Python NLTK Training

NLTK

Trainer: Amir Othman

TINFOTECH

Website:www.tertiarycourses.com.my
Email: enquiry@tertiaryinfotech.com

Agenda

Module 1 Get Started on NLTK

- What is NLTK
- Install NLTK
- Download NLTK Data and Packages

Module 2: Text Analysis with NLTK

- Analyzing Text
- Tokenize
- Stop Words
- Stemming
- Lemmatizing
- POS Tagging
- NER & Chucking

Agenda

Module 3: Corpus

- What is Text Corpus
- Corpus Attributes
- Create Your Own Corpus
- WorldNet
- Synset, Lemma, Definition
- Text Statistics

Module 4: Text Classification

- What is Text Classification
- Steps for Text Classification
- Save and Load Model
- Scikit Learn Classifiers with NLTK

Agenda

Module 5: Advanced Topics

- **Grammar Parser**
- **N-gram**

Prerequisite

Basic Python knowledge is assumed

Exercise Files

You can download the exercise files from

https://github.com/amirothman/nltk-training

Module 1
Get Started on NLTK

What is Natural Language Processing?

- The goal of Natural Language Processing (NLP) is for computers to understand human language in order to perform tasks like making appointment, buying things
- Full understanding and representing the meaning of language is much harder than object recognition in computer vision.

Applications of NLP

- Spelling Check, Keyword Search, Finding Synonyms
- Extracting information from websites such as product price, dates…
- Machine translation
- Speech recognition
- Chatbot, automating customer support
- Complex question answering

What is NLTK?

- NLTK is a leading platform for building Python programs to work with human language data.
- It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet,
- It provides a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning
- http://www.nltk.org/

Install NLTK

For PC:

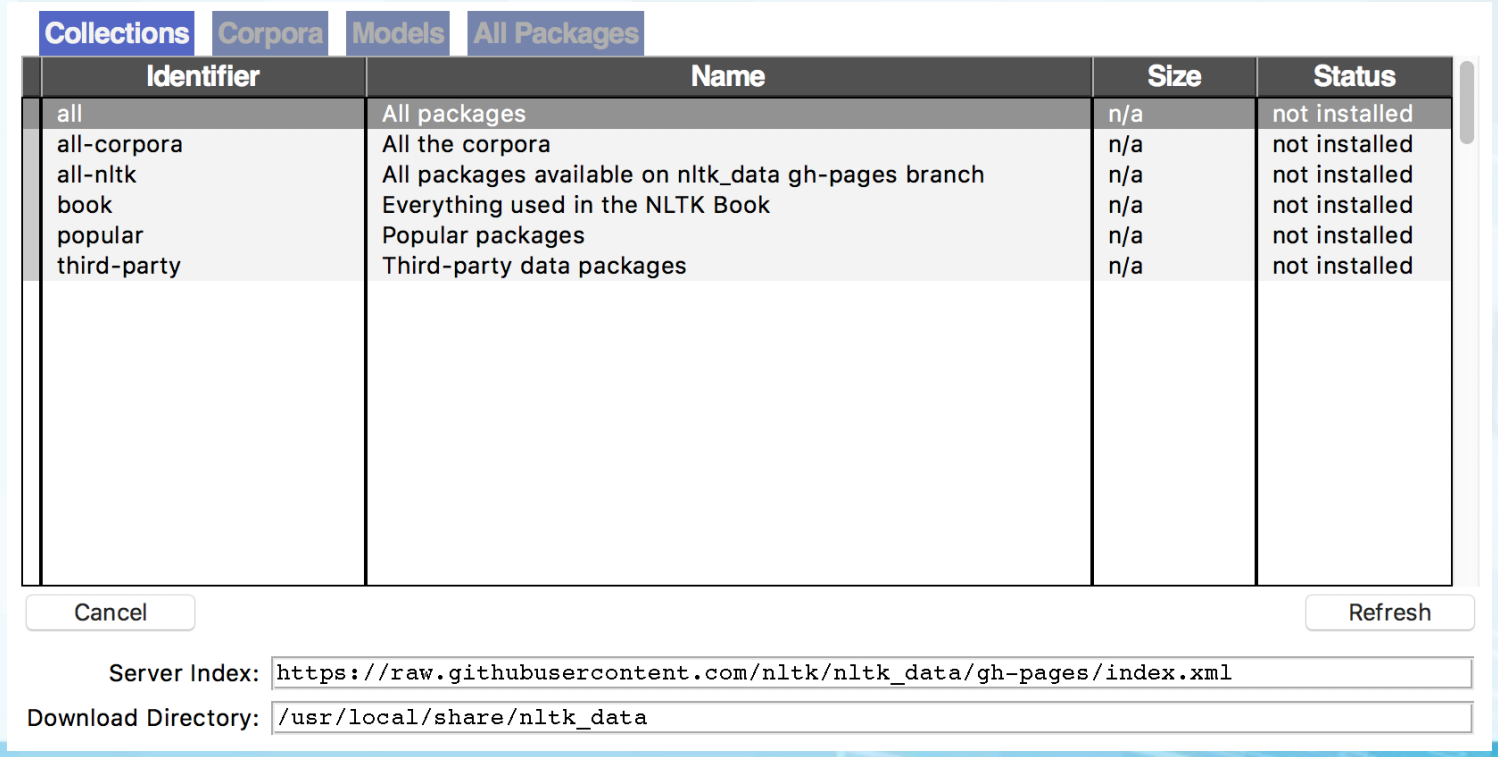
pip install nltk

For Mac

pip3 install nltk

Download Data & Packages

- Change the download directory to /usr/local/share/nltk_data
- download all packages



Check NLTK

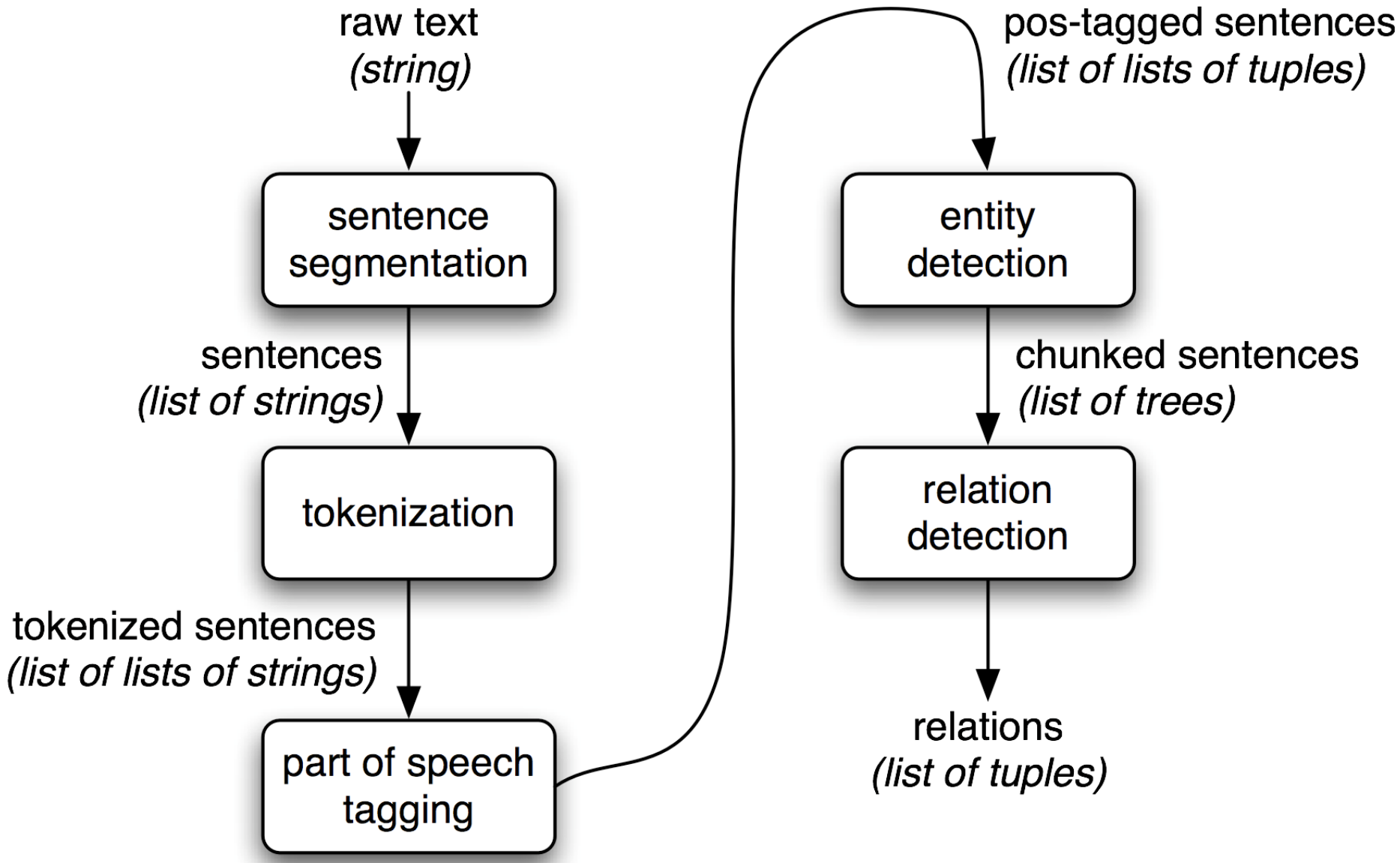
```
import nltk
from nltk.corpus import brown
brown.words()
```

Module 2
Analyzing Text with NLTK

Analyzing Text

- The raw text of the document is split into sentences using a sentence tokenizer
- Each sentence is further subdivided into words using a word tokenizer.
- Each sentence is tagged with part-of-speech tags
- Named entity detection to search for potentially interesting entities in each sentence

Analyzing Text



Sentence Tokenize

```
from nltk.tokenize import sent_tokenize


text = "Hello, Mr. Tan, how are you? Do you find NLTK fun and easy to use? I hope you do in this class"


sent_tokenize(text)
```

Word Tokenize

```
from nltk.tokenize import  word_tokenize

text = "Hello, Mr. Tan, how are you? Do you find NLTK fun and easy to use? I hope you do in this class"

word_tokenize(text)
```

Exercise: Tokenize

get the text from internet:

```
from urllib import request
url =
"http://www.gutenberg.org/files/2554/2554.txt"
response = request.urlopen(url)
text = response.read().decode('utf8')
```

Determine the # of tokenized sentences and words

Time: 5mins

Stop Words

- Text may contain stop words like ‘the’, ‘is’, ‘are’.
- Stop words can be filtered from the text to be processed.
- There is no universal list of stop words in nlp research
- NLTK contains a list of stop words.

Stop Words in NLTK

set(stopwords.words('english'))

'you', 'have', 'why', 'here', 'very', 're', 'mightn', 'there', 'we', 'our', 'his', 'a', 'theirs', 'again', 'to', 'only', 'yours', 'doesn', 'when', 'himself', 'too', 'each', 'should', 'wasn', 'having', 'just', 'me', 'couldn', 'which', 'doing', 'ma', 'the', 'isn', 'y', 'mustn', 'an', 'under', 'do', 'and', 'are', 'such', 'themselves', 'shouldn', 'weren', 'but', 'those', 'other', 'can', 'of', 'because', 'so', 'than', 'then', 'through', 'she', 'while', 'ain', 'during', 'does', 'itself', 'until', 'most', 'from', 'these', 'in', 'nor', 'some', 'hers', 'he', 'haven', 'more', 'has', 'needn', 'did', 'by', 't', 'am', 'm', 'between', 'ourselves', 'or', 'further', 'was', 'they', 'both', 'her', 'off', 'own', 'about', 'been', 'not', 'that', 'at', 'will', 'where', 'd', 'who', 'now', 'it', 'i', 'against', 'once', 'up', 'your', 'him', 'yourself', 'what', 'on', 'into', 'wouldn', 'as', 'o', 'out', 'for', 'few', 've', 'hadn', 'yourselves', 'whom', 'before', 'above', 'no', 'is', 'their', 'aren', 'were', 'myself', 'be', 'don', 'all', 'any', 'my', 'herself', 'had', 'didn', 'its', 'hasn', 'with', 'll', 'below', 'down', 'won', 'being', 'ours', 'them', 's', 'this', 'how', 'over', 'after', 'shan', 'same', 'if'

Filter Stop Words

```
words = word_tokenize(text)

filtered = [w for w in words if not w in stop_words]
```

Stemming and Lemmatization

- Stemming and Lemmatization are the basic text processing methods for English text
- Stemming is the process for reducing derived words to their stem, base or root form
- Lemmatisation is the process of grouping together different derived word so they can be analysed as a single item

Stemming

- Stemming are techniques to find out the root/stem of a word eg
  - user
  - users
  - used
  - use

The stem/root for above words is "use"

- Stemming is use for
  - Matching similar words
  - Reduce indexing size

Basic Stemming Methods

- Remove ending
  - "uses" to "use"
  - "used" to "use"

- Transform word
  - "using" to "use"

Stemming with NLTK

NLTK provides several famous stemmers interfaces, such as

- Porter Stemmer
- Lancaster Stemmer
- Snowball Stemmer

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
[ps.stem(w) for w in words]
```

Lemmatizing

- Lemmatisation is closely related to stemming. The difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech.
- The NLTK Lemmatization method is based on WordNet’s built-in morphy function.

Lemmatizing in NLTK

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

[lemmatizer.lemmatize(w) for w in words]
```

POS Tagging

- Part-of-speech tagging is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context—i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph. Eg identification of words as nouns, verbs, adjectives, adverbs, etc.
- It is one of the most important text analysis tasks used to classify words and label them according the tagset.

Common POS Tags

NN noun, singular eg desk

NNS noun plural eg desks

VB verb eg take

VBD verb eg took

VBG verb, present participle eg taking

VBN verb, past participle eg taken

List of POS tags

http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

POS Tags - Noun & Verb

```
# Noun
import nltk
print(nltk.pos_tag(['cat']))


# Verb
import nltk
print(nltk.pos_tag(['run']))
```

POS Tags - Adjective & Adword

```
# Adjective
import nltk
print(nltk.pos_tag(['delicious','food']))

# Adword
import nltk
print(nltk.pos_tag(['run','slowly']))
```

POS Tagging on Tokenized Text

```
words = word_tokenize(text)
tagged = nltk.pos_tag(words)
```

POS Tagging on Corpus

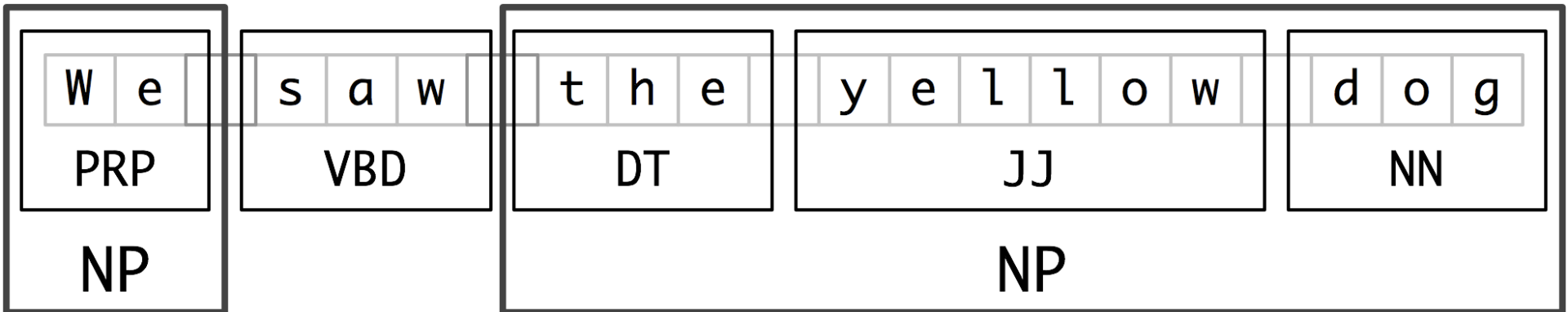
```
from nltk.corpus import brown
brown_tagged_sents = brown.tagged_sents(categories='news')
brown_sents = brown.sents(categories='news')
```

Named Entity Detection

- Named Entity Recognition (NER) is probably the first step towards information extraction from unstructured text.
- It basically means extracting what is a real world entity from the text Eg people, places, things, locations, monetary figures, and more.

Chunking

- The basic technique for NER or entity detection is chunking, which segments and labels multi-token sequences
- The smaller boxes show the word-level tokenization and part-of-speech tagging, while the large boxes show higher-level chunking. Each of these larger boxes is called a chunk.



Chunking in NLTK

```
from nltk import chunk
tree = chunk.ne_chunk(tagged)
print(tree)
tree.draw()
```

Module 3
Text Corpus

Text Corpus

- A text corpus is a large body of text
- NLTK includes a small selection of texts from the Project Gutenberg electronic text archive.

To access the Gutenberg corpus:

```
from nltk.corpus import gutenberg
gutenberg.fileids()
```

Basic Copus Attributes

Fileid : gutenberg.fileids()

Characters : gutenberg.raw("fileid"))

Words : gutenberg.words("fileid")

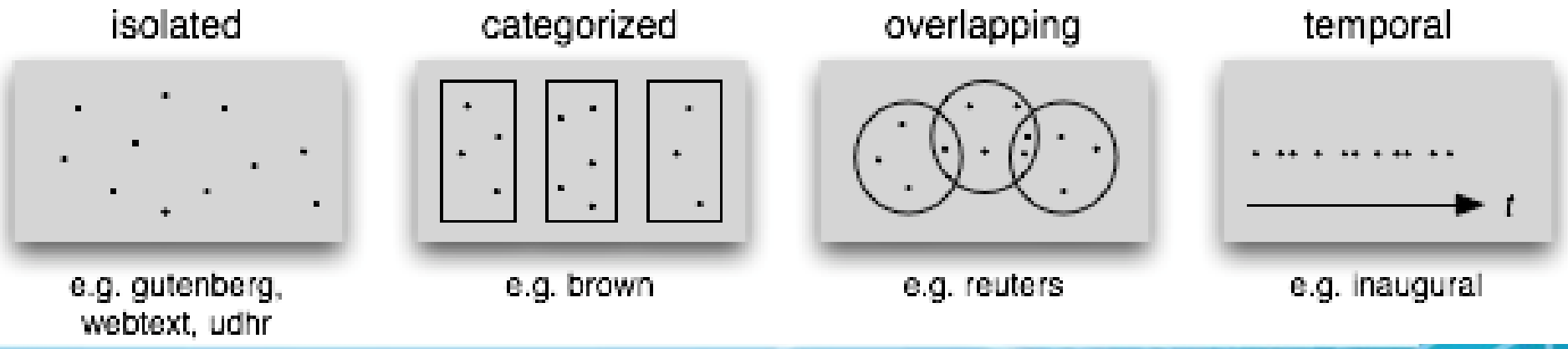
Sentence : gutenberg.sents("fileid")

Exercise: Text Corpus

tokenize the austen-sense.txt from gutenberg and show the first 5 tokens.

Text Corpus Structure

- The simplest corpus is just a collection of texts eg gutenberg.
- Some corpus are grouped into single category eg brown, movie_review or group into multiple overlapping categories eg reuters



Categories

movie_reviews.categories(fileid)

Ex: Categories

Determine the category for brown corpus for fileid = 'cl08'

Load Your Own Corpus

```
from nltk.corpus import PlaintextCorpusReader
corpus_root = 'corpus'
my_corpus = PlaintextCorpusReader(corpus_root, '.*')
```

Ex: Create Your Own Corpus

- Create a new folder and add a few text files.
- Load your corpus

Time: 10 mins

WorldNet

WordNet is a semantically-oriented dictionary of English, similar to a traditional thesaurus but with a richer structure.

NLTK includes the English WordNet, with 155,287 words and 117,659 synonym sets.

https://wordnet.princeton.edu/

Synset

Synset, or "synonym set" is a collection of synonymous words (or "lemmas")

```
from nltk.corpus import wordnet
wordnet.synsets("motorcars")
wordnet.synsets("automobiles")
wordnet.synsets("cars")

wordnet.synset('car.n.01').lemma_names()
```

Lemmas

```
wordnet.synset('car.n.01').lemma_names()
```

Definition

wordnet.synset('car.n.01').definition()

Semantic Similarity

```
w1 = wordnet.synset('ship.n.01')
w2 = wordnet.synset('boat.n.01')
w1.wup_similarity(w2)
```

Exercise: WordNet

Determine the similarity of "rat" and "mouse" from WordNet

Conditional Frequency Distr.

We can obtain counts for each genre of interest using NLTK's conditional frequency distribution

```
nltk.ConditionalFreqDist(
    (genre, word)
    for genre in brown.categories()
    for word in brown.words(categories=genre)
```

Text Statistics

```
distr = nltk.FreqDist(words)
distr.most_common(5)
distr["python"]
```

Ex: Text Statistics

Import the movie_reviews corpus and determine the top 15 most common words

Hint:

```
from nltk.corpus import movie_reviews
words = [w.lower() for w in movie_reviews.words()]
distr = nltk.FreqDist(words)
```

Module 4
Text Classification

Text Classification

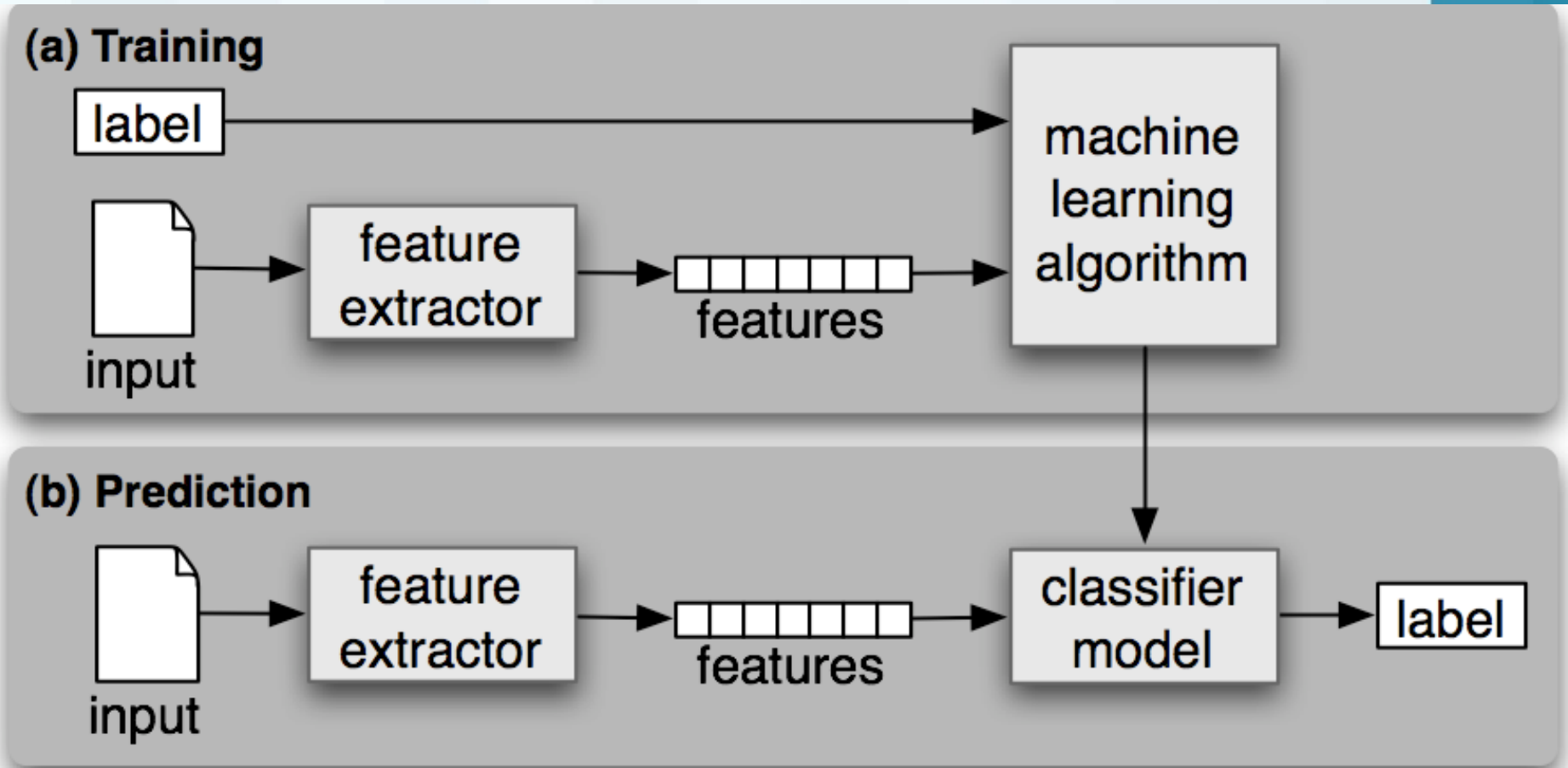
The goal of text classification is to classify text to a label

Eg

- classify the sentiment of a text - positive or negative
- classify the category of a news - political, home, international, sport...
- classify the type of email - spam or not spam

Supervised Text Classification

A classifier is called supervised if it is built based on training corpora containing the correct label for each input



Terms for Text Classification

Feature: refer to some property of a token

Featureset: a dictionary that maps from a feature name to feature label.

Feature extractor: a function that takes a token as its input, and returns a featureset describing that token

Steps for Text Classification

Step 1: Load data

Step 2: Extract feature set

Step 3: Split the feature set to training/testing

Step 4: Load the classifier

Step 5: Train the classifier on training set

Step 6: Evaluate the classifier on testing set

Step 1: Load Data

```
from nltk.corpus import names

labeled_names = ([(name, 'male') for name in
names.words('male.txt')]
    + [(name, 'female') for name in
names.words('female.txt')])

import random
random.shuffle(labeled_names)
```

Step 2: Feature Extractor

The first step in creating a classifier is to extract the feature and form a feature set

Eg: extract the last letter of a name as a feature

```
def feature_extractor(name):
    return {'last_letter': name[-1]}

featureset = [(feature_extractor(name), gender)
for (name, gender) in labeled_names]
```

Step 3: Split Test/Training Data

- We then divide the resulting list of feature set into a training set and a test set.
- The training set is used to train a classifier

```
train_set, = featureset[500:]
test_set = featureset[:500]
```

Step 4/5: Load & Train Classifier

```
import nltk
classifier =
nltk.NaiveBayesClassifier.train(train_set)
```

Step 6: Evaluation/Prediction

```
classifier.classify(gender_features('Neo'))
nltk.classify.accuracy(classifier, test_set)
```

Challenge: Text Classification

Perform text classification on movie_reviews corpus for positive and negative sentiment analysis

Hint:

from nltk.corpus import movie_reviews

Time: 15 mins

Save Model

```
import pickle
save_classifier = open("naivebayes.pickle","wb")
pickle.dump(classifier, save_classifier)
save_classifier.close()
```

Load Model

```
import pickle
classifier_f = open("naivebayes.pickle", "rb")
classifier = pickle.load(classifier_f)
classifier_f.close()
```

Scikit Learn Classifiers

- scikit-learn (http://scikit-learn.org) is a machine learning library for Python.
- It supports many classification algorithms, including SVMs, Naive Bayes, logistic regression, decision trees
- NLTK has a Scikit Learn API - SklearnClassifier

```
from nltk.classify.scikitlearn import SklearnClassifier
from sklearn.naive_bayes import MultinomialNB
classifier = SklearnClassifier(MultinomialNB())
```

Ex: Scikit Learn Classifiers

Compare the following SciKit Learn classifiers for gender classification

SklearnClassifier(MultinomialNB())

SklearnClassifier(LogisticRegression())

SklearnClassifier(SVC())

Time: 5mins

Evaluation

In order to decide whether a classification model is accurately capturing a pattern, we must evaluate that model.

There are 3 metrics for evaluation:

- Accuracy
- Precision and Recall
- Confusion Matrices

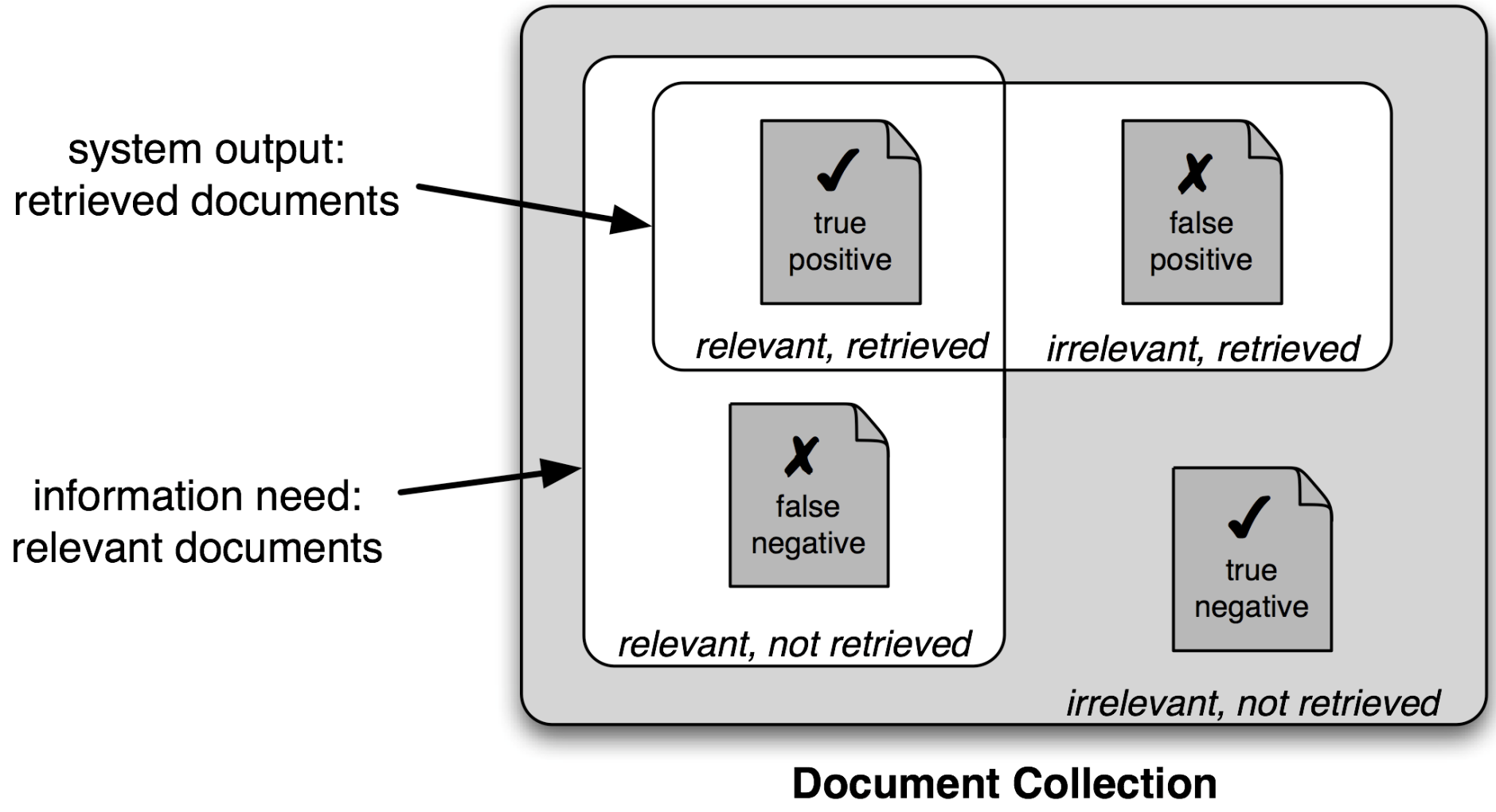
Accuracy

The simplest metric that can be used to evaluate a classifier, accuracy, measures the percentage of inputs in the test set that the classifier correctly labeled

```
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
```

Precision and Recall

Precision = TP/(TP+FP), Recall = TP/(TP+FN)



Confusion Matrix

- Confusion matrix is a table where each cell [i,j] indicates how often label j was predicted when the correct label was i.
- The diagonal entries indicate labels that were correctly predicted, and the off-diagonal entries indicate errors

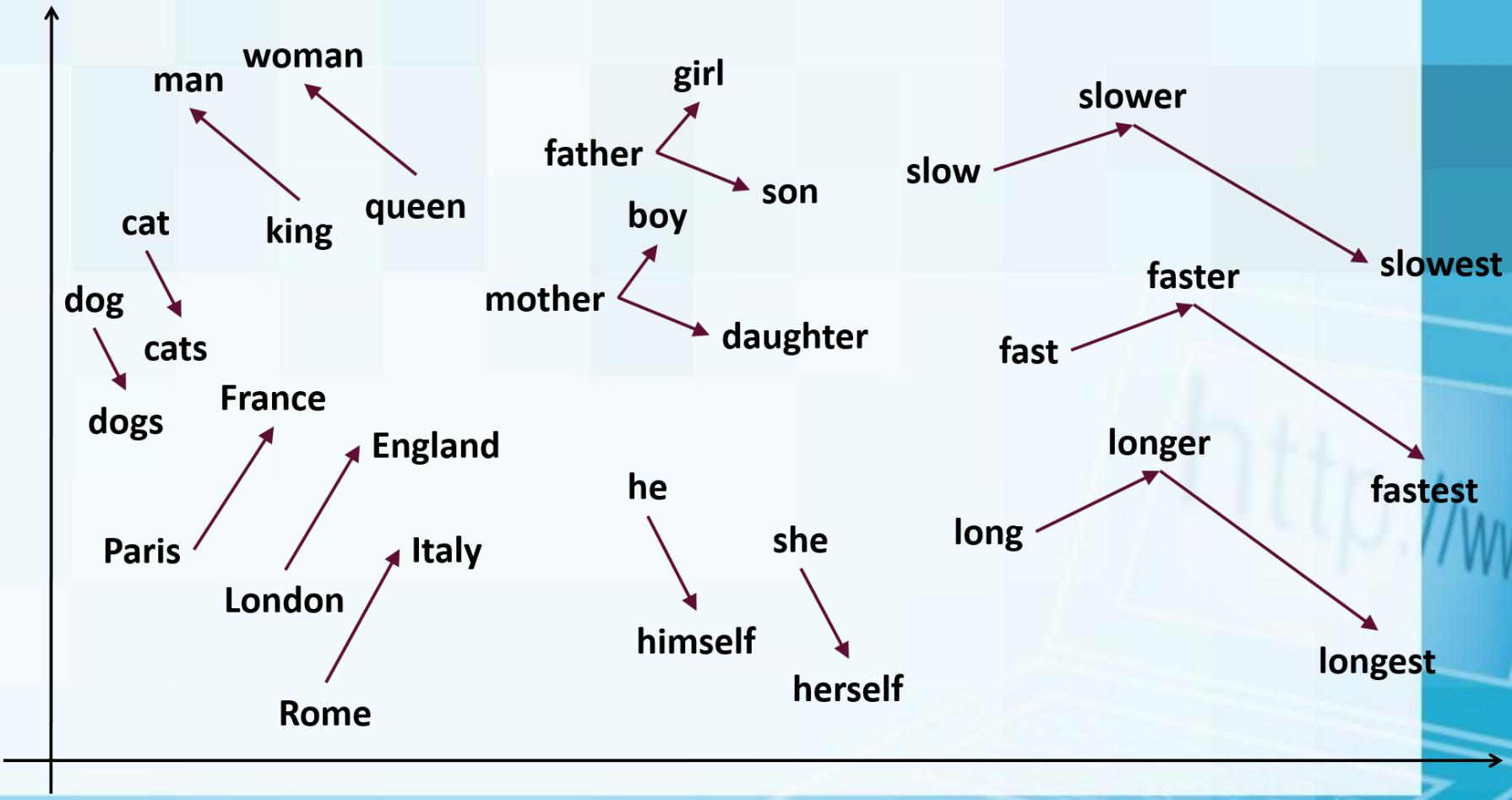
Module 5
Language Modelling

Word2Vec

- Traditional NLP treat words as atomic units, similar to indexes in vocabulary. There is no notion of similarity between words.
- Word to Vector (Word2Vec) represent words with similarity. Word2Vec was first proposed by Tomas Mikolov from Google in 2013

Word Similarities

- Many words are related.



N-gram Model

N-gram are basically a set of co-occurring words within a given window Eg "The quick brown fox jumps over the lazy dog.".

For N=2 (known as bigrams),

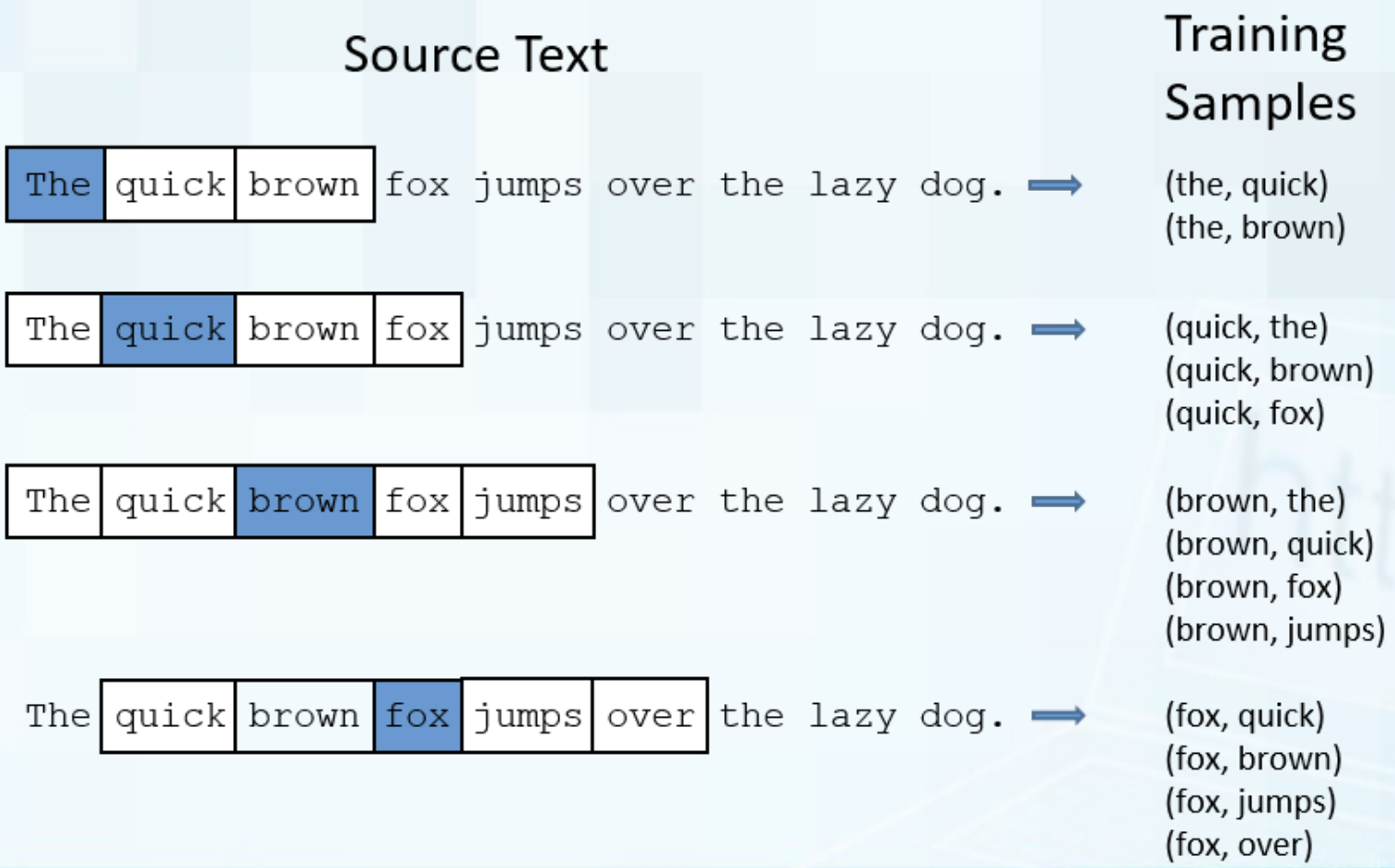
- The quick
- quick brown
- brown fox
- fox jumps
- jumps over
- over the
- the lazy
- lazy dog

Word2Vec

- N-gram model treat words as atomic units, similar to indexes in vocabulary. There is no notion of similarity between words.
- Word2Vec represent words with similarity.
- Word2Vec was first proposed by Tomas Mikolov from Google in 2013
- There are 2 methods to implement Word2Vec: Sink-gram and Continuous Bags-of-Words (CBOW).

Sink-gram Model

The quick brown fox jumps over the lazy dog.



Continuous Bags-of-Words Model

- Continuous Bags-of-Words (CBOW) remove the non-linear hidden layer and the projection layer is shared for all words.
- Order of words in the history does not influence the word vector.

History

1986 - Back Propagation for Learning (Rumelhart et al)

2003 - Neural Probabilistic Language Model (Bengio et al)

2008 - NLP (Collobert et al)

2013 - Word2Vec (Mikolov et al)

Grammar Parser

```
from nltk import CFG
grammar = 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 -> '
```

Resources

http://www.nltk.org/
http://www.nltk.org/book/

Summary
Parting Message

Q & A

Feedback

https://goo.gl/EDezXH

Thank You!

Amir Othman
0 11-3697 8924
othman.amir@gmail.com