Agenda of course

Intro to NLP with ML techniques

Deep Learning

NLP with Deep Learning

Intro to NLP

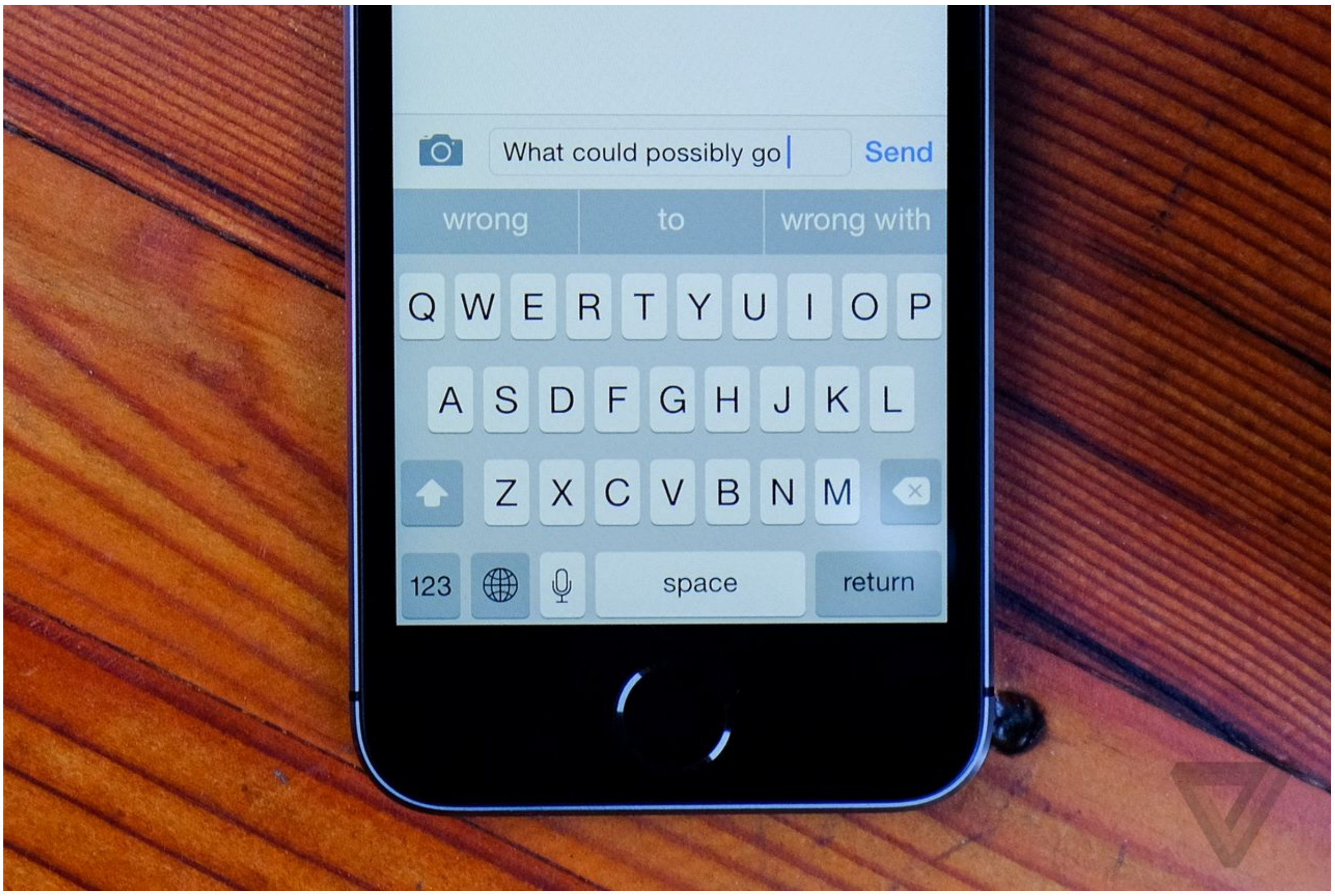
What is NLP

NLP is the study of various algorithms to analyze and understand natural language such as English, Hindi etc.

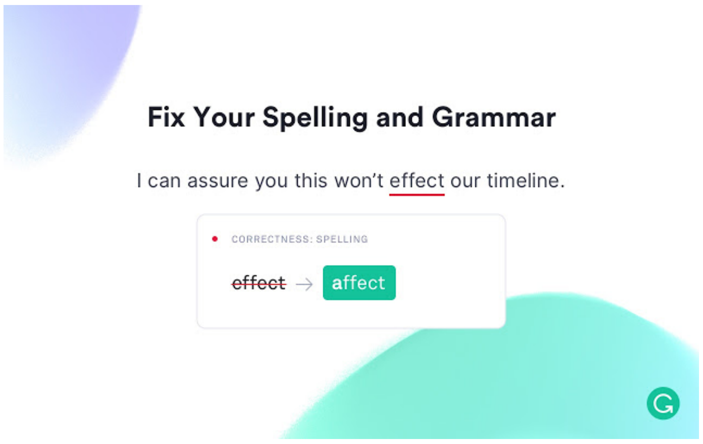
We use NLP in our day-to-day life

Voice-assistants and chat-bots

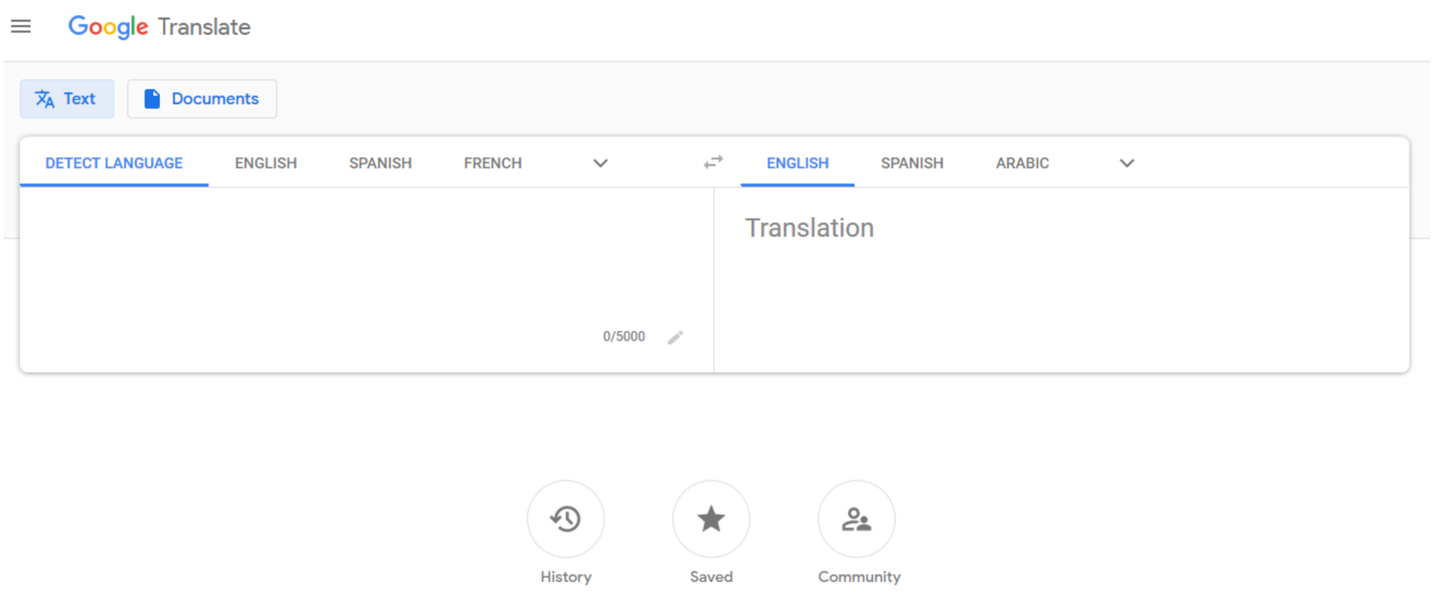
Autocomplete feature on android and gmail



Spelling and grammar checking


Fix Your Spelling and Grammar
I can assure you this won't effect our timeline.
CORRECTNESS: SPELLING
~~effect~~ → affect

Language Translation



NLP algorithms

1. Rule-based :- Hand-crafted rules by humans like using multiple if-else loops
2. ML based
3. DL based

Text classification

Task of classifying a piece of text into a particular category

1. Classifying a text as spam or not like gmail does automatically
2. Finding sentiment of the review

But before doing this

All ML algorithms deal with **numbers**, they don't know how to deal with text

So we need a way to map the text to numbers which can be understood by algorithms

Also there is a lot of unwanted text that don't contribute to the task of text classification

This is where **preprocessing** comes in

Lets discuss few things to be done before sending to algorithms

Tokenization

Tokenization describes splitting paragraphs into sentences, or sentences into individual words.

```
IN:"He did not try to navigate after the first bold flight, for the reaction had taken something out of his soul.

"OUT:['He', 'did', 'not', 'try', 'to', 'navigate', 'after', 'the', 'first', 'bold', 'flight', ',', 'for', 'the', 'reaction', 'had', 'taken', 'something', 'out', 'of', 'his', 'soul', '.']
```

One hot encoding

Convert text to numbers

50000 unique words

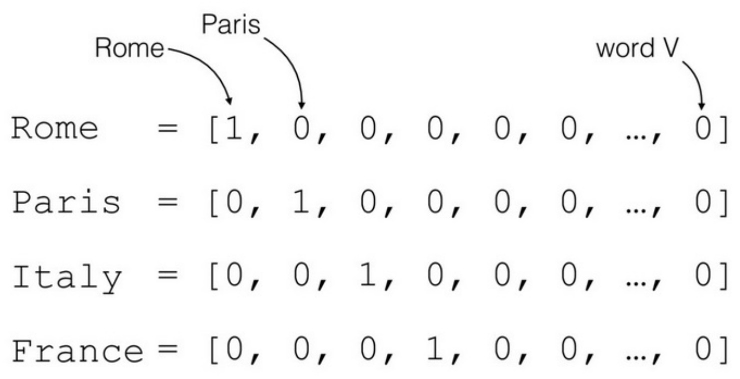
Hello, Hii, How, you

Hello - [1, 0, 0, 0] 1

Hii - [0, 1, 0, 0] 2

How - [0, 0, 1, 0] 3

You - [0, 0, 0, 1] 4



```
Rome   = [1, 0, 0, 0, 0, 0, …, 0]
Paris  = [0, 1, 0, 0, 0, 0, …, 0]
Italy  = [0, 0, 1, 0, 0, 0, …, 0]
France = [0, 0, 0, 1, 0, 0, …, 0]
```

Capitalization

Convert all words to smaller ones to have a lesser vocabulary

hello , how, you , world

Punctuation removal

Remove unnecessary punctuation which might not add any value to our task

Stopword removal

A majority of the words in a given text are connecting parts of a sentence rather than showing subjects, objects or intent. Word like “the” or “and” can be removed by comparing text to a list of stopword.

The sandwich is tasting very good

sandwich tasting good

```
 IN:
['He', 'did', 'not', 'try', 'to', 'navigate', 'after', 'the', 'first', 'bold', 'flight', ',', 'for', 'the', 'reaction', 'had', 'taken', 'something', 'out', 'of', 'his', 'soul', '.']

OUT:
['try', 'navigate', 'first', 'bold', 'flight', ',', 'reaction', 'taken', 'something', 'soul', '.']
```

Stemming

Much of natural language machine learning is about sentiment of the text. Stemming is a process where words are reduced to a root by removing inflection through dropping unnecessary characters, usually a suffix. There are several stemming models, including Porter and Snowball. The results can be used to identify relationships and commonalities across large datasets.

go going gone -> go

fumble fumbling -> fumbl

```
 IN:
["It never once occurred to me that the fumbling might be a mere mistake."]

OUT:
['it', 'never',  'onc',  'occur',  'to',  'me',  'that',  'the', 'fumbl',  'might', 'be', 'a', 'mere',  'mistake.'],
```

It is easy to see where reductions may produce a “root” word that isn’t an actual word. This doesn’t necessarily adversely affect its efficiency, but there is a danger of “overstemming” were words like “universe” and “university” are reduced to the same root of “univers”.

Lemmazation

Lemmazation is an alternative approach from stemming to removing inflection. By determining the part of speech and utilizing WordNet’s lexical database of English, lemmazation can get better results.

```
 The stemmed form of leafs is: lea
The stemmed form of leaves is: lea

The lemmatized form of leafs is: leaf
The lemmatized form of leaves is: leaf
```

Lemmazation is a more intensive and therefor slower process, but more accurate. Stemming may be more useful in queries for databases whereas lemmazation may work much better when trying to determine text sentiment.

Bag of Words

Represent a sentence using count of words

Sentences

Hello World

Hello Jessy

Hello Jessy Hello World

Vocabulary

Hello, World, Jessy

One-hot encoding

Hello - [1, 0, 0]

World - [0, 1, 0]

Jessy - [0, 0, 1]

Bag of words

Hello World - [1, 1, 0]

Hello Jessy - [1, 0, 1]

Hello Jessy Hello World - [2, 1, 1]

Now do same for

1.Good god

2. God is great

Vocabulary

good, god, is, great

good - [1, 0, 0, 0]

god - [0, 1, 0, 0]

is - [0, 0, 1, 0]

great - [0, 0, 0, 1]

good god - [1, 1, 0, 0]

god is great - [0, 1, 1, 1]

food not good using browser -

food good not using browser +

n-grams

Sometimes using a single word as indicator might not be enough

For example good might indicate positive sentiment but if you keep not before that the sentiment revereses

Since bag-of-words loses the structure of words we might need to capture more than 1-word phrases as features as well

These are called n-grams

So for below sentences

Sentences

Hello World

Hello Jessy

Hello Jessy Hello World

Hello, World, Jessy - 1 grams

2-grams will be

Hello, World, Jessy - 1 grams

Hello World, Hello Jessy, Jessy Hello - 2 grams

Hello Jessy Hello, Jessy Hello World - 3 grams

8

Hello - [1, 0, 0, ,,,, 0]- 8

World - [0, 1, 0, ,,,,0] - 8

..

Hello World - [0,0,0,1,0,0,0,0] - 8

..

..

Hello Jessy Hello - [0,0,0,0,0,0,1,0] - 8

..

Hello World Hello Jessy - [2, 1, 1, 1, 1, 0, 0, 0]

Vocabulary is

Hello, World, Jessy, Hello World, Hello Jessy, Jessy Hello

Now do for 3-grams

TF-IDF Vectorizer

Count vectorizer may not be a good representation always

Say a word is continuously repeated everywhere then it might be just a usual word and even if it's count is high it shouldn't effect the model decision

To solve this we can replace count vector with a tf-idf vector

tf-idf stands for *Term frequency-inverse document frequency*

tf-idf weight is composed by two terms

1. Normalized Term Frequency (tf)
2. Inverse Document Frequency (idf)

tf indicates how many times the word is repeated in a document/sentence

Doc 1: Ben studies about computers in Computer Lab. **Doc 2:** Steve teaches at Brown University. **Doc 3:** Data Scientists work on large datasets.

DOC 1	BEN	STUDIES	COMPUTER	LAB
tf	1	1	2	1

Vector Space Representation for Doc 1 : **[1, 1, 2, 1]**

tf for document 2:

DOC 2	STEVE	TEACHES	BROWN	UNIVERSITY
tf	1	1	1	1

Vector Space Representation for Doc 2 : **[1, 1, 1, 1]**

tf for document 3:

DOC 3	DATA	SCIENTISTS	WORK	LARGE	DATASETS
tf	1	1	1	1	1

Vector Space Representation for Doc 3 : **[1, 1, 1, 1, 1]**

Since we are dealing with the term frequency which rely on the occurrence counts, thus, longer documents will be favoured more. To avoid this, normalize the **term frequency**

DOCUMENTS	||D||
1	7
2	5
3	6

Normalized TF for Document 1:

DOC1	BEN	STUDIES	COMPUTER	LAB
Normalized Tf	0.143	0.143	0.286	0.143

Vector Space Representation for Document 1 : **[0.143, 0.143, 0.286, 0.143]**

Normalized tf for document 2:

DOC 2	STEVE	TEACHES	BROWN	UNIVERSITY
NormalizedTf	0.2	0.2	0.2	0.2

Vector Space Representation for Document 2 : **[0.2, 0.2, 0.2, 0.2]**

Normalized tf for document 3:

DOC 3	DATA	SCIENTISTS	WORK	LARGE	DATASETS
NormalizedTf	0.167	0.167	0.167	0.167	0.167

Vector Space Representation for Document 3 : **[0.167, 0.167, 0.167, 0.167, 0.167]**

Inverse Document Frequency – idf

200

W1 - 180

W2 - 20

idf(W1) = log(200/180)

idf(W2) = log(200/20)

It typically measures how important a term is. The main purpose of doing a search is to find out relevant documents matching the query. Since `tf` considers all terms equally important, thus, we can't only use term frequencies to calculate the weight of a term in the document. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scaling up the rare ones. Logarithms helps us to solve this problem.

First of all, find the document frequency of a term t by counting the number of documents containing the term:

```
idf(t) = log(N/ df(t))
```

```
idf(computer) = log(Total Number Of Documents / Number Of Documents with term Computer in it)
```

```
 The term Computer appears in Document1

idf(computer) = log(3 / 1)
          = 1.5849
```

Given below is the **idf** for terms occurring in all the documents-

GIVEN	NO. OF DOCUMENTS IN WHICH TERM APPEARS(NT)	IDF = LOG(N/NT)
Ben	1	Log(3/1)=1.5849
Studies	1	Log(3/1)=1.5849
Computer	1	Log(3/1)=1.5849
Lab	1	Log(3/1)=1.5849
Steve	1	Log(3/1)=1.5849
Teaches	1	Log(3/1)=1.5849
Brown	1	Log(3/1)=1.5849
University	1	Log(3/1)=1.5849
Data	1	Log(3/1)=1.5849
Scientists	1	Log(3/1)=1.5849
Work	1	Log(3/1)=1.5849
Large	1	Log(3/1)=1.5849
Dataset	1	Log(3/1)=1.5849

```
tf-idf(t, d) = tf(t, d)* idf(t, d)
```