Data Preparation

Jueun Lee, Subir Das, Jannik Schmitt, Vedant Shah, Maksim Urazov

Dataset

Basic Feature Extraction

	total_words	total_char	stopwords	total_punc	total_num	total_uppercase
0	9	60	5	0	0	0
1	28	161	13	1	0	0
2	19	107	8	0	0	1
3	17	117	5	0	0	0
4	15	100	6	0	0	0
5	22	161	8	0	0	0
6	5	27	0	0	0	5
7	21	128	9	0	0	0
8	19	124	6	0	0	1
9	7	57	1	0	0	0

Term Frequency (TF)

Term frequency is how many times a term appears in a particular document in corpus.

TF = (Number of times term T appears in the particular row) / (number of terms in that row)

Application : In any sector, well known example of TF is Google.

Inverse Document Frequency (IDF)

Inverse Document frequency is how many times a term appears in a particular document in corpus.

IDF(t) = log(Total number of rows / the number of rows in which the word was present).

Application : Retrieve information, Keyword Extraction

Term Frequency - Inverse Document Frequency

TF_IDF = Term Frequency * Inverse Document Frequency

	words	tf	idf	tf_idf
0	we	3279.0	4.503231	14766.095450
1	aperiodically.	1.0	10.190207	10.190207
2	our	2798.0	5.127612	14347.057336
3	customers	537.0	6.932110	3722.543136
4	what	593.0	7.625257	4521.777581

Bag-of-Words

Bag-of-Words(BOW) is a way of extracting features from the text to use in machine learning algorithms.

Example : Assume couple of sentence as separate document where 7 unique words.

“Today is sunny day”	*[1,1,1,1,0,0,0]*
“Today temperature is good”	*[1,1,0,0,1,1,0]*
“Today is presentation day”	*[1,1,0,1,0,0,1]*

List of all unique words to create vector
[“Today”, “is”, “sunny”, “day”, “temperature”, “good”, “presentation”]→ Assign 1 into vector if present, else 0

Part of Speech

Part of Speech find out how a word is used in the document. Dealt with English grammar eight main parts of speech - nouns, pronouns, adjectives, verbs, adverbs, prepositions, conjunctions and interjections.

Named Entity Recognition

Part-of-Speech(POS)-Tagging

• Labeling a word in a text (corpus) as corresponding to a POS, based on definition and context.

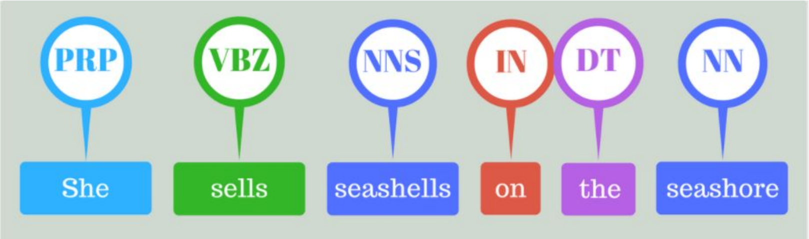
• Ambiguity:

“Give me your answer”, “answer” - noun

“Answer the question”, “answer” - verb

• Use cases: parse trees, base for tasks like NER (most named entities are Nouns), regexp, text-to-speech.

Pic: https://github.com/dhirajhr/POS-Tagging



Named Entity Recognition (NER)

- NER is an information extraction technique that automatically identifies named entities in an unstructured text and classifies them into predefined categories
- Person names, companies, locations, time, money, percentages, etc.
- Useful for analyzing unstructured text
    - Applications: emails, social media posts, customer support, online surveys, product reviews etc.

Methods

1. NLTK

- The Natural Language Toolkit is a suite of libraries and programs for symbolic and statistical NLP for English

- Classification, tokenization, stemming, tagging, parsing, etc.

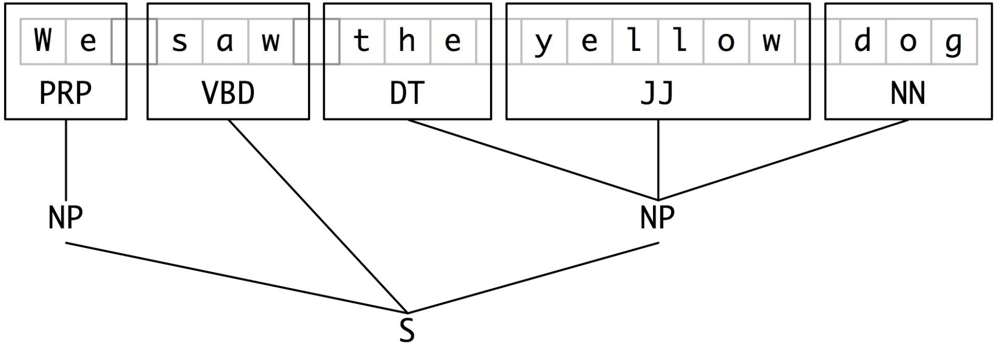
NLTK Workflow

1. Tokenize

   [This, is, an, example]
2. POS-tag

   NN-noun singular, JJ-adjective, DT-determiner, etc.
3. Chunk

   a DT + record NN = NP
4. Perform NER



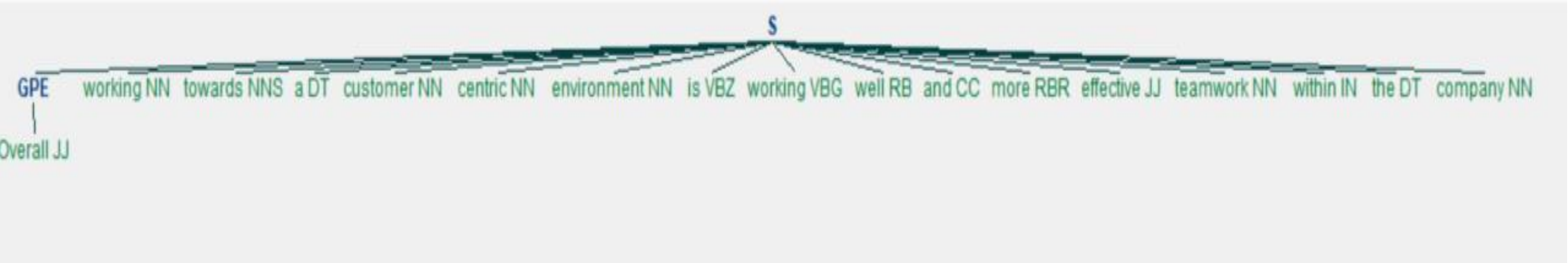
https://www.nltk.org/images/chunk-treerep.png

NLTK - NER

Overall - geopolitical entity (no)

Daimler AG - person (no-2)

```
{('AI', 'ORGANIZATION'),
 ('Apollo Project', 'PERSON')
 ('BYD', 'ORGANIZATION'),
 ('Baidu', 'GPE'),
 ('Daimler AG', 'PERSON'),
 ('Dongfeng', 'PERSON'),
 ('Ford', 'ORGANIZATION'),
 ('Grab', 'PERSON'),
 ('Honda', 'GPE'),
 ('Hyundai', 'PERSON'),
 ('Intel', 'ORGANIZATION'),
 ('Microsoft', 'PERSON'),
 ('Nvidia', 'GPE'),
 ('ZTE', 'ORGANIZATION')}
```



https://medium.com/@b.terryjack/nlp-pretrained-named-entity-recognition-7caa5cd28d7b

2. spaCy

- spaCy is one of the open_source NLP libraries, designed specifically for production use

- Useful for building information extraction or natural language understanding systems, or to pre-process text for deep learning.

- Tokenization, Part-of-speech (POS) tagging, lemmatization, Named Entity Recognition(NER) etc.

spaCy

Named Entity Recogntion with spaCy

- spaCy provides an efficient statistical system for named entity recognition

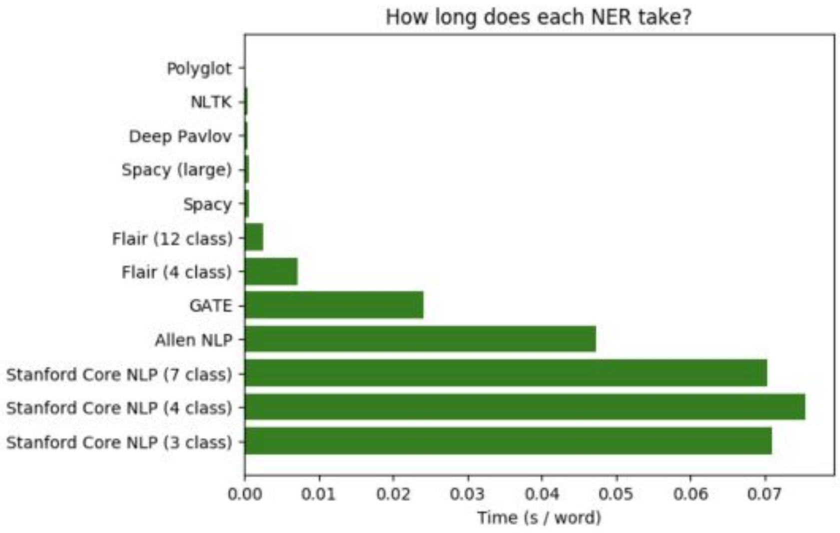
- The default model identifies a range of named and numeric entities
    - companies, locations, organizations, products etc.

- spaCy‘s models strongly depend on the examples they were trained on

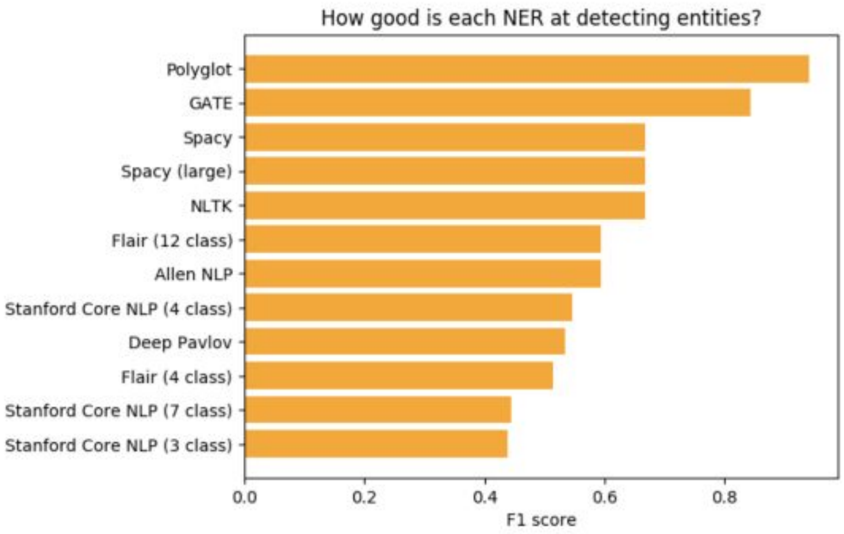
- Addition of arbitrary classes, by training the model with new examples.

Performance Comparison

- speed of predictions



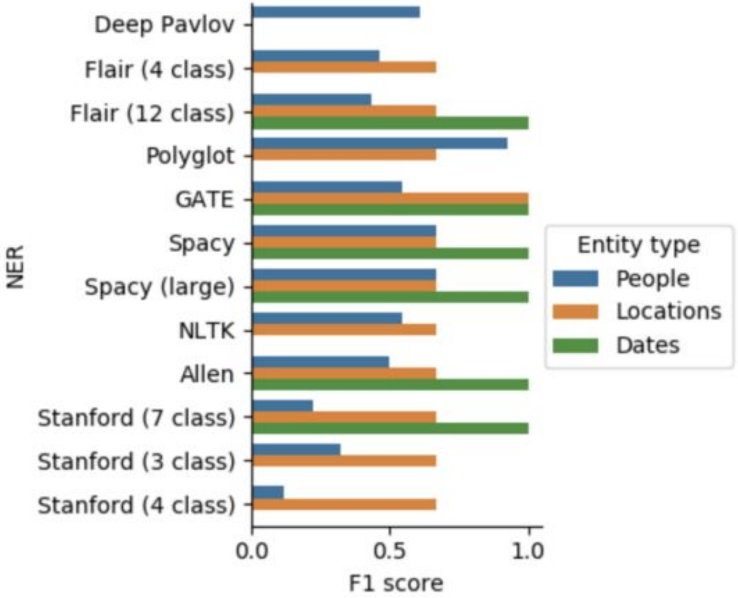
- average F1 score



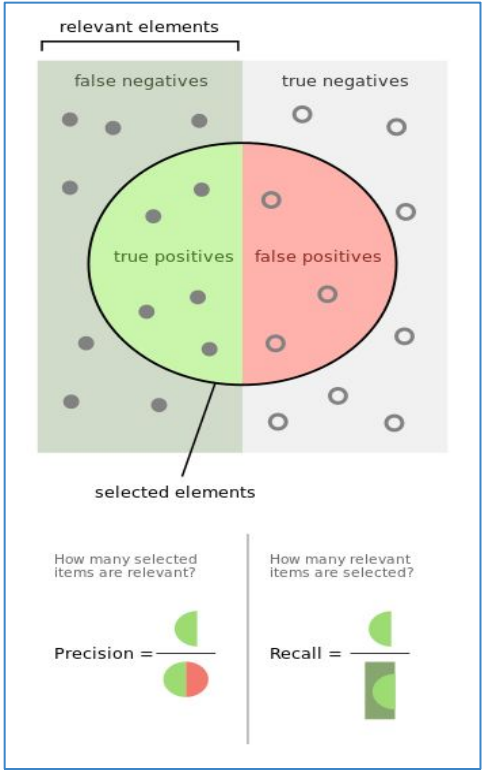
https://medium.com/@b.terryjack/nlp-pretrained-named-entity-recognition-7caa5cd28d7b

Performance Comparison

- F1 scores for People, Locations and Dates

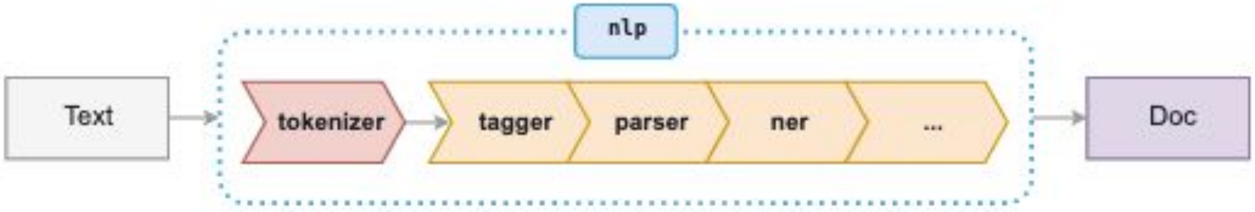


What is F1 score:



https://medium.com/@b.terryjack/nlp-pretrained-named-entity-recognition-7caa5cd28d7b
https://en.wikipedia.org/wiki/F1_score

Language Processing Pipelines



- Processing pipeline ‚nlp‘
  1) Tokenize a text to produce a Doc object
  2) The Doc is processed in several steps
- When ‘nlp’ is called on a text, the entire background pipeline returns the Doc objects.
- Named entities can be accessed as the ‘ents’ property of a Doc

https://spacy.io/usage/processing-pipelines

Named Entity Recogntion with spaCy

- Example data
    - ‘Apple is looking at buying U.K. startup for $1 billion‘

TEXT	START	END	LABEL	DESCRIPTION
Apple	0	5	ORG	Companies, agencies, institutions.
U.K.	27	31	GPE	Geopolitical entity, i.e. countries, cities, states.
$1 billion	44	54	MONEY	Monetary values, including unit.

https://spacy.io/usage/linguistic-features

Named Entity Recogntion with spaCy

- Example data

	report_grouping	question_text	comments	ner
0	Large Department	Please tell us what is working well.	we do what our customers need, we communicate aperiodically.	[]
1	Large Department	Please tell us what is working well.	customs business development continues to grow and expand, through the use of our internal business network & the team of people which are in place at the moment	[(customs business development, ORG)]
2	Large Department	Please tell us what is working well.	i think the team work hard, are committed to continuous improvement and doing a good job for our customers.	[]
3	Large Department	Please tell us what is working well.	overall working towards a customer centric environment is working well and more effective teamwork within the company	[]
4	Large Department	Please tell us what is working well.	customer centricity is a growing culture in the company creating a very positive customer experience	[]

3. A Multiple Label Classification

- **Aim:** We are trying to classify words to labeled entities.
- **Objective**
  - To have a model to identify entities from the data provided to us.
  - To have a classification model with pre-defined labels
  - A model functional for multiple classifications labels

The Training Data

- Taking an already existing data which is pre annotated.
- The dataset without the non-NER’s = 160667
- The tokens are 1 word tokens split based on white-spacing. Except for labels where multiple words are essential for an entity like organizations.
- The accuracy of the model: 0.8064
- List of Entities in the training Data as of now with their Corresponding Descriptions

Dataset

Tags	Description
O	All tokens that are not a Entity
geo	Geographical Entity
gpe	Organization
per	Person
org	Geopolitical Entity
tim	Time indicator
art	Artifact
nat	Event
eve	Natural Phenomenon

To Be Added Tags
Telephone
Email Address

Vectorization

- We use Term Frequency - Inverse Document Frequency (Tf-idf) method to vectorized the tokenized data.
- Term Frequency is the Number of times the terms are present in the corpus multiplied with,
- Inverse Document Frequency is the number of documents where the term is present.

Another Method

Count Vectorizer function from sklearn package.
Convert a collection of text documents to a matrix of token counts and produces a sparse matrix.

Keras Dense Model and Predicting Classes

- We create a basic sequential model of Recurrent Neural Network Layers.
- We run all the Labels into the learning together.
- The model predicts the probability of the word for each label trained.
- The label with the highest probability is taken to be the classified label for the token
- The precision and recall for the model right now:

Precision and Recall

	Predicted	Actual_Tag
0	0	[geo]
1	2	[per]
2	1	[geo]
3	0	[geo]
4	1	[org]
...	...	...
14951	1	[org]
14952	0	[geo]
14953	0	[geo]
14954	1	[org]
14955	2	[per]

	precision	recall	f1-score	support
0	0.85	0.95	0.90	7540
1	0.86	0.71	0.78	4065
2	0.94	0.90	0.92	3351

Tag Number	Tags
0	geo
1	org
2	per

NER: Road Ahead:

- Adding Entity label which are not in the data set like telephone numbers and email addresses.
- Test the model on the actual dataset.
  - Use N-Grams for tokenizing the comments and vectorize N-Grams
  - Check the model efficiency
- The model can be modified:
  - The training set can have complete sentence as an input and one labelled output. This means we will have multiple models for each entity.
  - Use LSTM- CRf Model. Conditional Random Field (CRF) model. It is a probabilistic graphical model that can be used to model sequential data such as labels of words in a sentence.

Spelling Correction

Background

As you have surely seen yourself, the dataset is riddled with typos.

Three types of Problems:

1. **Non-word errors**, e.g. “*lihgt*” instead of “*light*”
2. **Abbreviations**, e.g. “*u*” instead of “*you*”
3. **Real-word errors,** e.g. “*tree*” instead of “*three*”

Three steps to a clean dataset:

1. identify misspelled word
2. based off of the misspelled word, find a list of potential candidates that could be “correct”
3. choose the best candidate

Current Solution: Norvig's Algorithm

- Initial Approach: Python Library pyspellchecker (based on a simple algorithm by Peter Norvig)

- achieved accuracy of 0.71 in his own test[1]
  - corpus was based on 6.5 MB of text data
  - pyspellchecker corpus based on 19.1 MB of text data → promises performance improvements

- uses frequency list with roughly 16.5 million words to detect errors
  - error-detection is context-insensitive, meaning that it considers all words contained in the language model as "correct" and all words not contained in the language model as incorrect

1. https://norvig.com/spell-correct.html

Current Solution: Norvig's Algorithm

Possible corrections are generated by editing the incorrect word using one (or two) of the following **operations**:

deletion	remove one letter	"erxample" → "example"
transposition	swap two adjacent letters	"examlpe" → "example"
replacement	change one letter to another	"axample" → "example"
insertion	add a letter	"exampe" → "example"

Applying these operations yields 54n+25 permutations for a word of length *n*. That means that the word "excessive" has **511** candidates <u>only within editing distance of 1.</u>

Current Solution: Norvig's Algorithm

Selecting the most likely candidate (pseudocode):

```
if word is known:
    return word
else if a candidate in edit1_distance_suggestions is known:
    return all known candidates in edit1_distance_suggestions
else if a candidate in edit2_distance_suggestions is known:
    return all known candidates in edit2_distance_suggestions
else: return word
```

Current Solution: Norvig’s Algorithm

<u>Selecting the most likely candidate (pseudocode):</u>

```
if word is known:
    r
else
    return all known candidates in edit1_distance_suggestions
else if a candidate in edit2_distance_suggestions is known:
    return all known candidates in edit1_distance_suggestions
else: return word
```

Whichever list of candidates is returned, it will be sorted in descending order of probability of the word occurring overall (based on our frequency list of words).

Where Norvig's Algorithm Fails Us

case	example: typed out word	example: suggested correction	example: actually correct word
correct word gets "in-corrected"	brexit	credit	brexit
changed into opposite meaning	irespective	respective	irrespective
changed into something semantically different	costumers	costumes	customers
no correction	workarunds	workarunds	workarounds

Next Steps

1. **Develop a labelled corpus based on our data.**
   This is necessary mostly for performance evaluation of different approaches.

2. **Evaluate the most successful means of finding the correct spelling.**
   Two sub-problems: candidate generation and candidate selection.

3. **Evaluate the most accurate means of error detection.**
   Context-agnostic and computationally lightweight vs. contextual and computationally demanding?

Currently in Development

1. Generate frequency list of used words in our dataset.
2. Train FastText word embeddings on our dataset.
3. Iterating through our frequency list (in descending order of frequency), query for nearest neighbors in our word embeddings.
4. Check nearest neighbors against a list of criteria for spelling mistakes; if criteria are met, store word-typo pair in a dict. e.g. (“irrespective”: [“irespective”, “irrrespective”])
5. Create an inverse lookup dict to find the “correct” word given an incorrect one.

Currently in Development

(Preliminary) Criteria for Spelling Mistakes[1]

1. Word must not be a known English word.
2. Mistake must occur at least n times.
3. Must have edit distance ≤ 2.
4. High vector similarity to the proper word (concrete threshold TBD).
5. Must have more than 3 characters.
6. Both words should have the first starting letter.

These criteria are inspired by a similar approach of Chirag Jain:
https://haptik.ai/tech/extract-spelling-mistakes-fasttext/

Currently in Development

<u>Advantage of this Approach</u>

Ideally, this approach significantly reduces the rate of False Positives.

- not considering unknown words as errors might increase usability on highly domain-specific data
- while generating the error dict is not computationally trivial, using the generated dict for spelling correction is

Currently in Development

Two To-Do's Down the Line

- this conservative **error detection** is context-insensitive
  - context-sensitive error detection will be the first thing to be tested once the demo of this approach is fully operational
- the demo currently in the works relies on frequency (like Peter Norvig's approach) for **candidate selection** as a baseline
  - alternative solutions will be evaluated and performance will be compared

Finally, please keep in mind that this is an exploratory approach; individual components (e.g. candidate generation) might be replaced by a different solution should performance not be satisfactory.

Currently in Development

<u>The Issue of Candidate Selection</u>

- this approach does not yet include a solution for candidate selection in scenarios where there are multiple candidates for a correct word

If you have any ideas for different approaches to candidate selection, now is the perfect time to discuss them!

Finally, please keep in mind that this is an explorative approach; individual components (e.g. candidate generation) might be replaced by a different solution should performance not be satisfactory.