren reeller Zahlen abbilden. Diese Vektoren helfen bei der Erfassung semantischer und syntaktischer Merkmale von Wörtern im Korpus. Dies geschieht beispielsweise durch die Kategorisierung ähnlicher Wörter, etwa durch die Gruppierung von Apfel, Mango und Banane als Früchte, wohingegen Hunde und Katzen als Tiere. Oder sogar die Darstellung von Wörtern anhand ihres Kontexts, was sich auf die semantischen Beziehungen zwischen Wörtern bezieht.

Eine der beliebtesten Einbettungsmethoden ist Word2vec, bei der jedes Wort in einen Merkmalsvektor codiert wird. Word2vec verwendet ein neuronales Netzwerk mit einer einzelnen verborgenen Schicht, um die Gewichtsmatrix der verborgenen Schicht zu lernen, auf deren Grundlage der Einbettungsvektor für jedes Wort berechnet wird. Die Größe der verborgenen Schicht wird auf

die Dimensionalität der resultierenden Wortvektoren eingestellt. Das Ziel dieses neuronalen Netzwerks besteht darin, diese Matrix basierend auf der Vorhersage von umgebenden Wörtern zu aktualisieren, zu lernen und zur Berechnung von Merkmalsvektoren zu verwenden. Dadurch implizieren wir Informationen über die semantische Beziehung in der Gewichtsmatrix.

Nehmen wir an, wir haben ein Korpuswörterbuch mit 5 Wörtern: ['Cat', 'climbe', 'the', 'tree', 'yesterday'] und eine versteckte Word2vec-Schicht aus 3 Neuronen mit der folgenden erlernten Gewichtsmatrix:

$$\begin{bmatrix} 1 & 13 & 5 \\ 2 & 12 & 43 \\ 0 & 10 & 30 \\ 25 & 18 & 70 \\ 5 & 23 & 44 \end{bmatrix}$$

dann sollten wir den folgenden Einbettungsvektor für das Wort „Tree“ erhalten:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 13 & 5 \\ 2 & 12 & 43 \\ 0 & 10 & 30 \\ 25 & 18 & 70 \\ 5 & 23 & 44 \end{bmatrix} = [25 \ 18 \ 70]$$

Word2vec Algorithmus basiert auf dem Konzept der Verteilungssemantik, das davon ausgeht, dass Wörter mit ähnlicher Bedeutung tendenziell in ähnlichen Kontexten auftauchen. Die Vektoren von Wörtern, die aufgrund ihres Kontexts als ähnlich beurteilt werden, werden durch Anpassen der Zahlen im Vektor näher zusammengebracht. Die Hauptidee der semantischen Suche besteht darin, ein bestimmtes Wort mithilfe von Einbettungsvektoren in seine wahrscheinlichen Kontexte zu übersetzen und dann im Korpus nach ähnlichen oder nahen Kontexten zu suchen. Diese Nähe (oder Ähnlichkeit) wird mit verschiedenen Methoden wie Kosinusähnlichkeit oder euklidischer Distanz berechnet. Die semantische Suche funktioniert durch Suchen, indem man sich die Vektordarstellungen der Abfragen anschaut und Einträge mit ähnlichen Vektordarstellungen im Korpus findet.

Wir haben zwei Haupttypen der semantischen Suche:

1. Symmetrische semantische Suche: Abfragen und Einträge im Korpus haben die gleiche Länge, z. B. die Suche nach ähnlichen Fragen.
2. Asymmetrische semantische Suche: Wir suchen nach Ähnlichkeiten zwischen kurzen Suchanfragen und langen Absätzen.

## Semantische Suche Übung

Die semantische Worteinbettung wird verwendet, um Wörter und Phrasen im Korpus in numerische Vektoren umzuwandeln, die die semantischen Beziehungen zwischen diesen Wörtern erfassen.

Im folgenden Beispiel verwenden wir einen Satztransformer-Decoder mit einem fein abgestimmten MiniLM-L6-v2-Modell, um die erforderlichen Einbettungen aus dem Text zu erstellen. Für jede

Abfrage erstellt der Decoder die entsprechende Einbettung und findet die fünf relevantesten Phrasen, indem er die Kosinusähnlichkeit zwischen der Abfrage und allen Phrasen im Korpus berechnet.

```
In [1]: # compute the cosine-similarity between the query and all entries in the corpus.  
# embedding techniques are used to represent words/text mathematically with numeric vectors using encoders/transformer  
# such as one-hot encoding..  
# SentenceTransformer('all-MiniLM-L6-v2') defines which embedding transformer model we like to use.  
# In this example, we load all-MiniLM-L6-v2, which is a MiniLM model fine tuned on a large dataset of over 1 billion training  
# pairs.  
# !pip install -U sentence-transformers  
  
from sentence_transformers import SentenceTransformer, util  
import torch  
  
encoder = SentenceTransformer('all-MiniLM-L6-v2')  
  
# Corpus with example sentences  
corpus = ['A man is eating food.',  
          'A man is eating a piece of bread.',  
          'The girl is carrying a baby.',  
          'A man is riding a horse.',  
          'A woman is playing violin.',  
          'Two men pushed carts through the woods.',  
          'A man is riding a white horse on an enclosed ground.',  
          'A monkey is playing drums.',  
          'A cheetah is running behind its prey.'  
]  
embeddings = encoder.encode(corpus, convert_to_tensor=True)
```

```
# Query sentences:  
queries = ['A man is eating pasta.', 'Someone in a gorilla costume is playing a set of drums.',  
          'A cheetah chases prey on across a field.'][  
  
# Find the closest 5 sentences of the corpus for each query sentence based on cosine similarity  
# top_k = min(5, len(corpus)) # for cases where the corpus is shorter than 5  
for q in queries:  
    query_embedding = encoder.encode(q, convert_to_tensor=True)  
  
    # We use cosine-similarity and torch.topk to find the highest 5 scores  
    cos_scores = util.cos_sim(query_embedding, embeddings)[0]  
    top_results = torch.topk(cos_scores, k=5)  
  
    print("\n\n=====\\n\\n")  
    print("Query:", q)  
    print("\\nTop 5 most similar sentences in corpus:")  
  
    for score, idx in zip(top_results[0], top_results[1]):  
        print(corpus[idx], "(Score: {:.4f})".format(score))
```

Alternativ können wir die Funktion `util.semantic_search` verwenden, um eine Kosinus-Ähnlichkeitssuche zwischen einer Liste von Abfrageeinbettungen und einer Liste von Korpus einbettungen durchzuführen. Diese Funktion arbeitet effizienter durch die parallele Verarbeitung von Abfragen, da sie bis zu 100 Abfragen für Korpora mit bis zu etwa 1 Million Einträgen parallel verarbeiten kann. Wir können `query_chunk_size` und `corpus_chunk_size` erhöhen, was bei großen Korpora zu einer höheren Geschwindigkeit führt, aber auch den Speicherbedarf erhöht. Der Algorithmus berechnet Kosinusähnlichkeiten einer Abfrage mit allen Einträgen und gibt ein Wörterbuch mit den Schlüsseln „corpus\_id“ und „score“ zurück, sortiert nach abnehmenden Kosinus-Ähnlichkeitswerten.

```
In [2]: # we can use util.semantic_search instead
# util.semantic_search performs a cosine similarity search between a list of query embeddings and a list of corpus embeddings.
# It can be used for Information Retrieval / Semantic Search for corpora up to about 1 Million entries.
# By default, up to 100 queries are processed in parallel.
# Further, the corpus is chunked into set of up to 500k entries.
# You can increase query_chunk_size and corpus_chunk_size, which leads to increased speed for large corpora,
# but also increases the memory requirement.
# returns a list with one entry for each query.
# Each entry is a dictionaries with the keys 'corpus_id' and 'score',
# sorted by decreasing cosine similarity scores.
query_embeddings = []
for q in queries:
    query_embedding = encoder.encode(q, convert_to_tensor=True)
    query_embeddings.append(query_embedding)

sim_scores = util.semantic_search(query_embeddings, embeddings, top_k = 5)
print(sim_scores)
```

```
[[{'corpus_id': 0, 'score': 0.7035486698150635}, {'corpus_id': 1, 'score': 0.5271987318992615}, {'corpus_id': 3, 'score': 0.18889547884464264}, {'corpus_id': 6, 'score': 0.10469925403594971}, {'corpus_id': 8, 'score': 0.0980304405093193}], [{"corpus_id": 7, "score": 0.6432532072067261}, {"corpus_id": 4, "score": 0.25641554594039917}, {"corpus_id": 3, "score": 0.1388726085424423}, {"corpus_id": 2, {"corpus_id": 6, "score": 0.11909151077270508}, {"corpus_id": 8, "score": 0.10798679292201996}], [{"corpus_id": 8, "score": 0.8253214359283447}, {"corpus_id": 0, "score": 0.1398952305316925}, {"corpus_id": 7, "score": 0.12919366359710693}, {"corpus_id": 6, "score": 0.10974173247814178}, {"corpus_id": 3, "score": 0.06497804820537567}]]
```

Wir werden nun die semantische Suche auf im US-Repräsentantenhaus vorgeschlagene Gesetzentwürfe anwenden. Den Datensatz können wir von [Kaggle](#) herunterladen. Wir werden die semantische Suche auf die Spalte ('summary') anwenden. Werfen wir einen Blick auf den Text im Zusammenfassungsfeld:

```
In [27]: # semantec_search on congress bills (draft Laws) dataset
import pandas as pd
WH_legis= pd.read_csv('house_legislation_116.csv')
# 'summary' column contains a summary of the bill
print(WH_legis['summary'][0])
WH_legis = WH_legis[WH_legis['summary'].notna()] # drop Nan rows
```

This bill addresses voter access, election integrity, election security, political spending, and ethics for the three branches of government. Specifically, the bill expands voter registration and voting access, makes Election Day a federal holiday, and limits removing voters from voter rolls. The bill provides for states to establish independent, nonpartisan redistricting commissions. The bill also sets forth provisions related to election security, including sharing intelligence information with state election officials, protecting the security of the voter rolls, supporting states in securing their election systems, developing a national strategy to protect the security and integrity of U.S. democratic institutions, establishing in the legislative branch the National Commission to Protect United States Democratic Institutions, and other provisions to improve the cybersecurity of election systems. This bill addresses campaign spending, including by expanding the ban on foreign nationals contributing to or spending on elections; expanding disclosure rules pertaining to organizations spending money during elections, campaign advertisements, and online platforms; and revising disclaimer requirements for political advertising. This bill establishes an alternative campaign funding system for certain federal offices. The system involves federal matching of small contributions for qualified candidates. This bill sets forth provisions related to ethics in all three branches of government. Specifically, the bill requires a code of ethics for federal judges and justices, prohibits Members of the House from serving on the board of a for-profit entity, expands enforcement of regulations governing foreign agents, and establishes additional conflict-of-interest and ethics provisions for federal employees and the White House. The bill also requires candidates for President and Vice President to submit 10 years of tax returns. This bill addresses voter access, election integrity, election security, political spending, and ethics for the three branches of government. Specifically, the bill expands voter registration and voting access, makes Election Day a federal holiday, and limits removing voters from voter rolls. The bill provides for states to establish independent, nonpartisan redistricting commissions. The bill also sets forth provisions related to election security, including sharing intelligence information with state election officials, protecting the security of the voter rolls, supporting states in securing their election systems, developing a national strategy to protect the security and integrity of U.S. democratic institutions, establishing in the legislative branch the National Commission to Protect United States Democratic Institutions, and other provisions to improve the cybersecurity of election systems. This bill addresses campaign spending, including by expanding

```
In [ ]: WH_legis.reset_index(inplace =True)
summary= WH_legis['summary']
queries =['medication pricing', 'foreigner residence in united states', 'climate change']
query_embeddings =[]
for q in queries:
    query_embedding = encoder.encode(q, convert_to_tensor=True)
    query_embeddings.append(query_embedding)
corpus_embeddings =encoder.encode(summary, convert_to_tensor=True)
sim_cos_scores = util.semantic_search(query_embeddings, corpus_embeddings, top_k = 10)# it takes some time

print(sim_cos_scores)
```

Als alternative Methode kann Approximate Nearest Neighbor (ANN) eine effizientere semantische Suche bei großen Korpora durchführen, da es eine Art Suchcluster oder -baum bildet, um die Suche zu beschleunigen, indem die Daten in kleinere Bruchteile ähnlicher Einbettungen aufgeteilt werden. ANN verwendet Einbettungsindizes, um einen Wald aus Indexbäumen basierend auf der Ähnlichkeit zwischen den Einbettungen aufzubauen. Der Indexwald kann anschließend effizient durchsucht werden und die Einbettungen mit der höchsten Ähnlichkeit (die nächsten Nachbarn) mit der Abfrageeinbettung können innerhalb von Millisekunden abgerufen werden.

AnnoyIndex aus dem Annoy-Python-Modul wird hier verwendet, um die Indizes zu erstellen und einen Indexwald mit 200 Bäumen aufzubauen.

```
In [28]: # Approximate Nearest Neighbor (ANN) can be helpful, since the data is partitioned into smaller fractions of similar embeddings.  
# that is KNN is first applied to group the data based on similarity.  
# the index forest can be searched efficiently and the embeddings with the highest similarity (the nearest neighbors) can be retr  
# even if you have millions of vectors. The main disadvantage is that some vectors with high similarity may be missed; that's  
# why this is called Approximate Nearest Neighbor  
#For all ANN methods, there are usually one or more parameters to tune that determine the recall-speed trade-off.  
#If you want the highest speed, you have a high chance of missing hits. If you want high recall, the search speed decreases.  
# AnnoyIndex() takes an argument representing the embedding size ;the number of features in an indexed vector;get the min embedding  
  
# install annoy  
!pip install annoy  
from annoy import AnnoyIndex  
# [min(len(emb) for emb in corpus_embeddings)]# result in 384  
embedding_size = 384  
n_tree= 200 # No. of clusters  
annoy_index = AnnoyIndex(embedding_size, 'angular')  
for i in range(len(corpus_embeddings)):  
    annoy_index.add_item(i, corpus_embeddings[i])  
  
annoy_index.build(n_tree) #apply ANN to build a forest of index trees (200 trees)  
#annoy_index.save(annoy_index_path)# to save the ANN model to a file
```

Requirement already satisfied: annoy in c:\users\la2022\anaconda3\lib\site-packages (1.17.1)

[notice] A new release of pip available: 22.3.1 -> 23.0  
[notice] To update, run: python.exe -m pip install --upgrade pip

Wir werden den vorherigen Indexwald verwenden, um nach den n Elementen aus unserem Korpus zu suchen, die basierend auf dem Kosinus-Distanzwert am ehesten mit dem Satz von Abfragen übereinstimmen.

In [55]:

```

top_k_hits = 5 # get the top 10 nearest trees
for q in queries:
    query_embedding = encoder.encode(q, convert_to_tensor=True)
    #Search the n closest items.
    #include_distances: The flag indicating whether to returns all corresponding distances.
    corpus_ids, scores = annoy_index.get_nns_by_vector(query_embedding, top_k_hits, include_distances=True)
    hits = []
    for i, score in zip(corpus_ids, scores):
        ## the scores returned by Annoy_index is euclidean distance,
        # we need to calculate the cosine distance(in case comparison with other cosine similarity methods)
        # the cosine distance is equals to 1 - e^2/2, where e is the euclidean distance value
        hits.append({'corpus_id': i, 'score': 1-((score**2) / 2)})
    print("\n Input question:", q)
    for hit in hits[0:top_k_hits]:
        print("\t{:.3f}\t{}".format(hit['score'], summary[hit['corpus_id']])))

```

PDP sponsor must report, both to drug manufacturers and to the CMS, specified information related to the determination and payment of such rebates. This bill makes a series of changes relating to the prices of prescription drugs under the Medicare prescription drug benefit and Medicare Advantage (MA) prescription drug plans (PDPs). Under current law, the Centers for Medicare & Medicaid Services (CMS) may neither negotiate the prices of covered drugs nor establish a formulary. The bill repeals these restrictions and instead specifically requires the CMS to (1) negotiate the prices of covered drugs; and (2) either establish a formulary for covered drugs, or require changes to PDP formularies that take into account CMS negotiations. If the CMS is unable to negotiate an appropriate price for a drug in accordance with certain criteria, the price must be the lowest of three specified options (e.g., the average price in other countries). The CMS must identify drugs that are subject to negotiation, with priority given to certain categories of drugs based on usage and cost. Additionally, drug manufacturers must issue rebates to the CMS for drugs dispensed to eligible low-income individuals. Subject to civil monetary penalties, a Medicare or MA PDP sponsor must report, both to drug manufacturers and to the CMS, specified information related to the determination and payment of such rebates.

Input question: foreigner residence in united states

0.480 This bill establishes additional security screening requirements for aliens seeking refugee status if they are nationals of Iraq or Syria. are stateless individuals whose last habitual residence was in one of those countries. or have

```

1 {
2   "cells": [
3     {
4       "cell_type": "code",
5       "execution_count": null,
6       "id": "de2b5270",
7       "metadata": {},
8       "outputs": [],
9       "source": [
10         "# compute the cosine-similarity between the query and all entries in the corpus.\n",
11         "# embedding techniques are used to represent words/text mathematically with numeric vectors using encoders/transformer\n",
12         "# such as one-hot encoding..\n",
13         "# SentenceTransformer('all-MiniLM-L6-v2') defines which embedding transformer model we like to use.\n",
14         "# In this example, we load all-MiniLM-L6-v2, which is a MiniLM model fine tuned on a large dataset of over 1 billion training\n",
15         "# pairs.\n",
16         "# !pip install -U sentence-transformers\n",
17         "\n"

```

```

18     "from sentence_transformers import SentenceTransformer, util\n",
19     "import torch\n",
20     "\n",
21     "encoder = SentenceTransformer('all-MiniLM-L6-v2')\n",
22     "\n",
23     "# Corpus with example sentences\n",
24     "corpus = ['A man is eating food.',\n",
25     "          'A man is eating a piece of bread.',\n",
26     "          'The girl is carrying a baby.',\n",
27     "          'A man is riding a horse.',\n",
28     "          'A woman is playing violin.',\n",
29     "          'Two men pushed carts through the woods.',\n",
30     "          'A man is riding a white horse on an enclosed ground.',\n",
31     "          'A monkey is playing drums.',\n",
32     "          'A cheetah is running behind its prey.'\n",
33     "]\n",
34     "embeddings = encoder.encode(corpus, convert_to_tensor=True)\n",
35     "\n",
36     "# Query sentences:\n",
37     "queries = ['A man is eating pasta.', 'Someone in a gorilla costume is playing a set of drums.',\n",
38     "           'A cheetah chases prey on across a field.']\n",
39     "\n",
40     "# Find the closest 5 sentences of the corpus for each query sentence based on cosine similarity\n",
41     "# top_k = min(5, len(corpus)) # for cases where the corpus is shorter than 5\n",
42     "for q in queries:\n",
43     "    query_embedding = encoder.encode(q, convert_to_tensor=True)\n",
44     "\n",
45     "    # We use cosine-similarity and torch.topk to find the highest 5 scores\n",
46     "    cos_scores = util.cos_sim(query_embedding, embeddings)[0]\n",
47     "    top_results = torch.topk(cos_scores, k=5)\n",
48     "\n",
49     "    print(\"Query:\", q)\n",
50     "    print(\"\\nTop 5 most similar sentences in corpus:\")\n",
51     "    print(\"\\n\")\n",
52     "\n",
53     "    for score, idx in zip(top_results[0], top_results[1]):\n",
54     "        print(corpus[idx], \"(Score: {:.4f})\".format(score))\n",
55   ]
56 },
57 {
58   "cell_type": "code",
59   "execution_count": null,
60   "id": "86f4588b",
61   "metadata": {},
62   "outputs": [],
63   "source": [
64     "# we can use util.semantic_search instead\n",
65     "# util.semantic_search performs a cosine similarity search between a list of query embeddings and a list of corpus embeddings.\n",
66     "# It can be used for Information Retrieval / Semantic Search for corpora up to about 1 Million entries.\n",
67     "# By default, up to 100 queries are processed in parallel.\n",
68     "# Further, the corpus is chunked into sets of up to 500k entries. \n",
69     "# You can increase query_chunk_size and corpus_chunk_size, which leads to increased speed for large corpora,\n",
70     "# but also increases the memory requirement.\n",
71     "# returns a list with one entry for each query.\n",
72     "# Each entry is a dictionaries with the keys ?corpus_id? and ?score?, \n",
73     "# sorted by decreasing cosine similarity scores.\n",
74     "query_embeddings =[]\n",
75     "for q in queries:\n",
76       query_embedding = encoder.encode(q, convert_to_tensor=True)\n",
77       query_embeddings.append(query_embedding)\n",
78     "\n",
79     "sim_scores = util.semantic_search(query_embeddings, embeddings, top_k = 5)\n",
80     "print(sim_scores)\n",
81   ]

```

```

82 },
83 {
84     "cell_type": "code",
85     "execution_count": null,
86     "id": "47785837",
87     "metadata": {},
88     "outputs": [],
89     "source": [
90         "# Text summarization on congress bills (draft laws) dataset\n",
91         "import pandas as pd\n",
92         "WH_legis= pd.read_csv('house_legislation_116.csv')\n",
93         "# 'summary' column contains a summary of the bill\n",
94         "print(WH_legis['summary'][0])\n",
95         "WH_legis = WH_legis[WH_legis['summary'].notna()] # drop Nan rows"
96     ],
97 },
98 {
99     "cell_type": "code",
100    "execution_count": null,
101    "id": "95bda031",
102    "metadata": {},
103    "outputs": [],
104    "source": [
105        "WH_legis.reset_index(inplace =True)\n",
106        "summary= WH_legis['summary']\n",
107        "queries =['medication pricing', 'foreigner residence in united states', 'climate change']\n",
108        "query_embeddings =[]\n",
109        "for q in queries:\n",
110            query_embedding = encoder.encode(q, convert_to_tensor=True)\n",
111            query_embeddings.append(query_embedding)\n",
112        corpus_embeddings =encoder.encode(summary, convert_to_tensor=True) \n",
113        sim_cos_scores = util.semantic_search(query_embeddings, corpus_embeddings, top_k = 10)# it takes some time\n",
114        "\n",
115        "print(sim_cos_scores)"
116    ],
117 },
118 {
119     "cell_type": "code",
120     "execution_count": null,
121     "id": "bcfcb6e5",
122     "metadata": {},
123     "outputs": [],
124     "source": [
125         "# Approximate Nearest Neighbor (ANN) can be helpful, since the data is partitioned into smaller fractions of similar embeddings.\n",
126         "# that is KNN is first applied to group the data based on similarity.\n",
127         "# the index forest can be searched efficiently and the embeddings with the highest similarity (the nearest neighbors) can be retrieved within milliseconds,\n",
128         "# even if you have millions of vectors.The main disadvantage is that some vectors with high similarity may be missed; that's\n",
129         "# why this is called Approximate Nearest Neighbor\n",
130         "#For all ANN methods, there are usually one or more parameters to tune that determine the recall-speed trade-off.\n",
131         "#If you want the highest speed, you have a high chance of missing hits. If you want high recall, the search speed decreases.\n",
132         "# AnnoyIndex() takes an argument representing the embedding size ;the number of features in an indexed vector;get the min embedding length\n",
133         "\n",
134         "# install annoy\n",
135         "!pip install annoy\n",
136         "from annoy import AnnoyIndex\n",
137         "# [min(len(emb) for emb in corpus_embeddings)]# result in 384\n",
138         "embedding_size = 384\n",
139         "n_tree= 200 # No. of clusters\n",
140         "annoy_index = AnnoyIndex(embedding_size, 'angular')\n",
141         "for i in range(len(corpus_embeddings)):\n",
142             annoy_index.add_item(i, corpus_embeddings[i])\n",
143         "\n",
144         "annoy_index.build(n_tree) #apply ANN to build a forest of index trees (200 trees)\n",
145         "#annoy_index.save(annoy_index_path)# to save the ANN model to a file "

```

```
146 ]
147 },
148 {
149     "cell_type": "code",
150     "execution_count": null,
151     "id": "332aa604",
152     "metadata": {},
153     "outputs": [],
154     "source": [
155         "top_k_hits =5 \n",
156         "for q in queries:\n",
157             "    query_embedding = encoder.encode(q, convert_to_tensor=True)\n",
158             "#Search the 5 closest items.\n",
159             "#include_distances: The flag indicating whether to returns all corresponding distances.\n",
160             "corpus_ids, scores = annoy_index.get_nns_by_vector(query_embedding, top_k_hits, include_distances=True)\n",
161             hits = []\n",
162             for i, score in zip(corpus_ids, scores):\n",
163                 ## the scores returned by Annoy_index is euclidean distance,\n",
164                 # we need to calculate the cosine distance(in case comparison with other cosine similarity methods) \n",
165                 # the cosine distance is equals to 1 - e^2/2, where e is the euclidean distance value\n",
166                 hits.append({'corpus_id': i, 'score': 1-((score**2) / 2)})\n",
167             print("\n Input question:\\", q)\n",
168             for hit in hits[0:top_k_hits]:\n",
169                 print("\t{:.3f}\t".format(hit['score']), summary[hit['corpus_id']])))"
170     ]
171 }
172 ],
173 "metadata": {
174     "kernelspec": {
175         "display_name": "Python 3 (ipykernel)",
176         "language": "python",
177         "name": "python3"
178     },
179     "language_info": {
180         "codemirror_mode": {
181             "name": "ipython",
182             "version": 3
183         },
184         "file_extension": ".py",
185         "mimetype": "text/x-python",
186         "name": "python",
187         "nbconvert_exporter": "python",
188         "pygments_lexer": "ipython3",
189         "version": "3.9.13"
190     }
191 },
192 "nbformat": 4,
193 "nbformat_minor": 5
194 }
```

## Textzusammenfassung

Bei der Textzusammenfassung im NLP handelt es sich um den Prozess, große Mengen von Korpora in kurzen Absätzen oder Überschriften zusammenzufassen, ohne dass wichtige Informationen verloren gehen oder die Bedeutung des Textes verändert wird. Die Sprache der Zusammenfassung sollte prägnant und unkompliziert sein, damit sie dem Leser die Bedeutung vermittelt. Da die Gesamtmenge der weltweit erstellten, aufgezeichneten, kopierten und verbrauchten Daten in den nächsten Jahren extrem ansteigt und Hunderte von Zettabyte übersteigt, besteht ein Bedarf an zusammengefassten Daten. Zusammenfassung bedeutet, dass Textdaten zu einfacheren, prägnanteren Umrissen minimiert werden, die wesentliche Details enthalten, die leichter analysiert werden können. Es besteht ein hoher Bedarf an Algorithmen für maschinelles Lernen, die lange Texte schnell zusammenfassen und genaue Erkenntnisse liefern können.

Hier sind einige Gründe, warum wir eine Textzusammenfassung benötigen:

1. Textzusammenfassung verkürzt die Lesezeit.
2. Die automatische Zusammenfassung kann die Suche nach notwendigen Informationen in riesigen Datenmengen beschleunigen.
3. Die Textzusammenfassungstechnik trägt zur Einsparung von Speicherplatz bei, was den Unternehmensanbietern weitere Vorteile bringt.
4. Die menschliche Zusammenfassung könnte voreingenommen und ungenau sein, was sich erheblich auf die Qualität der Informationen auswirken kann.

Die Zusammenfassung von Texten ist für verschiedene NLP-Aufgaben von Vorteil, darunter die Klassifizierung von Texten, Zusammenfassungen juristischer Texte, Zusammenfassungen von Nachrichten und die Generierung von Schlagzeilen.

Es gibt zwei Arten der Textzusammenfassung:

1. Extraktive Textzusammenfassung: Dabei werden Schlüsselwörter aus einem Text extrahiert und isoliert, wodurch der Text zu einer zusammengefassten Version gekürzt wird. Im Vergleich zu abstrakten Methoden wird die extractive Textzusammenfassung einfacher angesehen. Eine der extractiven Zusammenfassungsmethoden ist der frequenzgesteuerte Ansatz, wobei die Häufigkeit von Wörtern als Indikator für die Wichtigkeit verwendet wird. Der erste Schritt mit dieser Methode ist jedem Wort im Text eine Gewichtung zuzuweisen, die die Häufigkeit dieses Wortes darstellt. Dementsprechend können wir dem gesamten Satz einen Wert zuweisen, indem wir einfach die Gewichte der einzelnen Wörter darin aufsummieren. Bei der extractiven Methode werden die Sätze nach ihrer Gewichtung geordnet, wobei die Sätze mit dem höchsten Rang die Informationen in allen zugehörigen Sätzen zusammenfassen können. PageRank ist ein weiterer Score, mit dem sich Inhalte im Web bewerten und ihnen so ein Wichtigkeitsgewicht zuweisen lassen. Diese Methode ist hauptsächlich von Google verwendet wird, um Webseiten in ihren Suchmaschinenergebnissen einzustufen.
2. Abstrakte Textzusammenfassung: Dabei werden die wichtigsten Informationen aus dem Text extrahiert, damit er mit einer neuen Reihe von Sätzen zusammengefasst wird. Diese Art der Zusammenfassung funktioniert ähnlich wie beim Menschen. Bei dieser Technik geht es darum, wichtige Teile zu identifizieren, den Kontext zu verstehen und eine zusammenhängende Zusammenfassung zu erstellen. Daher funktioniert es besser mit Deep-Learning-Modellen wie LSTM und Sequenz-zu-Sequenz-Methoden.

Python stellt eine Reihe von Bibliotheken zur Verfügung, die die Textzusammenfassung unterstützen, wie NLTK, Gensim und spaCy.

## Textzusammenfassung Übung

Wir werden noch mal die im US-Repräsentantenhaus vorgeschlagene Gesetzentwürfe. Den Datensatz können wir von [Kaggle](#) herunterladen. Wir werden die Textzusammenfassung auf die Spalte ('summary') anwenden.

In [27]: # Text summarization on congress bills (draft laws) dataset

```
import pandas as pd
WH_legis= pd.read_csv('house_legislation_116.csv')
# 'summary' column contains a summary of the bill
print(WH_legis['summary'][0])
WH_legis = WH_legis[WH_legis['summary'].notna()] # drop Nan rows
```

This bill addresses voter access, election integrity, election security, political spending, and ethics for the three branches of government. Specifically, the bill expands voter registration and voting access, makes Election Day a federal holiday, and limits removing voters from voter rolls. The bill provides for states to establish independent, nonpartisan redistricting commissions. The bill also sets forth provisions related to election security, including sharing intelligence information with state election officials, protecting the security of the voter rolls, supporting states in securing their election systems, developing a national strategy to protect the security and integrity of U.S. democratic institutions, establishing in the legislative branch the National Commission to Protect United States Democratic Institutions, and other provisions to improve the cybersecurity of election systems. This bill addresses campaign spending, including by expanding the ban on foreign nationals contributing to or spending on elections; expanding disclosure rules pertaining to organizations spending money during elections, campaign advertisements, and online platforms; and revising disclaimer requirements for political advertising. This bill establishes an alternative campaign funding system for certain federal offices. The system involves federal matching of small contributions for qualified candidates. This bill sets forth provisions related to ethics in all three branches of government. Specifically, the bill requires a code of ethics for federal judges and justices, prohibits Members of the House from serving on the board of

Für die Textzusammenfassung sollten wir zunächst den Text bereinigen, indem wir Satzzeichen und Stopwörter entfernen.

Der Ansatz basiert darauf, jedem Wort im Korpus eine Gewichtung zuzuweisen. Die Gewichtung stellt die Häufigkeit dieses Wortes im Text dar. Anschließend weisen wir dem gesamten Satz einen Wert zu, indem wir einfach die Gewichte jedes darin enthaltenen Wortes summieren. Bei der extraktiven Methode werden die Sätze nach ihrem Gewicht geordnet, wobei die höchstrangigen Sätze die Informationen in allen zusammenhängenden Sätzen zusammenfassen können.

```
In [30]: # text summarization
import nltk
import spacy
from nltk.tokenize import sent_tokenize,word_tokenize
from nltk.corpus import stopwords
from string import punctuation
nltk.download('punkt')
nltk.download('stopwords')
nlp = spacy.load('en_core_web_sm')
data = summary[1]
doc= nlp(data)
word_tokens=[token.text for token in doc]
sentence_tokens= [sent for sent in doc.sents]
stop_w = stopwords.words('English')
from string import punctuation
word_frequencies= {} #dictionary data type
for word in word_tokens:
    if word.lower() not in stop_w:
        if word.lower() not in punctuation:
            if word not in word_frequencies.keys():
                word_frequencies[word] = 1
            else:
                word_frequencies[word] += 1
max_frequency=max(word_frequencies.values())
for word in word_frequencies.keys():
    word_frequencies[word]=word_frequencies[word]/max_frequency
```

```
[nltk_data] Downloading package punkt to
[nltk_data]      C:\Users\la2022\AppData\Roaming\nltk_data...
[nltk_data]      Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]      C:\Users\la2022\AppData\Roaming\nltk_data...
```

Jetzt berechnen wir die Satzgewichte für den ersten Korpus in unserem Datensatz, um die Zusammenfassung dieses Textes zu erstellen. Die 20 % der am besten bewerteten Sätze werden verwendet, um eine Zusammenfassung dieses Korpus zu generieren.

```
In [39]: # calculation of sentence scores:
from heapq import nlargest
sentence_scores = {}
for sent in sentence_tokens:
    for word in sent:
        if word.text.lower() in word_frequencies.keys():# extract word frequency/weight
            if sent not in sentence_scores.keys():
                sentence_scores[sent]=word_frequencies[word.text.lower()]
            else:
                sentence_scores[sent]+=word_frequencies[word.text.lower()]
# determination of the number of sentences in the summary
# 20% of the sentences to be included
sent_no = int(len(sentence_tokens)* 0.2)
# get the top 20% ranked sentences
top_sent = nlargest(sent_no, sentence_scores,key=sentence_scores.get)
summary_words=[word.text for word in top_sent]
summary=''.join(summary_words)
print(summary)
```

Specifically, the CMS must negotiate maximum prices for (1) insulin products; and (2) at least 25 single source, brand name drugs that do not have generic competition and that are among the 125 drugs that account for the greatest national spending or the 125 drugs that account for the greatest spending under the Medicare prescription drug benefit and Medicare Advantage (MA). Specifically, the CMS must negotiate maximum prices for (1) insulin products; and (2) at least 25 single source, brand name drugs that do not have generic competition and that are among the 125 drugs that account for the greatest national spending or the 125 drugs that account for the greatest spending under the Medicare prescription drug benefit and Medicare Advantage (MA). Among other things, the bill (1) requires drug manufacturers to issue rebates to the CMS for covered drugs that cost \$100 or more and for which the average manufacturer price increases faster than inflation; and (2) reduces the annual out-of-pocket spending threshold, and eliminates beneficiary cost-sharing above this threshold, under the Medicare prescription drug benefit.

```
1 {
2   "cells": [
3     {
4       "cell_type": "code",
5       "execution_count": null,
6       "id": "9e461f8d",
7       "metadata": {},
8       "outputs": [],
9       "source": [
10         "import pandas as pd\n",
11         "WH_legis= pd.read_csv('house_legislation_116.csv')\n",
12         "# 'summary' column contains a summary of the bill\n",
13         "print(WH_legis['summary'][0])\n",
14         "WH_legis = WH_legis[WH_legis['summary'].notna()] # drop Nan rows\n",
15         "WH_legis.reset_index(inplace =True)\n",
16         "summary= WH_legis['summary']"
17       ],
18     },
19     {
20       "cell_type": "code",
21       "execution_count": null,
22       "id": "8a045679",
23       "metadata": {}},
```

```
24     "outputs": [],
25     "source": [
26       "# text summarization\n",
27       "import nltk\n",
28       "import spacy\n",
29       "from nltk.tokenize import sent_tokenize,word_tokenize\n",
30       "from nltk.corpus import stopwords\n",
31       "from string import punctuation\n",
32       "nltk.download('punkt')\n",
33       "nltk.download('stopwords')\n",
34       "# spacy.load will load the trained langauge pipeline, that will be used to tokenize our corpus..\n",
35       "nlp = spacy.load('en_core_web_sm')\n",
36       "data = summary[1]\n",
37       "doc= nlp(data)\n",
38       "word_tokens=[token.text for token in doc]\n",
39       "sentence_tokens= [sent for sent in doc.sents]\n",
40       "stop_w = stopwords.words('English')\n",
41       "from string import punctuation\n",
42       "word_frequencies= {} #dictionary data type\n",
43       "for word in word_tokens:\n",
44         "if word.lower() not in stop_w:\n",
45           "if word.lower() not in punctuation:\n",
46             "if word not in word_frequencies.keys():\n",
47               "word_frequencies[word] = 1\n",
48             "else:\n",
49               "word_frequencies[word] += 1\n",
50       "max_frequency=max(word_frequencies.values())\n",
51       "for word in word_frequencies.keys():\n",
52         "word_frequencies[word]=word_frequencies[word]/max_frequency# normalization\n",
53       "print(len(word_frequencies)) "
54     ],
55   },
56   {
57     "cell_type": "code",
58     "execution_count": null,
59     "id": "0d0316d1",
60     "metadata": {},
61     "outputs": [],
62     "source": [
63       "# calculation of sentence scores:\n",
64       "from heapq import nlargest\n",
65       "sentence_scores = {}\n",
66       "for sent in sentence_tokens:\n",
67         "for word in sent:\n",
68           "if word.text.lower() in word_frequencies.keys():# extract word frequency/weight\n",
69             "if sent not in sentence_scores.keys():\n",
70               "sentence_scores[sent]=word_frequencies[word.text.lower()]\n",
71             "else:\n",
72               "sentence_scores[sent]+=word_frequencies[word.text.lower()]\n",
73       "# determination of the number of sentences in the summary\n",
74       "# 20% of the sentences to be included\n",
75       "sent_no = int(len(sentence_tokens)* 0.2)\n",
76       "# get the top 20% ranked sentences\n",
77       "top_sent = nlargest(sent_no, sentence_scores,key=sentence_scores.get)\n",
78       "summary_words=[word.text for word in top_sent]\n",
79       "summary=''.join(summary_words)\n",
80       "print(summary)"
81     ],
82   },
83   {
84     "cell_type": "code",
85     "execution_count": null,
86     "id": "ed24ccf2",
87     "metadata": {},
```

```
88     "outputs": [],
89     "source": [
90       "print(top_sent)"
91     ]
92   ],
93 ],
94 "metadata": {
95   "kernelspec": {
96     "display_name": "Python 3 (ipykernel)",
97     "language": "python",
98     "name": "python3"
99   },
100  "language_info": {
101    "codemirror_mode": {
102      "name": "ipython",
103      "version": 3
104    },
105    "file_extension": ".py",
106    "mimetype": "text/x-python",
107    "name": "python",
108    "nbconvert_exporter": "python",
109    "pygments_lexer": "ipython3",
110    "version": "3.9.13"
111  }
112 },
113 "nbformat": 4,
114 "nbformat_minor": 5
115 }
```