



---

# Natural Language Processing



---

# Natural Language Processing

---

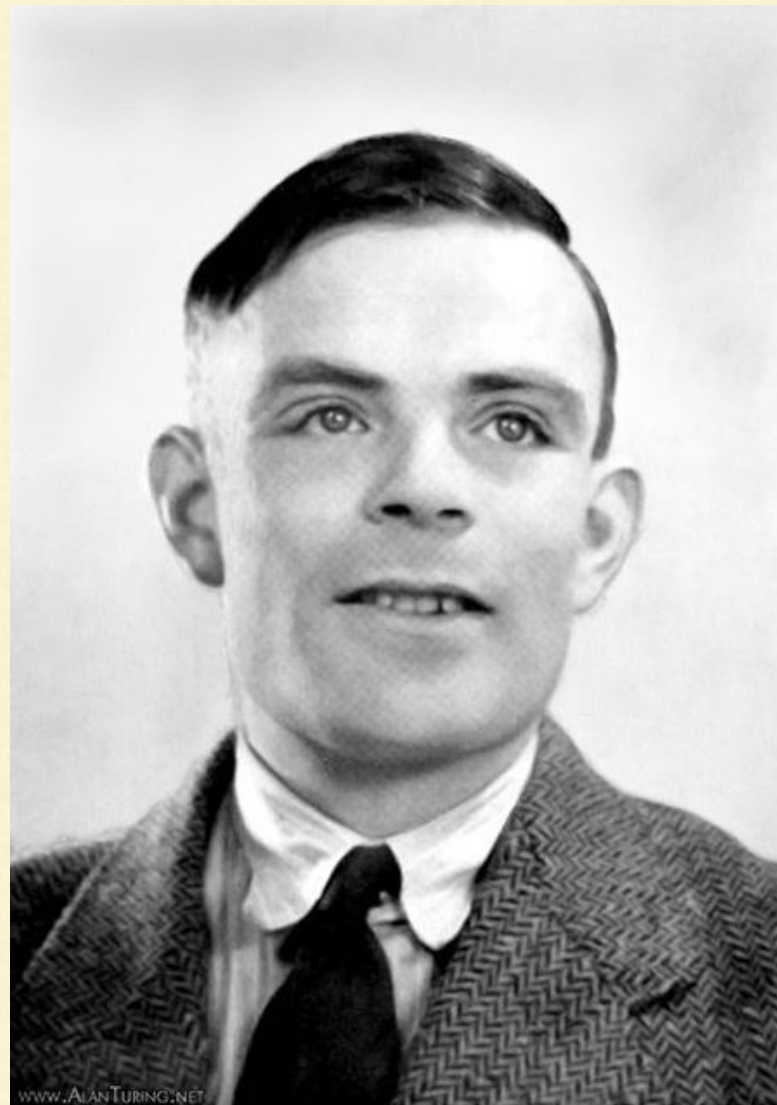
- Natural-language processing (NLP) is an area of computer science and artificial intelligence concerned with the interactions between **computers** and **human languages**
- In particular how to program computers to fruitfully process large amounts of natural language data

---

# Natural Language Processing

---

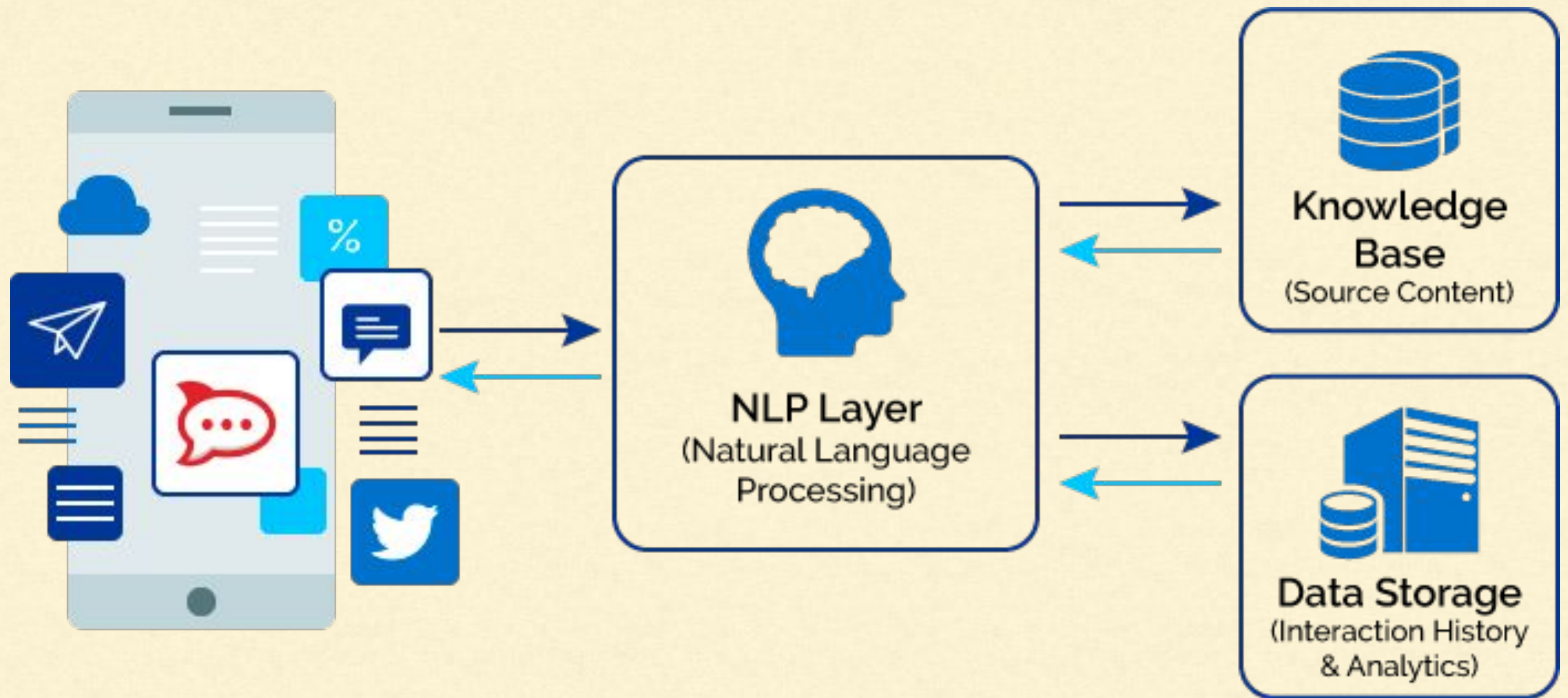
In 1950, Alan Turing published an article titled "Computing Machinery and Intelligence" which proposed what is now called the Turing test as a criterion of intelligence.





# Natural Language Processing

Basic Structure of a NLP application( chatbot considered below )



# Natural Language Processing



**Knowledge  
Base**  
(Source Content)

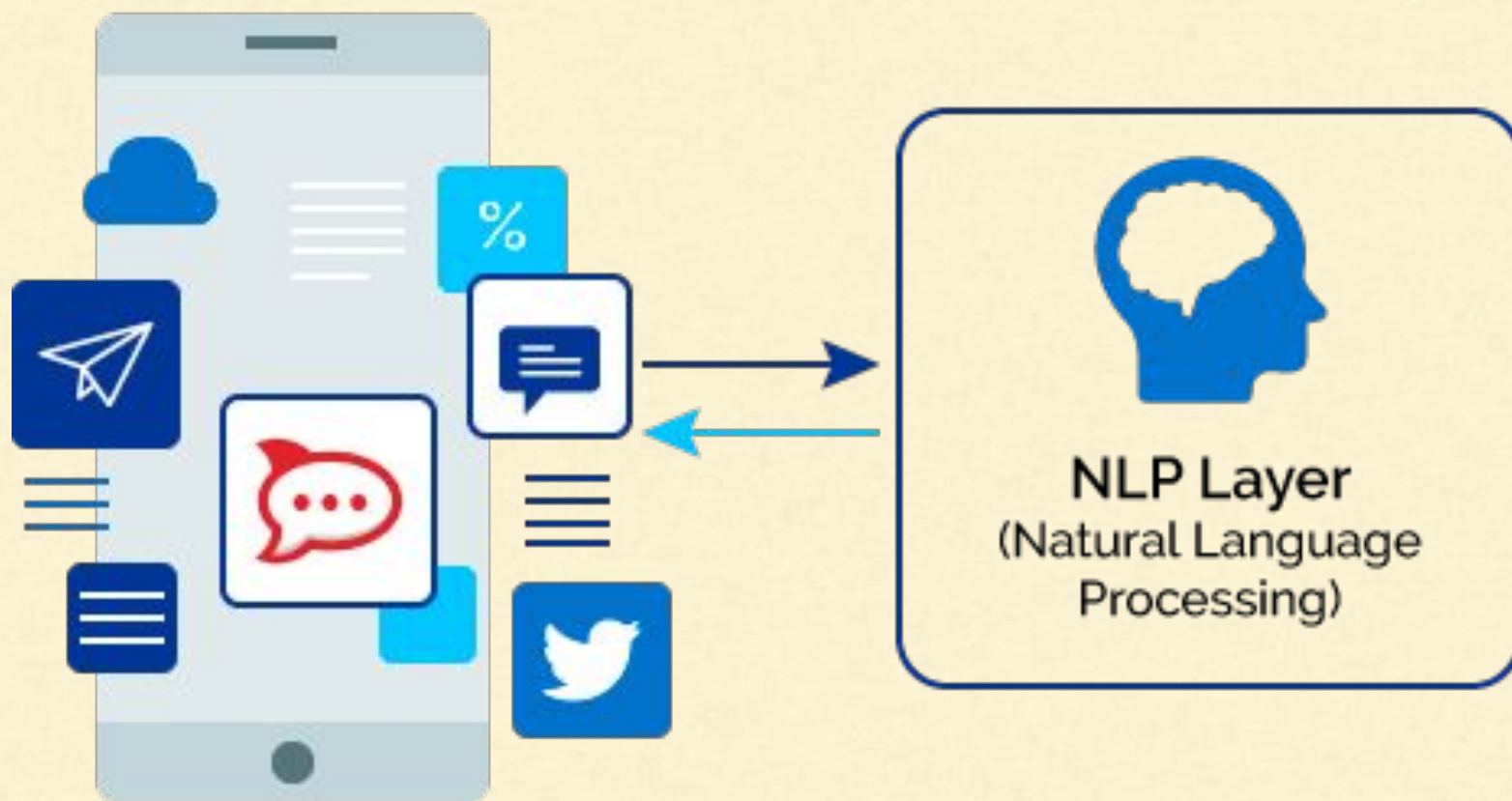
**Knowledge Base** – It contains the database of information that is used to equip chatbots with the information needed to respond to queries of customers request.



**Data Storage**  
(Interaction History  
& Analytics)

**Data Store** – It contains interaction history of chatbot with users.

# Natural Language Processing



**NLP Layer** – It translates users queries (free form) into information that can be used for appropriate responses.

**Application Layer** – It is the application interface that is used to interact with the user



# Natural Language Processing - Applications

- **Speech Recognition** - The task of speech recognition is to map an acoustic signal containing a spoken natural language utterance into the corresponding sequence of words intended by the speaker.



---

# Natural Language Processing - Applications

---

- **Text Classification** - Given an example of text, predict a predefined class label





# Natural Language Processing - Applications

- **Caption Generation** - It is the problem of describing the contents of an image.



↑ a living room with a couch and a television



↑ a man riding a bike on a beach

# Natural Language Processing - Applications

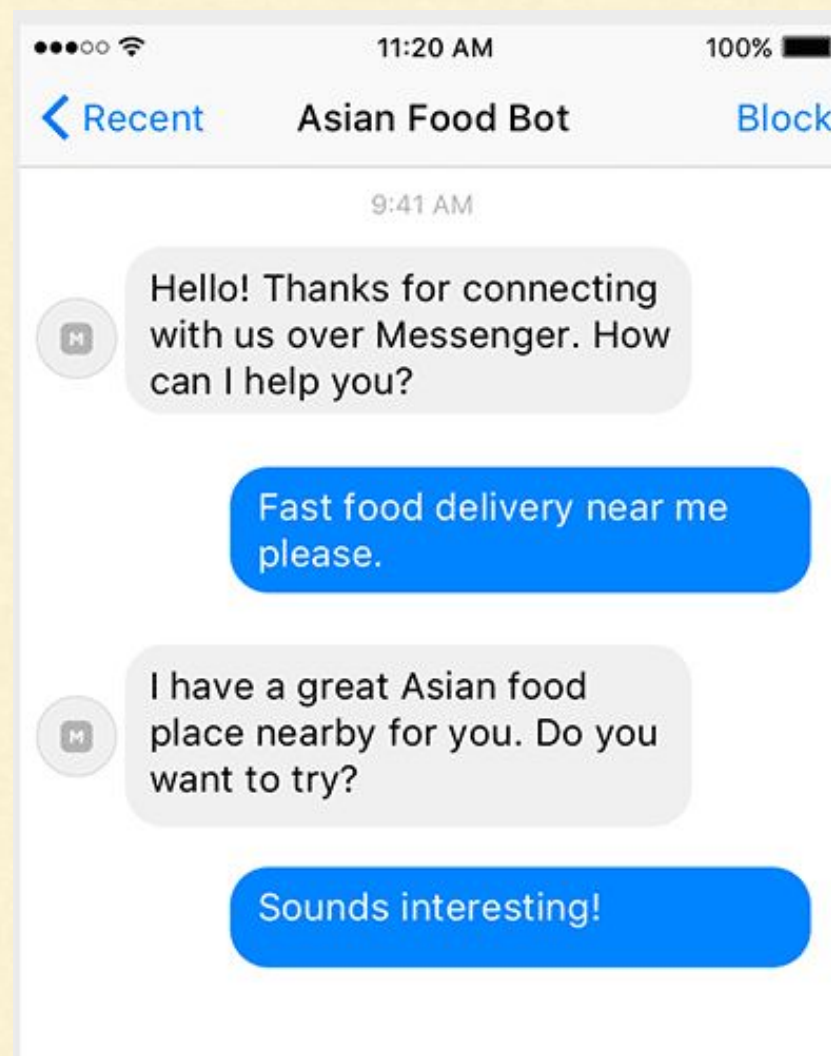
- **Machine Translation** - Machine translation is the problem of converting a source text in one language to another language.





# Natural Language Processing - Applications

- **Question Answering** - It is the problem where given a subject, such as a document of text, answer a specific question about the subject.





---

# Natural Language Processing - Tools

---

The most popular Natural Language Processing Tools are:

- [Stanford's Core NLP Suite](#)
- [Natural Language Toolkit](#)
- [Apache Lucene and Solr](#)
- [Apache OpenNLP](#)
- [Text Blob which is a wrapper over the NLTK library](#)

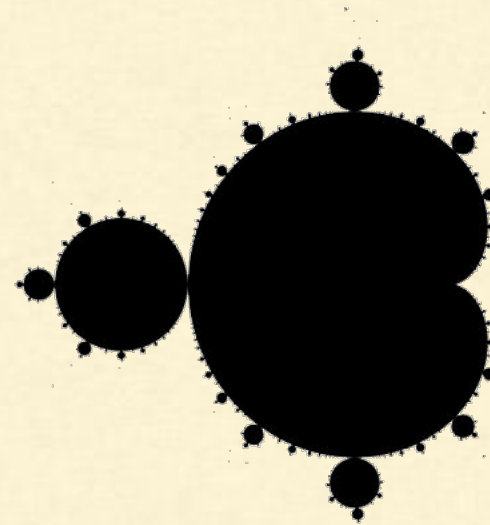
---

# Natural Language Processing - Tools

---

Let us use the TextBlob library of Python to Build a program that makes a quiz out of a provided text.

It is basically a usage of NER - [Named-entity recognition](#)



TextBlob

---

# Natural Language Processing - TextBlob

---

Let us begin by importing TextBlob and then selecting a text.

```
>>> from textblob import TextBlob
```

Now you can either load a text from a file as

```
>>> f = open('filename.txt')  
>>> text = f.read()
```

Or assign the text to a variable as

```
>>> text = "World War II (often abbreviated to  
WWII or WW2), also known as the Second World War,  
was a .... "
```



---

# Natural Language Processing - TextBlob

---

Next we'll convert our text to a TextBlob object.

```
>>> text = TextBlob(text)
```

Now we are ready to apply different methods on our text.

---

# Natural Language Processing - TextBlob

---

Let us understand a few things about the TextBlob api -

**text.sentences** - gives the sentences in a text

**sentences.tags** - gives the tags for each of the word in sentence.  
It returns a list of tuples with the word being the first element of the tuple and the tag being the second.

---

# Natural Language Processing - TextBlob

---

Now to generate our quiz

- We will extract each sentence
- We will replace all the nouns and proper nouns with a blank from each sentences.
- To make it easy we will remove only after the fourth word in the sentence.



---

# Natural Language Processing - TextBlob

---

```
>>> ww2b = TextBlob(ww2)
>>> for sentence in ww2b.sentences:
    new_sentence = sentence
    for index, tag in enumerate(sentence.tags):
        if tag[1] in ('NN', 'NNP') and index > 3:
            new_sentence =
new_sentence.replace(tag[0], "_____")
print(new_sentence)
print("\n===== \n")
```

**Run it on Notebook**

---

# Natural Language Processing - Tools

---

**Let's write a program to find Related Posts using Python's  
Scikit Learn**



---

# Natural Language Processing - Tools

---

We are given the task of finding the most related posts from a bunch of posts.

**The tricky thing that we have to tackle first is how to turn text into something on which we can calculate similarity ??**



---

# Natural Language Processing - Tools

---

## How to do it ??

### Bag of Words Approach -

It totally ignores the order of words and simply uses word counts as their basis.

In this model, a text, such as a sentence or a document is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.

the dog is on the table

0	0	1	1	0	1	1	1
are	cat	dog	is	now	on	table	the

---

# Natural Language Processing - Tools

---

## Vectorization

- For each word in the post, its occurrence is counted and noted in a vector.
- This step is also called **vectorization**.
- The vector is typically huge as it contains as many elements as words occur in the whole dataset.

---

# Natural Language Processing - Tools

---

## Vectorization - Example

For the two statements - "How to format my hard disk" and " Hard disk format problems " the vectors are shown below

Word	Occurence in post 1	Occurence in post 2
disk	1	1
format	1	1
how	1	0
hard	1	1
my	1	0
problems	0	1
to	1	0

**aka Term Document Matrix**



---

# Natural Language Processing - Tools

---

## Vectorization - Using Scikit learn

```
>>> from sklearn.feature_extraction.text import  
CountVectorizer
```

```
>>> vectorizer = CountVectorizer(min_df=1)
```

The min\_df parameter determines how CountVectorizer treats seldom words

- If it is set to an integer, all words occurring less than that value will be dropped
- If it is a fraction, all words that occur in less than that fraction of the overall dataset will be dropped.

---

# Natural Language Processing - Tools

---

## Vectorization - Using Scikit learn

```
>>> content = ["How to format my hard disk", " Hard  
disk format problems "]
```

```
>>> X = vectorizer.fit_transform(content)
```

```
>>> vectorizer.get_feature_names()
```

```
['disk', 'format', 'hard', 'how', 'my', 'problems',  
'to']
```

**Run it on Notebook**

---

# Natural Language Processing - Tools

---

## Vectorization - Using Scikit learn

```
>>> print(X.toarray().transpose())
```

```
[[1 1]
 [1 1]
 [1 1]
 [1 0]
 [1 0]
 [0 1]
 [1 0]]
```

This means that the first sentence contains all the words except "problems", while the second contains all but "how", "my", and "to".



---

# Natural Language Processing - Tools

---

## Finding Distance

We can measure distance between two vectors using the Euclidean Distance.

But first we will normalize each vectors.

The `scipy.linalg` module provides a function called `norm`.

The **`norm()`** function calculates the Euclidean norm (shortest distance)

---

# Natural Language Processing - Tools

---

## Finding Distance

```
>>> def dist_norm(v1, v2):  
    v1_normalized = v1/sp.linalg.norm(v1.toarray())  
    v2_normalized = v2/sp.linalg.norm(v2.toarray())  
    delta = v1_normalized - v2_normalized  
    return sp.linalg.norm(delta.toarray())
```

---

# Natural Language Processing - Tools

---

## Applying Everything we learnt on a toy dataset

Now we will consider 5 toy posts and find the similarity with a given post.

```
>>> post1 = "This is a toy post about machine learning.  
Actually, it contains not much interesting stuff."  
>>> post2 = "Imaging databases can get huge."  
>>> post3 = "Most imaging databases save images  
permanently."  
>>> post4 = "Imaging databases store images."  
>>> post5 = "Imaging databases store images. Imaging  
databases store images. Imaging databases store images."
```



---

# Natural Language Processing - Tools

---

## Applying Everything we learnt on a toy dataset

Now we will build our vectorizer

```
>>> posts = [post1, post2, post3, post4, post5]
>>> X_train = vectorizer.fit_transform(posts)
>>> num_samples, num_features = X_train.shape
>>> print("#samples: %d, #features: %d" %
(num_samples, num_features))
```

```
#samples: 5, #features: 24
```

As we provided 5 different posts and there are 24 different words in them.

---

# Natural Language Processing - Tools

---

## **Applying Everything we learnt on a toy dataset**

Finally we will iterate through all the vectors of the posts and find their distance with the new post.

**Perform on Notebook**

---

# Natural Language Processing

---

**Now let us analyse a collection of text documents using Scikit Learn**





---

# Natural Language Processing

---

**In this section we will see how to:**

- Load the file contents and the categories
- Extract feature vectors suitable for machine learning
- Train a linear model to perform categorization
- Use a grid search strategy to find a good configuration of both the feature extraction components and the classifier

---

# Natural Language Processing

---

## Loading the 20 newsgroups dataset

To load the dataset use the code-

```
>>> categories = ['alt.atheism',  
'soc.religion.christian', 'comp.graphics', 'sci.med']  
>>> from sklearn.datasets import fetch_20newsgroups  
>>> twenty_train = fetch_20newsgroups(subset='train',  
categories=categories, shuffle=True, random_state=42)
```

---

# Natural Language Processing

---

## Loading the 20 newsgroups dataset

To load the dataset use the code-

```
>>> categories = ['alt.atheism',  
'soc.religion.christian', 'comp.graphics', 'sci.med']  
>>> from sklearn.datasets import fetch_20newsgroups  
>>> twenty_train = fetch_20newsgroups(subset='train',  
categories=categories, shuffle=True, random_state=42)
```



---

# Natural Language Processing

---

## Analysing our dataset

The **target\_names** holds the list of the requested category names:

```
>>> twenty_train.target_names  
['alt.atheism', 'comp.graphics', 'sci.med',  
'soc.religion.christian']
```

The files themselves are loaded in memory in the data attribute

```
>>> len(twenty_train.data)  
2257  
  
>>> len(twenty_train filenames)  
2257
```

---

# Natural Language Processing

---

## Analysing our dataset

Content of the first lines of the first loaded file

```
>>> print("\n".join(twenty_train.data[0].split("\n")[:3]))  
From: sd345@city.ac.uk (Michael Collier)  
Subject: Converting images to HP LaserJet III?  
Nntp-Posting-Host: hampton
```

The category integer id of each sample is stored in the target attribute

```
>>> twenty_train.target[:10]  
array([1, 1, 3, 3, 3, 3, 3, 2, 2, 2])
```

---

# Natural Language Processing

---

**Now we will apply the bag of word approach**

Tokenizing text with scikit-learn

```
>>> from sklearn.feature_extraction.text import  
CountVectorizer  
  
>>> count_vect = CountVectorizer()  
  
>>> X_train_counts =  
count_vect.fit_transform(twenty_train.data)  
  
>>> X_train_counts.shape  
(2257, 35788)
```



---

# Natural Language Processing

---

Occurrence count is a good start but there is an issue -

- longer documents will have higher average count values than shorter documents

To avoid these potential discrepancies we

- Divide the number of occurrences of each word in a document by the total number of words in the document: these new features are called **tf for Term Frequencies**.

---

# Natural Language Processing

---

**How can we improve tf ?**

Downscale weights for words that occur in many documents in the corpus and are therefore less informative than those that occur only in a smaller portion of the corpus.

This downscaling is called **tf-idf** for “**Term Frequency times Inverse Document Frequency**”.

---

# Natural Language Processing - Tf-idf

---

- Tf-idf stands for **term frequency-inverse document frequency**
- Tf-idf weight is often used in
  - Information retrieval and
  - Text mining
- This weight is a used to evaluate
  - How important a word is to a
  - Document in a collection or corpus



---

# Natural Language Processing - Tf

---

## Term Frequency

- Measures how frequently a term occurs in a document
- It is possible that a term would appear
  - Much more times in long documents than shorter ones
  - This is why we normalize TF

**$TF(t) = (\text{Number of times term } t \text{ appears in a document}) / (\text{Total number of terms in the document})$**

---

# Natural Language Processing - IDF

---

## Inverse Document Frequency

- Measures how important a term is
- In TF, all terms are considered equally important
- However some words and stop words appear a lot of time
  - But have least importance
- In IDF we weight down frequent terms
  - And scale up rare terms

$$\text{IDF}(t) = \log_e(\text{Total number of documents} / \text{Number of documents with term } t \text{ in it})$$

---

# Natural Language Processing - Tf-idf

---

## Example

- Consider a document containing 100 words and the word cat appears 3 times
- The term frequency(tf) for cat is
  - $(3 / 100) = 0.03$



---

# Natural Language Processing - Tf-idf

---

## Example

- Now, assume we have 10 million documents and
  - The word cat appears in 1,000 of these
- The inverse document frequency(idf) is
  - $\log(10,000,000 / 1,000) = 4$
- Tf-idf weight is the product of tf and idf
  - $0.03 * 4 = 0.12$

---

# Natural Language Processing

---

**Now let us apply tf-idf to our example**

```
tfidf_transformer = TfidfTransformer()  
  
>>> X_train_tfidf =  
tfidf_transformer.fit_transform(X_train_counts)  
  
>>> X_train_tfidf.shape  
(2257, 35788)
```

---

# Natural Language Processing

---

## Training a classifier

We'll start with a **naïve Bayes classifier**, which provides a nice baseline for this task.

```
>>> from sklearn.naive_bayes import MultinomialNB  
>>> clf = MultinomialNB().fit(X_train_tfidf,  
twenty_train.target)
```

The multinomial variant of Naive Bayes is one the most suitable for word counts tasks.



---

# Natural Language Processing

---

## **Now let us make a prediction on a new document**

To try to predict the outcome on a new document we need to extract the features using almost the same feature extracting chain as before

- We will first transform the new document to count vectors
- Then we'll transform it with the `tfidf_transformer`.
- Finally we'll call the `predict` method of the classifier

---

# Natural Language Processing

---

```
>>> docs_new = ['God is love', 'OpenGL on the GPU is fast']
>>> X_new_counts = count_vect.transform(docs_new)
>>> X_new_tfidf = tfidf_transformer.transform(X_new_counts)

>>> predicted = clf.predict(X_new_tfidf)

>>> for doc, category in zip(docs_new, predicted):
...     print('%r => %s' % (doc, twenty_train.target_names[category]))
...
'God is love' => soc.religion.christian
'OpenGL on the GPU is fast' => comp.graphics
```

**Run it on Notebook**

---

# Natural Language Processing

---

**Now let us combine all the steps in the form of a pipeline**

```
>>> from sklearn.pipeline import Pipeline
>>> text_clf = Pipeline([('vect', CountVectorizer()),
                        ('tfidf', TfidfTransformer()),
                        ('clf', MultinomialNB()),])
```

We can now train the model with a single command

```
>>> text_clf.fit(twenty_train.data, twenty_train.target)
Pipeline(...)
```



---

# Natural Language Processing

---

**Now let us perform performance evaluation on the test set**

```
>>> import numpy as np
>>> twenty_test = fetch_20newsgroups(subset='test',
...     categories=categories, shuffle=True,
random_state=42)
>>> docs_test = twenty_test.data
>>> predicted = text_clf.predict(docs_test)
>>> np.mean(predicted == twenty_test.target)

0.834...
```

I.e., we achieved 83.4% accuracy

---

# Natural Language Processing

---

**Now we'll use a different classifier and compute the performance metrics**

```
>>> from sklearn.linear_model import SGDClassifier
>>> text_clf = Pipeline([('vect', CountVectorizer()),
...                       ('tfidf', TfidfTransformer()),
...                       ('clf', SGDClassifier(loss='hinge', penalty='l2',
...                                             alpha=1e-3, random_state=42,
...                                             max_iter=5, tol=None)),
... ])
```

---

# Natural Language Processing

---

**Now we'll use a different classifier and compute the performance metrics**

```
>>> text_clf.fit(twenty_train.data, twenty_train.target)
```

```
Pipeline(...)
```

```
>>> predicted = text_clf.predict(docs_test)
```

```
>>> np.mean(predicted == twenty_test.target)
```

```
0.912...
```

---

# Natural Language Processing

---

## Parameter tuning using grid search

Since there are different parameters which we can choose, we'll apply grid search to find the best parameters

```
>>> parameters = {'vect__ngram_range': [(1, 1), (1, 2)],  
...               'tfidf__use_idf': (True, False),  
...               'clf__alpha': (1e-2, 1e-3),  
... }
```

Here we'll be applying grid search for the parameters - `ngram_range`, `use_idf` and `alpha`.



---

# Natural Language Processing

---

## Parameter tuning using grid search

If we have multiple CPU cores at our disposal, we can tell the grid searcher to try these eight parameter combinations in parallel with the `n_jobs` parameter.

```
>>> gs_clf = GridSearchCV(text_clf, parameters,  
n_jobs=-1)  
  
>>> gs_clf = gs_clf.fit(twenty_train.data[:400],  
twenty_train.target[:400])
```

---

# Natural Language Processing

---

## Predicting and finding best score

```
>>> twenty_train.target_names[gs_clf.predict(['God is  
love'])[0]]  
'soc.religion.christian'  
>>> gs_clf.best_score_  
0.900...
```

---

# Natural Language Processing

---

## Predicting and finding best score

```
>>> for param_name in sorted(parameters.keys()):  
...     print("%s: %r" % (param_name,  
gs_clf.best_params_[param_name]))  
  
...  
clf__alpha: 0.001  
tfidf__use_idf: True  
vect__ngram_range: (1, 1)
```

---

# Natural Language Processing - Tools

---

## Overview of Stanford Core NLP





---

# Natural Language Processing - Stanford NLP

---

- Stanford CoreNLP provides a set of human language technology tools.
- It can give
  - The base forms of words,
  - Their parts of speech,
  - Whether they are names of companies, people, etc.,
  - Mark up the structure of sentences in terms of phrases and syntactic dependencies,
  - Indicate which noun phrases refer to the same entities, indicate sentiment

---

# Natural Language Processing - Stanford NLP

---

## **Choose Stanford CoreNLP if you need:**

- An integrated NLP toolkit with a broad range of grammatical analysis tools
- A fast, robust annotator for arbitrary texts, widely used in production
- A modern, regularly updated package, with the overall highest quality text analytics

---

# Natural Language Processing - Stanford NLP

---

**Choose Stanford CoreNLP if you need:**

- Support for a number of major (human) languages
- Available APIs for most major modern programming languages
- Ability to run as a simple web service

---

# Natural Language Processing - Stanford NLP

---

## Programming languages and operating systems

Stanford CoreNLP is written in **Java**; recent releases require Java 1.8+.

You can interact with **CoreNLP** via the **command-line** or its **web service** using languages like Javascript, Python etc.



---

# Natural Language Processing - Stanford NLP

---

## Programming languages and operating systems

You can use **Stanford CoreNLP** from

- The command-line, via its original Java programmatic API,
- Via the object-oriented simple API,
- Via third party APIs for most major modern programming languages,  
Or via a web service.

It works on **Linux**, **macOS**, and **Windows**

---

# Natural Language Processing

---

## More coming up on Cloudu00d7Lab

- Word2vec - Vector Representations of Words
- Deep Learning - LSTM - Long Short-Term Memory
- GloVe - Global Vectors for Word Representation
- spaCY - Industrial-Strength Natural Language Processing in Python
- Hands-on using Stanford CoreNLP
- List of APIs available for chatbots etc

# Thank You

---

<https://discuss.cloudxlab.com>

[reachus@cloudxlab.com](mailto:reachus@cloudxlab.com)

