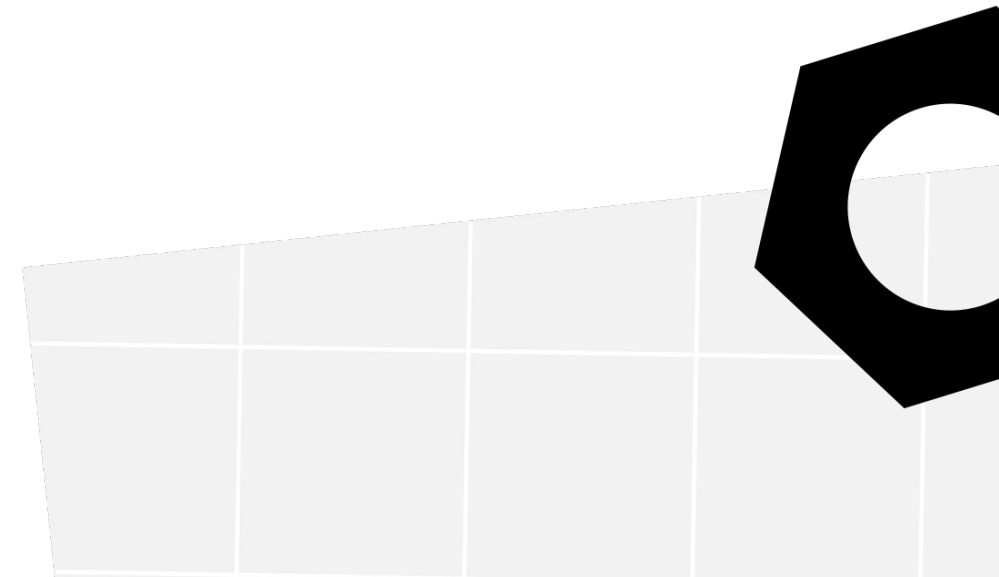




# Przetwarzanie języka naturalnego

**Kurs Data Science**





# Agenda



- Wprowadzenie do NLP
- Operacje na tekście
- Biblioteka Spacy, NLTK
- Tokenizacja
- Stemming
- Lematyzacja
- Klasyfikacja tekstu

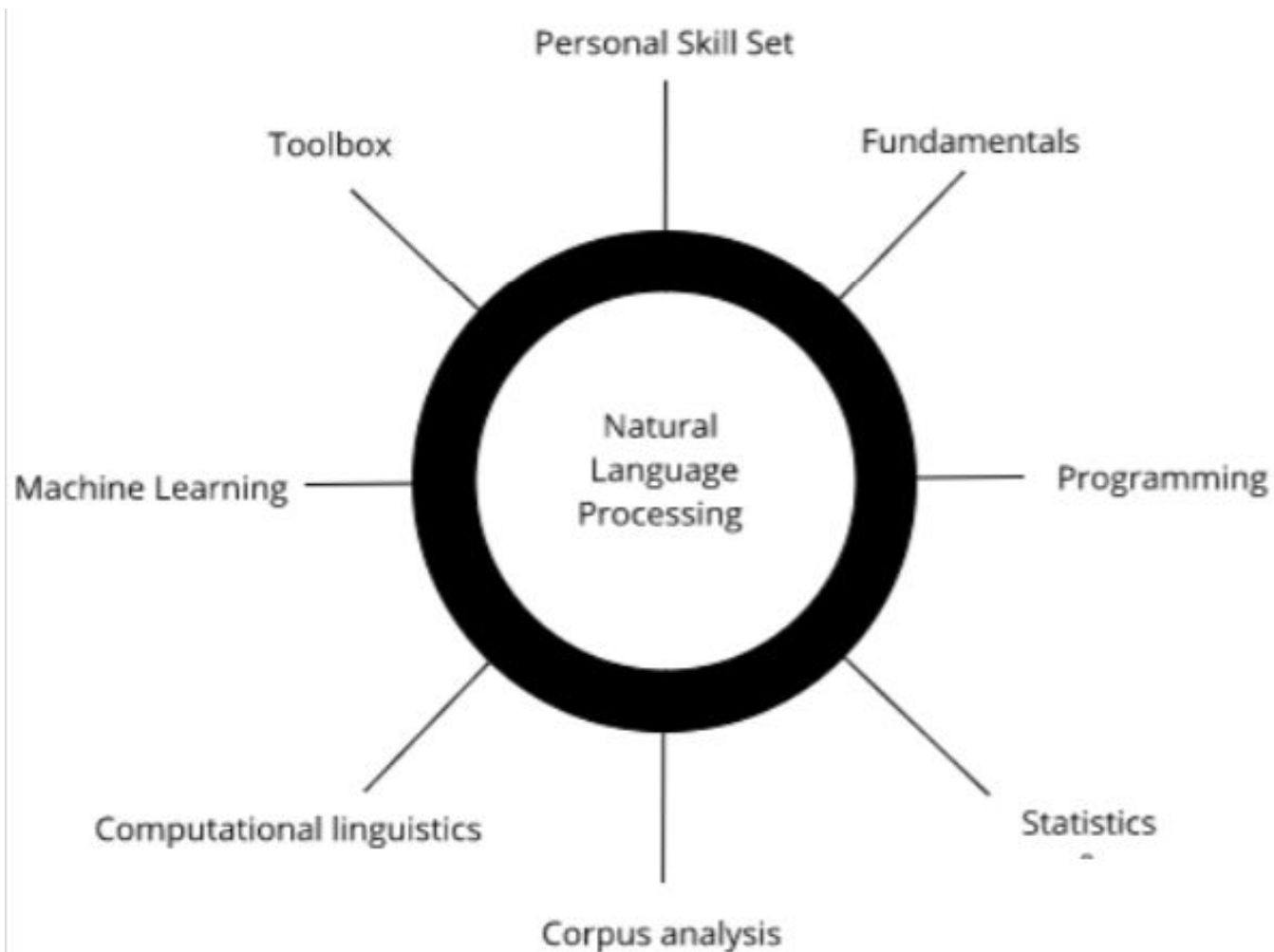
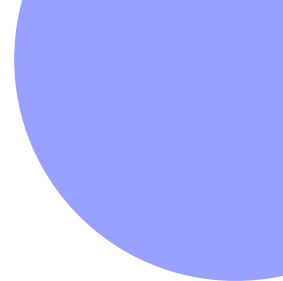
# Przetwarzanie języka naturalnego – NLP (ang. Natural Language Processing)



**Przetwarzanie języka naturalnego** (ang. natural language processing, NLP) – interdyscyplinarna dziedzina, łącząca zagadnienia sztucznej inteligencji i językoznawstwa, zajmująca się automatyzacją analizy, rozumienia, tłumaczenia i generowania **języka naturalnego** przez komputer.



# Koncepcja NLP





# Dane i problemy



- **Dane:**

- Korpusy językowe
- Internet (facebook, wikipedia, twitter etc.)

- **Problemy:**

- Machine translation
- Text summarization
- Question answering
- Sentiment analysis



# Podstawowe podejścia



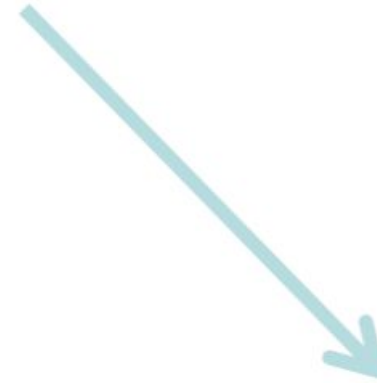
Dwa podstawowe podejścia




## **Analiza formalna**

- Bazy wiedzy
- Drzewa leksykalne
- Rozumienie tekstu

+



## **Analiza statystyczna**

- Etykietowane dane
  - Zliczanie
  - Uczenie maszynowe
- 



# Najpopularniejsze narzędzia dla języka angielskiego



Spacy

NLTK

Stanford  
CoreNLP (dla  
Javy!)



# Najpopularniejsze narzędzia dla języka polskiego



- Spacy
- Zespół Inżynierii Lingwistycznej PAN:
  - Ciekawe rozwiązania: CLIP czy CLARIN-PL
- Grupa technologii językowych Politechniki Wrocławskiej





# Zasoby językowe

## Korpusy

- [Narodowy Korpus Języka Polskiego](#)
- [British National Corpus](#)

## Ontologie:

- [Wikipedia](#)



# NLTK

- Biblioteka posiadająca wbudowany corpus
- Możemy się dostać do listy przygotowanych corpusów używając kodu:

```
import nltk.corpus
dir(nltk.corpus) # Python shell
print dir(nltk.corpus) # Pycharm IDE syntax
```



# NLTK – zastosowanie



- Biblioteka która umożliwia nam:
  - Procesowanie języka w Pythonie
  - Dostanie się do tzw. Lexical Corpora
  - Kategoryzowanie i tagowanie słów
  - Uczenie aby klasyfikować tekst
  - Budowanie cech opartych na gramatyce



# Przykład importowania korpusu



```
import nltk
from nltk.corpus import brown as cb
from nltk.corpus import gutenberg as cg
```



# Spacy

```
# pip install -U spacy
# python -m spacy download en_core_web_sm
import spacy

# Load English tokenizer, tagger, parser and NER
nlp = spacy.load("en_core_web_sm")

# Process whole documents
text = ("When Sebastian Thrun started working on self-driving cars at "
        "Google in 2007, few people outside of the company took him "
        "seriously. "I can tell you very senior CEOs of major American "
        "car companies would shake my hand and turn away because I wasn't "
        "worth talking to," said Thrun, in an interview with Recode earlier "
        "this week.")
doc = nlp(text)

# Analyze syntax
print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
print("Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB"])

# Find named entities, phrases and concepts
for entity in doc.ents:
    print(entity.text, entity.label_)
```



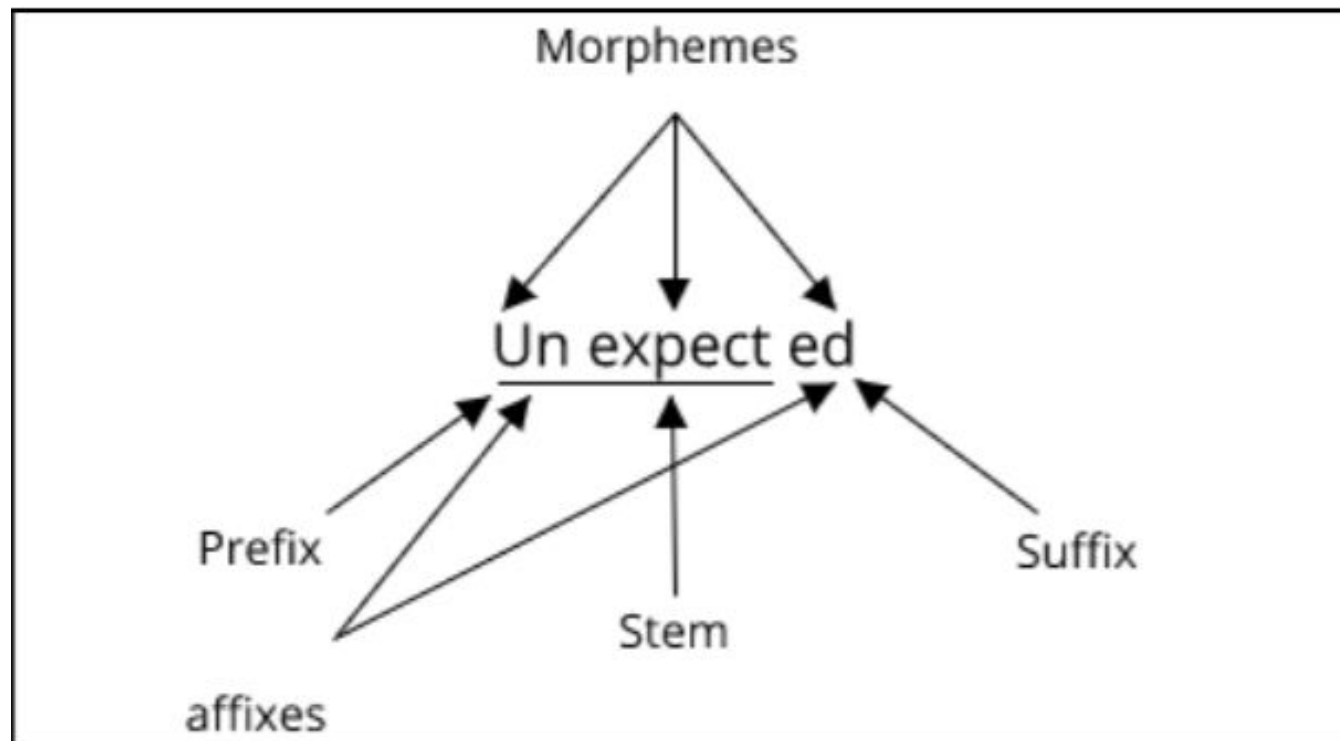
# Trochę teorii



- **Fonetyka** – nauka zajmująca się systematyką dźwięków w językach mówionych
- **Morfologia** – dlaczego „impossible” jest poprawne a „imread” już nie
- **Składnia**: co sprawia, że „I lecture computational linguistics” jest poprawne, natomiast „I lecturing computational linguistics” już nie
- **Semantyka**: dlaczego „I borrowed it from Jim” jest poprawne, a „I borrowed it to Jim” już nie? Albo czym „robbery by the lake” i „robbery by the fugitive” się różnią?
- **Pragmatyka**: dlaczego zdanie „taxes always go down” jest niepoprawne?

# Co to jest analiza morfologiczna?

- W jaki sposób tworzone są słowa używając tzw. morfemów





# Od teorii do praktyki



- **Tokenizer** – dzieli tekst na słowa i znaki interpunkcyjne:
  - Sporo pułapek, np. At 8 o'clock I didn't feel good. => |At|8|o'clock||I|did|n't|feel|good|.|
  - Wyrażenia regularne (*ang. regular expressions*)
- **POS (part of speech)**, oznacza części mowy
  - Potrzebne między innymi do parsowania zdań
- **N-grams** – tzw. n-tki słów opisane prawdopodobieństwem ich wystąpienia
  - Mogą posłużyć do podpowiadania słów lub generowania tekstów





# Od teorii do praktyki



- **Parser** – sprawdza składnię, określa części zdania:
  - W praktyce zależy nam na stworzeniu drzewa (drzew) składniowych
  - Trudne zadanie dla języków naturalnych
    - I saw a man on a hill with telescope
  - Oparte na regułach lub statystyce (zliczaniu)
- **Semantyka** – najtrudniejszy etap
  - Brak gotowych rozwiązań
  - Techniki mocno zależne od zastosowania

# Wektoryzacja

WEKTORYZACJA

①	TO	JEST	BIAŁY	KOT.					
②	TO	JEST	MÓJ	DOM.					
③	LUBIĘ	BIAŁY	KOLOR.						
④	BIAŁY	KOT	1	BIAŁY	DOM.				
	TO	JEST	BIAŁY	KOT	MÓJ	DOM	LUBIĘ	KOLOR	1
①	1	1	1	1	0	0	0	0	0
②	1	1	0	0	1	1	0	0	0
③	0	0	1	0	0	0	1	1	0
④	0	0	2	1	0	1	0	0	1

Wektoryzacja  
zlicza  
wystąpienie  
słowa

# TF-IDF

Step 1 : Calculate TF

Step 1.1 : Term Count for each document

Document 1		Document 2	
Term	Term Count	Term	Term Count
this	1	this	1
is	1	is	1
a	2	another	2
sample	1	example	3

Step 1.2 : Now calculate total number of words in each document

Document 1 : Total words are = 5

Document 2 : Total words are = 7

Step 1.3 : Now calculate TF

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

$$tf("this", d_1) = \frac{1}{5} = 0.2$$

$$tf("this", d_2) = \frac{1}{7} \approx 0.14$$

# TF-IDF w praktyce (z użyciem text blob)

```
from __future__ import division
from textblob import TextBlob
import math

def tf(word, blob):
    return blob.words.count(word) / len(blob.words)

def n_containing(word, bloblist):
    return 1 + sum(1 for blob in bloblist if word in blob)

def idf(word, bloblist):
    x = n_containing(word, bloblist)
    return math.log(len(bloblist) / (x if x else 1))

def tfidf(word, blob, bloblist):
    return tf(word, blob) * idf(word, bloblist)

text = 'tf idf, short form of term frequency, inverse document frequency'
text2 = 'is a numerical statistic that is intended to reflect how important'
text3 = 'a word is to a document in a collection or corpus'

blob = TextBlob(text)
blob2 = TextBlob(text2)
blob3 = TextBlob(text3)
bloblist = [blob, blob2, blob3]
tf_score = tf('short', blob)
idf_score = idf('short', bloblist)
tfidf_score = tfidf('short', blob, bloblist)
print "tf score for word short--- " + str(tf_score)+"\n"
print "idf score for word short--- " + str(idf_score)+"\n"
print "tf x idf score of word short--- "+str(tfidf_score)
```



# TF-IDF z użyciem sklearn ćwiczenie



- Zrób to co powyżej używając biblioteki sklearn



# Stop words



- Stop lista (po angielsku zwana „stop words”) to lista słów, które właściwie nie mają znaczenia dla algorytmu. Najczęściej są to spójniki, przyimki, zaimki – słowa, które nie niosą za sobą zbyt dużo treści.
- Przykład: Idę do sklepu spożywczego po pomidory i jabłka.

Z powyższego zdania bez utraty znaczenia możemy spróbować pozbyć się słów „do”, „po” oraz „i”.



# Jedno zdanie może mieć wiele interpretacji

**I saw a man on a hill with a telescope**

- There is a man on a hill, and I am watching him with my telescope
- There is a man and he's on a hill that also has a telescope on it
- I'm on a hill, and I saw a man using telescope





# Ujednolicenie form wyrazów: lematyzacja lub stemming.



- **Stemming** to obcięcie wszelkiego rodzaju przedrostków i przyrostków, mające na celu dotarcie do nieodmiennego „rdzenia” reprezentującego wyraz. Sam rdzeń niekoniecznie musi być poprawnym słowem. Algorytm stemmera nie musi być zależny od języka.
- **Lematyzacja** to sprowadzenie słowa do jego podstawowej postaci. W przypadku czasownika będzie to bezokolicznik, w przypadku rzeczownika – mianownik liczby pojedynczej. Do wykonania tego zadania potrzebny jest słownik lub rozbudowany zestaw reguł fleksyjnych dla danego języka.





# Różnice między stemmingiem a lematyzacją



- Stemming zawsze działa na pojedynczym słowie bez wiedzy na temat całego kontekstu
- Lematyzacja skupia się na znaczeniu słowa w zdaniu
- Stemming służy do grupowania słów o podobnym znaczeniu
- Lematyzacje wykorzystuje się aby tworzyć słowniki takie jak Wordnet



# Normalizacja



- **Czym jest normalizacja?**

Polega na takim przetworzeniu tekstu, żeby miał spójną formę, która ułatwi dalszą interpretację tego tekstu. W zależności od potrzeb, różnie może wyglądać taka normalizacja.



# Przykłady zastosowania normalizacji




- Zmiana wielkości liter na małe i wielkie
- Rozwinięcie skrótów
- Normalizacja skrótowców
- Konwersja wyrazów numerycznych i słowno-numerycznych do postaci słownej
- Usunięcie lub zamiana znaków interpunkcyjnych
- Usuwanie (lub zamienianie) znaków diakrytycznych
- Usunięcie elementów niezwiązanych z tekstem



# Gramatyki formalne



- Oparte na teorii Chomskiego, gramatyki (tworzone ręcznie lub automatycznie) to zestawy reguł, które definiują, jak może wyglądać poprawne zdanie w danym języku (naturalnym, jak polski, albo sztucznym, jak C++).
  - Gramatyka może składać się z poniższych reguł:
    - NP  $\rightarrow$  ART N
    - ART  $\rightarrow$  a
    - ART  $\rightarrow$  the
    - N  $\rightarrow$  cat
    - N  $\rightarrow$  dog
- 

# Wyrażenia regularne

Symbol	Znaczenie	Przykład
.	Dowolny znak.	k.t pasuje do „kot” i „kat”
+	Powtórzenie poprzedzającego znaku jeden lub więcej razy.	kot+ pasuje do „kot”, „kott”, „kott” itp.
*	Powtórzenie poprzedzającego znaku zero lub więcej razy.	kot* pasuje do „ko”, „kot”, „kott” itp.
{m,n}	Powtórzenie poprzedzającego znaku od m do n razy.	ko{2,4}t pasuje do „koot”, „koooot” i „koooot”.
^	Początek linii.	^k – literka k zapisana na początku linii
\$	Koniec linii	k\$
[abc]	Znaki ze zbioru, wyszczególnione	k[ao]t pasuje do „kot” i „kat”.
[a-c]	Znaki ze wskazanego przedziału	[2-4] to cyfra „2”, „3” lub „4”.
\b	Granica słowa	.+a\b to dowolne słowo kończące się na literą „a”, np. „torba”.
\d	Cyfra	\d\d pasuje na przykład do „23”
\s	Biały znak	k\s t pasuje do „k t”
\S	Znak niebiały	k\S t pasuje do „kot”, „kit”, ale nie do „k t”



# Word sense disambiguation



- Co wybrane słowo znaczy w danym kontekście
- Słowo wieloznaczne, homonimy
- Potrzebne zasoby:
  - Statystyki występowania słów
  - Słowniki
- Maksymalna trafność: zdolności ludzkie
- Minimalna: częstość najpopularniejszego znaczenia



# Word sense disambiguation – zastosowanie



- **Podejście statystyczne:**

- Bierzemy korpus etykietowanych tekstów
- Dla wskazanego słowa określamy wszystkie jego konteksty w korpusie
- Korzystamy z twierdzenia Bayesa i maksymalizujemy  $P(w|c)$

- **Podejście słownikowe:**

- Porównujemy kontekst słowa z definicją słownikową



# Metody syntaktyczne

- Analiza składniowa, która mówi nam o pewnym logicznym znaczeniu danego zdania
- Przykład: zdanie *School go a boy* nie przekazuje logicznie swojego znaczenia, struktura gramatyczna jest niepoprawna
- Tak więc ta analiza mówi nam czy zdanie ma swoją logikę, strukturę i czy zdanie jest poprawne




# Metody syntaktyczne – praktyka

```
# This script is for generating parsing tree by using NLTK.
# We are using python wrapper for stanford CoreNLP called-"pycorenlp" to generate Parsing result for us.
# NLTK gives us tree representation of stanford parser.
import nltk
from nltk import CFG
from nltk.tree import *
from pycorenlp import StanfordCoreNLP
from collections import defaultdict

# Part 1: Define a grammar and generate parse result using NLTK
def definegrammar_paserresult():
    Grammar = nltk.CFG.fromstring("""
S -> NP VP
PP -> P NP
NP -> Det N | Det N PP | 'I'
VP -> V NP | VP PP
Det -> 'an' | 'my'
N -> 'elephant' | 'pajamas'
V -> 'shot'
P -> 'in'
""")
    sent = "I shot an elephant".split()
    parser = nltk.ChartParser(Grammar)
    trees = parser.parse(sent)
    for tree in trees:
        print tree

# Part 2: Draw the parse tree
def draw_parser_tree():
    dp1 = Tree('dp', [Tree('d', ['the']), Tree('np', ['dog'])])
    dp2 = Tree('dp', [Tree('d', ['the']), Tree('np', ['cat'])])
    vp = Tree('vp', [Tree('v', ['chased']), dp2])
    tree = Tree('s', [dp1, vp])
    print(tree)
    print(tree.pformat latex qtree())
```



```
# Part 3: Stanford parser wrapper library "pycorenlp"
# you need to install pycorenlp as well as you need to download stanford-corenlp-full-* from stanford corenlp website.
def stanford_parsing_result():
    text = """ I shot an elephant. The dog chased the cat. School go to boy. """
    nlp = StanfordCoreNLP('http://localhost:9000')
    res = nlp.annotate(text, properties={
        'annotators': 'tokenize,ssplit,pos,depparse,parse',
        'outputFormat': 'json'
    })
    print(res['sentences'][0]['parse'])
    print(res['sentences'][2]['parse'])

if __name__ == "__main__":
    print "\n-----Parsing result as per defined grammar-----"
    definegrammar_pasrereult()
    print "\n-----Drawing Parse Tree-----"
    draw_parser_tree()
    print "\n-----Stanford Parser result-----"
    stanford_parsing_result()
```



# Analiza semantyczna

- Analiza leksykalna, która koncentruje się na zrozumieniu pojedynczych słów w strumieniu tekstu, natomiast analiza semantyczna skupia się na większych fragmentach tzw. chunkach
- Analizę semantyczną można przeprowadzić na poziomie frazy, na poziomie zdań, akapitów, a czasem także na poziomie dokumentów



# Klasyfikacja tekstu – przykłady

- Podział maili na spam i niespam
- Analiza sentymentów
- Kategoryzacja artykułów
- Rozpoznawanie języka
- Przypisanie kategorii do towarów
- Wypadki przy pracy



# Klasyfikacja tekstu - przykładowe algorytmy

- Naive Bayes
- Random forest
- SVM
- Regresja logistyczna
- Sieci neuronowe

# Klasyfikacja tekstu



mapa myśli: klasyfikacja tekstu



# Name entity recognition



- Wykrywanie encji w tekście:
  - Osoby
  - Miejsca
  - Organizacje
  - Określenia czasu
  - Wartości
- Świetne wyniki dla języka angielskiego (Stanford)



# Question answering

- Automatyczne odpowiadanie na pytania
- Głównie związane z poszukiwaniem wiedzy i porad w internecie
- Wyszukiwarki internetowe tego używają
- Bardzo dobre wyniki






# Analiza sentymentu


- Firmy są zainteresowane opinią o produktach
- Ludzie szukają opinii przed zakupem
- W ostatnich latach bardzo popularne zagadnienie w NLP
- Ściśle powiązane z analizą mediów społecznościowych (twitter, facebook)

# Sentymment vs opinia

## „Sentymment” vs Opinia



Bardziej związany z  
uczuciami




Bardziej związana z  
przemyśleniami i  
sposrzeżeniami



# Analiza sentymentu



- Potrzebujemy dwóch definicji:
    - Czym jest pojedyncza opinia?
    - Jaka jest ogólna opinia? (w tzw. populacji)
  - **Opinia to:**
    - Encja: opisywany obiekt
    - Aspekt: cecha opisywanego obiektu
    - Sentyment: +, -, neu, ocena, liczba gwiazdek, emocja
    - Właściciel: osoba wypowiadająca opinię
    - Czas: moment kiedy opinia została wyrażona
- 

# Przykład zastosowania analizy sentymentu

## Przykład

*“I bought an **iPhone** a few days ago. It is such a nice **phone**. The **touch screen** is really cool. The **voice quality** is clear too. It is much better than my old **Blackberry**, which was a terrible **phone** and so **difficult to type** with its **tiny keys**. However, **my mother** was mad with me as I did not tell her before I bought the **phone**. She also thought the phone was too **expensive**, ...”*

## Feature Based Summary of iPhone:

### Feature1: **Touch screen**

Positive: 212

- The **touch screen** was really cool.
- The **touch screen** was so easy to use and can do amazing things.

...

Negative: 6

- The **screen** is easily scratched.
- I have a lot of difficulty in removing finger marks from the **touch screen**.

...

### Feature2: **voice quality**



# POS-tagging



- Potrzebujesz otagowanego corpusu
- Wybierz cechy
- Trenuj używając algorytmu drzewa decyzyjnego z pakietu sklearn
- Sprawdź skuteczność
- Spróbuj przewidzieć POS tagi używając własnego przetrenowanego modelu

# Przykład użycia Stanford POS-tagger

```
from pycorenlp import StanfordCoreNLP
nlp = StanfordCoreNLP('http://localhost:9000')

def stnfordpostagdemofunction(text):
    output = nlp.annotate(text, properties={
        'annotators': 'pos',
        'outputFormat': 'json'
    })
    for s in output["sentences"]:
        for t in s["tokens"]:
            print str(t["word"]) + " --- postag ---" + str(t["pos"])

if __name__ == "__main__":
    stnfordpostagdemofunction("This is a car.")
```



# POS-tagging – ćwiczenie



- Spróbuj używając biblioteki TreeTagger, aby wygenerować POStagging.
- Pomocny będzie do tego poniżej omawiany link z google colaboratory:  
[https://colab.research.google.com/drive/1d7LO\\_0665DYw6DrVJXXautJAJzHHqYOm](https://colab.research.google.com/drive/1d7LO_0665DYw6DrVJXXautJAJzHHqYOm)
- Treetagger możesz pobrać z:  
<https://www.cis.lmu.de/~schmid/tools/TreeTagger/>



# Wyzwania w POS-tagging

- Identyfikacja właściwego POStagu dla określonych słów w niejednoznacznej strukturze składni jest trudne, jeśli słowo ma inne znaczenie kontekstowe wówczas POStagger może generować niewłaściwe tagi POS.
- Opracowanie taggera POS dla indyjskich jest trudne, ponieważ dla niektórych języków trudno znaleźć odpowiedni zbiór danych.





# Name entity recognition



- Segregowanie encji do zdefiniowanej klasy
- Różne narzędzia NER mają różnie zdefiniowane klasy
- Dostępny np. Stanford NER ma:
  - W 1 wersji 3 klasy lokalizacja, osoba i organizacja
  - W 2 wersji: lokalizacja osoba, organizacja i różny typ
  - W 3 wersji: osoba, lokalizacja, organizacja, pieniądze, procent, data i czas



# NER w spaCy



- Klasy NER dostępne w spaCy:
  - PERSON – identyfikuje imię osoby
  - NORP – identyfikuje narodowość, poglądy polityczne i religijne
  - FACILITY – kategoria identyfikująca budynki, lotniska, autostrady
  - ORG – organizacje, instytucje
  - GPE – miasta, kraje
  - LOC – klasa dla „nie-GPE” lokalizacji
  - PRODUCT – klasa zawierająca produkty, jedzenie, pojazdy itd.
  - EVENTS – klasa dla wydarzeń sportowych, wojen, nazwane huragany
  - WORK\_OF\_ART – tytuły książek, piosenek
  - LANGUAGE – tagi dla nazwanych języków



# NER – wyzwania



- Narzędzia NER uczą się na zamkniętym zestawie danych, system NER nauczony na 1 zbiorze danych działa dobrze na tym zbiorze danych, a źle na innym, wymagałoby to stworzenia uniwersalnego NER.
- Czasami nazwa lokalizacji jest też nazwiskiem osoby, NER nie radzi sobie kiedy ta sama nazwa jest nazwą lokalizacji, organizacji i osoby.
- Budowanie NER np. dla mikroblogów jest dużym wyzwaniem.



# NER – ćwiczenie

- Poniżej mamy link do google colab obrazujący NER we frameworku Spacy:

<https://colab.research.google.com/drive/1cuOG4M9Y2keSTflp6roY1tZwgBabkwBR>

- Spróbuj wykonać to dla języka polskiego



# Bag of words



- Przeanalizujemy 2 dokumenty:

*Text document 1: John likes to watch cricket. Chris likes cricket too*

*Text document 2: John also likes to watch movies.*

- Na podstawie tych dwóch dokumentów możemy uzyskać poniższą listę:

*List of words = ["John", "likes", "to", "watch", "cricket", "Chris", "too", "also", "movies"]*

**Poniższa lista to właśnie bag of words!**

- Więcej:  
<https://colab.research.google.com/drive/1nEQl4tfKYitDJFZnLHS7EwgWDhHiz6Jk>

# Bag of words – czyli jak zrozumieć tekst

- Mamy 3 recenzje:

Review 1: This movie is very scary and long

Review 2: This movie is not scary and is slow

Review 3: This movie is spooky and good

- Możemy teraz wziąć każde słowo i zliczyć jego wystąpienie w tekście:

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	0	1	1	0	1	0	0	8
Review 3	1	1	1	0	0	0	1	0	0	1	1	6



# Szkic BoW

1. Jeżeli nowe zdanie zawiera nowe słowo to zmienia to nam rozmiar słownika, co zmienia też rozmiar wektora.
2. Wektory zawierają wiele 0.
3. Nie mamy żadnych informacji na temat gramatyki tekstu.
4. Następnie używamy tf-idf.



# BoW a TF-IDF



- Bag of word po prostu tworzy zbiór wektorów zawierających liczbę wystąpień słów w dokumencie.
- Podczas gdy samo tf-idf zawiera informacje o wystąpieniu słów ważniejszych i mniej ważnych .
- Wektory bag of word są łatwe do zinterpretowania.
- Jednak zazwyczaj tf-idf lepiej sprawdza się w modelach uczenia maszynowego.

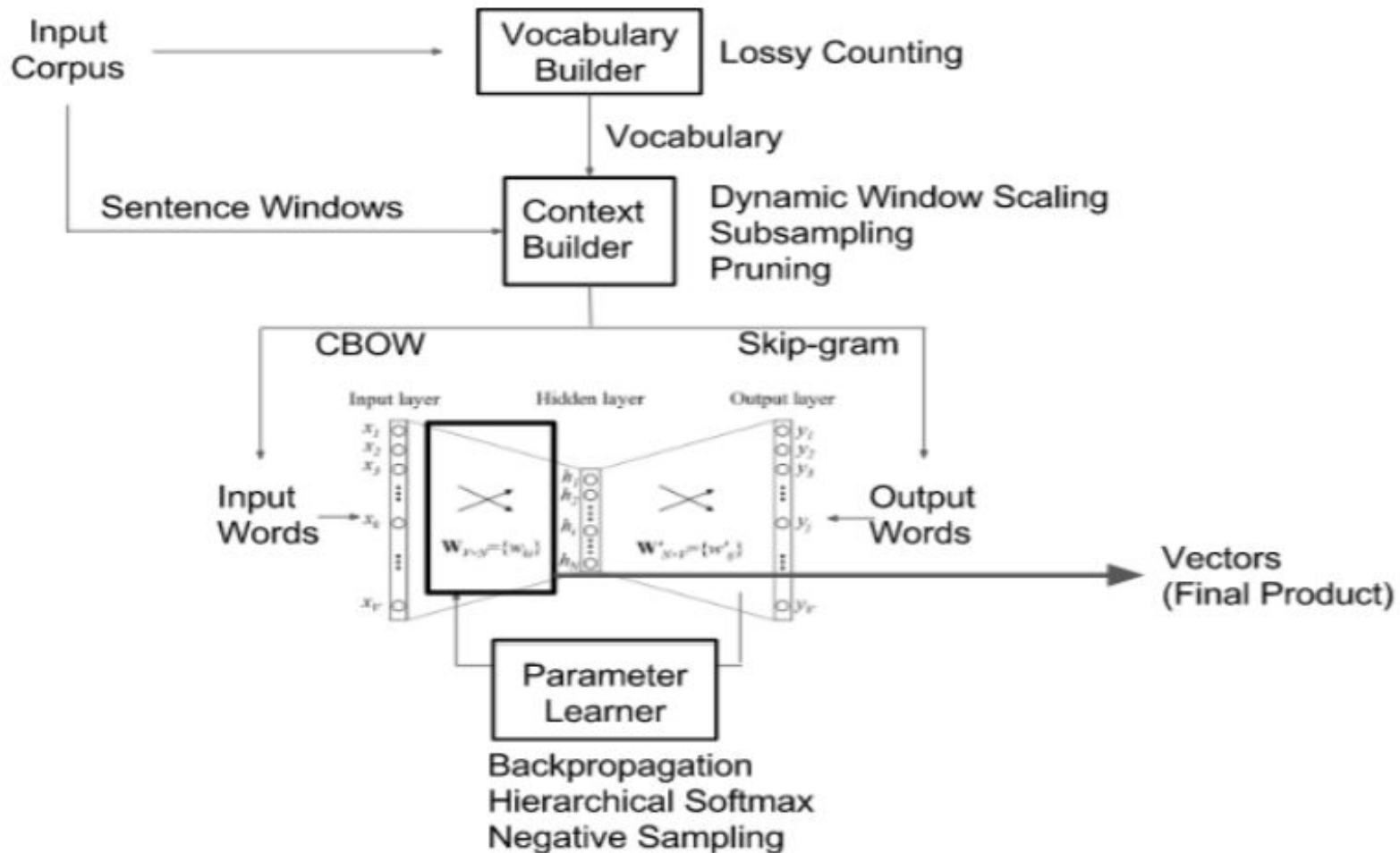




# Word2vec

- Został opracowany przy użyciu 2-warstwowych sieci neuronowych.
- Zawiera dużo danych tekstowych lub korpusów z danymi tekstowymi, z których następnie generuje zestaw wektorów z podanego tekstu.
- Tworzymy wielowymiarową przestrzeń wektorów, która ma czasem kilkaset wymiarów.

# Architektura word2vec



# Przykład word2vec

Source Text	Training Samples			
<table><tr><td>The</td><td>quick</td><td>brown</td></tr></table> fox jumps over the lazy dog. →	The	quick	brown	(the, quick) (the, brown)
The	quick	brown		
The <table><tr><td>quick</td><td>brown</td><td>fox</td></tr></table> jumps over the lazy dog. →	quick	brown	fox	(quick, the) (quick, brown) (quick, fox)
quick	brown	fox		
The quick <table><tr><td>brown</td><td>fox</td><td>jumps</td></tr></table> over the lazy dog. →	brown	fox	jumps	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
brown	fox	jumps		
The quick brown <table><tr><td>fox</td><td>jumps</td><td>over</td></tr></table> the lazy dog. →	fox	jumps	over	(fox, quick) (fox, brown) (fox, jumps) (fox, over)
fox	jumps	over		



# Word2vec



- To zbiór modeli służących do osadzania słów.
- Word2vec przyjmuje jako informacje olbrzymią ilość tekstu i tworzy przestrzeń wektorową, zwykle każdemu niezwykłemu słowu przypisywany jest tzw. wektor porównawczy.
- „Człowieka można odróżnić po organizacji, którą prowadzi”, podobnie słowo można rozpoznać po zebraniu często używanych przy nim słów, na tej możliwości opiera się Word2Vec.
- Word2Vec ma dwie odmiany, jedną zależną od modelu Skip Gram, a drugą zależną od modelu Continuous Bag of words.



# Model skip gram



- Biorąc pod uwagę słowo informacyjne w zdaniu, system przewiduje, na ile prawdopodobne jest, że każde słowo w kontekście jest bliskie temu słowu. Dane wejściowe dla systemu neuronowego to zestawy słów, które zawierają słowo informacyjne i jego kontekst.
- Na przykład, rozważmy zdanie "The quick brown fox jumps over the lazy dog" a rozmiar okna wynosi 2. Przykładami treningowymi będą: ("the", "quick"), ("the", "brown"), ("quick", "the"), ("quick", "brown"), ("quick", "fox") itd. Wektor ten będzie miał  $n$  składowych (na przykład  $n=10000$ , po jednej dla każdego słowa w naszym słowniku) i umieścimy "1" na pozycji odpowiadającej słowu wejściowemu, a "0" na wszystkich pozostałych pozycjach.



# CBOW (Continuous Bag of Words)

- Przeciwnieństwo skipgram.
- Biorąc pod uwagę kontekst słów (otaczający słowo) w zdaniu, sieć przewiduje, jakie jest prawdopodobieństwo, że każde słowo w słowniku jest tym słowem.
- W modelu Continuous Bag-of-Words próbujemy przewidzieć słowo używając słów obejmujących je (słów kontekstowych), wkład w model to one-hot zakodowany wektor słów ustawień wewnątrz rozmiaru okna.



# Realizacja modelu



- **Rozmiar:** określa wymiarowość wektora słów, określa liczbę tokenów używanych do zaprezentowania danego słowa
- **Okno:** maksymalna odległość między słowem docelowym a słowem sąsiednim. Na przykład weźmy frazę „agama to gad” z 4 słowami (przypuśćmy, że nie wykluczamy słów stopujących). Jeśli rozmiar okna wynosi 2 to na wektor słowa „agama” mają bezpośredni wpływ słowa „jest” i „a”.
- **Min\_count:** ignoruje wszystkie słowa o łącznej częstotliwości mniejszej niż ta. Na przykład, jeśli częstotliwość występowania słów jest wyjątkowo niska, to słowo to może zostać uznane za nieważne.
- **Sg:** wybiera algorytm uczenia: 1 dla Skip-Gram; 0 dla CBOW (Continuous Bag of Words).
- **Pracownicy:** liczba wątków roboczych używanych do trenowania modelu.



# Kroki budowy modelu

1. Wyczyść dane.
2. Zbuduj korpus.
3. Wytrenuj model Word2Vec.
4. Wizualizuj reprezentacje t-SNE dla najczęściej używanych słów.



# Przykładowy rezultat

```
model.most_similar('walking')
```

```
↳ /usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated `most_similar` (N
    """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion of the second argument of issub
    if np.issubdtype(vec.dtype, np.int):
[('walkng', 0.7617052793502808),
 ('waling', 0.7350203990936279),
 ('locationwalking', 0.7182509303092957),
 ('spitting', 0.7120993733406067),
 ('goldmine', 0.6016842722892761),
 ('downtownwaterfront', 0.5912593007087708),
 ('conga', 0.5825145840644836),
 ('ltrain', 0.5787146091461182),
 ('wandering', 0.5749132633209229),
 ('walkable', 0.5730565786361694)]
```

*Words similar to walking*



# Polecane materiały związane z tematyką bloku

- <https://spacy.io/>
- <https://www.nltk.org/>
- <https://nlp.stanford.edu/>