NLP without Annotated Dataset

NLP Pipeline and Text Representation

Sowmya Vajjala

Seminar für Sprachwissenschaft, University of Tübingen, Germany

11 January 2021

Class Outline

- Quick recap of last class
- How do we build NLP systems?
- A typical NLP Pipeline
- Case study of a real-world NLP pipeline
- Building an NLP Pipeline - an example
- Sharing some personal experiences from an industry project.
- Feature Engineering- How do we represent text for NLP?

As per student suggestions, let us take a 5 min break each at about 45 min, and at about 1.5 hrs.

Last Class: Quick Recap

- What is NLP, where is it useful?
- Why is NLP difficult?
- Some common NLP tasks
- levels of language processing
- NLP methods

- Any questions on anything from the last class or about the course so far?

General Housekeeping

- New zoom room
- Messages about teams, termpapers etc.
- Participation in forums (and new forums)
- Breaks in between from now on (2 5 min breaks)
- Today's class also hopefully has more interaction/questions to you
- Some extra stuff (text representation) in slides today

What does NLP involve?

- Some basic linguistic ideas (e.g., words, morphology, part of speech, syntax etc.)
- Different tasks (e.g., POS tagging, parsing, coreference resolution, information extraction etc)
- Different applications (e.g., Machine Translation, Chatbots, Question Answering etc.)
- Many algorithms (e.g., naive bayes, HMMs, LSTMs, Transformers etc)

How do these come together when you build an NLP system for some industry use case?

Option 1: Utilize Existing Services

We don't always have to build everything ourselves. We can use a pay as you go service from a third party provider.
An example: Microsoft's machine translation API

```
import os, requests, uuid, json
subscription_key = "XXXX"
endpoint = "https://api-nam.cognitive.microsofttranslator.com"
path = '/translate?api-version=3.0'
params = '&to=de' #From English to German (de)
constructed_url = endpoint + path + params
headers = {
    'Ocp-Apim-Subscription-Key': subscription_key,
    'Content-type': 'application/json',
    'X-ClientTraceId': str(uuid.uuid4())
}

body = [{'text' : 'How good is Machine Translation?'}]
request = requests.post(constructed_url, headers=headers, json=body)
response = request.json()

print(json.dumps(response, sort_keys=True, indent=4, separators=(',', ': ')))
```

source: Practical NLP, Ch 7

Output


```
[
    {
    "detectedLanguage": {
          "language": "en",
          "score": 1.0
    },
    "translations": [
          {
               "text": "Wie gut ist maschinelle Ãbersetzung?",
               "to": "de"
          }
    ]
    }
]
```

Advantages and Disadvantages

- Advantage: You don't have to worry about setting stuff up, hiring a large NLP team, maintaining the NLP system etc.
- Disadvantages:
  1. This only works if you have problem that exactly meets the specifications of such an available API
  2. No possibility of customization/modification
  3. Depending on how much you use, costs may escalate

  Note: You still have to think whether this approach is a long term solution for your problem.

Short Exercise

- Spend 5 minutes on the internet and look for NLP services from Cloud providers such as Google, Microsoft, Amazon, Watson etc.
- Make a note of what sort of services do they provide for NLP applications.

What is more likely to happen, though?

- Say you have custom problem (e.g., classify customer tweets about your product into "actionable" and "non-actionable" ones.)
- Will any of these off the shelf solutions work?

What is more likely to happen, though?

- Say you have custom problem (e.g., classify customer tweets about your product into "actionable" and "non-actionable" ones.)
- Will any of these off the shelf solutions work?
- What do we need to design a solution for this problem??

What is more likely to happen, though?

- Say you have custom problem (e.g., classify customer tweets about your product into "actionable" and "non-actionable" ones.)
- Will any of these off the shelf solutions work?
- What do we need to design a solution for this problem??
- First of all, we need "data". What Data? for what?

What is more likely to happen, though?

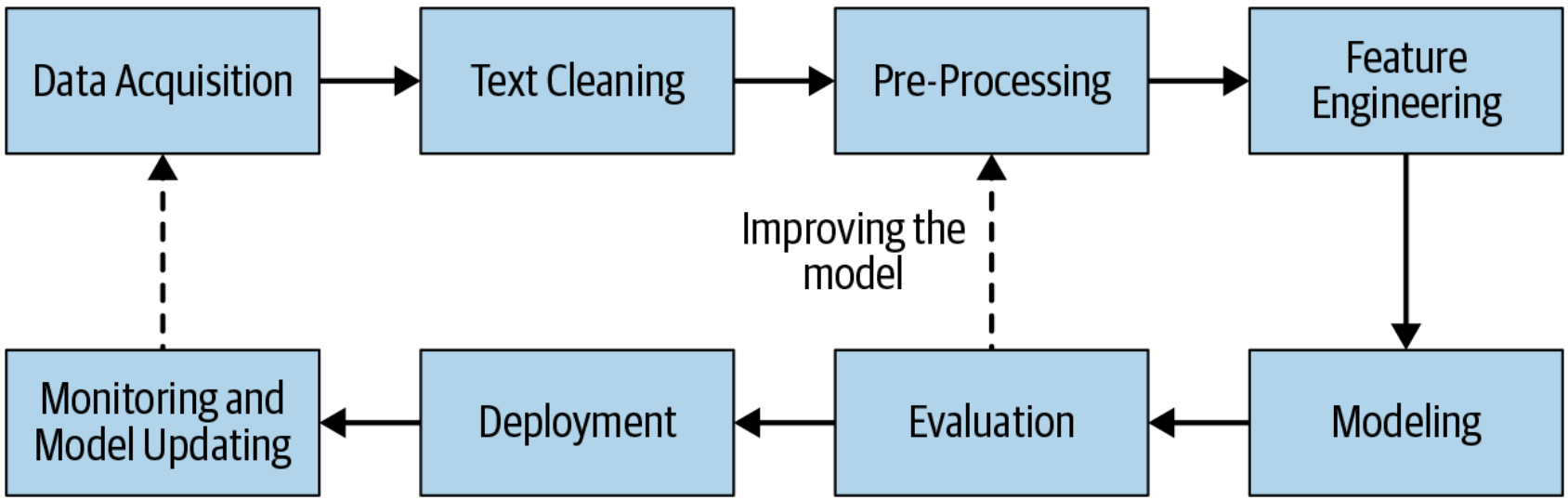
- Say you have custom problem (e.g., classify customer tweets about your product into "actionable" and "non-actionable" ones.)
- Will any of these off the shelf solutions work?
- What do we need to design a solution for this problem??
- First of all, we need "data". What Data? for what?
  1. To "train" NLP models
  2. To "evaluate" and validate NLP models

What is more likely to happen, though?

- Say you have custom problem (e.g., classify customer tweets about your product into "actionable" and "non-actionable" ones.)
- Will any of these off the shelf solutions work?
- What do we need to design a solution for this problem??
- First of all, we need "data". What Data? for what?
  1. To "train" NLP models
  2. To "evaluate" and validate NLP models
- ... and then?

NLP System Development Pipeline

A Common Case



Data Acquisition

- Where do we get our data from?

Data Acquisition

- Where do we get our data from?
- Let us say you are working in a software company, and they asked you to develop a customer ticket routing system that looks at a ticket text, and classifies it into three categories: technical, sales, other. Where will your data come from?

Data Acquisition

- Where do we get our data from?
- Let us say you are working in a software company, and they asked you to develop a customer ticket routing system that looks at a ticket text, and classifies it into three categories: technical, sales, other. Where will your data come from?
- In an ideal scenario, we have some historical data of customer tickets along with this routing information.
- But what if they were just routing to the right team, but not storing that information anywhere in the past? We don't have the training data we want!

How do we get our Data?

- use a public NLP dataset (e.g., https://datasets.quantumstat.com/)

How do we get our Data?

- use a public NLP dataset (e.g., https://datasets.quantumstat.com/)
- scrape the data from the web (e.g., customer support forums of other products, if they are available and are tagged with categories)
- work together with your customer support team and gradually collect the data you want.
- using pattern matching and other such methods to create some data to train your own models and gradually improve this baseline.
- setting up data annotation experiments

.... and so on.

Text Extraction and Cleaning

- What, according to you, is the format of data you see in NLP?

Text Extraction and Cleaning

- What, according to you, is the format of data you see in NLP?
- Data can come in all forms: PDF, Docs, HTML files, scanned png files, tables etc.
- Text extraction and cleanup refers to the process of extracting raw text from the input data by removing all the other non-textual information.
- Text extraction may not involve NLP per se, but it defines the rest of your NLP pipeline. Bad text extraction = Bad NLP system.

Text Extraction: Some facts

- PDF to text conversion is hard and imperfect. Not all pdfs can be efficiently parsed.

Text Extraction: Some facts

- PDF to text conversion is hard and imperfect. Not all pdfs can be efficiently parsed.
- When we are extracting text from images, We may see some characters not rendered properly, or some words extracted with spelling mistakes etc
- If all of the "dataset" is a large collection of pdf documents with scanned text along with some tables: it is like the worst of all NLP worlds :-)
- We don't have a single solution that works for all. There are tools like Amazon Textract, but they are not perfect.

Text Pre-processing

- Sentence segmentation and word tokenization

Text Pre-processing

- Sentence segmentation and word tokenization
- Stop word removal, stemming and lemmatization, removing digits/punctuation, lowercasing, etc.

Text Pre-processing

- Sentence segmentation and word tokenization
- Stop word removal, stemming and lemmatization, removing digits/punctuation, lowercasing, etc.
- Normalization, language detection, code mixing, transliteration, etc.

Text Pre-processing

- Sentence segmentation and word tokenization
- Stop word removal, stemming and lemmatization, removing digits/punctuation, lowercasing, etc.
- Normalization, language detection, code mixing, transliteration, etc.
- POS tagging, parsing, coreference resolution, etc.

(you probably are already familiar with some of these.)

Questions so far?

Feature Engineering

- The goal of feature engineering is to capture the characteristics of the text into a numeric vector that can be understood by the ML algorithms.
- There are primarily two ways of feature extraction in NLP:
  1. hand crafted features (e.g., number of words/sentences in a document, number of spelling errors/sentence etc.)
  2. automatically extracted features (e.g., Bag of words, Bag of N-grams etc.)

Modeling

- Heuristics based systems (e.g., regular expressions, rule based matching etc)
- Learning from data: Machine learning (e.g., logistic regression, support vector machines etc.), deep learning
- Model ensemble (combining predictions from multiple models)
- Model cascade (using one model's prediction as input to another model)

.....

Evaluation

How well is my NLP approach doing?

- Intrinsic evaluation focuses on intermediary objectives, while extrinsic focuses on evaluating performance on the final objective.

Evaluation

How well is my NLP approach doing?

- Intrinsic evaluation focuses on intermediary objectives, while extrinsic focuses on evaluating performance on the final objective.
- Consider a email spam classification system:
  1. Intrinsic evaluation will focus on measuring the system performance using precision and recall.

Evaluation

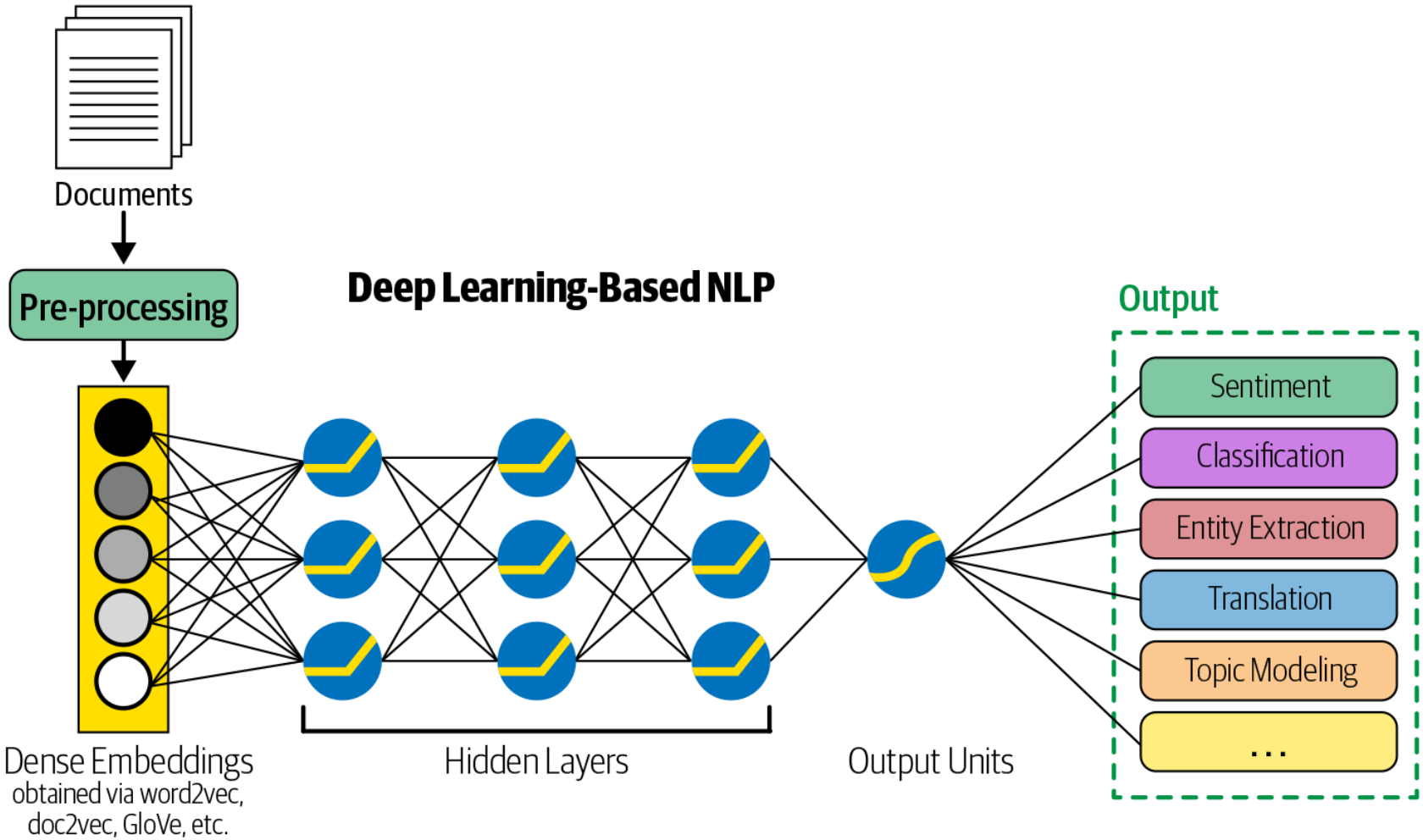
How well is my NLP approach doing?

- Intrinsic evaluation focuses on intermediary objectives, while extrinsic focuses on evaluating performance on the final objective.
- Consider a email spam classification system:
  1. Intrinsic evaluation will focus on measuring the system performance using precision and recall.
  2. Extrinsic evaluation will focus on measuring the time a user wasted because a spam email went to their inbox or a genuine email went to their spam folder.

Deployment, Monitoring and Model Updating

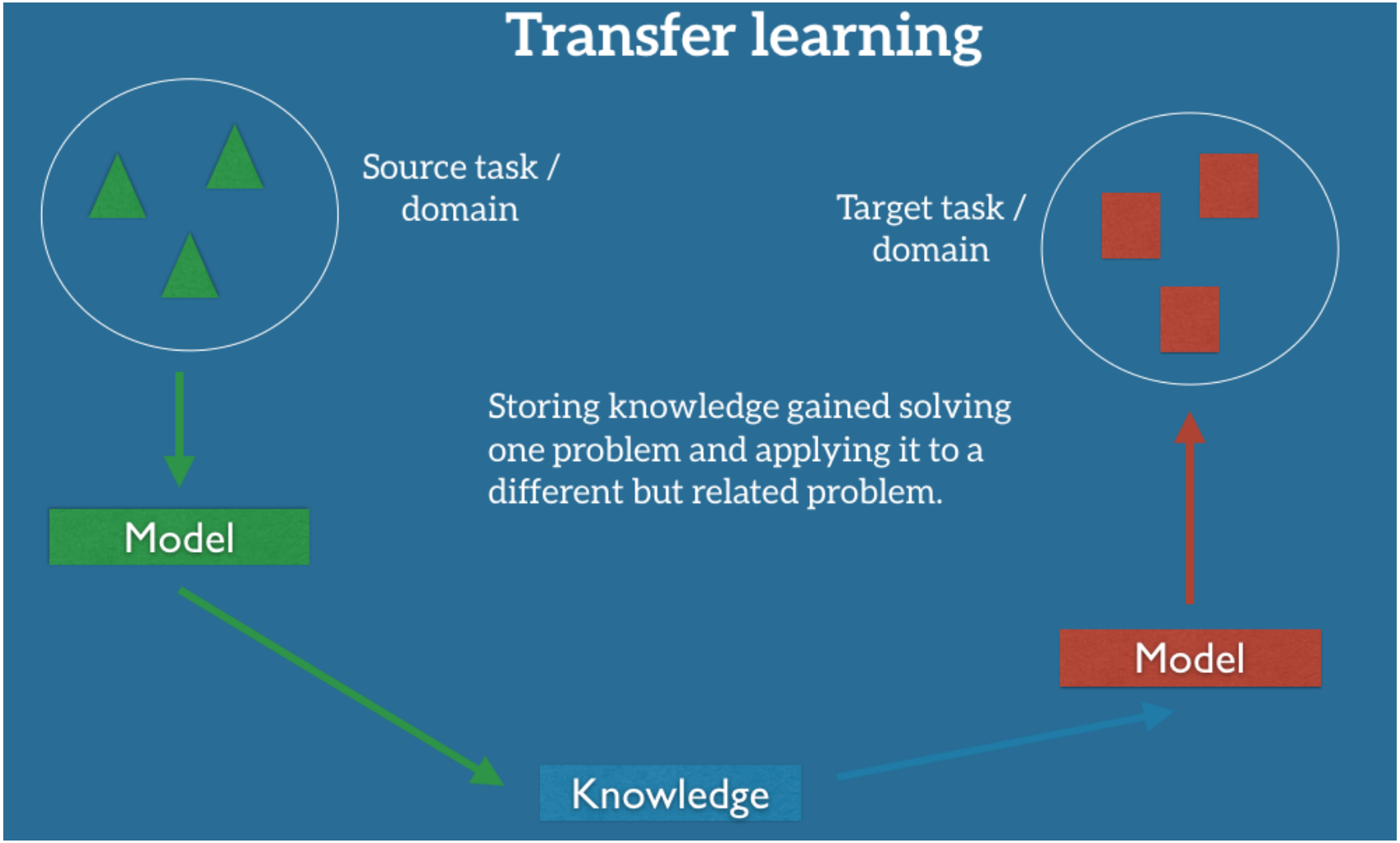
- Once we have a good model, we have to deploy it in the context of a larger system (e.g., spam classification is a part of an email software)
- A common approach: deploy the NLP system as a micro service/web service
- Model monitoring: e.g., using a performance dashboard showing the model parameters and key performance indicators
- Model updating: Model has to stay current, with changing data. So, we should have some way of regularly updating, evaluating and deploying a model.

Pipeline Variation: Deep Learning Pipeline



[source](https://blog.aylien.com/leveraging-deep-learning-for-multilingual/)

Pipeline Variation: Transfer Learning Pipeline



[source](https://ruder.io/transfer-learning/)

NLP Pipeline: Some Questions

- What is the difference between traditional pipeline and deep learning pipeline?
- What are some advantages and disadvantages of deep learning?
- What are some advantages and disadvantages of transfer learning?
- If you already know some ML/deep learning usage, how many steps of this pipeline did you learn/think about so far?

Pipeline Variation: Automated ML

- Automated ML is the process of automating the process of building machine learning based systems (provided you have your data in place) so that even those with minimal ML knowledge can also use it.
- The goal is to automate the iterative process of training/testing multiple models, parameter settings etc.
- Google, Microsoft etc provide AutoML options for various tasks.

Advantages and Disadvantages

Advantages:

- We don't need an expert machine learning or NLP team
- We don't have to worry about training/tuning the model well
- We can get by with writing minimum "machine learning" related code ourselves.

Disadvantages:

- This works only if we have a large amount of training data already in place.
- It is still hard to customize if we have some custom pre/post processing, feature engineering etc.

Short Exercise

- Spend 5 minutes on the internet and look for what AutoML feature of Google and Microsoft looks like.

A real-world NLP pipeline: Uber's COTA

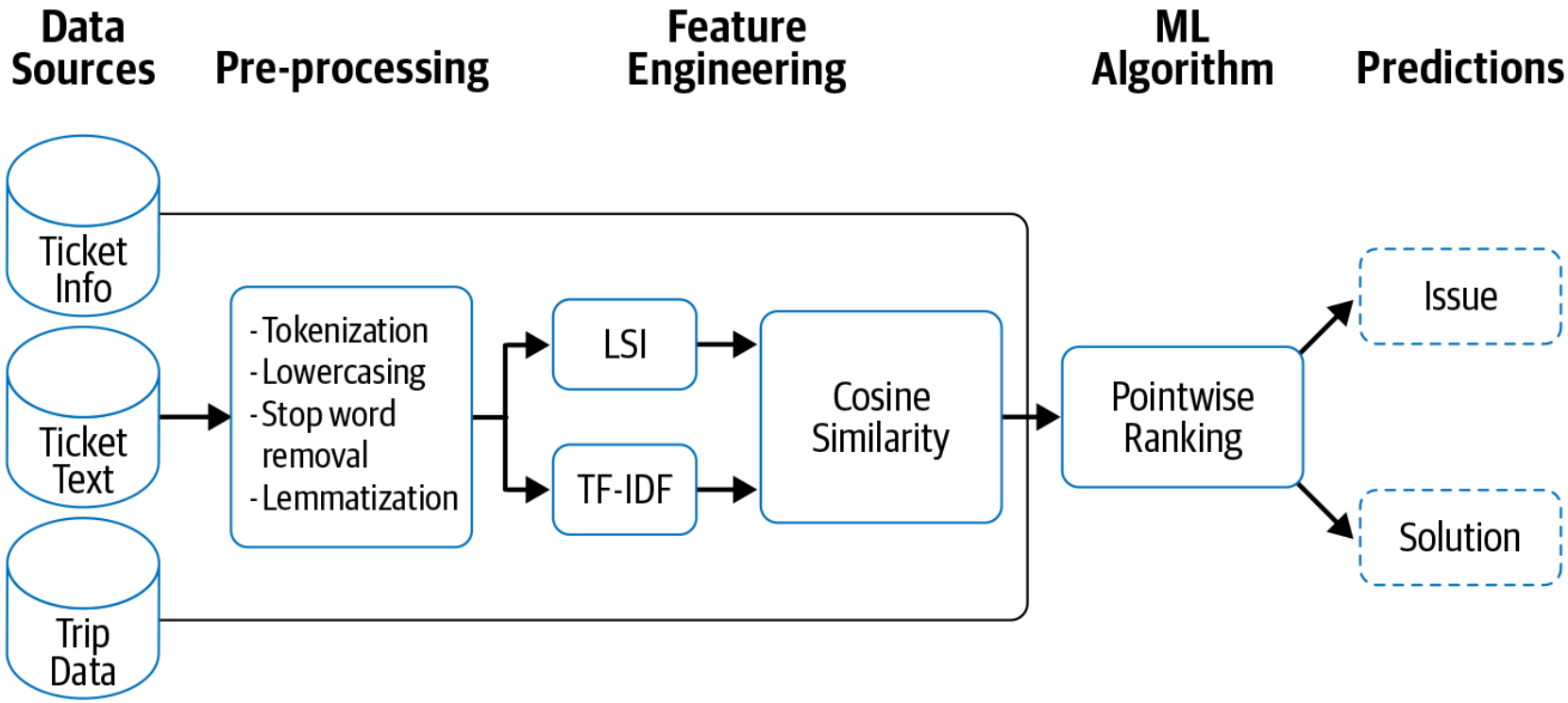
- It is a tool used within Uber to help agents do better customer support, by supporting quick and efficient issue resolution for a majority of Uber's support tickets.
- (i.e., instead of asking the customer to answer several questions related to the issue, automate that process so that agents can react more quickly.)

A real-world NLP pipeline: Uber's COTA

- It is a tool used within Uber to help agents do better customer support, by supporting quick and efficient issue resolution for a majority of Uber's support tickets.
- (i.e., instead of asking the customer to answer several questions related to the issue, automate that process so that agents can react more quickly.)
- Task: identify the issue type, and find out the right resolution based on ticket text, and other info such as trip data.

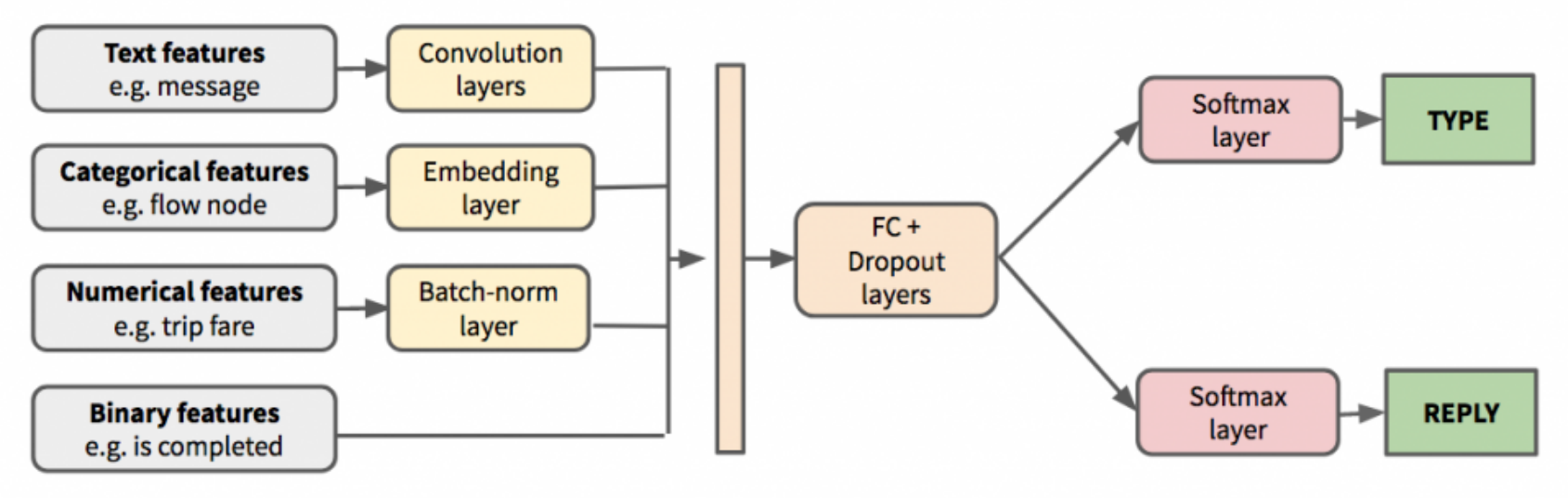
"COTA can reduce ticket resolution time by over 10 percent while delivering service with similar or higher levels of customer satisfaction"

Uber COTA V1



[source](https://eng.uber.com/cota/)

Uber COTA V2: A Deep Learning Pipeline



[source](https://eng.uber.com/cota-v2//)

NLP Pipeline: Some Questions

- What is one important thing you learnt from this quick look at COTA?

NLP Pipeline: Some Questions

- What is one important thing you learnt from this quick look at COTA?
- For me, it is:
  1. start with a relatively straight forward approach, and build incrementally.
  2. We don't have to start with deep learning.
  3. Have your extrinsic evaluation measures in place (here: speed of resolution or improved customer support experience etc) along with intrinsic ones

Few More Questions

- What according to you is the most important part of the NLP Pipeline?

Few More Questions

- What according to you is the most important part of the NLP Pipeline?
- Should our data to train an NLP system come from only a single source?

Few More Questions

- What according to you is the most important part of the NLP Pipeline?
- Should our data to train an NLP system come from only a single source?
- When are rule based "models" relevant?
- What can be a rule based approach to developing a sentiment analyzer?

A code Example: Sentiment Analyzer

- source code for Training and Using a model

Text Extraction and Pre-processing

```
#Read the training file and do the required pre-processing
def getData(file_path):
    #col1: Move cat. #last col: Text
    fh = open(file_path)
    texts = []
    cats = []
    fh.readline() #Header
    for line in fh:
        temp = line.split(",")
        text = " ".join(temp[1:])
        texts.append(" ".join(removepunct_tokenize_stem(text)))
        cats.append(temp[0])
    fh.close()
    return texts,cats
```

Text Extraction and Pre-processing

pre-processing

```
#Stemming. Optional.
def stem_tokens(tokens, stemmer):
    stemmed = []
    for item in tokens:
        stemmed.append(stemmer.stem(item))
    return stemmed

#Remove punctuation, and perform stemming - again, optional. Just a choice I made.
#You can build a classifier without doing these - it will work, may be not as efficiently.
def removepunct_tokenize_stem(text):
    text = "".join([ch for ch in text if ch not in string.punctuation]) #Remove punctuation
    tokens = word_tokenize(text)
    stemmer = PorterStemmer()
    final = stem_tokens(tokens, stemmer)
    return finals
```

Feature Engineering and Model Building

```
#Learns a classifier with the specified features in CountVectorizer
#and saves the model so that we can use it again to make predictions
def save_model(texts,cats,model_file):
    vectorizer =  CountVectorizer(analyzer = "word", tokenizer = None,
                  preprocessor = None, stop_words = None, ngram_range=(1,2), min_df=10)
   # classifier =  LinearSVC(max_iter=500)
    classifier = LogisticRegression(max_iter=100,class_weight="balanced",random_state=1234)
    pipeline = Pipeline([('vectorizer', vectorizer), ('pac', classifier)])
    pipeline.fit(texts,cats)
    joblib.dump(pipeline, model_file)
    print("Model saved as: ", model_file)
```

Notes

We normally explore:

- multiple vectorization methods (single words, characters, word ngrams etc), multiple settings within one method (e.g., top 1000 features, top 5000, with or without certain pre-processing etc) (More on this at the end of today's class)
- multiple training approaches (e.g., logistic regression, decision trees etc) and different parameter settings (e.g., number of training iterations, any other tunable parametres etc)
- various evaluation measures (e.g., accuracy, F1 score, precision, recall etc)

... to build many different models and choose the best one, and save that one.

Using the Classifier

```
#Makes prediction for a single string.
def makePredictionsForAString(inputstr,classifier):
    converted = [" ".join(removepunct_tokenize_stem(inputstr))]
    all_predictions = list(classifier.predict_proba(converted)[0])
    predict_prob = max(all_predictions)
    print(predict_prob)
    prediction = classifier.predict(converted)[0]
    print(inputstr[:50]+"...", prediction)
```

NLP Pipeline: A Personal Experience

Scenario: NER for legal documents (e.g., agreements etc) which are PDFs.

NLP Pipeline: A Personal Experience

Scenario: NER for legal documents (e.g., agreements etc) which are PDFs.
Our options:

- use an off the shelf solution, tune it to your domain, and then deploy

NLP Pipeline: A Personal Experience

Scenario: NER for legal documents (e.g., agreements etc) which are PDFs.
Our options:

- use an off the shelf solution, tune it to your domain, and then deploy
- build your own NER model, using rules

NLP Pipeline: A Personal Experience

Scenario: NER for legal documents (e.g., agreements etc) which are PDFs.
Our options:

- use an off the shelf solution, tune it to your domain, and then deploy
- build your own NER model, using rules
- build your own NER using machine learning/deep learning

Off the shelf NER

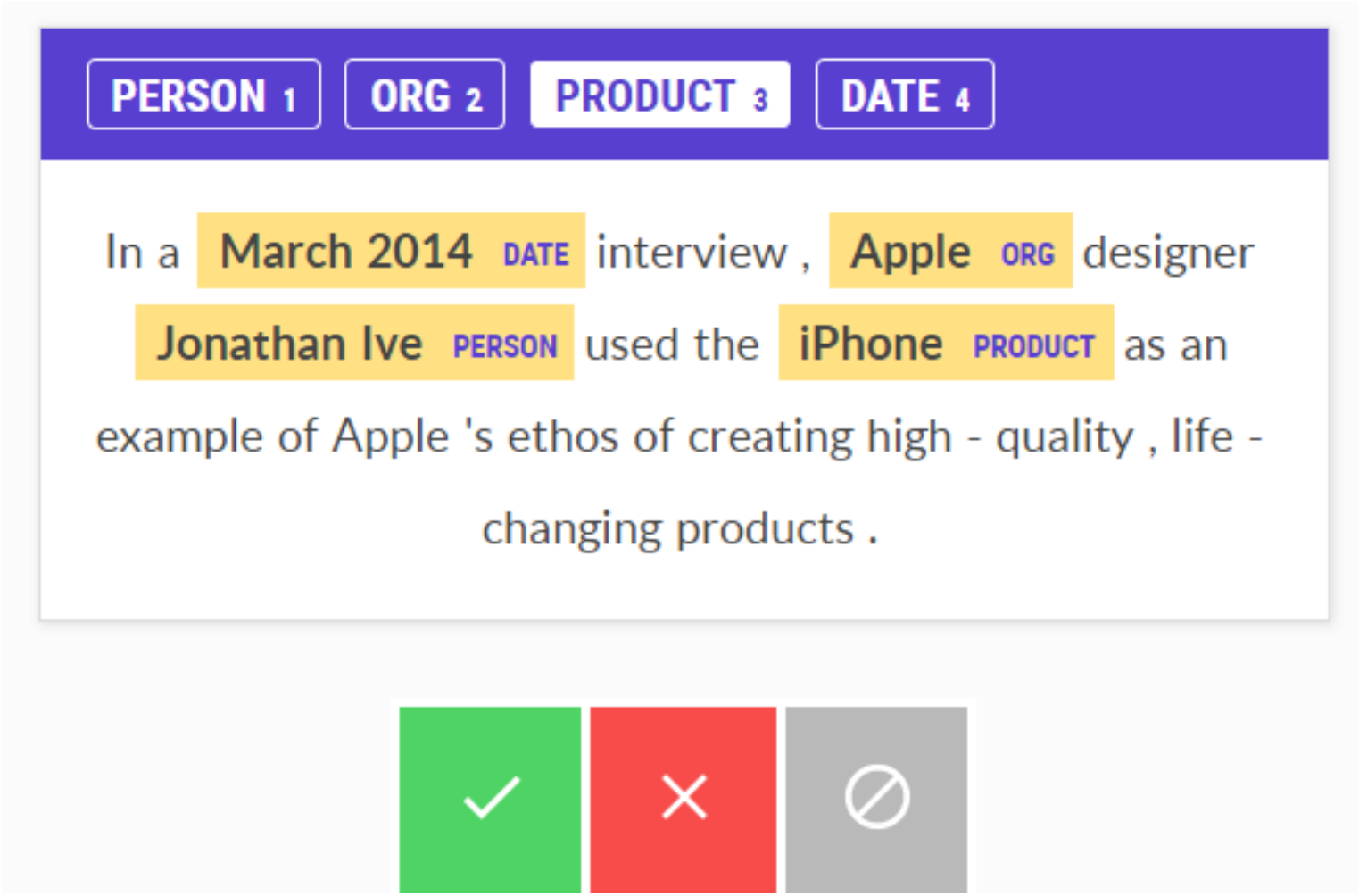
```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")

for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)

Output:
Apple 0 5 ORG
U.K. 27 31 GPE
$1 billion 44 54 MONEY
```

off the shelf NER + Adapt to your domain

using "active learning": annotate examples that the model does not know yet, and the model adapts, as it sees them.



https://prodi.gy/docs/named-entity-recognition

Build your own NER Model using rules

- a lookup table with a large collection of names of people/organizations/etc relevant for your organization

Build your own NER Model using rules

- a lookup table with a large collection of names of people/organizations/etc relevant for your organization
- a bunch of hand crafted rules (e.g., a proper noun followed by "was born" may indicate a person) etc.

Spacy's Entity Ruler is a useful tool for this kind of an approach.

Build your own NER model using machine/deep learning

- using feature engineering + machine learning
- using spacy's training pipeline (deep learning)
- using transfer learning

What we did

- Explored various NLP libraries and cloud providers. We figured they don't do so well for our use case.
- Trained our own NER with a standard dataset (CONLL-03), and with simple features, as a base NER model.
- Used prodi.gy for domain specific annotation, and added these examples to the existing model to update it (and tune it for legal docs)
- Added some rules/heuristics on top of a machine learning model
- Explored a model ensemble with 2-3 models
- Looked at training with noisy text (i.e., output of pdf to text conversion + sentence/word segmentation)

... ...

Some Observations

- off the shelf NER is not perfect, but it is a good starting point.
- If you have only a small amount of annotated data for your domain, you can explore transfer learning, and gradually collect more domain specific data.
- A reliable approach afterwards: rules + (feature engineering + model)
- If you want to use deep learning, check if it is better than the above process first (for any NLP problem)
- Remember: your implementation should also be deployable, not just accurate.

... ...

More observations

- There is no single model. We have to set up a model monitoring/updating pipeline using some evaluation criteria.
- There is no single training set or test set. They will also evolve with time.

More observations

- There is no single model. We have to set up a model monitoring/updating pipeline using some evaluation criteria.
- There is no single training set or test set. They will also evolve with time.
- NLP tools we use in our pipeline are not perfect. Even a simple thing as text extraction or tokenization can have many unresolved issues. While our models are all very valuable effort, these steps are, too.

More observations

- There is no single model. We have to set up a model monitoring/updating pipeline using some evaluation criteria.
- There is no single training set or test set. They will also evolve with time.
- NLP tools we use in our pipeline are not perfect. Even a simple thing as text extraction or tokenization can have many unresolved issues. While our models are all very valuable effort, these steps are, too.
- No model can solve the problem of data quality. So, focus on getting good quality data to solve your problem first.

More observations

- There is no single model. We have to set up a model monitoring/updating pipeline using some evaluation criteria.
- There is no single training set or test set. They will also evolve with time.
- NLP tools we use in our pipeline are not perfect. Even a simple thing as text extraction or tokenization can have many unresolved issues. While our models are all very valuable effort, these steps are, too.
- No model can solve the problem of data quality. So, focus on getting good quality data to solve your problem first.
- Build a solution incrementally. Don't jump into the most complex solution first. Eventually, you want your stuff to be reliable, and not too expensive to maintain in short/long term.

... ...

Summary

- When we read about NLP in the news/research updates, we usually see models everywhere.
- However, there is much more behind an NLP system
- I hope this session introduced some of the issues behind the scenes.

A (not so quick) detour: Text Representation

In "Feature Engineering", I said:

- There are primarily two ways of feature extraction in NLP:
  1. hand crafted features (e.g., number of words/sentences in a document, number of spelling errors/sentence etc.)
  2. automatically extracted features (e.g., Bag of words, Bag of N-grams etc.)

  - What exactly do these mean??

What is text representation?

- Any form of data text/image/video/audio should be converted into some form of numeric representation so that it can be processed by NLP/Machine Learning algorithms later
- In NLP, this conversion of raw text to a suitable numerical form is called text representation
- We will use text representation and feature extraction/representation as synonyms in this course.

How is text different from any other form of data?

- How do we do feature extraction with images? An image is stored in a computer as a matrix of pixels where each cell[i,j] in the matrix represents pixel i,j of the image. This matrix representation accurately represents the complete image.

How is text different from any other form of data?

- How do we do feature extraction with images? An image is stored in a computer as a matrix of pixels where each cell[i,j] in the matrix represents pixel i,j of the image. This matrix representation accurately represents the complete image.
- A video can be seen as a sequence of frames/images. So, it can be represented as a sequential collection of matrices.

How is text different from any other form of data?

- How do we do feature extraction with images? An image is stored in a computer as a matrix of pixels where each cell[i,j] in the matrix represents pixel i,j of the image. This matrix representation accurately represents the complete image.
- A video can be seen as a sequence of frames/images. So, it can be represented as a sequential collection of matrices.
- Consider speech—it is transmitted as a wave. To represent it mathematically, we sample the wave and record its amplitude (height).

What about text?

Hand crafted features to represent text

- When we know the domain well, it is possible to represent text as a collection of hand crafted features.
- e.g., when I am categorizing sentiment, I can create a list of positive words, and a list of negative words, and use the % of positive and negative words in a document as its two dimensional text representation.
- Assuming we come across a new problem/dataset, and we don't know where to begin, what can we do?

Hand crafted features to represent text

- When we know the domain well, it is possible to represent text as a collection of hand crafted features.
- e.g., when I am categorizing sentiment, I can create a list of positive words, and a list of negative words, and use the % of positive and negative words in a document as its two dimensional text representation.
- Assuming we come across a new problem/dataset, and we don't know where to begin, what can we do?
- in NLP, it is common to use what I will call today as **automatic text representations**, where we don't have to explicitly encode any information.

Consider a Toy corpus

Table 3-1. Our toy corpus

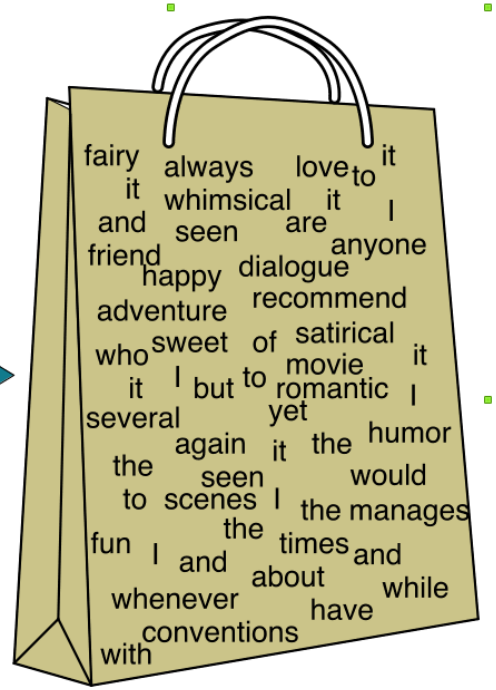
D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

Let us start with some simple text representations

- The vocabulary in this corpus, ignoring case-differences and punctuation consists of 6 words: [dog, bites, man, eats, meat, food]
- We can organize the vocabulary in any order. In this example, we simply take the order in which the words appear in the corpus.
- Every document in this corpus can now be represented with a vector of size six.

The Bag of Words Representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

source: https://web.stanford.edu/~jurafsky/slp3/4.pdf

Bag of Words

- for our toy corpus, where the word IDs are dog = 1, bites = 2, man = 3, meat = 4 , food = 5, eats = 6
- D1 ("Dog bites man") becomes [1 1 1 0 0 0]
- D4 ("Man eats food") becomes [0 0 1 0 1 1].

Code for BoW representation

```
from sklearn.feature_extraction.text import CountVectorizer

documents = ["Dog bites man.", "Man bites dog.", "Dog eats meat.", "Man eats food."]

#remove punctuation and lowercase words:
processed_docs = [doc.lower().replace(".","") for doc in documents]

count_vect = CountVectorizer()

#Build a BOW representation for the corpus
bow_rep = count_vect.fit_transform(processed_docs)

#Look at the vocabulary mapping
print("Our vocabulary: ", count_vect.vocabulary_)

#See the BOW rep for first 2 documents
print("BoW representation for 'dog bites man': ", bow_rep[0].toarray())
print("BoW representation for 'man bites dog: ",bow_rep[1].toarray())

#Get the representation using this vocabulary, for a new text
temp = count_vect.transform(["dog and dog are friends"])
print("Bow representation for 'dog and dog are friends':",

temp.toarray())
```

Bag of Words ...

- If we run this code, we'll notice that the BoW representation for a sentence like "dog and dog are friends" has a value of 2 for the dimension of the word "dog," indicating its frequency in the text.

Bag of Words ...

- If we run this code, we'll notice that the BoW representation for a sentence like "dog and dog are friends" has a value of 2 for the dimension of the word "dog," indicating its frequency in the text.
- Sometimes, we don't care about the frequency of occurrence of words in text and we only want to represent whether a word exists in the text or not. Researchers have shown that such a representation without considering frequency is useful for sentiment analysis.

Bag of Words ...

- If we run this code, we'll notice that the BoW representation for a sentence like "dog and dog are friends" has a value of 2 for the dimension of the word "dog," indicating its frequency in the text.
- Sometimes, we don't care about the frequency of occurrence of words in text and we only want to represent whether a word exists in the text or not. Researchers have shown that such a representation without considering frequency is useful for sentiment analysis.
- In such cases, we just initialize CountVectorizer with the binary=True option keeping everything else the same.

```
count_vect = CountVectorizer(binary=True)
```

- This results in a different representation for the same sentence.

Bag of Words ...

- If we run this code, we'll notice that the BoW representation for a sentence like "dog and dog are friends" has a value of 2 for the dimension of the word "dog," indicating its frequency in the text.
- Sometimes, we don't care about the frequency of occurrence of words in text and we only want to represent whether a word exists in the text or not. Researchers have shown that such a representation without considering frequency is useful for sentiment analysis.
- In such cases, we just initialize CountVectorizer with the binary=True option keeping everything else the same.

```
count_vect = CountVectorizer(binary=True)
```

- This results in a different representation for the same sentence.

Advantages of BoW

- BoW is fairly simple to understand and implement.
- Documents sharing vocabulary have their vector representations closer to each other in Euclidean space. In our toy corpus, distance between D1 and D2 is 0 as compared to the distance between D1 and D4, which is 2.
- We have a fixed-length representation for any sentence of arbitrary length.

Disadvantages of BoW

- The size of the vector increases with the size of the vocabulary. One way to control this is to take the first N most frequent words in the entire corpus.
- t does not capture the similarity between different words that mean the same thing. Say we have three documents: “I run”, “I ran”, and “I ate”. BoW vectors of all three documents will be equally apart.
- This representation does not have any way to handle out of vocabulary words (i.e., new words that were not seen in the corpus that was used to build the vectorizer).
- it is a “bag” of words—word order information is lost in this representation. Both D1 and D2 will have the same representation in this scheme.

Bag of N-grams

- BoN breaking text into chunks of n contiguous words (or tokens) called n-grams, instead of individual words
- This can help us capture some context,
- The corpus vocabulary, V, is then nothing but a collection of all unique n-grams across the text corpus.
- Then, each document in the corpus is represented by a vector of length —V—. This vector simply contains the frequency counts of n-grams present in the document and zero for the n-grams that are not present.

Bag of N-grams with the Toy Corpus

- The set of all bigrams in the corpus is as follows: dog bites, bites man, man bites, bites dog, dog eats, eats meat, man eats, eats food
- The bigram representation for the first two documents is as follows: D1 : [1,1,0,0,0,0,0,0], D2 : [0,0,1,1,0,0,0,0]

Bag of N-grams with the Toy Corpus

- The set of all bigrams in the corpus is as follows: dog bites, bites man, man bites, bites dog, dog eats, eats meat, man eats, eats food
- The bigram representation for the first two documents is as follows: D1 : [1,1,0,0,0,0,0,0], D2 : [0,0,1,1,0,0,0,0]
- Note that the BoW scheme is a special case of the BoN scheme, with n=1. n=2 is called a "bigram model," and n=3 is called a "trigram model." and so on.

Bag of N-grams code

```
#n-gram vectorization example with count vectorizer and
#uni, bi, trigrams
count_vect = CountVectorizer(ngram_range=(1,3))

#Build a BOW representation for the corpus
bow_rep = count_vect.fit_transform(processed_docs)

#Look at the vocabulary mapping
print("Our vocabulary: ", count_vect.vocabulary_)

#Get the representation using this vocabulary, for a new text
temp = count_vect.transform(["dog and dog are friends"])
print("Bow representation for 'dog and dog are friends':"
      , temp.toarray())
```

Pros and Cons of Bag of N-grams

- It captures some context and word-order information in the form of n-grams.
- Thus, resulting vector space is able to capture some semantic similarity. Documents having the same n-grams will have their vectors closer to each other in Euclidean space as compared to documents with completely different n-grams.

Pros and Cons of Bag of N-grams

- It captures some context and word-order information in the form of n-grams.
- Thus, resulting vector space is able to capture some semantic similarity. Documents having the same n-grams will have their vectors closer to each other in Euclidean space as compared to documents with completely different n-grams.
- We still have the out of vocabulary issue, and large feature vector issue.

TF-IDF representation

- In BoW/BoNgrams, all the words in the text are treated as equally important—there's no notion of some words in the document being more important than others.
- TF-IDF, or term frequency–inverse document frequency, aims to quantify the importance of a given word relative to other words in the document and in the corpus.

TF-IDF representation

- In BoW/BoNgrams, all the words in the text are treated as equally important—there's no notion of some words in the document being more important than others.
- TF-IDF, or term frequency–inverse document frequency, aims to quantify the importance of a given word relative to other words in the document and in the corpus.

TF(t,d)=(Number of occurrences of term t in document d)/(Total number of terms in the document d) IDF(t)=$\log_e$(Total number of documents in the corpus)/(Number of documents with term t in them)

TFIDF Calculation

Table 3-2. TF-IDF values for our toy corpus

Word	TF score	IDF score	TF-IDF score
dog	⅓ = 0.33	$\log_2(4/3) = 0.4114$	0.4114 * 0.33 = 0.136
bites	⅙ = 0.17	$\log_2(4/2) = 1$	1* 0.17 = 0.17
man	0.33	$\log_2(4/3) = 0.4114$	0.4114 * 0.33 = 0.136
eats	0.17	$\log_2(4/2) = 1$	1* 0.17 = 0.17
meat	1/12 = 0.083	$\log_2(4/1) = 2$	2* 0.083 = 0.17
food	0.083	$\log_2(4/1) = 2$	2* 0.083 = 0.17

The TF-IDF vector representation for a document is then simply the TF-IDF score for each term in that document. So, for D_1 we get

Dog	bites	man	eats	meat	food
0.136	0.17	0.136	0	0	0

TF-IDF code

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()
bow_rep_tfidf = tfidf.fit_transform(processed_docs)
print(tfidf.idf_) #IDF for all words in the vocabulary
print(tfidf.get_feature_names()) #All words in the vocabulary.

temp = tfidf.transform(["dog and man are friends"])
print("Tfidf representation for 'dog and man are friends':\n", temp.toarray())
```

Note: There are some variations of the formula in practice.

Pros and Cons of TFIDF

- We can use the TF-IDF vectors to calculate similarity between two texts using a similarity measure like Euclidean distance or cosine similarity.
- TF-IDF is a commonly used representation in application scenarios such as information retrieval and text classification.

Pros and Cons of TFIDF

- We can use the TF-IDF vectors to calculate similarity between two texts using a similarity measure like Euclidean distance or cosine similarity.
- TF-IDF is a commonly used representation in application scenarios such as information retrieval and text classification.
- However, despite the fact that TF-IDF is better than the vectorization methods we saw earlier in terms of capturing similarities between words, it still suffers from the curse of high dimensionality.

Embeddings

Distributionally simliar words are words that are likely to occur in similar contexts.

- If we're given the word "USA," distributionally similar words could be other countries (e.g., Canada, Germany, India, etc.) or cities in the USA.
- If we're given the word "beautiful," words that share some relationship with this word (e.g., synonyms, antonyms) could be considered distributionally similar words.

Embeddings

Distributionally simliar words are words that are likely to occur in similar contexts.

- If we're given the word "USA," distributionally similar words could be other countries (e.g., Canada, Germany, India, etc.) or cities in the USA.
- If we're given the word "beautiful," words that share some relationship with this word (e.g., synonyms, antonyms) could be considered distributionally similar words.

Modern day NLP is based text representations which **learn** such semantic relationships in a dense, low dimensional space (compared to sparse, high dimensional space we saw earlier)

Word Embeddings

- pre-trained: such embedding representations are already learnt by training on a large database such as wikipedia. We can download the learnt models and directly use them.
- e.g., word2vec, glove, fastText etc.

```
from gensim.models import Word2Vec, KeyedVectors
pretrainedpath = "NLPBookTut/GoogleNews-vectors-negative300.bin"
w2v_model = KeyedVectors.load_word2vec_format(pretrainedpath
              , binary=True)
print('done loading Word2Vec')
print(len(w2v_model.vocab))
#Number of words in the vocabulary.
print(w2v_model.most_similar['beautiful'])
W2v_model['beautiful']
```

`most_similar('beautiful')` returns the most similar words to the word "beautiful." The output is shown below. Each word is accompanied by a similarity score. The higher the score, the more similar the word is to the query word:

```
[('gorgeous', 0.8353004455566406),
 ('lovely', 0.810693621635437),
 ('stunningly_beautiful', 0.7329413890838623),
 ('breathtakingly_beautiful', 0.7231341004371643),
 ('wonderful', 0.6854087114334106),
 ('fabulous', 0.6700063943862915),
 ('loveliest', 0.6612576246261597),
 ('prettiest', 0.6595001816749573),
 ('beatiful', 0.6593326330184937),
 ('magnificent', 0.6591402292251587)]
```

`w2v_model` returns the vector for the query word. For the word "beautiful," we get the vector as shown in Figure 3-6.

Document Embedding

- We can obtain the vector representation for an entire text by averaging individual word vectors.

```
import spacy
import en_core_web_sm

# Load the spacy model. This takes a few seconds.
nlp = en_core_web_sm.load()

# Process a sentence using the model
doc = nlp("Canada is a large country")

#Get a vector for individual words
#print(doc[0].vector) #vector for 'Canada', the first word
print(doc.vector) #Averaged vector for the entire sentence
```

The OOV problem

- We can also train our own word embeddings, but both pre-trained and self-trained word embeddings depend on the vocabulary they see in the training data.
- What should we do when we encounter a new word in a document?

The OOV problem

- We can also train our own word embeddings, but both pre-trained and self-trained word embeddings depend on the vocabulary they see in the training data.
- What should we do when we encounter a new word in a document?
- A simple approach that often works is to exclude those words from the feature extraction process so we don't have to worry about how to get their representations.
- Another way to deal with the OOV problem for word embeddings is to create vectors that are initialized randomly, where each component is between –0.25 to +0.25, and continue to use these vectors.
- There are also other approaches that handle the OOV problem by modifying the training process by bringing in characters and other subword-level linguistic components.

Beyond Words: Contextual representations

- In all the representations we've seen so far, we notice that one word gets one fixed representation. Can this be a problem?
- Well, to some extent, yes. Words can mean different things in different contexts.
- For example, the sentences "I went to a bank to withdraw money" and "I sat by the river bank and pondered about text representations" both use the word "bank." However, they mean different things in each sentence.
- Well, to some extent, yes. Words can mean different things in different contexts. For example, the sentences "I went to a bank to withdraw money" and "I sat by the river bank and pondered about text representations" both use the word "bank." However, they mean different things in each sentence.

SOTA: Universal Text Representations

- Neural architectures such as recurrent neural networks (RNNs) and transformers were used to develop large-scale models of language (BERT, and beyond), which can be used as pre-trained models to get text representations.
- Process: Take a large pre-trained model, and "fine-tune" it to a given task/dataset.
- These models have shown significant improvements on some fundamental NLP tasks, such as question answering, semantic role labeling, named entity recognition, and coreference resolution, to name a few.
- There are a lot of custom models, especially for BERT, such as: SciBERT, LAWBert, FinBERT, PatentBERT, BioBERT etc.

An example using BERT

```
from transformers import AutoTokenizer, AutoModel, pipeline
modelpath = "bert-base-uncased"
model = AutoModel.from_pretrained(modelpath, cache_dir=cache_dir)
tokenizer = AutoTokenizer.from_pretrained(modelpath, cache_dir=cache_dir)
nlp = pipeline('feature-extraction')
myrep = nlp(sometext)[0][0]
```

Check examples here

Conclusion

- we saw different techniques for representing text, starting from the basic approaches to state-of-the-art DL methods.
- When should we use what?

Conclusion

- we saw different techniques for representing text, starting from the basic approaches to state-of-the-art DL methods.
- When should we use what?
- For some applications, such as text classification, it's more common to see vectorization approaches and embeddings as the go-to feature representations for text.
- For some other applications, such as information extraction, it's more common to look for handcrafted, domain-specific features.
- Quite often, a hybrid approach that combines both kinds of features are used in practice.
- Having said that, BoW, BoN, TFIDF approaches are a great starting point!

Resources

Jupyter notebooks with code examples showing different ways of representing text: https://github.com/practical-nlp/practical-nlp/tree/master/Ch3

Assignment 1 Description

- 15% of the total grade (for those submitting a term paper), 30% of the grade (for those submitting without).
- Question 1: Explore the free trials of NLP service providers or libraries and write a small piece of code that takes a sentence as input and does any one of the following: machine translation, named entity recognition, key phrase extraction, part of speech tagging, relation extraction (again, just write code for any functionality among these!)
- Question 2: Look for a PDF to text conversion library, write a small piece of code that converts a pdf file you input into plain text.
- For both these questions, write do own your observations about the tool(s) you used (ease of use, accuracy of the result etc).
- Submit the assignment as follows: 1 pdf file with the observations, and all the associated code files put into one zip file.

ToDo - before next class

- Participate in the forum and ask any questions about today's class under "NLP Pipeline" forum.
- Think about where do we usually find corpora? If you want some new kind of corpus, how will you go about looking for it?
- Think about what kind of file formats will we encounter when we say "corpora".
- Teams. Think about your teams/paper you chose. If you don't know anyone or can't form a team, message me, I will assign a team for you by Friday.

We will talk more about data acquisition in the next class.