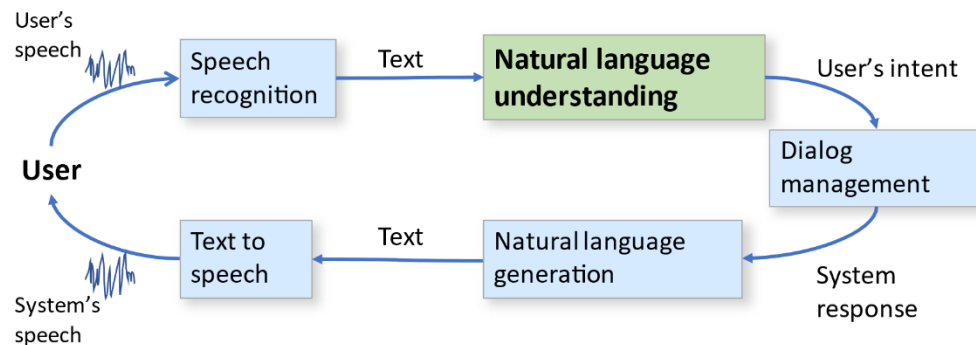


Chapter 1: Natural Language Understanding, Related Technologies, and Natural Language Applications

与大多数西方语言不同，书面中文不使用空格分隔单词



Intent: "Make a flight reservation"

Entities:

departure

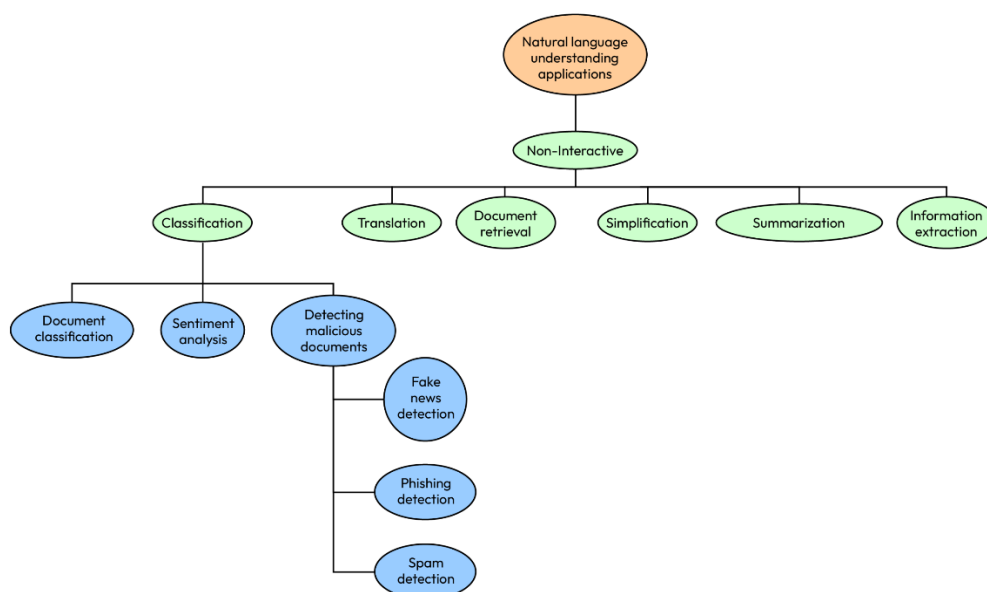
destination

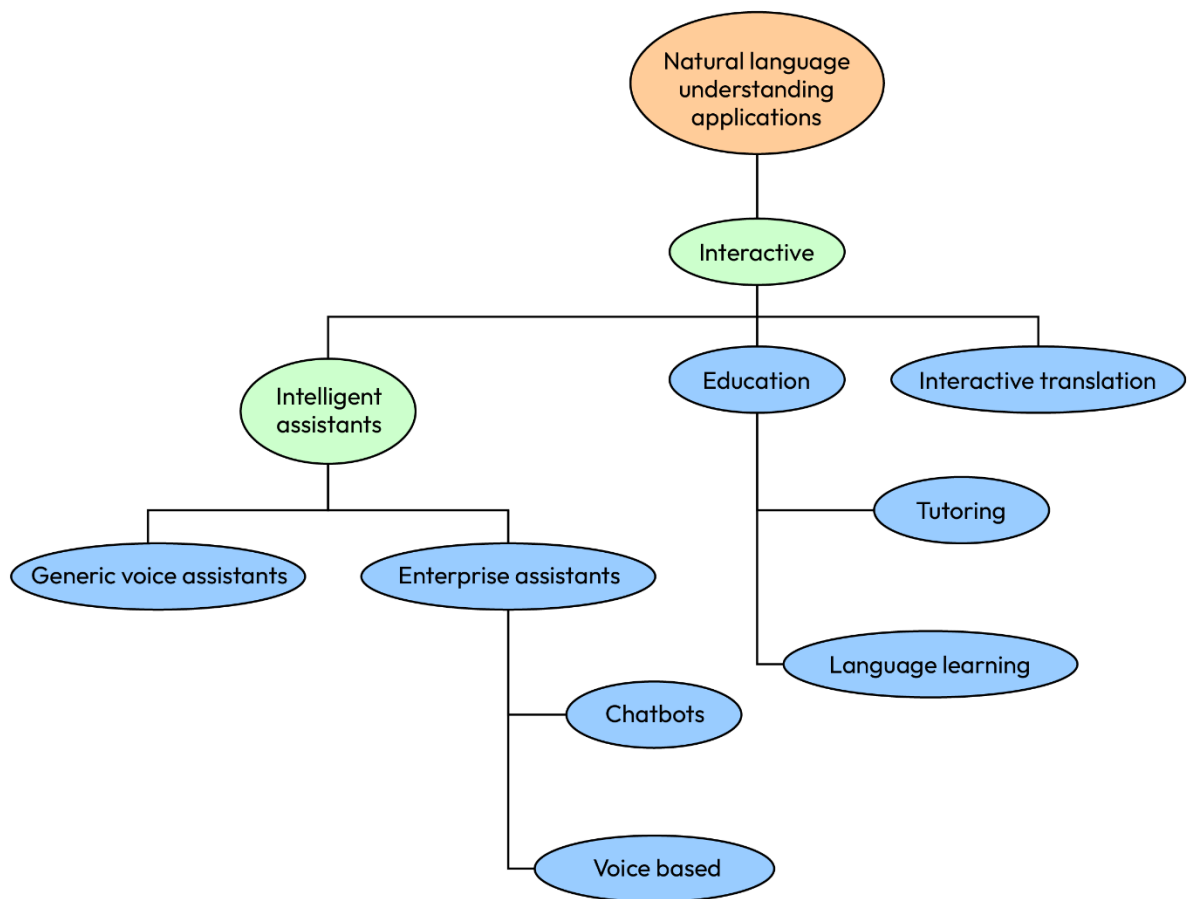
Utterance: I want to book a flight from

Boston

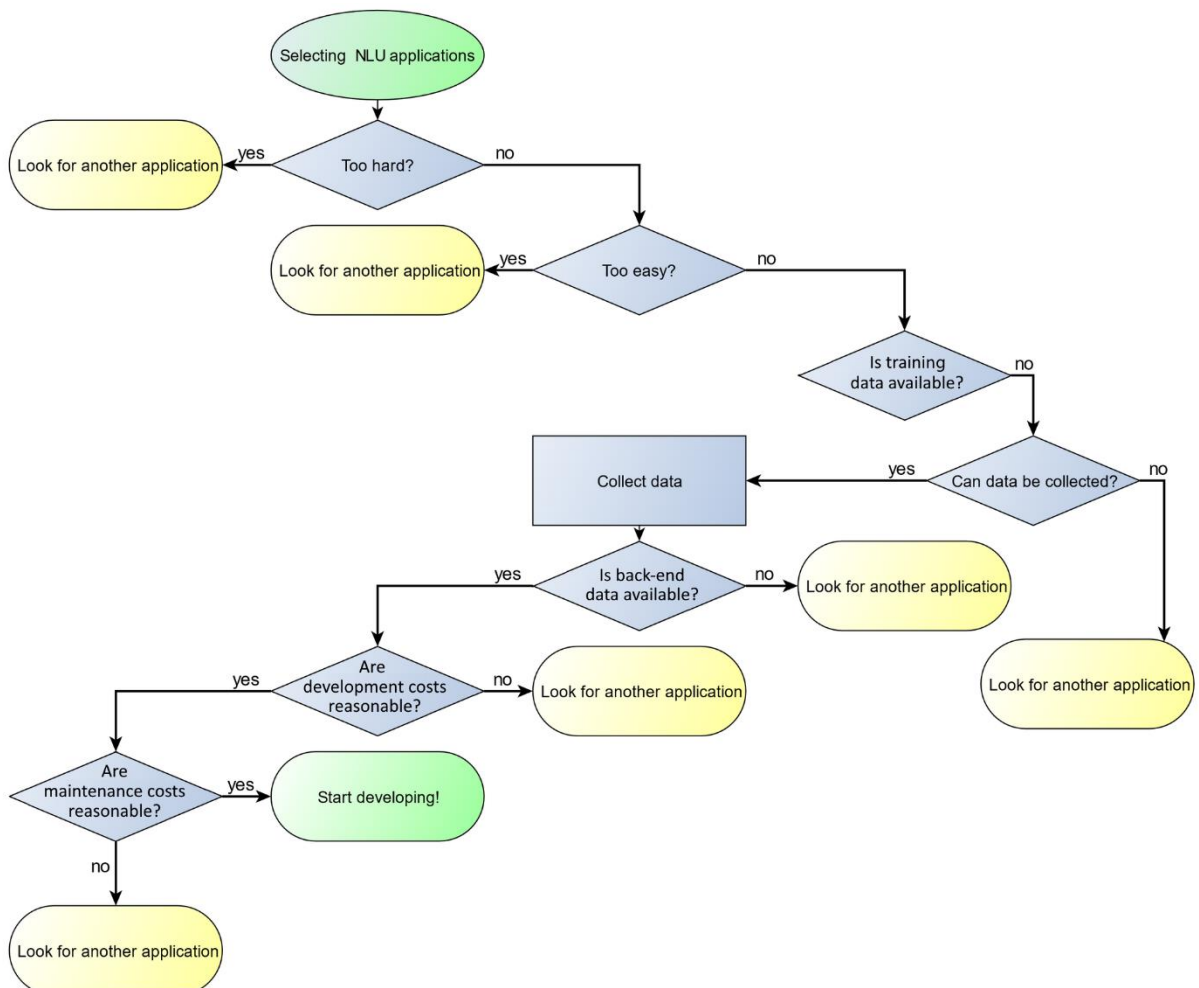
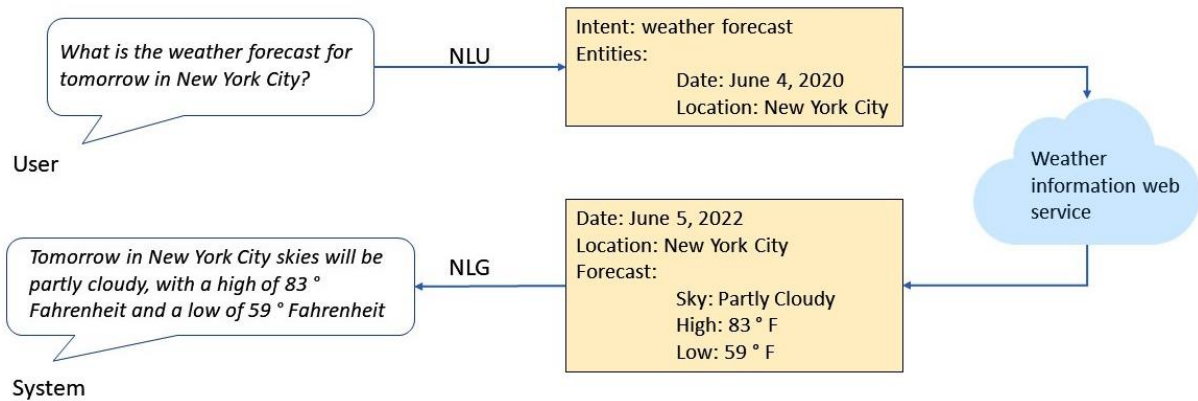
to

Philadelphia



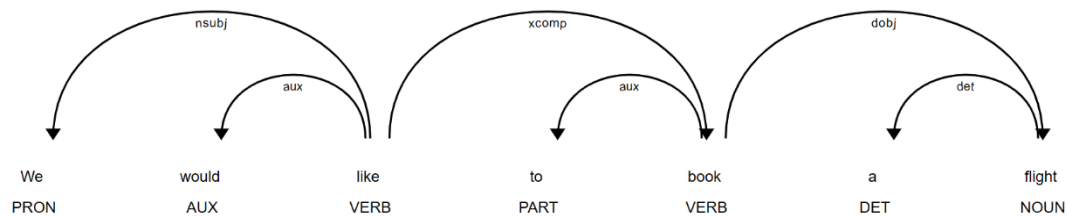


Chapter 2: Identifying Practical Natural Language Understanding Problems



Chapter 3: Approaches to Natural Language Understanding

– Rule-Based Systems, Machine Learning, and Deep Learning

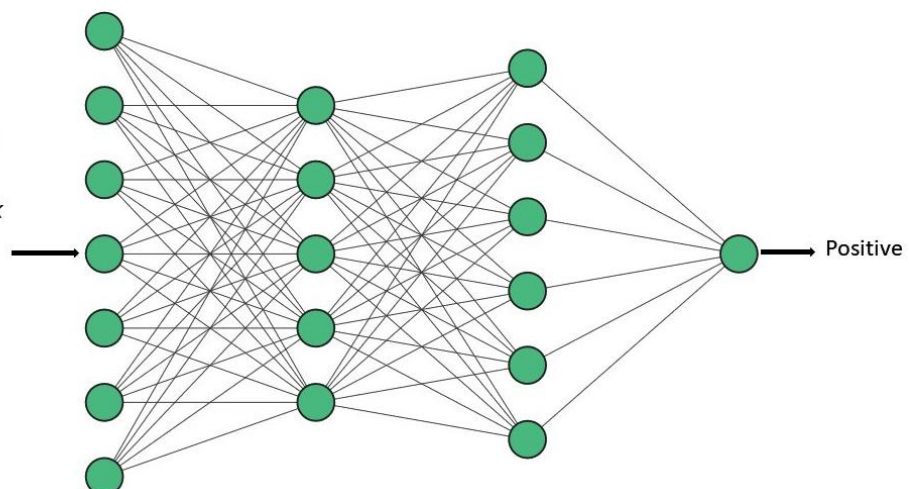


- S: (n) **airplane**, [aeroplane](#), [plane](#)
 - [direct hyponym](#) / [full hyponym](#)
 - [part meronym](#)
 - [domain term category](#)
 - [direct hyponym](#) / [inherited hyponym](#) / [sister term](#)
 - S: (n) [heavier-than-air craft](#)
 - S: (n) [aircraft](#)
 - S: (n) [craft](#)
 - S: (n) [vehicle](#)
 - S: (n) [conveyance](#), [transport](#)
 - S: (n) [instrumentality](#), [instrumentation](#)
 - S: (n) [artifact](#), [artefact](#)
 - S: (n) [whole](#), [unit](#)
 - S: (n) [object](#), [physical object](#)
 - S: (n) [physical entity](#)
 - S: (n) [entity](#)

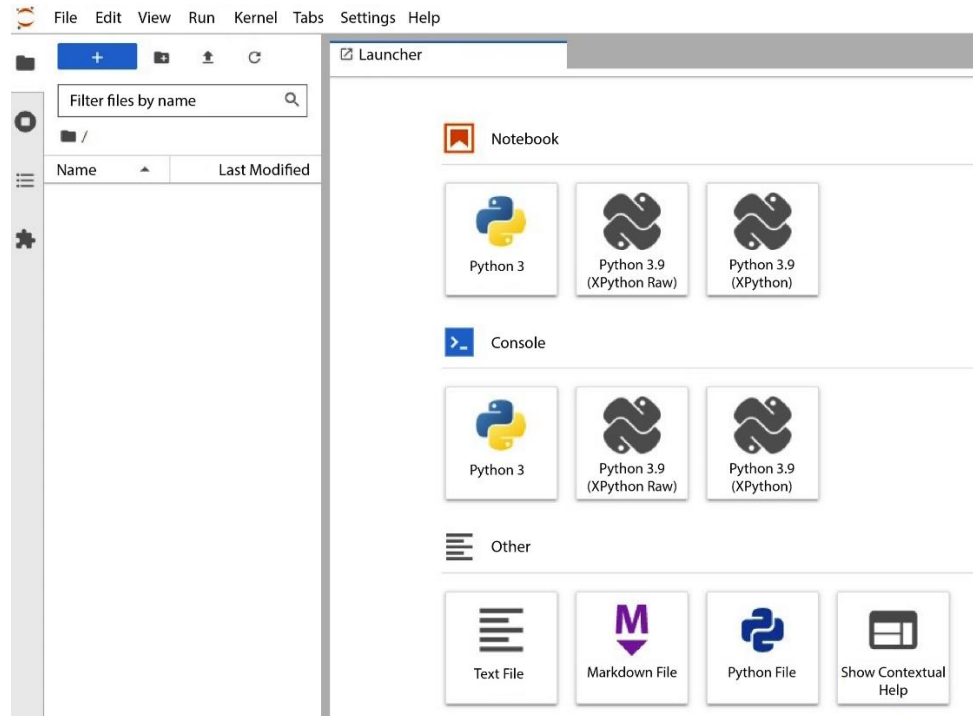
Book a flight to **London GPE** on **United ORG** for less than \$1000 dollars **MONEY** .

1. I'm looking for a nearby Chinese restaurant that's highly rated.
2. An Italian restaurant within five miles of here.
3. Are there any inexpensive Middle Eastern places that aren't too far away?

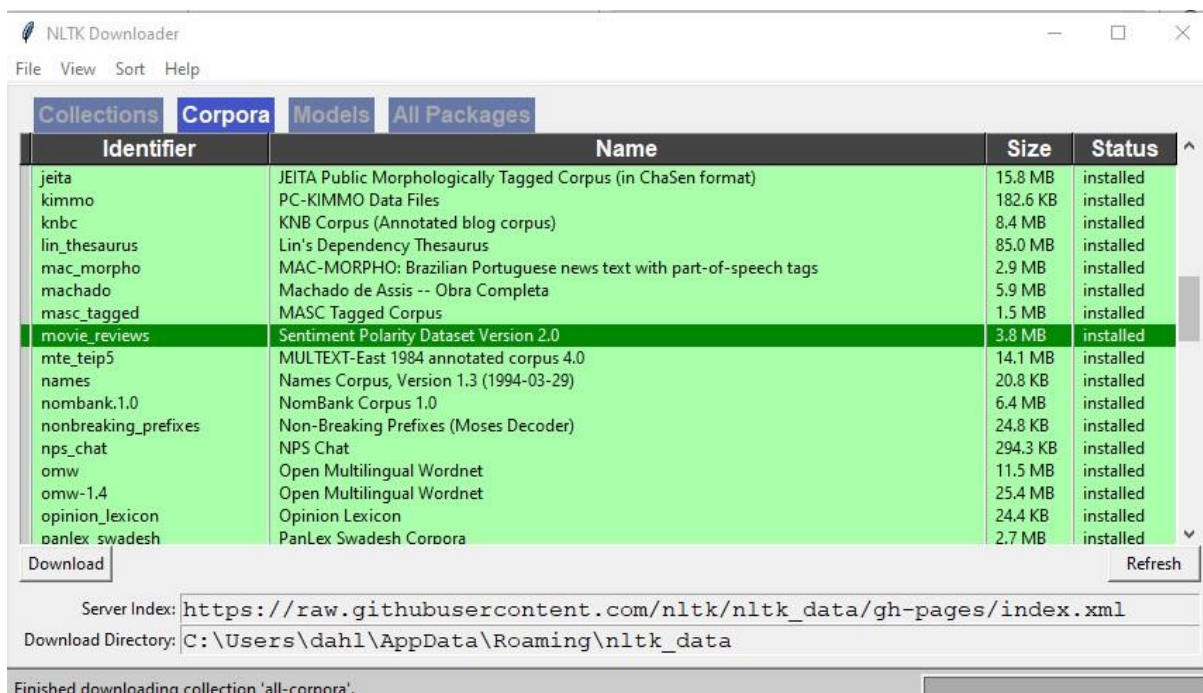
I was concerned that this chair, although comfortable, might break before I had it for very long because the legs were so thin...

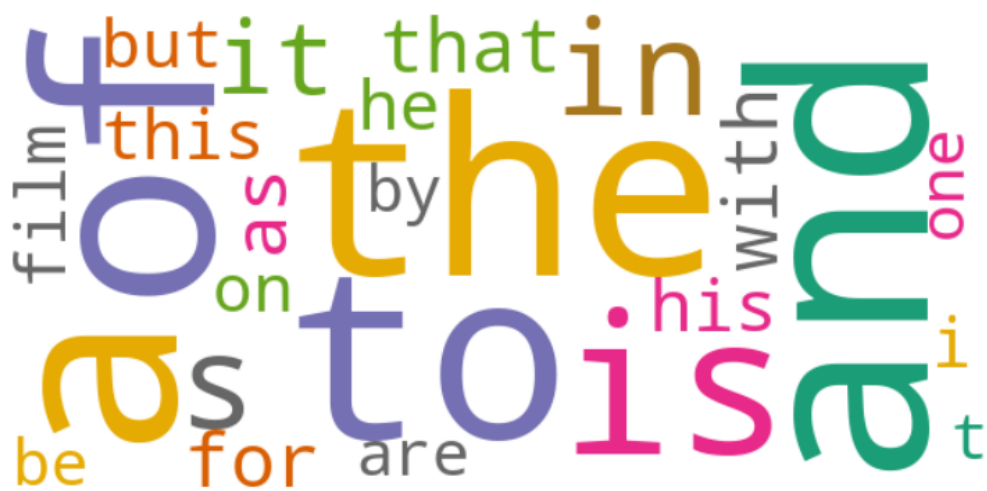
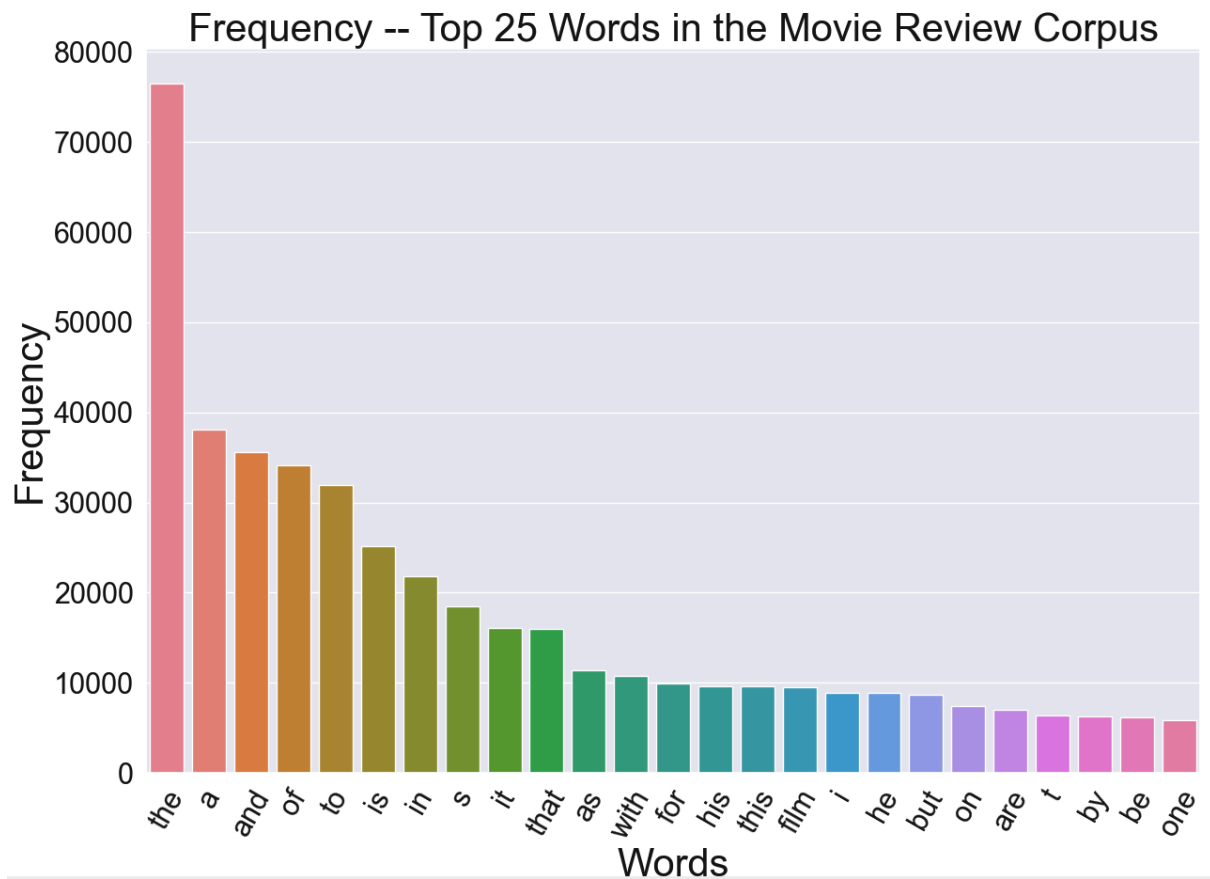
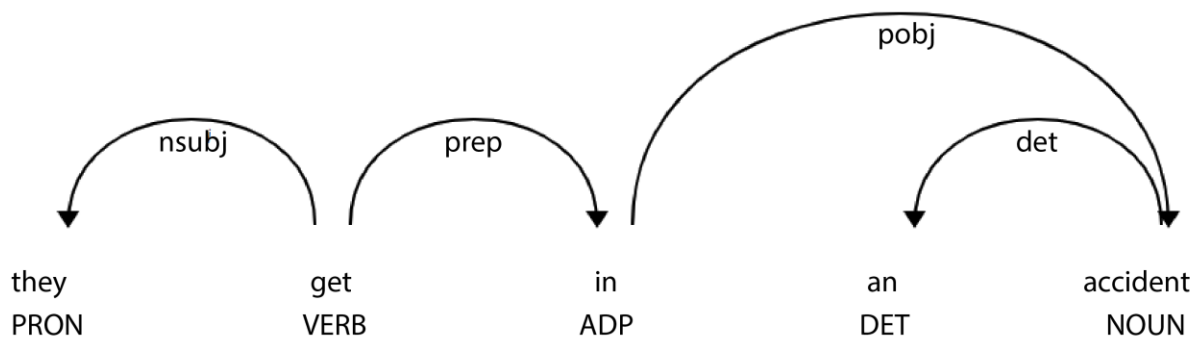


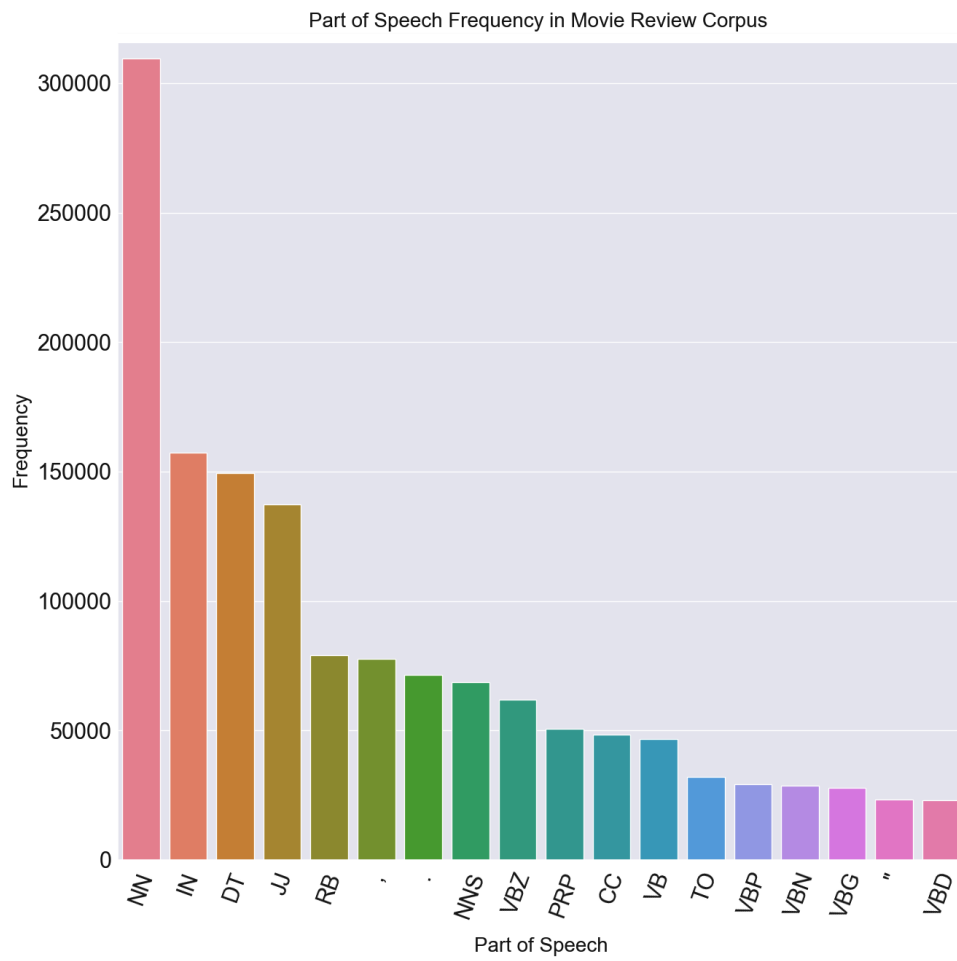
Chapter 4: Selecting Libraries and Tools for Natural Language Understanding



we'd like to book a flight from **boston GPE** to **new york GPE**







Chapter 5: Natural Language Data – Finding and Preparing Data

```
1 import demoji
2
3 happy_birthday = "Happy birthday!🎂"
4
5 text_with_emojis_replaced = demoji.replace_with_desc(happy_birthday)
6 print(text_with_emojis_replaced)
7
8 text_with_emojis_removed = demoji.replace(happy_birthday, "")
9 print(text_with_emojis_removed)
10
```

Happy birthday!:birthday cake:
Happy birthday!

```
import nltk
from nltk import word_tokenize

# a set of a few sentences to illustrate tokenization
text = ["Walk here.", "Walk here.", "Don't walk here.", "$100"]

print("Split on white space")

for sentence in text:
    tokenized = sentence.split(" ")
    print(tokenized)

print("Using NLTK tokenization")

for sentence in text:
    tokenized = word_tokenize(sentence)
    print(tokenized)
```

```
1 mixed_text = "WALK! Going for a walk is great exercise."
2 mixed_words = nltk.word_tokenize(mixed_text)
3 print(mixed_words)
4
5 lower_words = []
6 for mixed_word in mixed_words:
7     lower_words.append(mixed_word.lower())
8 print(lower_words)

['WALK', '!', 'Going', 'for', 'a', 'walk', 'is', 'great', 'exercise', '.']
['walk', '!', 'going', 'for', 'a', 'walk', 'is', 'great', 'exercise', '.']
```



```
import nltk
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
text_to_stem = "Going for a walk is the best exercise. I've walked every evening this week."
tokenized_to_stem = nltk.word_tokenize(text_to_stem)
stemmed = [stemmer.stem(w) for w in tokenized_to_stem]
print(stemmed)
```

```
['go', 'for', 'a', 'walk', 'is', 'the', 'best', 'exercis', '.', 'I', "'ve", 'walk', 'everi',
'even', 'thi', 'week', '.']
```

```
1 import nltk
2 nltk.download("wordnet")
3 from nltk.stem.wordnet import WordNetLemmatizer
4 from nltk import word_tokenize, pos_tag
5 from nltk.corpus import wordnet
6 from collections import defaultdict
7
8 # align names for parts of speech between WordNet and part of speech tagger.
9 tag_map = defaultdict(lambda: wordnet.NOUN)
10 tag_map["J"] = wordnet.ADJ
11 tag_map["V"] = wordnet.VERB
12 tag_map["R"] = wordnet.ADJ
13
14 lemmatizer = WordNetLemmatizer()
15 text_to_lemmatize = (
16     "going for a walk is the best exercise. i've walked every evening this week"
17 )
18 print("text to lemmatize is: \n", text_to_lemmatize)
19
20 tokens_to_lemmatize = nltk.word_tokenize(text_to_lemmatize)
21 lemmatized_result = ""
22 for token, tag in pos_tag(tokens_to_lemmatize):
23     lemma = lemmatizer.lemmatize(token, tag_map[tag[0]])
24     lemmatized_result = lemmatized_result + " " + lemma
25 print("lemmatized result is: \n", lemmatized_result)
```

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

text to lemmatize is:

going for a walk is the best exercise. i've walked every evening this week

lemmatized result is:

go for a walk be the best exercise . i 've walk every evening this week

```
1 from nltk.corpus import stopwords
2 nltk.download('stopwords')
3 nltk_stopwords = nltk.corpus.stopwords.words('english')
4 print(nltk_stopwords)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd",
'es', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
'hemselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are',
'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', '
'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', '

```

```

1 import spacy
2 nlp = spacy.load('en_core_web_sm')
3 spacy_stopwords = nlp.Defaults.stop_words
4 print(spacy_stopwords)

```

```

{'m', 'now', 'very', 'elsewhere', 'out', 'used', 'besides', 'mine', 're', 'top', 'each', 'whence', 'd', 'enough',
'would', 'mostly', 'that', 'such', 'for', 'everything', 'some', 'n't', 'its', 'becomes', 'm', 'also', 'own', 'call',
g', 'no', 'side', 'myself', 'toward', 'amount', 'did', 'otherwise', 'ca', 'n't', 'how', 'so', 'which', 'whatever',
s', 'almost', 'eleven', 'nevertheless', 'seemed', 'he', 'anyhow', 'front', 'always', 'therein', 'please', 'fifteen',
l', 'although', 'behind', 'bottom', 'much', 'yours', 'too', 'herein', 'made', 'without', 'after', 'hence', 'are', '

```

```

# define a sample text and tokenize it
text_to_remove_punct = "going for a walk is the best exercise!! i've walked, i believe, every evening this week."
tokens_to_remove_punct = nltk.word_tokenize(text_to_remove_punct)

# remove punctuation
tokens_no_punct = [word for word in tokens_to_remove_punct if word.isalnum()]
print(tokens_no_punct)

```

```

['going', 'for', 'a', 'walk', 'is', 'the', 'best', 'exercise', 'i', 'walked', 'i', 'believe', 'every', 'evening',
'this', 'week']

```

```

from spellchecker import SpellChecker

spell_checker = SpellChecker()

# find words that may represent spelling errors
text_to_spell_check = "Ms. Ramalingam voted against the bill"
tokens_to_spell_check = nltk.word_tokenize(text_to_spell_check)
spelling_errors = spell_checker.unknown(tokens_to_spell_check)

for misspelled in spelling_errors:
    # Get the one `most likely` answer
    print(misspelled, " should be", spell_checker.correction(misspelled))

```

```

ms. should be is
ramalingam should be None
against should be against

```

System: where do you live?

User: I live in Texas

Class token substitution: I live in <state_name>.

Chapter 6: Exploring and Visualizing Data

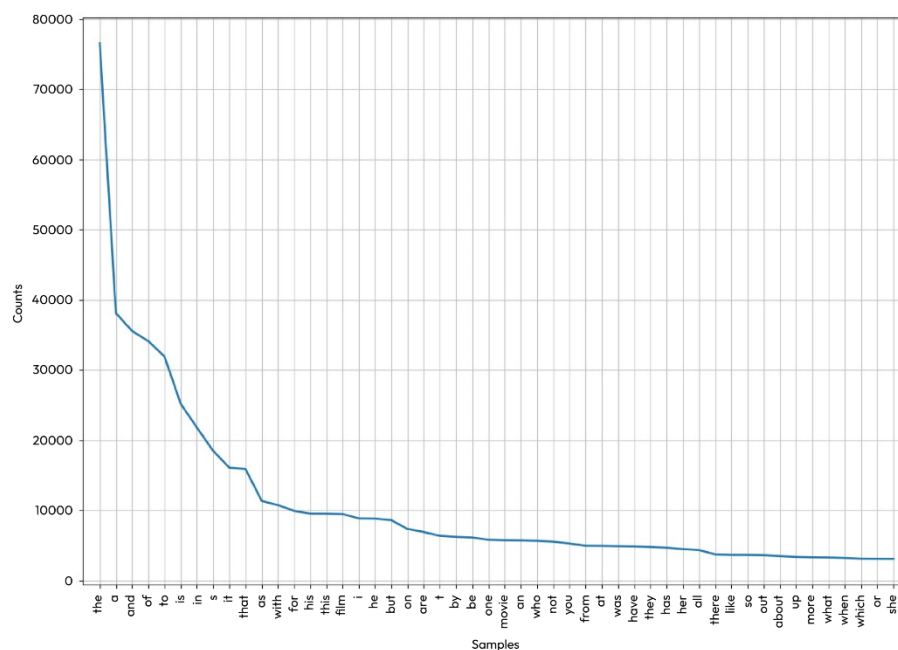
```
1 import nltk
```

```
1 #import the training data
2 from nltk.corpus import movie_reviews
```

```
3 corpus_words = movie_reviews.words()
4 print(len(corpus_words))
5 print(corpus_words)
```

```
1 # remove punctuation
2 words_no_punct = []
3 for word in corpus_words:
4     if word.isalnum():
5         words_no_punct.append(word)
6 freq = nltk.FreqDist(words_no_punct)
7 #common words
8 print("Common Words:", freq.most_common(50))
```

Common Words: [('the', 76529), ('a', 38106), ('and', 35576), ('of', 34123), ('to', 31937), ('is', 25195), ('in', 21822), ('s', 18513), ('it', 16107), ('that', 15924), ('as', 11378), ('with', 10792), ('for', 9961), ('his', 9587), ('this', 9578), ('film', 9517), ('i', 8889), ('he', 8864), ('but', 8634), ('on', 7385), ('are', 6949), ('t', 6410), ('by', 6261), ('be', 6174), ('one', 5852), ('movie', 5771), ('an', 5744), ('who', 5692), ('not', 5577), ('you', 5316), ('from', 4999), ('at', 4986), ('was', 4940), ('have', 4901), ('they', 4825), ('has', 4719), ('her', 4522), ('all', 4373), ('there', 3770), ('like', 3690), ('so', 3683), ('out', 3637), ('about', 3523), ('up', 3405), ('more', 3347), ('what', 3322), ('when', 3258), ('which', 3161), ('or', 3148), ('she', 3141)]



```

1 import nltk
2 from nltk.corpus import stopwords
3
4 stop_words = list(set(stopwords.words('english')))
5 print(len(stop_words))
6 print(stop_words[0:50])

```

179

```

['each', 'did', 'such', "should've", 'can', "shouldn't", 'him', 'as', 'there', 'n
ow', 'out', 'o', 'during', 'because', 'yourselves', 've', 'up', 'ours', 'the', 't
o', 'ma', 'both', 'am', 'after', 'all', 'your', 'hers', "mightn't", 'hadn', 'me',
"that'll", 'isn', 'same', 'from', 'just', 'he', 'she', 'haven', "hasn't", 'on',
'who', 'yourself', 'had', 'whom', 'we', 'down', 'shan', 'i', 'under', 'no']

```

```

1 languages = stopwords.fileids()
2 print('Stopwords for ', len(languages), ' languages are included in NLTK')
3 print(languages)

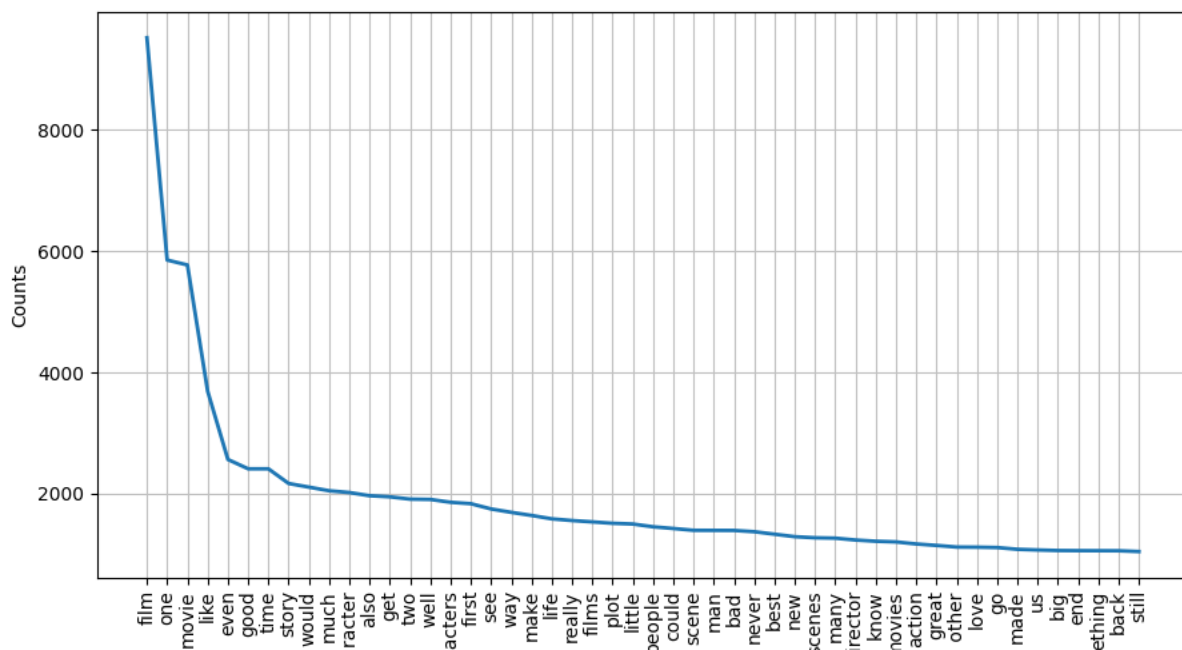
```

Stopwords for 29 languages are included in NLTK

```

['arabic', 'azerbaijani', 'basque', 'bengali', 'catalan', 'chinese', 'danish', 'dutch',
'english', 'finnish', 'french', 'german', 'greek', 'hebrew', 'hinglish', 'hungarian', 'in
donesian', 'italian', 'kazakh', 'nepali', 'norwegian', 'portuguese', 'romanian', 'russia
n', 'slovene', 'spanish', 'swedish', 'tajik', 'turkish']

```



```

1 import nltk
2 import pandas as pd
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5
6 frequency_cutoff = 25
7 all_fdist = nltk.FreqDist(freq_without_stopwords).most_common(frequency_cutoff)
8

```



```

1 import nltk
2 from nltk.util import ngrams
3 from nltk.corpus import movie_reviews
4
5 frequency_cutoff = 25
6
7 # collect the words from the corpus
8 corpus_words = movie_reviews.words()
9
10 # remove punctuation and stopwords
11 cleaned_corpus = clean_corpus(corpus_words)

```

```

13 # collect the bigrams in the corpus
14 bigrams = ngrams(cleaned_corpus,2)
15
16 # make a list from the bigrams
17 list_bigrams = list(bigrams)
18
19 # put together the bigrams into a single string
20 consolidated_bigrams = []
21 for bigram in list_bigrams:
22     consolidated_bigram = bigram[0] + " " + bigram[1]
23     consolidated_bigrams.append(consolidated_bigram)

```

```

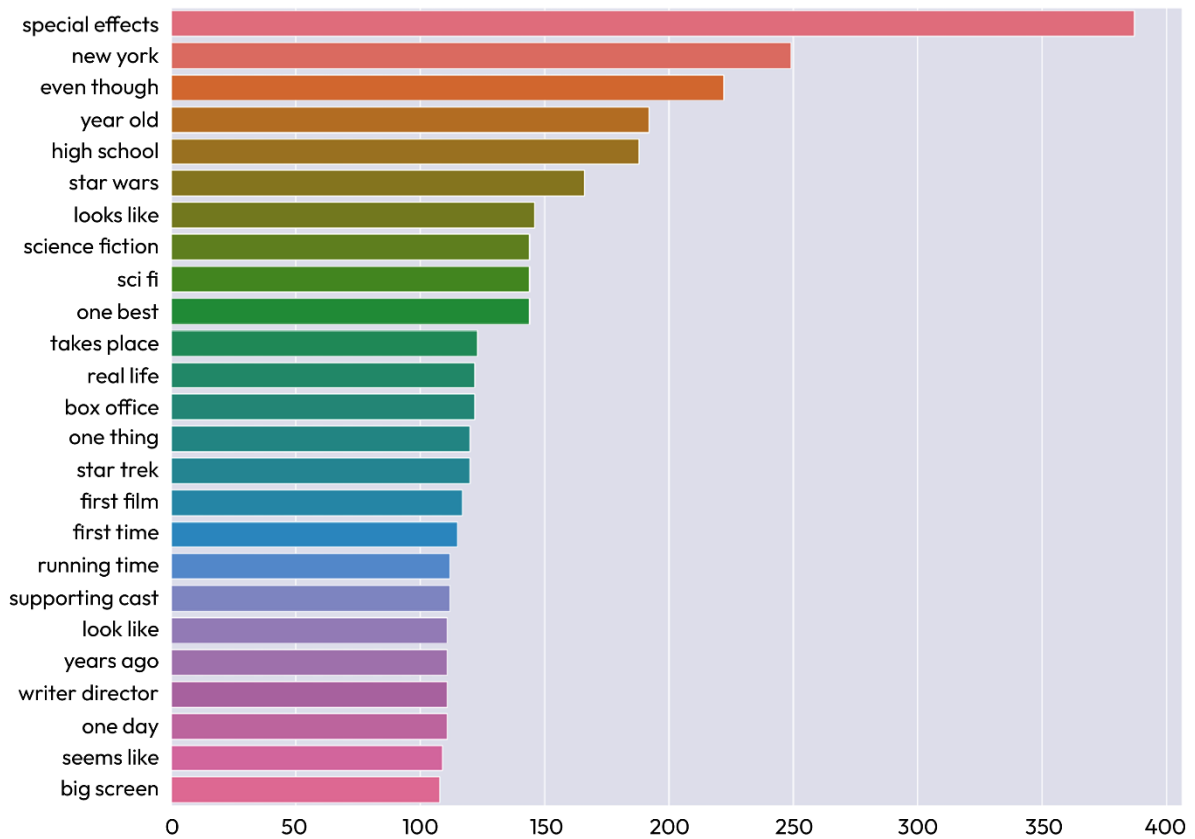
1 import random
2
3 #import the training data
4 from nltk.corpus import movie_reviews
5 corpus_words = movie_reviews.words()
6
7 # remove punctuation and stopwords
8 cleaned_corpus = clean_corpus(corpus_words)
9
10 all_words = nltk.FreqDist(w for w in cleaned_corpus)
11 max_words = 1000
12 word_features = list(all_words)[:max_words]

```

```

25 # make a frequency distribution from the bigrams
26
27 freq_bigrams = nltk.FreqDist(consolidated_bigrams).most_common(frequency_cutoff)
28 # Convert to a Pandas series
29 all_fdist = pd.Series(dict(freq_bigrams))
30
31 # set figure and axis variables and set sizes for the x and y axes
32 fig, ax = plt.subplots(figsize=(50,40))
33
34 # create a bar graph using Seaborn
35 sns.set(font_scale=2)
36 # display the bigrams on the y-axis and the counts on the x-axis
37 all_plot = sns.barplot(x=all_fdist.values, y=all_fdist.index, ax=ax)
38
39 plt.show()

```



```

13
14 def document_features(document):
15     features = {}
16     for word in word_features:
17         if word in document:
18             features[word] = 1
19         else:
20             features[word] = 0
21     return features
22
23 # make a list of documents
24 documents = [(list(movie_reviews.words(fileid)), category)
25              for category in movie_reviews.categories()
26              for fileid in movie_reviews.fileids(category)]
27
28 random.shuffle(documents)

```

```

30 # collect features, that is, words that occur in a document
31 featuresets = [(document_features(document), category) for (document,category) in documents]
32
33 #remove categories for display
34 docnumber = 0
35 new_featuresets = {}
36 for featureset in featuresets:
37     new_featureset = featureset[0]
38     new_featuresets[docnumber]= new_featureset
39     docnumber += 1
40
41 # display the words that occur in the first 10 documents, the bag of words
42 df_featuresets = pd.DataFrame.from_dict(data = new_featuresets, orient = 'index', columns = word_features)
43 df_featuresets.head(10)

```

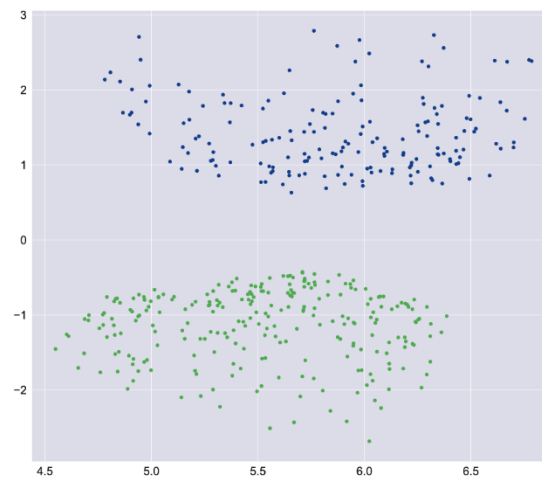

| | film | one | movie | like | even | good | time | story | would | much | ... | spielberg | development | etc | language | blue | proves | vampire | seemingly | basic | caught |
|---|------|-----|-------|------|------|------|------|-------|-------|------|-----|-----------|-------------|-----|----------|------|--------|---------|-----------|-------|--------|
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 6 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 7 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

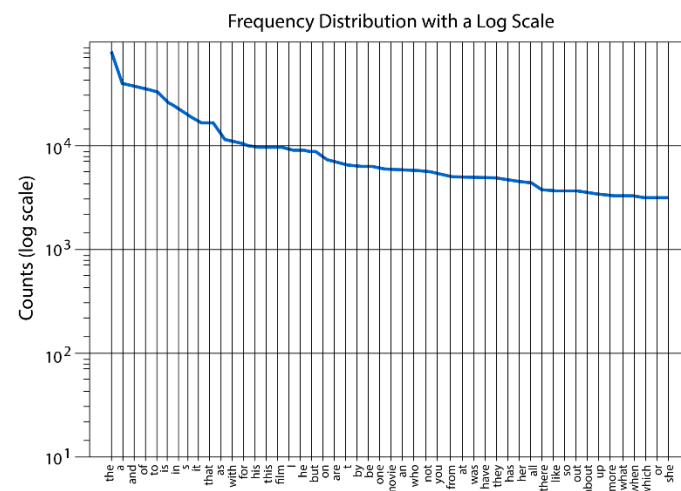
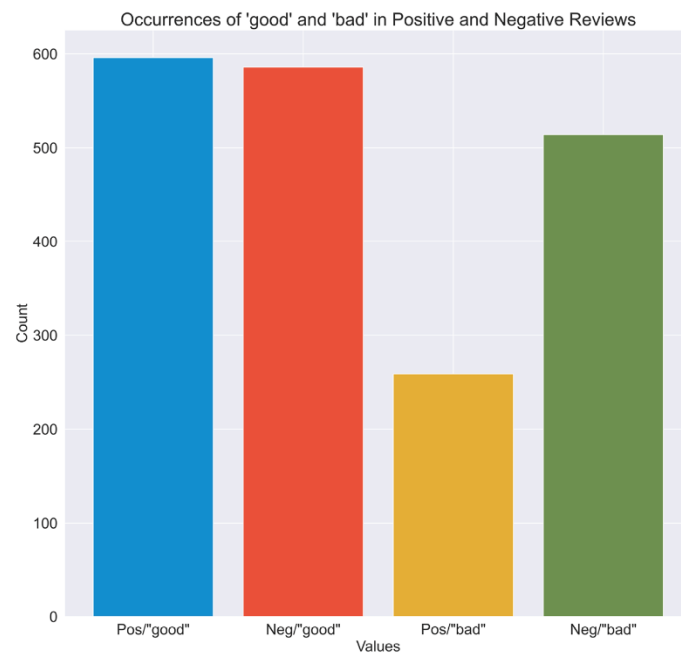
10 rows × 1000 columns

```

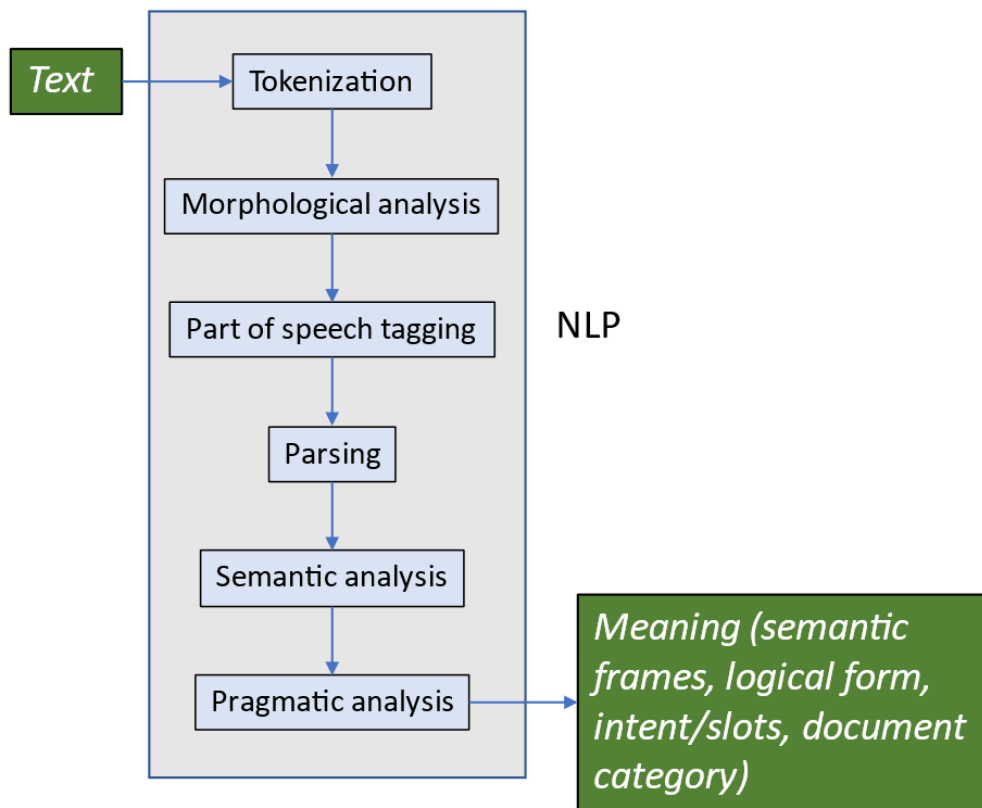
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 from sklearn.cluster import KMeans
5 from sklearn.decomposition import TruncatedSVD
6
7 true_k = 2
8
9 # truncatedSVD for reducing dimensions to 2 for display
10 truncatedSVD = TruncatedSVD(n_components = 2)
11 X_2D = truncatedSVD.fit_transform(df_featuresets)
12
13 kmeans = KMeans(n_clusters = true_k,
14                 init='k-means++',
15                 max_iter=100, # Maximum iterations
16                 n_init=10) # Number of times to run the k-means algorithm
17
18 result = kmeans.fit(X_2D)
19 labels = result.labels_
20
21 cm = plt.get_cmap('Accent')
22
23 # plot clusters in different colors
24 for cluster in range(true_k):
25     current_color = cm(1.*cluster/(true_k))
26     plt.scatter(X_2D[labels == cluster, 0], X_2D[labels == cluster, 1],
27               color = current_color, label='cluster ' + str(cluster))
28
29 plt.rcParams["figure.figsize"] = (20,20)
30 plt.rcParams['font.size'] = '12'
31 plt.show()

```





Chapter 7: Selecting Approaches and Representing Data



| | film | one | movie | like | even | good | time | story | would | much | ... | spielberg | development | etc | language | blue | proves | vampire | seemingly | basic | caught |
|---|------|-----|-------|------|------|------|------|-------|-------|------|-----|-----------|-------------|-----|----------|------|--------|---------|-----------|-------|--------|
| 0 | 3 | 2 | 0 | 1 | 1 | 0 | 3 | 0 | 2 | 4 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 6 | 0 | 3 | 0 | 1 | 3 | 0 | 0 | 1 | 2 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 2 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 3 | 9 | 3 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 4 | 5 | 3 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 4 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 5 | 2 | 1 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 4 | 4 | 3 | 0 | 1 | 2 | 3 | 1 | 0 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 4 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 2 | 6 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

10 rows × 1000 columns

```

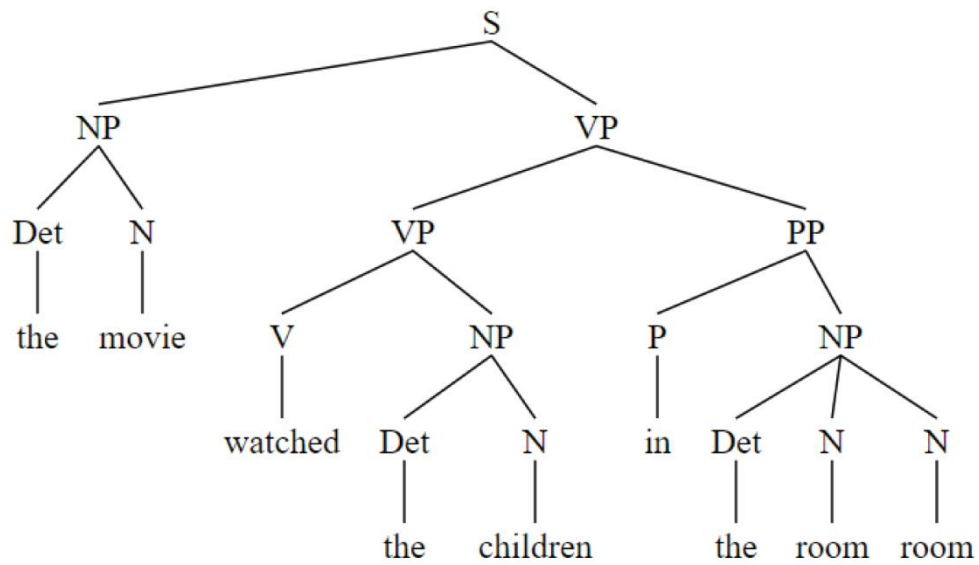
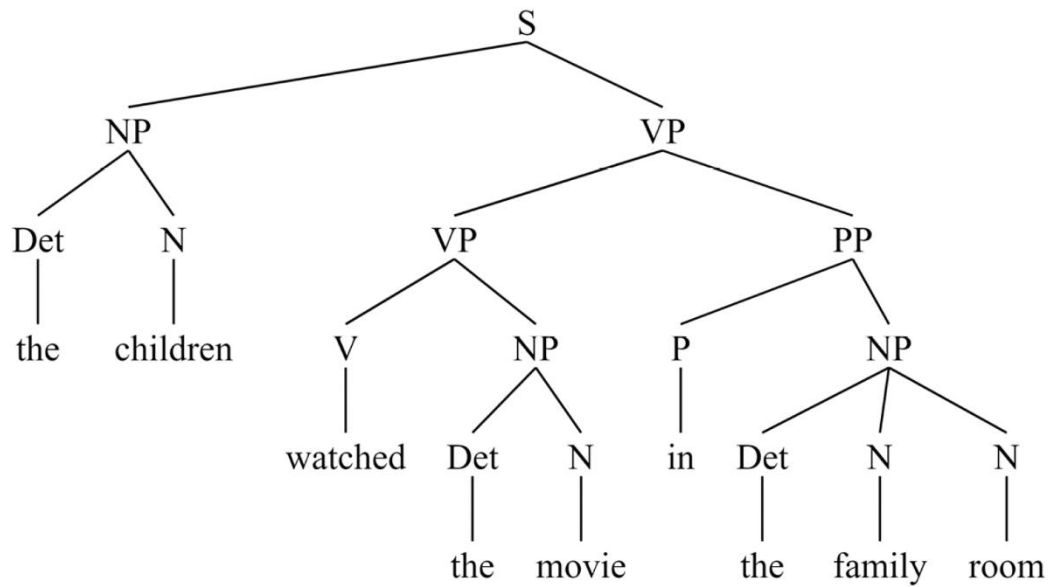
1 import string
2 import os
3 from sklearn.feature_extraction.text import TfidfVectorizer
4
5 # consider only the most common words
6 max_tokens = 200
7
8 #this tokenizer will be used to tokenize the inputs
9 def tokenize(text):
10     tokens = nltk.word_tokenize(text)
11     return tokens
12
13 path = './movie_reviews/'
14 token_dict = {}
15
16 # Look at all the files in the given path, there are 2,000 files
17 for dirpath, dirs, files in os.walk(path):
18     for f in files:
19         fname = os.path.join(dirpath, f)
20         with open(fname) as review:
21             text = review.read()
22             token_dict[f] = text.lower().translate(str.maketrans('', '', string.punctuation))
23
24 #get a new tfidf vectorizer
25 tfidfVectorizer = TfidfVectorizer(input = "content",
26                                   use_idf = True,
27                                   tokenizer = tokenize,
28                                   max_features = max_tokens,
29                                   stop_words = 'english')
30
31 # use the vectorizer to compute the tfidf of the dataset
32 tfidf = tfidfVectorizer.fit_transform(token_dict.values())
33
34 # the feature names are the words (tokens) in the dataset
35 tfidf_tokens = tfidfVectorizer.get_feature_names_out()
36
37 final_vectors = pd.DataFrame(
38     data = tfidf.toarray(),
39     columns = tfidf_tokens
40 )
41
42 final_vectors

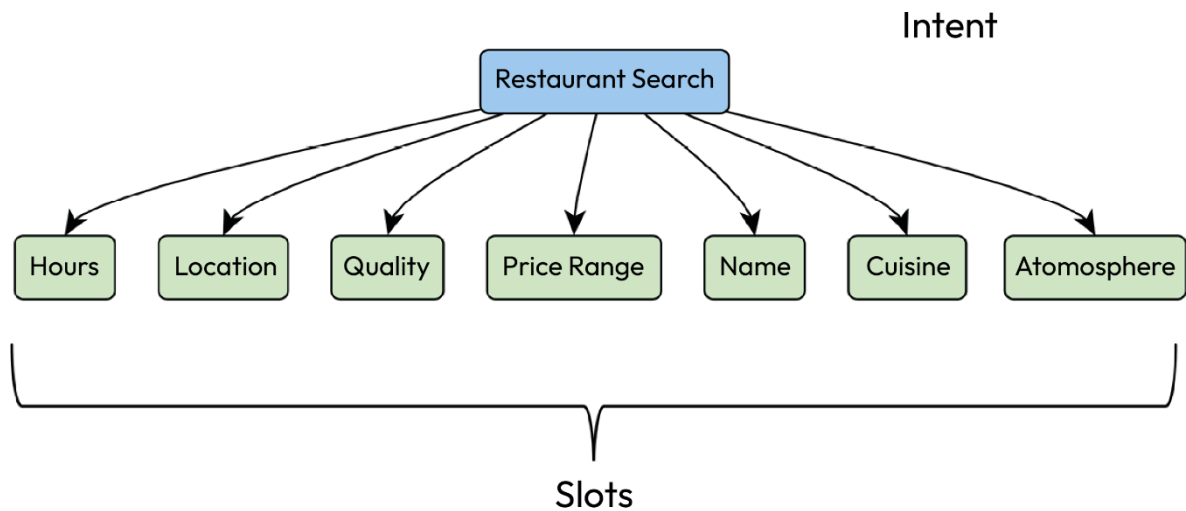
```

| | 2 | acting | action | actor | actors | actually | alien | american | audience |
|------|----------|----------|----------|----------|----------|----------|-------|----------|----------|
| 0 | 0.092834 | 0.000000 | 0.000000 | 0.000000 | 0.069898 | 0.134576 | 0.0 | 0.084837 | 0.130063 |
| 1 | 0.000000 | 0.266255 | 0.132484 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.000000 | 0.000000 |
| 2 | 0.000000 | 0.094612 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.115113 | 0.264719 |
| 3 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.160810 | 0.000000 | 0.0 | 0.000000 | 0.000000 |
| 4 | 0.000000 | 0.000000 | 0.000000 | 0.210719 | 0.000000 | 0.090622 | 0.0 | 0.000000 | 0.087584 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1995 | 0.000000 | 0.046388 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.056440 | 0.000000 |
| 1996 | 0.000000 | 0.000000 | 0.000000 | 0.171716 | 0.000000 | 0.147697 | 0.0 | 0.000000 | 0.000000 |
| 1997 | 0.000000 | 0.000000 | 0.133597 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.000000 | 0.000000 |
| 1998 | 0.000000 | 0.000000 | 0.000000 | 0.089530 | 0.159988 | 0.000000 | 0.0 | 0.000000 | 0.000000 |
| 1999 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.000000 | 0.000000 |

2000 rows × 200 columns

Chapter 8: Rule-Based Techniques





can you recommend a casual **ATMOSPHERE** italian **CUISINE** within walking distance **LOCATION**

| Name | Purpose | Model format | Processing | Result |
|---------------------|---|---------------------------|--------------------------------------|---|
| Regular expressions | Recognize and format fixed phrases such as dates, times, addresses, and phone numbers | Regular expression syntax | Regular expression parser | Match object, string |
| Syntactic parsing | Recognize and label grammatical structures | Context-free grammar | Parsing (for example, chart parsing) | Constituent tree, dependency graph |
| Semantic analysis | Identify semantic relationships among elements of a sentence | Pattern rules | Semantic parsing | Slot-value pairs (slots are also referred to as "entities") |

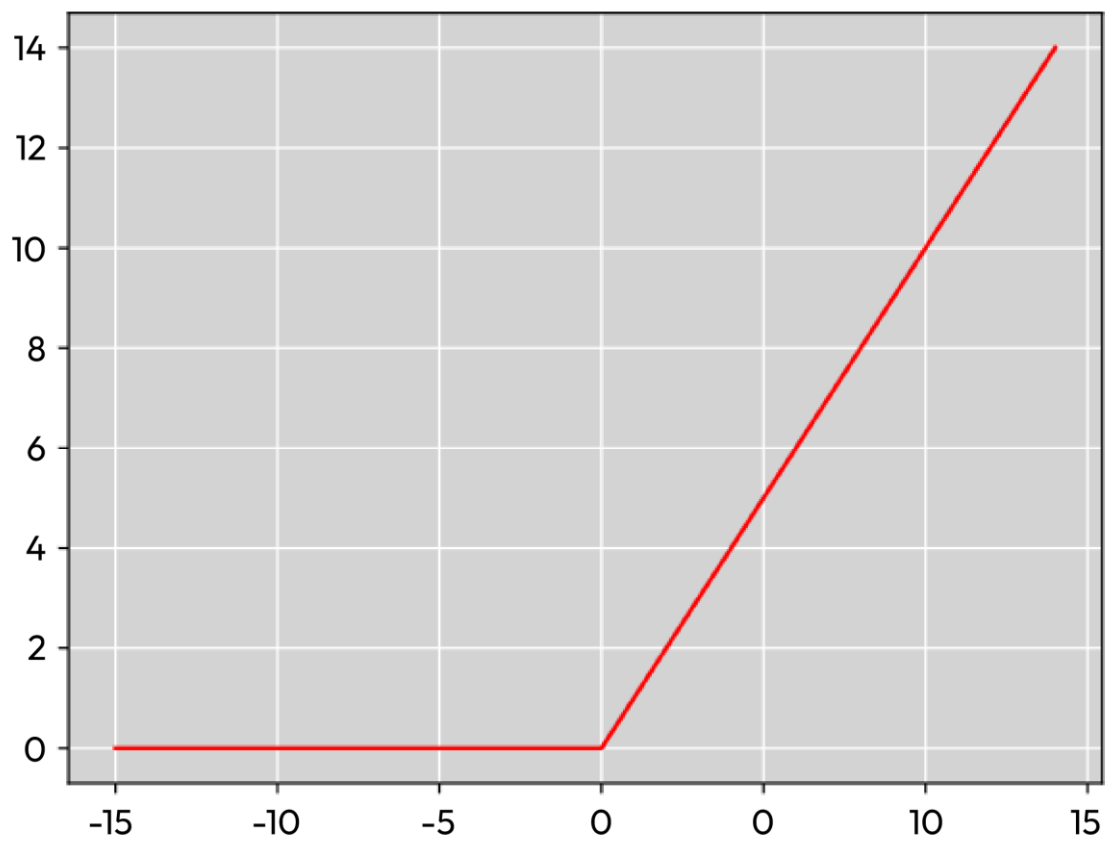
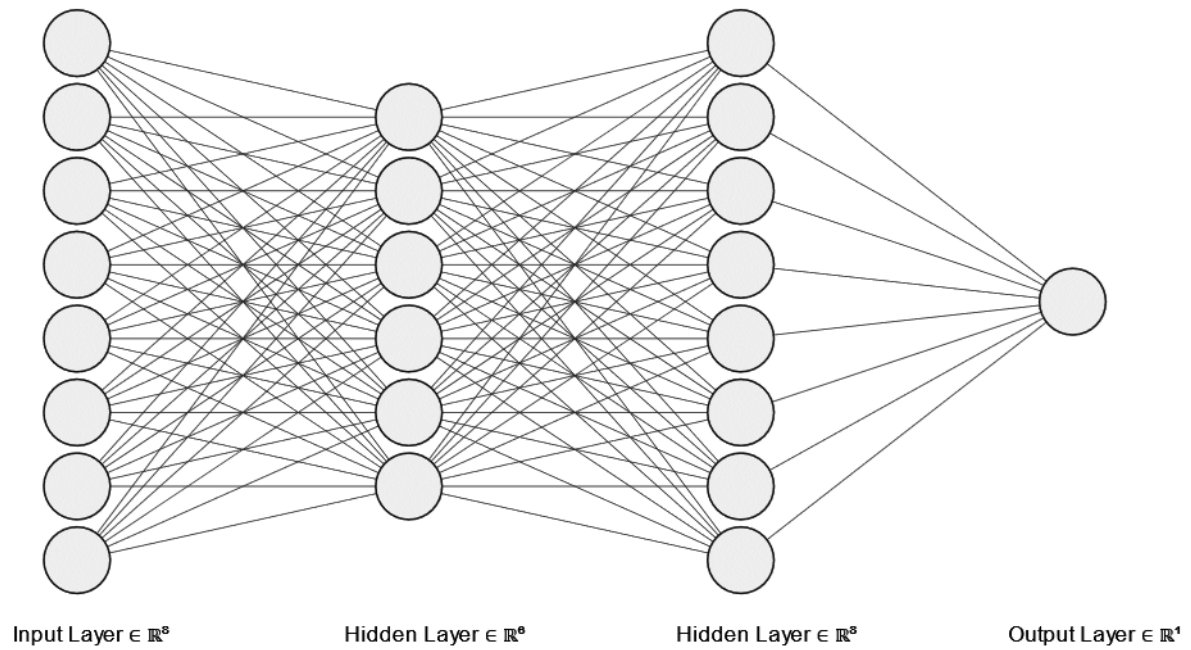
Chapter 9: Machine Learning Part 1 – Statistical Machine Learning

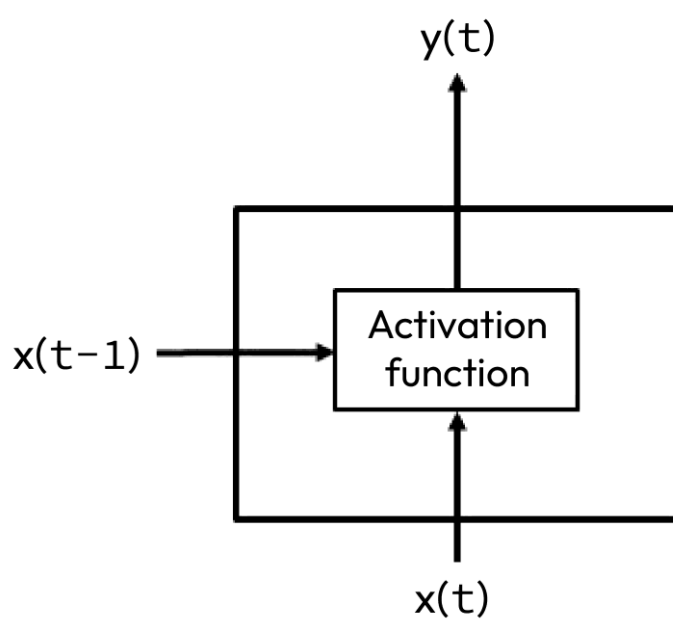
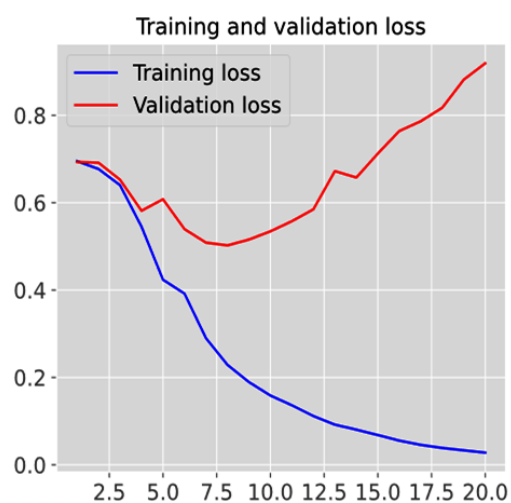
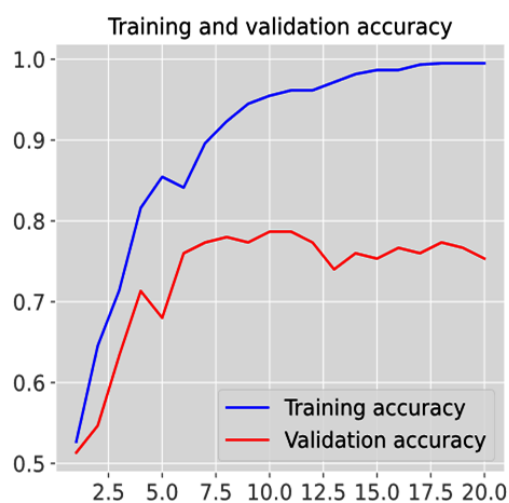
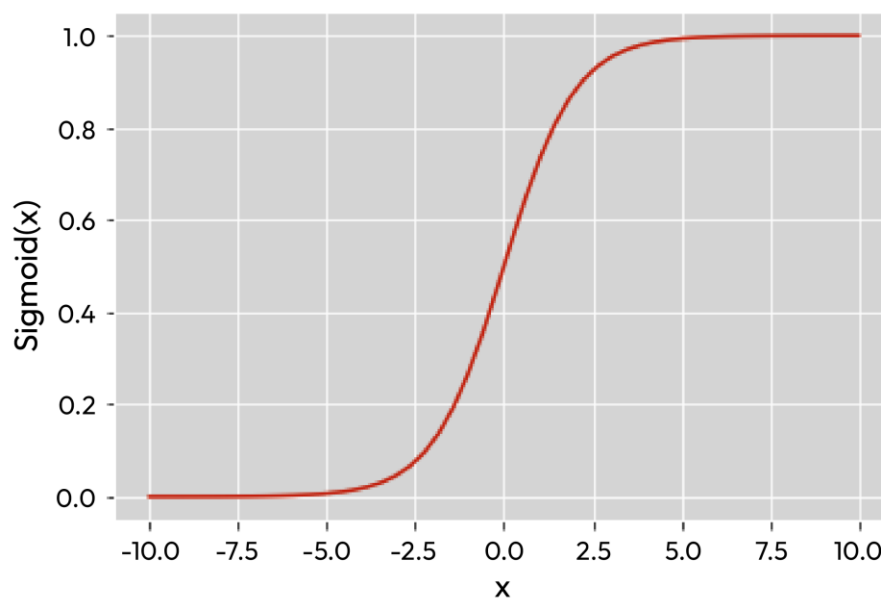
O → show
O → me
B-GENRE → science
I-GENRE → fiction
I-GENRE → films
O → directed
O → by
B-DIRECTOR → steven
I-DIRECTOR → spielberg

restaurant_search

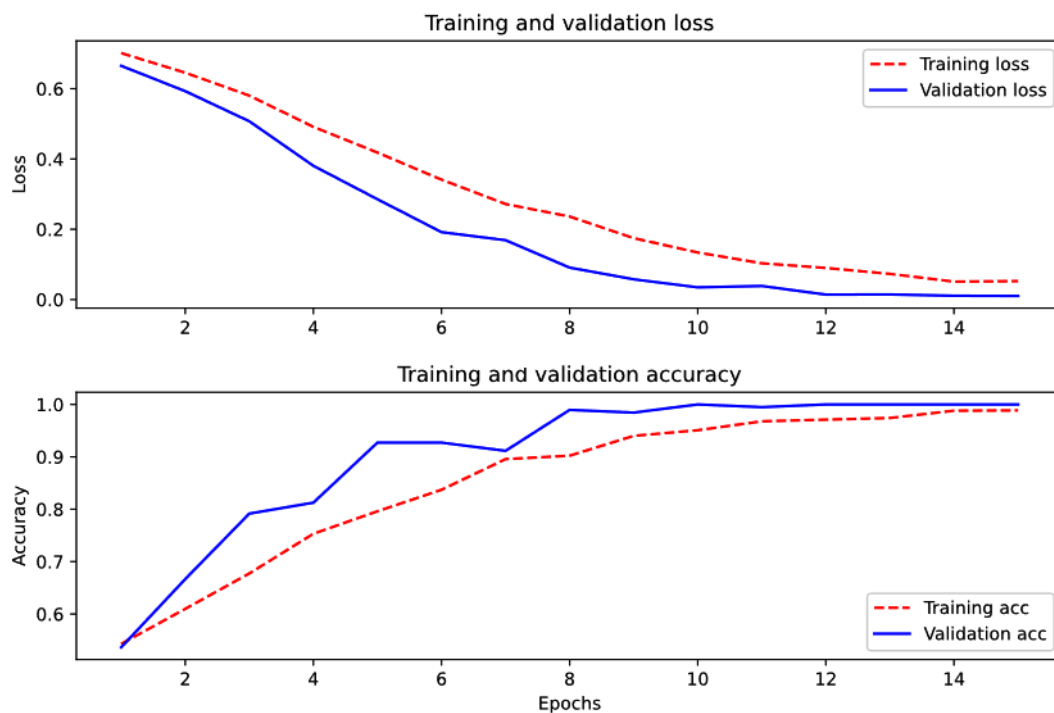
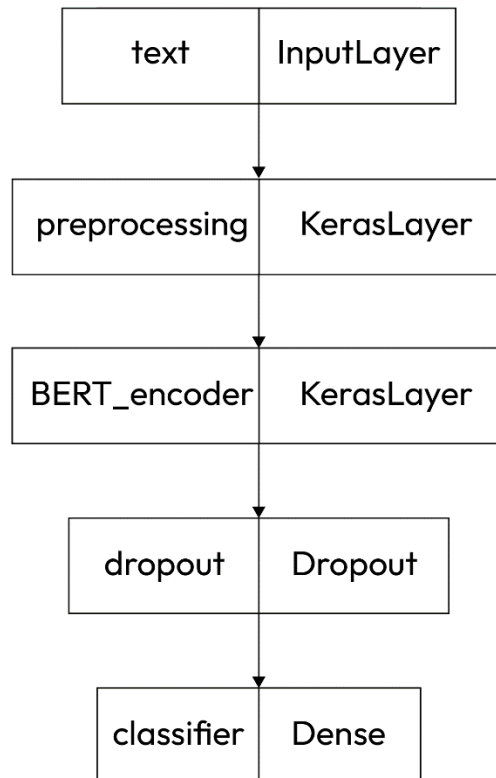
- I want to get some [lunch](meal)
- I am searching for a [dinner](meal) spot
- I'm looking for a place in the [north](location) of town
- show me some [good](quality) [Chinese](cuisine) restaurants in the [north](location)
- how about a [Mexican](cuisine) restaurant [downtown](location)
- Are there any [Indian](cuisine) spots near here
- Italian restaurants on the [west side] (location)
- looking for [German](cuisine) places in the [south](location)
- what [Greek](cuisine) places are near [12345](location)
- help me find a [casual](atmosphere)[Asian fusion](cuisine) place
- I am looking for a [French](cuisine) restaurant [nearby](location)
- I am looking for a [nice] (quality) [Mexican](cuisine) or [thai](cuisine) place that's [not too expensive](price)
- [cozy](atmosphere) [barbecue](cuisine) restaurant

Chapter 10: Machine Learning Part 2 – Neural Nets and Deep Learning Techniques






Chapter 11: Machine Learning Part 3 – Transformers and Large Language Models



ChatGPT - Question (prompt)



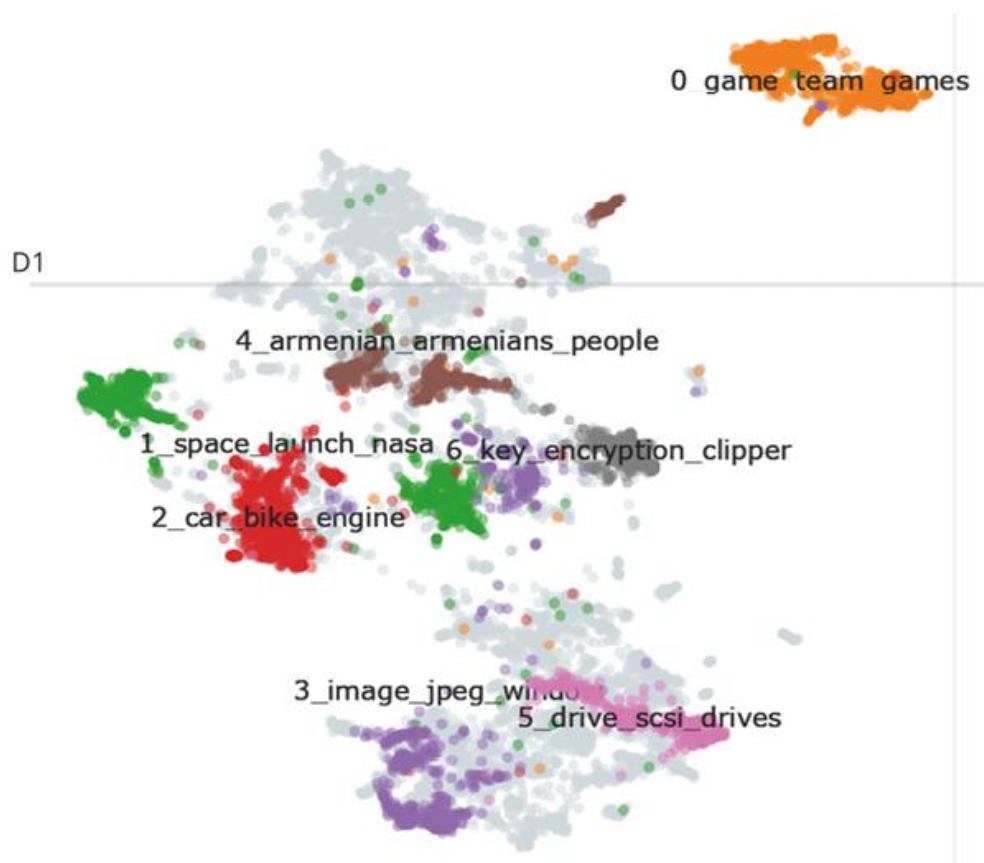
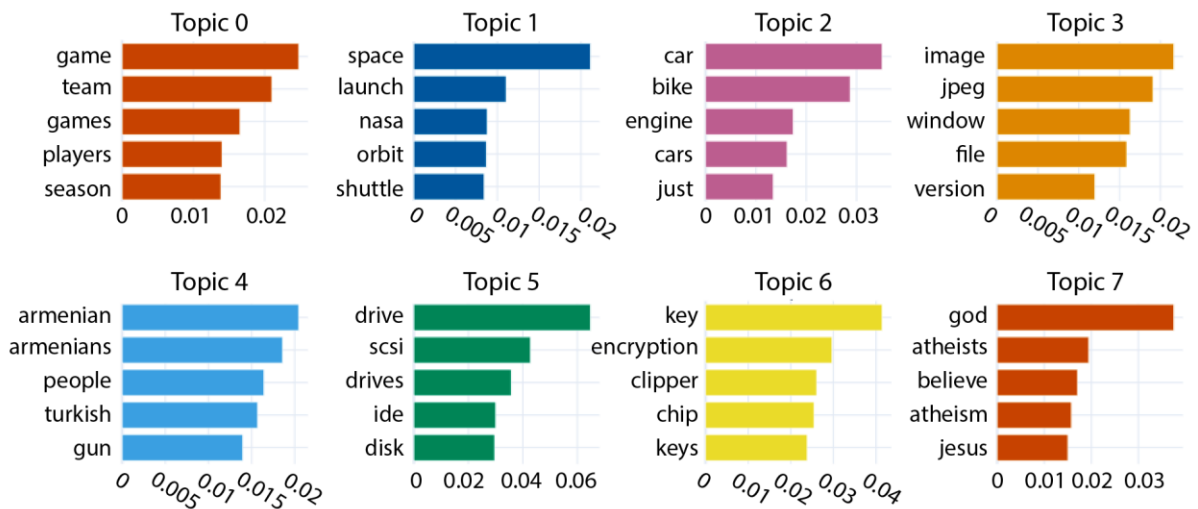
Model: ChatGPT App - text-davinci-002-render-sha 

Sure, here are 10 examples of how someone might ask for their checking balance:

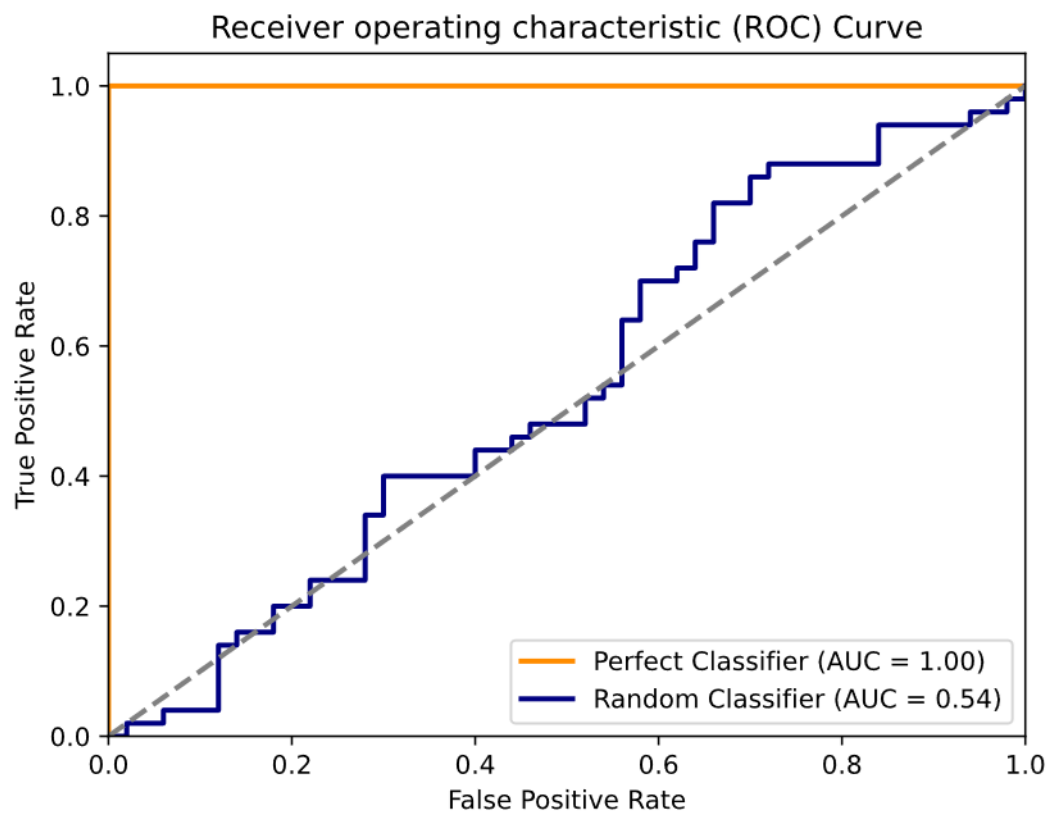
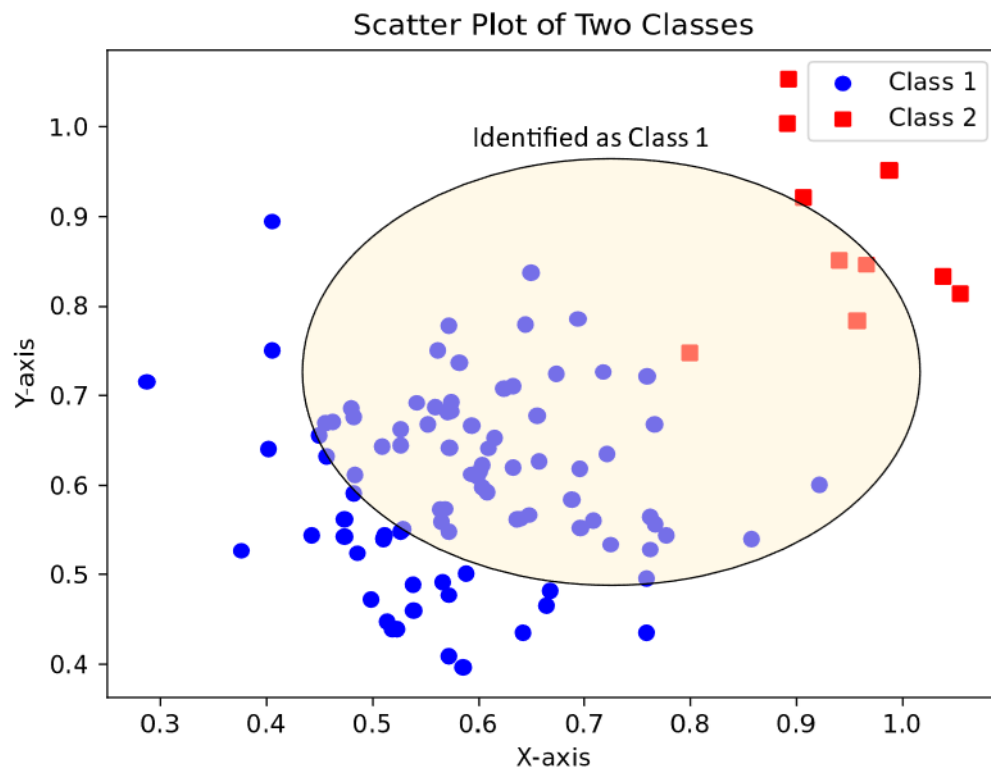
- "What's my checking account balance?"
- "Can you tell me how much money is in my checking account?"
- "I need to know how much money I have in my checking account. Can you help me with that?"
- "Could you please give me my current checking account balance?"
- "What is my current balance in my checking account?"
- "I'd like to check my checking account balance. Can you assist me with that?"
- "Can you provide me with my checking account balance, please?"
- "How much money do I have available in my checking account right now?"
- "May I know the balance in my checking account, please?"
- "What's the balance on my checking account statement?"

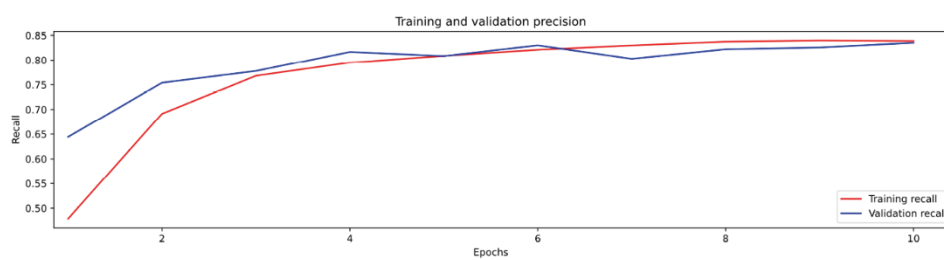
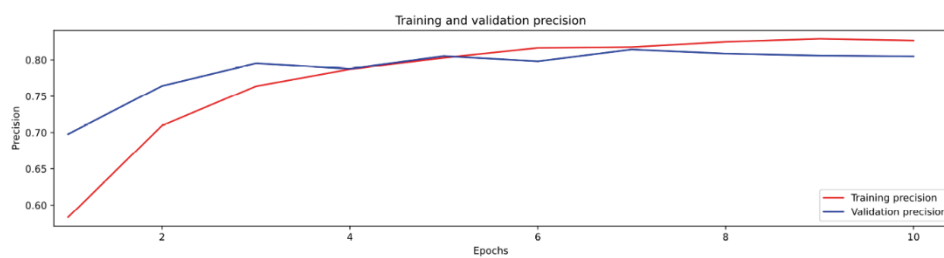
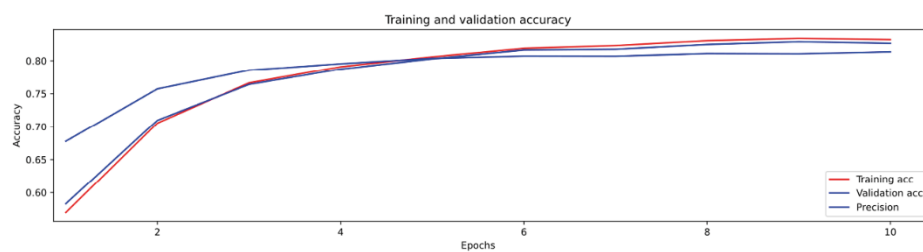
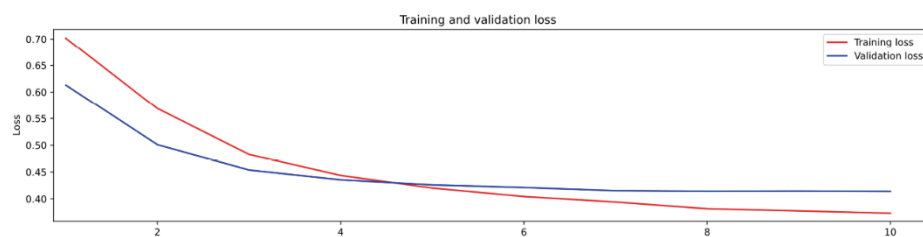
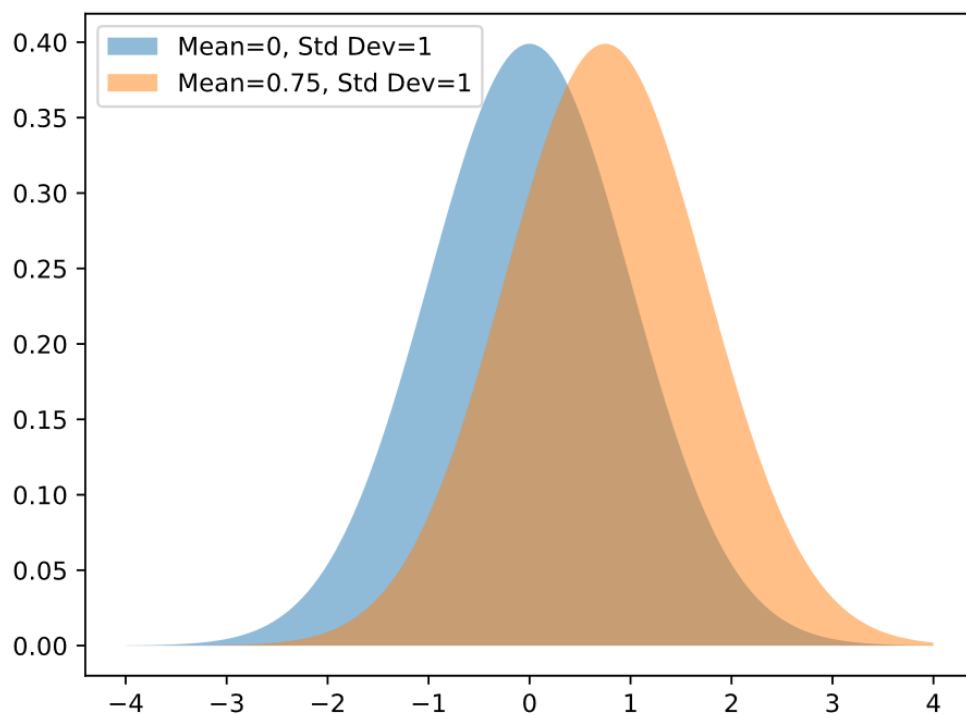
Chapter 12: Applying Unsupervised Learning Approaches

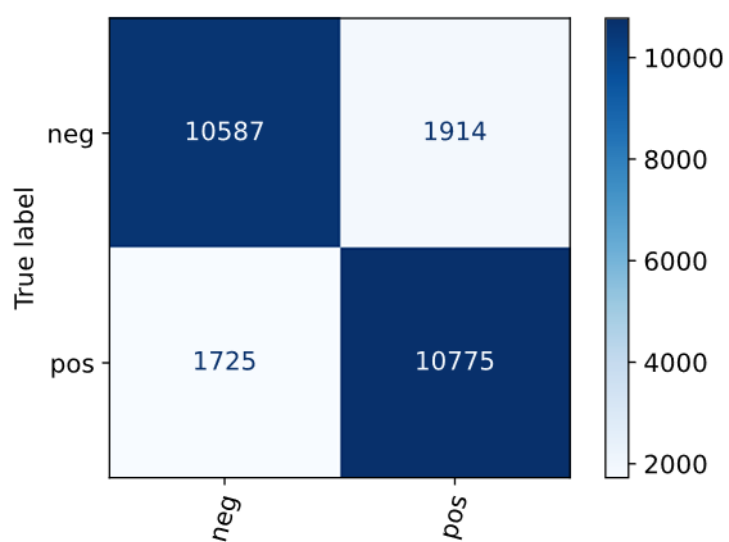
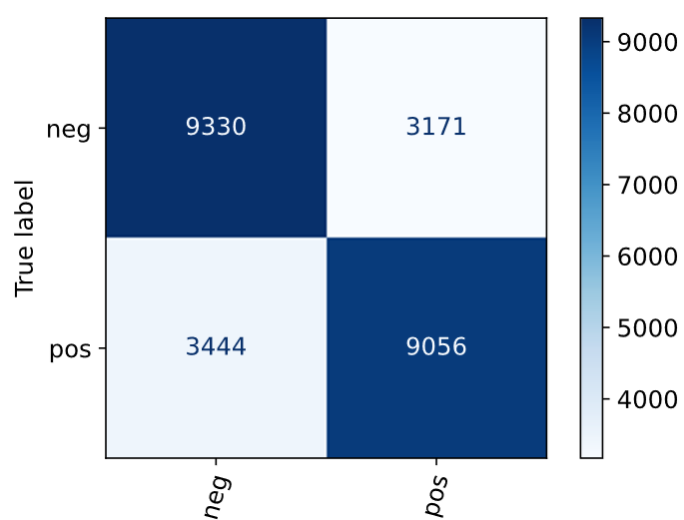
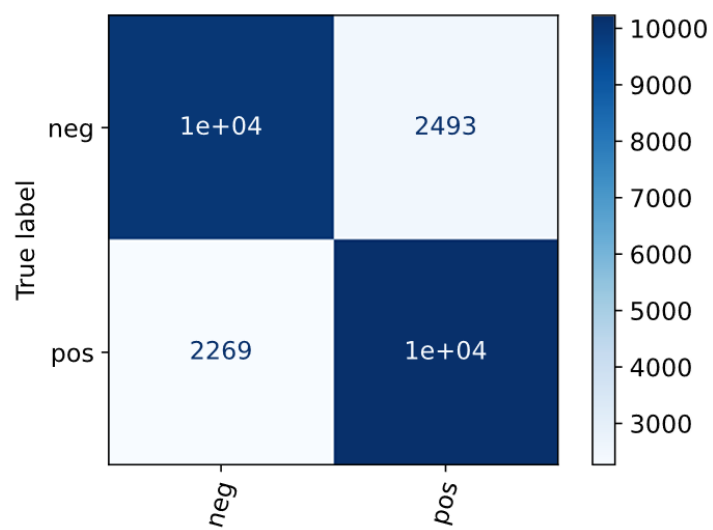
Topic Word Scores



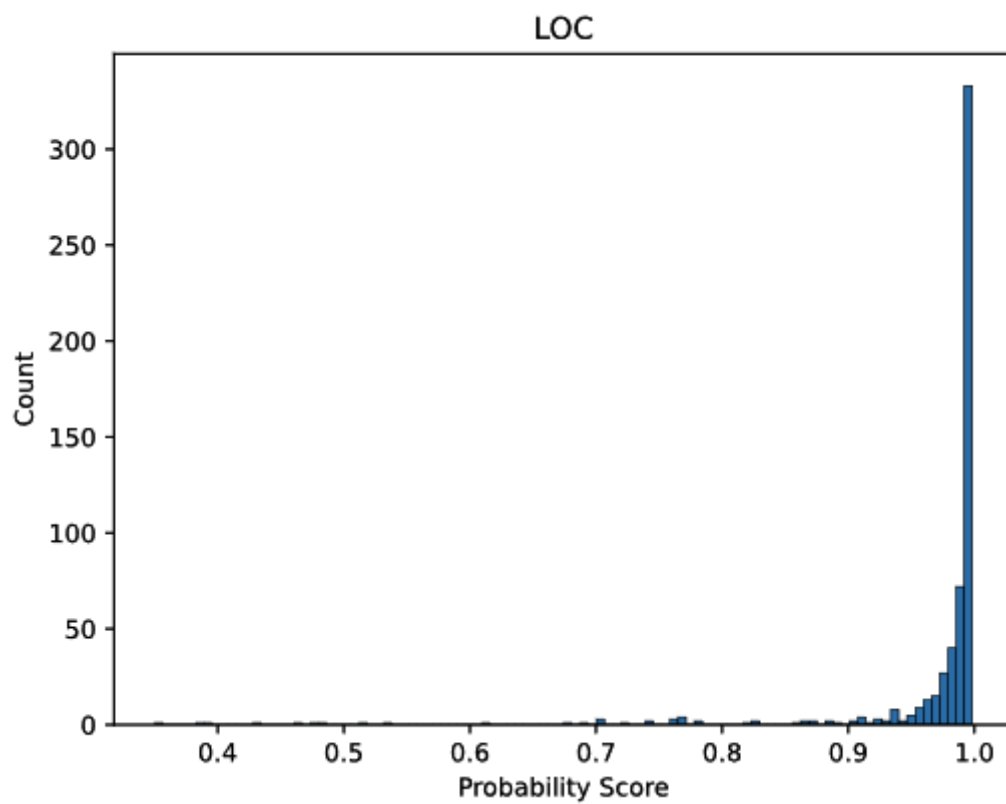
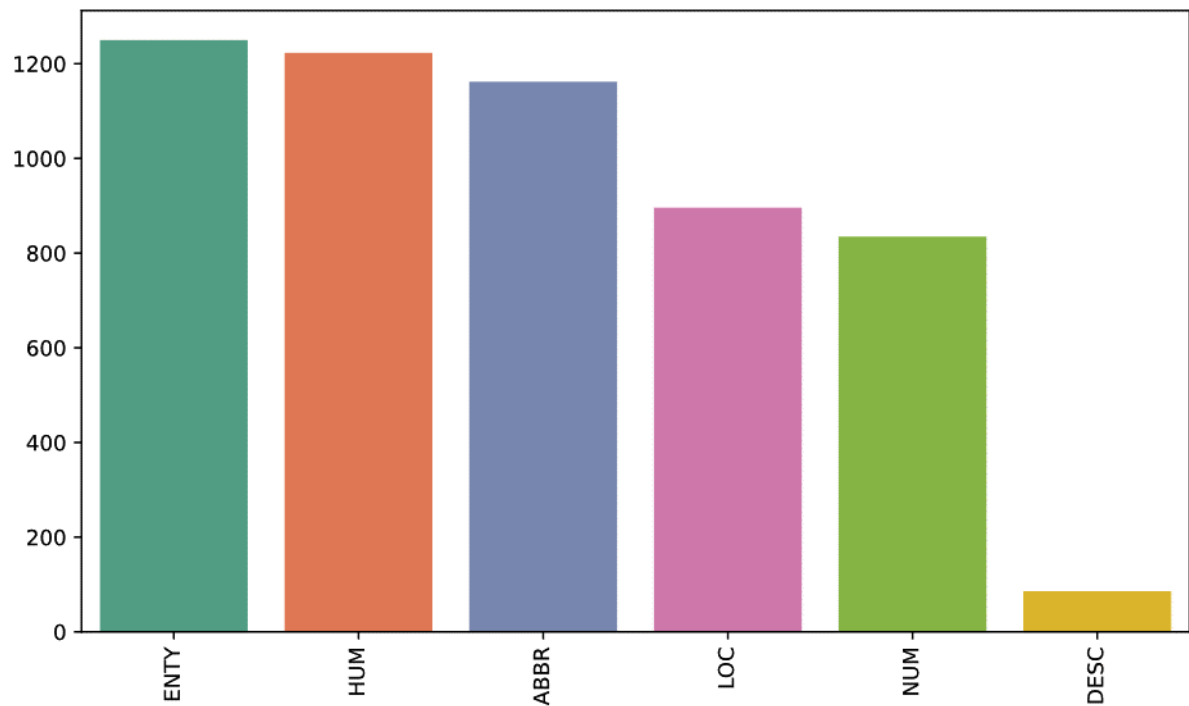
Chapter 13: How Well Does It Work? – Evaluation

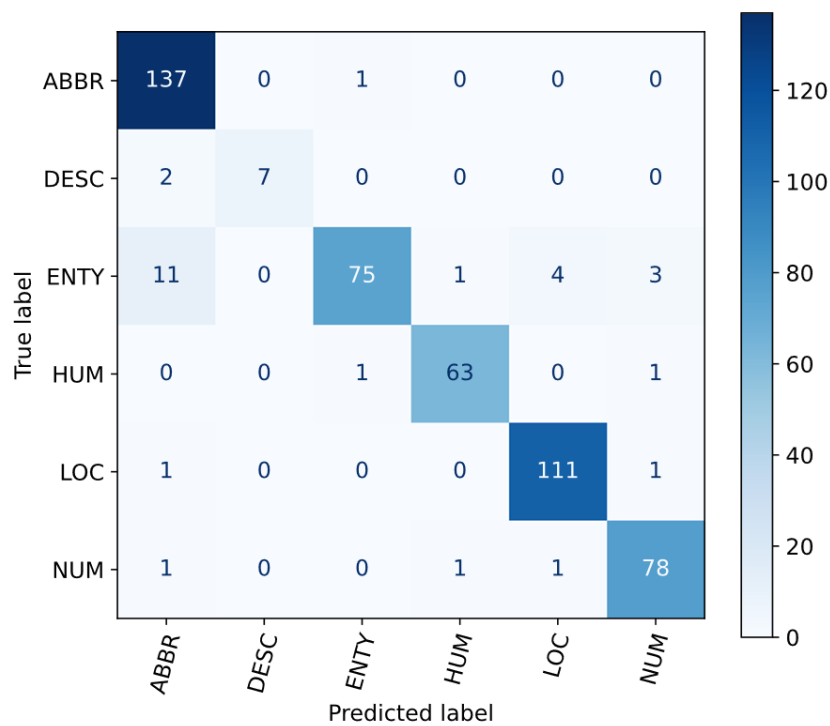
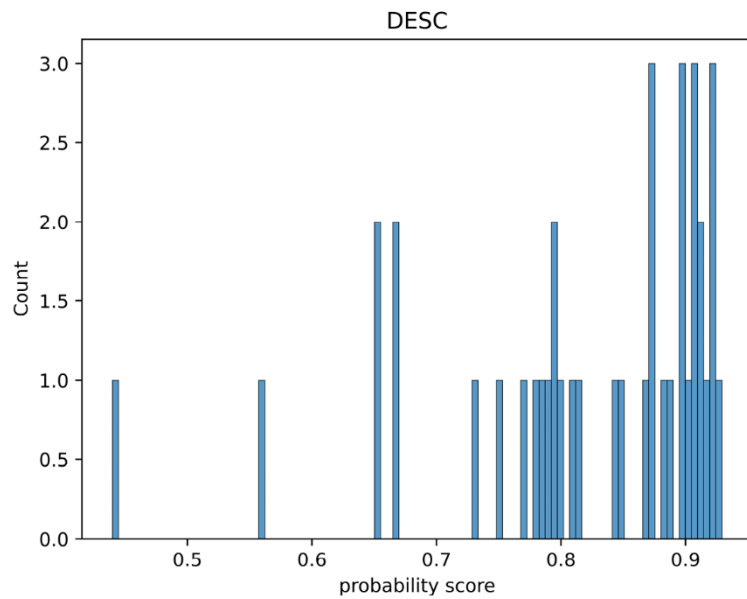






Chapter 14: What to Do If the System Isn't Working





generate 20 requests to find local restaurants of different cuisines and price ranges



1. Find local Italian restaurants with a moderate price range.
2. Search for nearby sushi restaurants with a high-end price range.
3. Discover Mexican restaurants in the area with an affordable price range.
4. Locate Indian restaurants nearby with a mid-range price.
5. Find Chinese restaurants in the vicinity with a budget-friendly price range.
6. Search for Thai restaurants in the area with a high-end price range.



generate 20 requests to find local restaurants of different cuisines and price ranges in an informal style

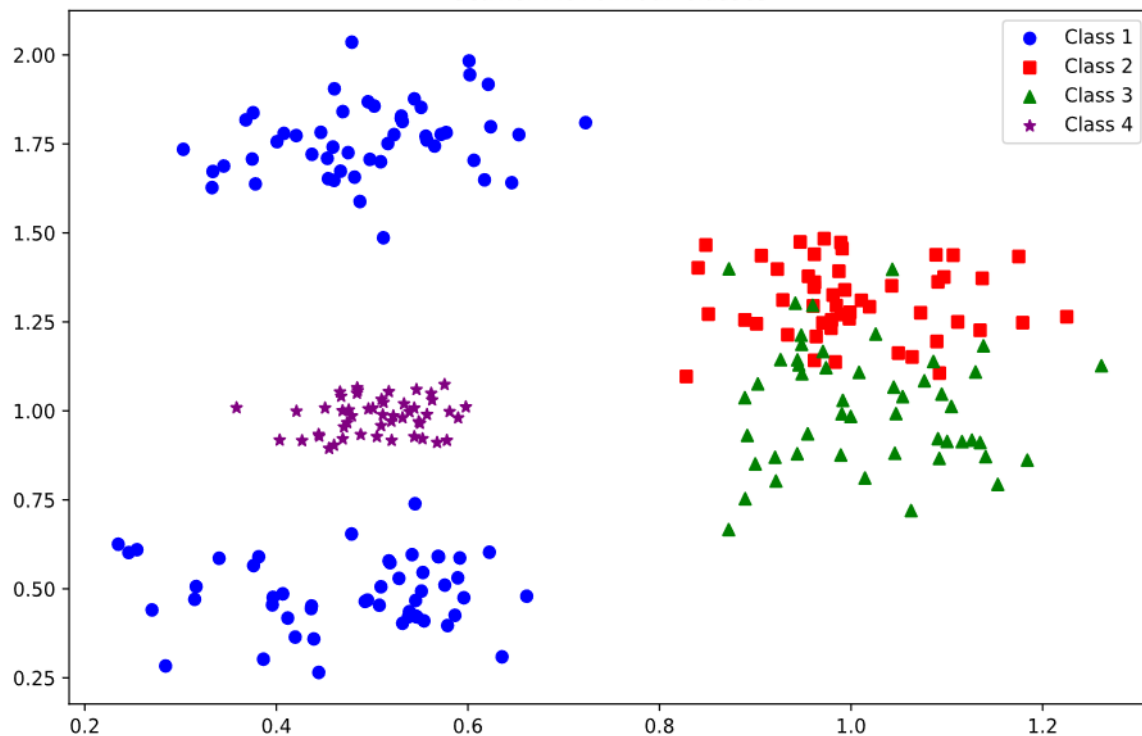


1. Yo, where can I find some dope Italian spots that won't break the bank?
2. I'm craving sushi like crazy! Any bougie joints in the area that I should hit up?
3. Dude, I need my Mexican food fix. Any affordable Mexican places around here?
4. Alright, I'm in the mood for some spicy Indian flavors. Any mid-range Indian restaurants nearby?
5. Chinese takeout sounds bomb right now. Hook me up with some budget-friendly Chinese spots, bro.
6. Craving some Thai food that's worth splurging on. Know any fancy Thai joints around here?

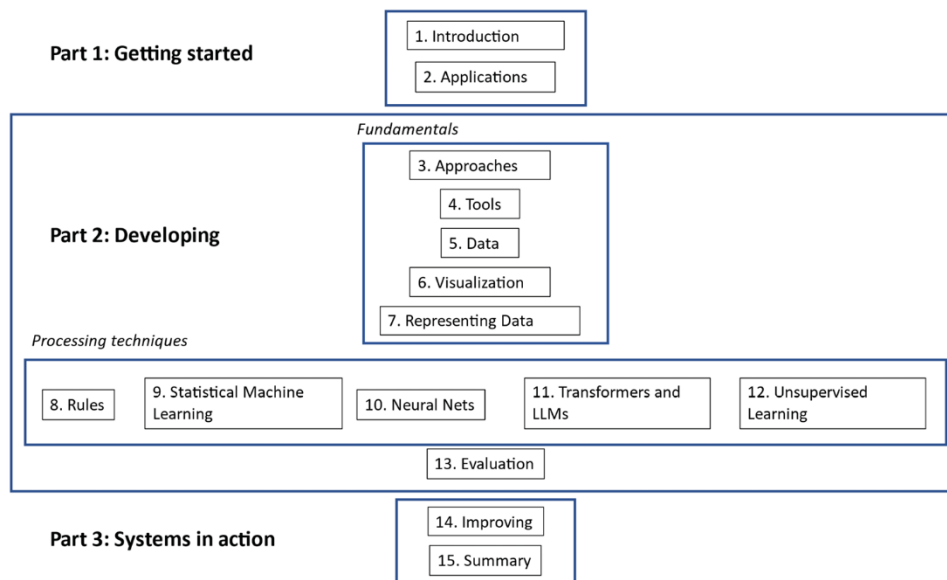
```
1 import os
2 import openai
3 openai.api_key = OPENAIKEY
4
5 completion = openai.ChatCompletion.create(
6     model="gpt-3.5-turbo",
7     temperature=1.5,
8     messages=[
9         {"role": "user", "content": "generate 6 ways of asking for local restaurants of different cuisines and price ranges"}
10    ]
11 )
12
13 result = completion.choices[0].message
14 result_content = result.get("content")
15 print(result_content)
16
```

1. "Can you recommend a good Italian restaurant that's not too expensive around here?"
2. "Are there any Thai restaurants that you would suggest trying in this area?"
3. "Could you point us in the direction of a steakhouse in town that won't break the bank?"
4. "We're looking for a Mexican restaurant with authentic cuisine. Any suggestions?"
5. "Do you know of any seafood restaurants that are a reasonable price nearby?"
6. "Where would you recommend going for some upscale dining? Possibly French cuisine."

Scatter Plot of Four Classes



Chapter 15: Summary and Looking to the Future



who is the current monarch of the united kingdom



As of my last knowledge update in September 2021, the current monarch of the United Kingdom is Queen Elizabeth II. She ascended to the throne on February 6, 1952, following the death of her father, King George VI. Queen Elizabeth II is the longest-reigning monarch in British history and has been serving as the queen for over six decades. However, please note that this information may have changed since then, so I recommend checking reliable sources or news outlets for the most up-to-date information on the current monarch of the United Kingdom.



To a Skylark

By Percy Bysshe Shelley

Hail to thee, blithe Spirit!
Bird thou never wert,
That from Heaven, or near it,
Pourest thy full heart
In profuse strains of unpremeditated art.

Higher still and higher
From the earth thou springest
Like a cloud of fire;
The blue deep thou wingest,
And singing still dost soar, and soaring ever singest.



describe watching a skylark fly up into the sky in a poetic manner in 50 words



In the boundless sky, a skylark takes flight,
Its melody transcending the realm of sight.
With grace and freedom, it ascends on high,
A symbol of dreams reaching for the sky.

