# Lecture 1: Intro

## Admin

Schedule:

* Lectures – 14:00 – 16:00
  + <https://uofglasgow.zoom.us/j/93850891064?pwd=RkJ0SHNVRExuRmYrQ3lXZmRVazhuZz09>
  + Meeting ID: 938 5089 1064
  + Passcode: 928653
  + Use Teams for chat
  + Recordings: Yes
* Labs
  + None in Week 1
  + Google Colab
    - Mini lab week 1
  + Material is examinable, including knowledge of basic libraries
  + Entirely online at the start
  + Reddit Data
    - Dataset from Reddit posts
    - Main posts with text content only
      * No replies
      * No “title-only” posts with images/videos/links
    - Data retrieved with Reddit API
  + Not assessed
* Office Hours
  + Wednesdays at 11-12

Technical

* Python
  + spacy
  + scikit-learn
  + transformers (from HuggingFace)
* Google Colab (branch of Jupyter Notebook)

Optional reading:

* <https://web.stanford.edu/~>

Assessment

* Coursework 20%
* Exam 80%

## Content Intro

Why make computers work with text?

* Humans interact with one another through language
  + Language can be represented as text
  + This makes text a convenient interface between humans and computers
* Text data is growing
  + Emails, chats, etc
* Unstructured, hard to process
  + No meaning (labels)

Text type

* **Input**
  + ex usage: News Aggregation, Search Tools, Email Suggestions
  + ex: Documents, Tweets, Voice commands, Search queries, Web pages, Medical records, Books
  + ex task: Document similarity
    - How to find similar documents?
* Output
  + simple ex: Some data (e.g., numbers) can be turned into their textual form with a number of rules
  + Advanced ex: Creative writing (GPT-3
  + advanced ex: <https://play.aidungeon.io> (AI-written text adventure game)
* Both I/O?
  + ex: Assistants (Siri, Alexa), Machine Translation (Google Translate), Email Suggestions

Type of data:

* Structured
* **Unstructured**

Names of field (overlapping different fields):

* Natural language processing (NLP)
* Computational linguistics
* Text Analytics

Language is not just rules

* Language uses reasoning to change meaning through context
  + “I saw the elephant with my telescope”

Text as Data has a long exciting history as a research area:

* Building with linguistics research
  + How does language work?
  + How do we learn language?
* Tied to computational performance
  + New CPUs and GPUs have enabled new advances
* The Internet provides an incredible source of example text
* Deep learning is changing the whole approach
  + New amazing abilities with language
    - New language systems are trained by asking them to complete a sentence, e.g., “It showed 9 o’clock on my \_\_\_”
  + ELMo and BERT deep learning approaches developed a new model for deep learning that could succeed at several different problems
  + OpenAI’s GPT-3 deep learning model showed incredible new abilities
    - Recipes
    - Github Copilot
  + In the past, bigger (and more complex) models haven’t translated into better performance
  + With transformer-based models, bigger (e.g., more parameters) models have been made
  + Hype for AI
    - Be sceptical of claims of human-level abilities
    - Most systems may be very good at specific tasks, but research on reliable generalisation is ongoing
  + Play with Text Generation
    - transformer.huggingface.co/doc/distill-gpt2

Summary of intro:

* Computers are now frequently using text as I/O
* Amount of text data is growing quickly

Approaches

* No approach is the best for all problems
* Each approach has strengths and weaknesses
  + Accuracy vs computational complexity
* Experiments needed to compare methods

Document Similarity

* If 2 posts overlap, we hypothesise that they are talking about a similar subject
* The more overlap, the higher similarity
* Data cleaning is often necessary
  + Font weight and size removed
  + URLs removed
    - Or valuable
  + Tables removed
    - Or just formatting
* Measuring similarity
  + Individual letters?
    - Order and grouping of letters is key
    - Encoding
      * ASCII: 128 possible characters (in 1 byte)
        + A-Z, a-z, 0-9, standard punctuation, various control chars
      * Unicode: variable-byte (2-4 bytes)
        + Encodes the alphabets of many languages
        + Emojis, math symbols
        + Frequently updated (v14 in Sep 2021)
        + Different versions, most often used UTF-8
        + Useful to know

mojibake (character errors)

Python has “encoding” parameter when loading files

* + - Advantage:
      * Detecting language
        + Different languages use different letters at different frequencies
        + There are better ways
  + Short sequences of letters?
    - Character n-grams (bigrams (2), trigrams (3), etc)
    - Extract by running moving window across text
    - US Defence department funds a lot of research
    - Strengths
      * More specific than individual letters
      * Easy to implement
    - Weaknesses
      * Small not specific enough
      * Large n are very rare
  + Words?
    - **Tokenization**
      * Splitting a section of text into individual tokens
      * Token (term) – meaningful sequence of characters
        + May split words for meaning, like “don’t”
        + Punctuation
      * Other languages can have very different tokenization challenges
        + e.g., Chinese characters
      * Using library: spaCy
    - Filtering using stopwords
      * **Stopword** – word with minimal meaning, e.g., a, I, etc
      * Disadvantage:
        + Important phrases might get filtered, like The Who
    - 2 words only map together if they match exactly
      * Case
        + If made case-sensitive, may lose meaning
      * Word forms (loved, loves)
        + May lose meaning
    - Breaking into stems/affixes (morphemes)
      * Morpheme – small meaningful unit
        + Stem – core meaning-bearing unit
        + Affix – bits and pieces
      * **Stemming**
        + Process of reducing inflected words to their stem/root form
        + Often rule-based
        + Generally works on single words with no context
        + Error-prone (computing to comput), but isn’t that bad
        + ex: Porter Stemmer

List of steps

* + - * Lemmatization
        + 2 words have same lemma if:

Same stem

Same part-of-speech (noun, verb, etc)

Essentially the same meaning

* + - * + Difference between lemmas and stems:

Lemmas take in the context of the sentence

* + - * Context of words matter
        + ex: like, don’t like
  + Word n-grams
    - Usually, pairs/triples of words
    - Better for context
    - Fairly powerful but computationally expensive
      * ~7 million words in English Wiktionary
* Standard pipeline

1. Data cleaning
2. Tokenizing
3. Normalize words
4. Stopwords removal

* Sets of tokens
  + Set – group of unique items
  + Documents can be represented as sets of tokens in them
  + Characteristics of sets:
    - No concept of frequency
    - Efficient computation when comparing
    - Most programming languages have sets as standard data structure
  + Operations
    - |X| - number
    - intersection
    - union
    - difference
  + Similarity with sets
    - Number of overlapping words?
      * sim(x,y) =
      * Needs to be normalized (sim = [0-1])
    - Overlap Coefficient
      * sim(X,Y)
      * Bounded between 0 and 1
        + 0 means no overlap
        + 1 means subset
    - Sorensen-Dice Coefficient
      * Bounded between 0 and 1
      * Will only be 1 if the 2 sets are exactly matching (no subsets)
    - Jaccard Similarity
      * Number of overlapping tokens divided by the number of unique tokens
        + Intersection over union (IOU)
      * Bounded between 0 and 1
      * Very popular
      * Satisfies triangle inequality
    - Tversky Index
      * Generalization of Jaccard and Sorensen-Dice
      * => Jaccard
      * => Sorensen-Dice
      * Asymmetric measure (sim(x,y) =/= sim(y,x))
  + Weaknesses
    - All words are treated equally (some should be more important)
    - Word frequencies ignored
    - 2 different words with similar meaning cannot match

# Lecture 2: Documentation Representation and Clustering

From a set to a vector:

1. Build vocab from all tokens in the corpus
   1. Each token is assigned an integer ID (from 0)
2. Build a vector where all the indices of the words in the set are all set to 1 (all others – 0)
   1. Boolean/One-hot encoding
   2. Length of vector –
   3. In practice, most values are 0
      1. vocab size >> document length
      2. (Standard English: ~150K-200K)
      3. => **Sparse** data
         1. Wasted memory if allocate entire vector
   4. Sparse representation:
      1. Pairs of TokenID-value are stored (only non-zeros)

Vectors for an Entire Corpus

* An entire corpus can be represented as a document-term matrix (DTM)
  + Each row is a document, represented as a vector of its word occurrences
* Alternatively, sparse versions can consist of lists of ID-value pairs (postings) – **index**. Common orientations:
  + Direct Index
    - document-oriented, Compressed Sparse Column (CSC)
  + Inverted Index
    - token-oriented, Compressed Sparse Row (CSR)

Beyond One-Hot Encoding

* Bag of Words Model – **Term Frequency**
  + Assumption: If a term occurs multiple
  + Relaxation of the binary occurrence assumption
  + Sparse Representation
  + Exercise:
    - D1: (9, 2), (6, 1), (4, 1), (3, 1)
    - D2: (18, 1), (22, 1), (3, 1), (9, 1)
  + Limitations:
    - Common words are not necessarily important (‘the’, ‘a’, etc)
      * Solution: stopwords (when able to use them)
    - Raw Term frequency (tf)
      * Term frequency (tf
      * Solution: Sublinear TF scaling, e.g., **log** frequency
        + Dampens effect of term frequency approach
    - How ‘good’ is each term?
      * All terms considered equally important if they have the same frequency
      * Solution: **Document frequency** – how many documents in the collection contain a given term
        + Rare terms are more informative than frequent terms
        + Goal – give higher weight for rare terms
* Document Frequency
  + Inverse Document Frequency weight
    - – document frequency of t
* TF
  + How many times the term appears in a doc
  + High value – document might be about this term
  + Low value – doc probably isn’t about the term
* DF
  + How many docs the term appears in
  + High value – term probably unimportant
  + Low value – important
* Inverse DF
  + Inverse
  + High value – important
  + Low value – unimportant

Zipf’s Law

* Statistical property of language
* straight log scale decreasing in a frequency (log) to rank (log) graph

tf-idf weighting

* Multiplication
* Best known weighting scheme for text similarity

Ex:

* TF – 3
* IDF – 4
* TF-IDF – 10.4

More advanced interpretations

* Collection frequency – times a word appears in a collection, rather than no. of docs it appears in
* Removing bias – normalising
* Probabilistic interpretation

Geometric Similarity

* Vector Space Model
  + Assumption: docs that are “close together” in vector space “talk about” the same things (are similar)
  + Formalise
    - Euclidean distance - (vectors)
      * Problem: distance is large for vectors of different lengths
    - Angle between vectors
      * Advantage: length not taken into account
      * Problem: inverse logic (small – similar)
      * Solution: Cosine
* Similarity – distance between points representing units of text

Clustering

* Motivation
  + Many uses
* Purpose
  + Discover underlying structure of data
* (Often) An unsupervised learning task
  + Not pre-defined
  + No labelled training data
  + Clusters are discovered by an algorithm
* Process:
  + 1 Derive representation
  + 2 Measure similarity
  + Apply clustering method
  + Check validity/quality of clustering
* Properties of Algorithms:
  + Partitioning criteria
    - Single level vs multi-level hierarchical partitioning (often more desirable)
  + Separation of clusters
    - Exclusive (1 object in only 1 cluster) vs overlapping (1 object may be in many clusters)
  + Hard versus fuzzy
    - Fuzzy clustering – an object belongs to every cluster with some weight (0-1)
    - Probabilistic clustering – similar
    - Weights must sum to 1
* Partitioning Clustering
  + Partitions a set of N documents into K disjoint clusters
  + Find the partition that optimises a specific criterion
  + Typically, a function of within-cluster similarity and between-cluster distance
  + Ex: K-Means Clustering
    - K – number of clusters
    - Input – dataset and K
    - Output - ???
    - process:
      * 0 Pick K random centres
      * 1 Assign each data point to its closest centre
      * 2 Recalculate the centres (average of assigned point)
      * (repeat Steps 1 and 2)
    - Advantage: Tends to converge quickly
    - Disadvantages:
      * Sensitive to choice of initial seeds (local minima)
        + Better methods for picking initial centres
      * Selecting K is tricky
        + Too small k – important clusters merged (lose info)
        + Too big k – too fine-grained
        + Solution – Elbow method

Elbow point in a graph

Compute distortion – average sum squared distance of each point to its centroid

Graph distortion against the number of clusters (K)

* + - * Measuring is difficult
        + Solution – silhouette coefficient

Cohesion score (a) – average distance of point i to the subjects in the same cluster (small – good)

Separation score (b) – min (average distance of i to objects in another cluster) (high – good)

s(i) = 1 – a/b if a < b

s(i) = 0 if a = b

s(i) = b/a – 1 if a > b

* + - * Not extensible
        + Solution – mini-batch K-means

Sample of population (random)

Dramatically reduce convergence time

Only slightly worse than running full K-Means

* + - * Produce clusters:
        + flat (not hierarchical)
        + non-overlapping (disjoint)
        + hard assignments
    - O(N\*K) (documents, clusters)
    - Choosing initial points:
      * Forgy partition
        + Random initial points
      * Random partition
        + For Step 1, random assignment instead
    - Resource: naftaliharris K-Mean Clustering Visualisation
    - Choosing K – **Elbow** method
* Hierarchical Clustering
  + Tree-like representation with clusters of highly similar documents nested
  + Agglomerative – Bottom-Up
    - Beginning with singletons
    - Merging them until the whole dataset becomes the root
  + Divisive – Top-Down
    - Recursively partitioning the entire dataset until singleton sets are reached
* K-Means vs Hierarchical
  + K-Means more efficient (O(
  + K-means usually produces clusters of similar quality
  + => K-Means usually choice for all-purpose
* Clustering Evaluation
  + Laugh (smell) test – inspect
  + Extrinsic – how much it helps solve another problem
    - Represent text using clusters (cluster IDs)
    - Did the external task evaluation improve?
  + Intrinsic – useful in and of itself
    - Help understand what’s in data
    - Internally consistent
      * Cohesion
      * Separation
* Conclusion
  + Theoretically solid and intuitively plausible
  + Clustering approaches assume that each document is only about a single topic
  + Entirely unsupervised
  + Anything can be clustered

# Lecture 3: Language Modelling

What is Language modelling?

* Task of building a predictive model of language
* Goal – Predict the probability of:
  + Next word in a sequence
  + A sentence of words

Applications:

* Speech recognition
* Optical Character Recognition
* Spelling correction
* Machine translation
* Authorship detection
* Auto-completion
* Search ranking
* Text segmentation
  + Split a long string of words without delimiters into words

The probability of observing a sequence is a measure of goodness/relevance

Historically: Using language statistics

* Simple, effective
* Still useful in many settings

Recently: Using neural networks

* Very expensive, highly effective
* Technique behind GPT3

# Lecture 4: Natural Language Processing (NLP)

Not understood by bag-of-words/n-gram models:

* Self-contained phrases
* Reference (anaphora) (ex: “it”)
* Subject/Object relationships
* Prepositional phrases
* Implications (Pragmatics)

I saw an elephant with a telescope. It was enormous!

NLP in general:

* Old field from AI/Linguistics
* Deals with knowledge extraction from text by leveraging syntactic and semantic elements
* Allows:
  + Building natural language interfaces to computers (devices)
  + Building intelligent machines which require knowledge, and a large portion of it is textual
* Goal: For computers to process (understand) natural language in order to perform useful tasks
* Ex: Virtual personal assistants (Siri, Google Assistant, Cortana)

Components:

* Syntax
  + Identifying **grammatical roles** of words in sentences, i.e., how words fit together to form valid sentences in a language
* Semantics
  + Worrying about the **meanings** of the words/phrases/sentences and how they relate to one another (who does what to whom?)
* Pragmatics
  + Worrying about how **communicative context** affects meaning

Other tasks

* Machine Translation
* Single/Multi-Doc Summarisation
* Entity disambiguation
  + Apple – company or fruit?
* Relation Extraction
* Semantic parsing
* Textual entailment

Difficulties:

* Contextual Dependence and Background Knowledge
* Garden Path Sentences
  + Do not appear to be syntactically correct on the first pass

NLP Pipeline

* Tokenization and lemmatization
* Sentence boundary detection
* Part-of-speech tagging
  + Nouns, verbs, adjectives, etc
* Parsing (dependency)
  + Diagramming sentences
* Named Entity Recognition
  + Detect and classify entities
* Coreference resolution
  + Resolve pronouns to named entities

Toolkits:

* Java
  + Stanford Core NLP pipeline
  + NLP4J
  + Apache OpenNLP
* Python

How

* Rule-based approaches
  + Linguistic + Domain knowledge
  + ex: tokenisation
  + Maintaining/updating rules manually is difficult
* Statistical methods for identifying patterns
  + **Supervised ML**
    - Training
      1. Collect a set of representative training documents
      2. Label each piece of text with its label
      3. Design feature extractors appropriate to the text and classes
      4. Train a classifier to predict the labels from the data
    - Testing

1. Receive a set of test text
2. Run model inference to label each piece of text
3. Appropriately output label
4. Evaluate correctness of predicted labels

* Learn from human interactions with computers
  + Mainly to acquire training data

Sequential NLP Structures

* Some NLP info can be formulated as a sequence of labels:
  + **Part of Speech Tagging**
    - PoS – basic syntactic analysis
    - Every word is assigned a tag based on its grammatical role
    - Tags:
      * Different in different languages/opinions
      * (Proper) Noun, (auxiliary) verb, adjective, determiner, ADP (prepositions), pronouns
      * + punct
      * [Full list in lecture]
    - State-of-the-art techniques achieve > 98% accuracy on news
    - Average PoS disagreement among expert human judges for the Penn Treebank was 3.5%
    - Difficult in:
      * Not well-formed domains (e.g., Twitter)
      * Resource-poor languages
  + **Named Entity Recognition (NER)**
    - Objects that can be referred to by a name, e.g., Joe Biden, Macbook Pro, Uni of Glasgow + quantities, dates/times, etc
    - Begin-Inside-Outside (BIO) labelling – whether a token starts with a NE/continuation of a NE/not a NE
    - Often paired with the task of extracting type of entity: person, product, organisation, location
    - Sometimes helpful to treat NEs as single tokens
    - Entity Linking
      * Given a NE, link to a canonical form of the NE
        + ex: Mr. Obama, Barack, Obama -> wiki/Barack\_Obama
      * Depends on context!
  + Word/Structure Segmentation
    - Word – e.g., Chinese
    - Structure – e.g., question or answer
* Each unit is assigned a label
* Labels are not independent

Sequential Labelling

* Create system to build structures automatically
  + Rule-based: Error-prone, building & maintaining rules is challenging
  + Supervised Learning: most common/effective approach
* Identify and generalise regularities found in a sample manually labelled training sequences. Training data from labelled treebanks
* Naïve Bayes
  + Simple approach: count & normalise probabilities
  + From Bayes Rule
  + Diagram

    Description automatically generated
  + Ignores sequential info
* **Hidden Markov Models**
  + Generalisation of Naïve Bayes to sequences
  + Assume underlying set of hidden (unobserved) states (labels) y, in which the model can be
  + Assume probabilistic generation of tokens from states
  + Assume current state dependent on previous state
  + Parts:
    - x\_i – observed states (words)
    - y\_i – hidden nodes (**tags** (meant to be predicted) (ex: noun, adj, etc))
    - Transitions: p(y\_i | y\_i-1) (horizontal arrows)
    - Emissions: p(x\_i | y\_i) (vertical arrows)
    - Training: learn transitions and emissions
      * Count and normalise
    - Inference: given x\_i, infer y\_i or p(y\_i)
  + Strategies:
    - Full
      * product of transition and emission and previous probability
    - Greedy
      * Without previous probability
  + Inference
    - Approximation: Use greedy approach – take the most likely tag at each position
    - Exact: Viterbi algorithm – score all non-zero edges in translation graph, pick the best path backwards

Parsing: Tree-Structured Prediction

* Motivation:
  + Many aspects of meaning can be learnt
    - NP (Noun Phrase) preceding VP (Verb Phrase) is likely the subject
    - NP following VP is likely object
  + Knowing basic units is helpful in modelling language
    - Used to predict/complete sentence (language modelling)
    - Reorganising sentences/simplifying (summarisation)
  + Many NLP applications use sentence structure to make decisions:
    - Relation extracting
* Overview:
  + Types:
    - Constituency/phrase-structure
    - **Dependency**
      * Given a sentence, draw edges between pairs of words and label them
        + Result – tree
        + Edge-labels between word pairs should convey the correct syntactic relation
      * **Diagram

        Description automatically generated**
      * Building the tree:
        + No single formalism/answer, methods are similarly effective (just need training data)
      * Formal definition:
        + Directed graph originating at a unique and artificiaully inserted root, which is always inserted as left-most value
      * Properties:
        + Weakly connected – replacing its edges with undirected edges results in a connected graph
        + Each word has only 1 incoming edge (except the root, which has 0)
        + Acyclic
      * Inferring:
        + Subject, direct/indirect object
    - Semantic/frame
  + Techniques/Algorithms
    - **Transition based**
      * Mechanism:
        + **Build parse tree by sequence of actions (transitions)**
        + **State:**

**Buffer – initialised with words**

**Stack – store state**

**Dependency arcs**

* + - * + Machine with stack and buffer
      * Transitions:
        + Types:
        + Irreversible
        + How to choose:

Multi-class prediction

Inputs: state = {stack, buffer, other features}

Targets: transitions{shift,

Classifier: score(ai | statei) at step i

Rule-based, SVM, NN, etc

Training:

Predict “gold” actions (from true tree)

Or: “dynamic oracle” – rewards good actions

* + - * O(N)
        + Only if you choose the right transitions
      * Performance
        + Very high, but still growing (91.8% in 2014, 97.4% in 2019)
        + Effective enough for real-world applications
* Summary
  + Syntactic parsing aims
  + Dependency parsing
    - Word-word relations, e.g., nsubj(ate, Elephant)
    - Parsing generates dependency trees by predicting/pruning edges
    - Transition-based parsing formulates edge prediction as a classification task
      * Predict 1 of 4 actions to take next
  + Widely used step in constructing deeper, semantic representations

Beyond Syntax

* Efforts to build semantic structures
  + ex: Abstract Meaning Representations
    - graph (not tree)

Summary

* What does NLP allow us to do?
  + Distinguish sentences based on syntactic relationships
    - PoS tags
    - Relationship in sentence tree
* Provides us with tools to model language beyond simple lexical (word) features
* Sequence models (HMMs, CRFs) allow us to label break-apart patterns in language
  + PoS tags
  + Sentence/word breaks
* Tree structured prediction
  + Predict complex patterns in language
  + Helpful for advanced applications (QA, Dialogue Systems, etc)
* Recent neural models can be effective even without these features

# Lecture 5: Text Classification

Overview: ML and Evaluation

Classification

* **Predicting** which of a predefined set of **classes** (**categories**) an object belongs to
  + i.e., assigning a **class** to an **object** based on **pre-trained** knowledge

Objects to classify – anything with words

* Article – identify subject

Applications:

* Topic categorisation
* Spam detection
* Sentiment analysis: positive/negative opinion
* Language identification
* Author attribution
* Domain-specific: Adult/Non-Adult
* Discourse: question/answer/clickbait

Types:

A picture containing diagram

Description automatically generated

* Binary
  + ex: spam/not spam
* Single-Label Multi-Class
  + ex: news categorisation: politics/entertainment/lifestyle/sports
* Multi-Label Multi-Class
  + Many items in document, each can belong to 0+ classes

Classification – Supervised Training

* Train, test, run
* Diagram

  Description automatically generated
* Supervised – Preparing Training Set
  + Labelled by human ‘assessors’
    - Consistency important
    - Resource-intensive
  + Quality vs Quantity
    - **Expensive** to prepare (because involves humans)
      * Must be accurate/consistent
    - Quantity can lead to **difficulties in training** and use of **specialised hardware** (GPUs/TPUs)
    - Usually good for multiple assessors to label same items and measure **inter-annotator agreement**
  + Training Model
    - Training set and tuning parameters
    - Learns statistical representation w.r.t. the salient features of each class/label to be predicted
* Supervised – Test and Use model
  + Model can:
    - make its own predictions for unobserved docs
    - practice against part of training set (reserved for testing) to fine-tune parameters

Feature Extraction

* Represent docs by their **most descriptive features**
* Document feature vectors
  + Types:
    - Text: Binary/real-valued values (TF, TF-IDF), etc
    - NLP/Linguistic (part-of-speech, dependency labels, etc)
    - Metadata (author, source, etc)
  + Typically, **most informative** terms and/or bigrams with bag of words feature representation

Doc classification formally:

* Input: doc feature vector d (also, instance
* Output: predicted class
* Goal: learn function
  + f – categorisation function (model)
  + d – feature vector
  + – prediction
* Terms:
  + X – set of docs to classify
  + Y – set of known answers/true labels for X
  + Y’ – set of predictions
* A picture containing clock, watch, gauge

  Description automatically generated

Types of features:

* Feats/Attributes:
  + Binary
  + Nominal (categories)
  + Ordinal (categories with ordering)
    - ex: small, medium, large
  + Continuous (numerical)
* No “text” – can be:
  + binary features of word occurrences (one-hot encoding, etc)
  + continuous features (TF/TF-IDF/etc) weights
* Each token is a feature
* **Text must be vectorised**

Additionally features

* ex: Author attribution might use word/sentence length and punctuation frequency

Classification algorithms

* Geometric algorithms:
  + **k-Nearest Neighbour**
    - Choose k of the “nearest” instances based on their distances
    - “Lazy learner”
      * Doesn’t build models explicitly
      * Can be relatively expensive
    - Does not “learn” importance of features performing the classification
    - Scaling issues
      * Attributes may need to be scaled to prevent distance measures from being dominated by one of the features
    - In practice: Feature scaling
      * Make sure input features on same scale
      * Min-max normalisation
      * Standardisation
      * Important for many classifiers (kNN, SVM, Logistic regression, etc)
  + **Support Vector Machines**
    - Pros:
      * Nice theoretical guarantees
      * Typically outperforms LR
      * Fast on small/medium datasets
      * No assumption of distributions
      * Few parameters to tune
    - Cons:
      * Need to select a kernel!
      * Need to tune kernel hyperparameters
      * May be slow, depending on kernel
      * Need regularisation
      * Some kernels can’t scale to large datasets
      * Not all kernels are probabilistic
      * No easy solution to multi-class problem
* Probabilistic algorithms:
  + **Naïve Bayes**
    - Pros:
      * Fast & efficient
      * Probabilistic
      * Interpretable
      * Very simple
      * Often relatively effective
      * Few (if any) parameters: Surprisingly robust
    - Cons:
      * Sensitive to errors on rare words
      * Assumption of one class per doc
      * Weak on rare categories
      * Weak on very large, noisy feature sets
      * Biased due to independence assumptions
  + **Logistic regression**
    - Pros:
      * Most widely used because general-purpose
      * Does better than Naïve Bayes – deals with dependencies between features
        + Should be at least as well
      * Easy to incorporate features (no assumption of a distribution)
      * Supports multiclass outputs
        + Softmax regression (maxent classifier)
      * Very fast, very scalable
    - Cons:
      * May need more data than NB
      * Can overfit on very sparse data – often used with regularisation
        + Regularisation penalises models with many high weight features
        + Regularisation parameters need to be tuned
  + **Neural networks**
    - Many different models (CNNs, RNNs, Transformers)
    - ex: Facebook’s FastText
      * Simple linear model
      * Input: bag-of-words with char n-grams
      * Vocab: 10M (100M) hashing
      * Supports hierarchical classification
      * Extremely fast and scalable
    - Binary/Multi-class classification
      * Binary – Logistic output
      * Multi-class - Softmax
* **Decision Trees**
  + Based on probabilities, geometric interpretation
  + Pros:
    - Cheap
    - Fast
    - Interpretable for small trees
    - Comparable accuracy
    - Handles diverse features
  + Cons:
    - Unstable
      * Small change in data leads to very different tree
    - Less accurate
    - Can be complex for large trees
    - Splitting criteria not perfect for all types of trees
* [table in slides comparing them all]

How to handle Multiclass classification

* Transformation to binary:
  + One vs all (OvA) – 1 binary classifier per class (positive – 1, all others negative)
  + All vs All (AvA) – discriminate between pairs of classes - classifiers
  + Issue: Doesn’t scale to large no. of classes

Toolkits

* Python
  + Scikit Learn
    - fit(X, Y)
      * X – size
      * Y – array of samples
    - predict(X)
    - score(X, Y)
    - Classifiers:
      * GaussianNB, LogisticRegression
      * DecisionTreeClassifier, RandomForestClassifier
      * SVC (SVM) (kernel specified, gamma)
  + NLTK
* Python/C++
  + fastText
  + Vowpal Rabbit
* Java/Scala
  + Apache Spark
  + Mahout
  + LingPipe
  + WEKA

Evaluation

* Need for a Dataset
  + Dataset of example instances (docs) with known pre-defined classes
  + Split into training and testing
* Steps:
  + Train
  + Evaluate
    - Make predictions
  + Analyse
    - Error Analysis
* Held-out set from test collection
  + Evaluate with docs
* Metrics
  + Accuracy
    - Count of correct predictions
      * T – true
      * F – false
      * P – positive
      * N – negative
    - Assumes equal cost for both TN & FN
    - Doesn’t show Class Imbalance Problem
      * one class may be rare, easy to get high accuracy by never guessing this
  + Contingency table (Confusion matrix)
    - Count correct/incorrect predictions
    - Select appropriate evaluation measure for type of classification task
  + Precision & Recall
    - Precision – exactness (what % of instances labelled as positive are actually positive?)
    - Recall – completeness (what percentage of positives were labelled as positive?)
  + Binary classification – F-measure
    - Harmonic mean of precision and recall
    - Can be tuned to give greater weight to one/other
    - Most widely used measure in NLP
  + Multiclass – Averaging
    - Micro-averaging
    - Macro-averaging
      * Often better
  + Receiver operating curve (ROC)
    - Graph showing performance at all thresholds
    - AUC – Area under ROC curve
      * Probability that the model ranks a random positive example more highly
  + [summary table in slides]
* Baselines
  + Most frequent (majority class) for each instance
  + Random – Assign instances to classes ranfomly
  + SKLearn – DummyClassifier
    - most\_frequent
    - Random (**stratified**)
* Statistical Testing
  + For statistically significant differences:
    - t-test
      * Most used for NLP
    - chi-squared
  + For measuring correlations:
    - Pearson Correlation coefficient
    - Kendall rank correlation coefficient
  + P-Hacking
    - Perform multiple test corrections (e.g., Bonferroni correction: divide p by no. of test conducted)

Scikit Learn Vectorising:

* fit(X) – builds vocab on input
* transform(X) – creates doc-term matrix (vectorising)
* NO – fit\_transform(X) – does both
* Multiple vectorisers
  + CountVectoriser
  + HashingVectoriser
  + TfidVectoriser

Problems:

* **Inductive Bias**
  + Assumptions because of too little data
* **Noise** in training data
  + Label noise: human assessors were not clear on task
  + Feature noise: someone had typo in review
* Features might **not be sufficient** for learning
* Overfitting
  + Modelling error that occurs when a function is too closely fit to a limited set of data points
  + Overly complex model
  + Data can have noise, overfitting takes into account noise
  + => good when training, bad when testing
* Underfitting
  + Too simple
* Class imbalance
  + Data with skewed
  + Solutions:
    - Downsampling (undersampling)
      * Training on a disproportionately low subset of majority class examples
      * May lead to underfitting
    - Upsampling (oversampling)
      * May lead to overfitting

Occam’s razor

* Given 2 models, prefer simpler
* Less of a chance of overfitting
* Consequences
  + Training Error
    - Overly optimistic
    - Error seen as opposite of Accuracy, F1

Workflow

* Don’t use all data for training
* Don’t measure (final) performance on training data
* Training, validation, test:
  + Training: model construction
  + Validation: setting hyperparameters
    - Prevent overfitting
    - middle of iterations
  + Testing: accuracy estimation
  + Starting point: 65/15/20 %

Cross-validation

* Randomly partition data into k mutually exclusive subsets
* SKLearn – GridSearchCV

Why isn’t it working?

* Some classifiers need more data
* How much data is needed?
* If too many features, perform **feature selection** to remove irrelevant ones
  + By Filtering
    - Assign score to each feature indicating how related x and y are
  + **By model search**
    - Forward selection
      * initialise empty
      * add feature which improves most
    - Backward elimination (ablation)
      * Algorithm:
        + initialise with all features
        + remove feature which improves most
      * tends to find better models
        + Interacting features
        + Frequently too expensive
* Contrast with Common practice: throw in every possible feature, let feature selection remove useless ones
* Presence of irrelevant features hurts generalisation

Ensemble methods: (Increasing Accuracy)

* Ensemble – combining set of heterogeneous classifiers to increase accuracy
* Methods:
  + Average classifier scores
  + Weighted average (linear combination)
  + Bagging – averaging prediction over collection of classifiers
  + Boosting – weighted vote with a collection of classifiers

# Lecture 6: Word Vectors

Representation of the meaning of a word

* Definition of meaning: (Webster dictionary)
  + idea represented
  + idea that a person wants to express
  + idea expressed in a medium (art, literature, etc)
* Linguistic meaning:
  + **signifier** (symbol) ⬄ **signified** (idea/thing)

Motivation for modelling meaning

* Ambiguity
  + One word might have different meanings
* Variability
  + One meaning might have different words, used differently in different scenarios

Representing meaning in a computer:

* Common (historical) solution: WordNet
  + A knowledge base containing lists of **synonym** sets and **hypernyms** (“is a” relationships)
  + Manually built tree
  + Problems:
    - No nuance between synonyms
    - No new words/slang
    - Subjective
    - **Requires human labour to create, update**
      * Especially with more languages
    - Hard to compute accurate word similarity
* Word embeddings, Word vectors

Distributional word similarity

* Intuition: 2 words are similar if they have **similar word contexts**
* Context – set of words appearing nearby (within fixed-size window)
  + shorter – syntactic representation (+- 1-3 very syntacticy)
  + longer – semantic (+- 4-10 semanticy)
  + In practice: ~5
* Cosine similarity: pretty good
* Cluster vectors to visualise similarity in co-occurrence matrices
* Sparse vs dense representations
  + Word co-occurrence vectors:
    - long (length |V|)
    - sparse (most elements are 0)
  + Storing and making computations with sparse vectors is problematic
  + Solution: use vectors that are
    - short (50-1000)
    - dense (most elements non-0)
    - to be used in ML

Vector Inputs to Neural Nets

* Sparse
  + Less efficient computation (gradient to be computed over vocab-sized dimension)
  + Most components being 0 can’t lead to parameter updates
    - SGD
* Dense
  + More efficient computation – Smaller dimension, non-0 elements

Word vectors (word embeddings)

* Dense vector – similar to vectors of words that appear in similar contexts
* IBM Model
  + Distributional word vectors (DWVs)
  + Word – sparse vector (pseudo-document) of the contexts around each occurrence
  + All words – BIG
    - |V|-dimensional vector
    - Stored as list of (term-id: weight) tuples
  + Cosine similarity can be used
  + Not dense – not easy to use with NNs
  + To obtain dense vectors, **Latent Semantic Indexing (LSI) / Singular Value Decomposition (SVD)**
    - SVD – principal components of maximum variance in the data => compact dense vectors
    - Number of principal components determines ???
  + Representing words as tf-idf vectors instead of tf vectors works well in practice
  + Algorithm for efficient implementation of DWVs:
    - Simple (slow):
      * Find all occurrences, merge corresponding vectors
      * Problem: O(n^2)
    - Efficient:
      * Slide a window of a given context size around text
      * Mid-point – anchor words
      * Around mid-point – context
  + Truncated SVD produces embeddings
    - Perform LSI/SVD on the distributional vectors for each word
    - => dense vectors
* Prediction-based models
  + Alternative way to get dense vectors
  + Most popular: Word2vec
  + Learn embeddings as part of the process of **word prediction**
  + “Fill in the blank”
  + Neural net language models
    - Train a neural network to predict either:
      * a word given the context
      * the context given a word
    - Learn dense embeddings for words in training corpus
  + Advantages:
    - Fast, easy to train (much faster than SVD)
    - Available online in word2vec (gensim in Python) package
    - Including sets of pretrained embeddings
  + Most algorithms use **Noise Contrastive Estimation**
    - Differences of Supervised Learning (SL) with NCE:
      * SL: **Per-instance** labels
      * NCE: Labels for each **instance-pair**
      * In NCE, labels are not annotated – they’re assigned with some heuristic (word2vec – local context)
    - Per-instance vs Instance-pairwise Labels
      * SL: given pairs of inputs and labels, learn map x->y
      * NCE: construct pairs of inputs and assign labels (binary) to denote how related these pairs are
  + Word2vec
    - Framework for learning word vectors using language modelling
    - Algorithm:

1. Start with large corpus of text
2. Every word in a fixed vocab (V) is represented by a vector (initially, random)
3. Go through each position t in text, which has a centre word c and context words o
4. Use similarity of word vectors for c and o to calculate probability of o, given c (or vice versa)
5. ???
   * + Skip-Ngram (*Lecture*)
       - Predict context words (position-independent) given centre word
       - Only parameters of the word vector are updated during each step of SGD
       - Vector for a context word gets updated when that word acts as the centre of a window
     + Continuous Bag of Words (CBOW)
       - Predict centre word from bag of context words (*Lab*)
     + <https://code.google.com/p/word2vec> - includes models and pre-trained embeddings
       - pre-trained – good because training takes a lot of data
     + Gensim: Python library that works with word2vec
     + **Problems**:
       - Words not in vocab (UNK) map to 1 vector
       - Phrases (ex: New York) not considered
     + Solution: Embedding character n-grams (FastText)
       - Treat character n-grams as the units to learn embeddings
       - Same skip-gram algorithm works with character n-grams and their contexts
       - Good practice to take range of different sized n-grams (practice: 3-5)
       - Can handle OOV (out-of-vocab) words
       - Good for lexemes (ex: -ed, -ing, -s)
   * Cool advantages:
     + Embeddings capture relational meaning!
       - ex: vector(king) – vector(man) + vector(woman) vector(queen)
       - ex: Paris – France + Italy Rome
     + ex: GloVe vectors word for comparative and superlative adjectives

* Evaluation
  + Commonly used:
    - Analogy Task
      * ex: (Berlin: Germany) == (Edinburgh: ?)
        + Compute NN(vec(Berlin) + vec(Germany) – vec(Edinburgh))
        + If got Scotland (groundtruth), then prediction is accurate
      * Compute average accuracy over several instances
    - Word Intrusion Task
      * Given a set of words, pick odd one out
    - Word Similarity Ranking Task
      * ex: Apple -> Orange, Grapefruit, Dog,
        + Compare embeddings, build ranking based on similarity
        + If ranking seems good => good
  + Extrinsic (task-based, end-to-end)
    - Word analogies
    - Question Answering
    - Spell Checking
    - Essay grading
  + Intrinsic
    - Correlation between algorithm and human word similarity ratings
    - Taking TOEFL multiple-choice vocab tests
* Post-transformation of Word2vec vectors
  + Retrofitting Word Vectors to Semantic Lexicons
    - Learns transformation matrix to make 2 word vectors that are semantically related in the Wordnet closer to each other
  + Sparse Overcomplete Word Vector Representations
    - Learns a linear layer with L1 regularisation to the dense vectors as (reasonably) sparse vectors
  + Bolukbasi et al. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings
    - Learns transformation matrix to neutralise the gender/race specific components of a word vector
* Generalisations
  + Node2Vec
    - Embeds nodes of a graph as vectors
  + WordNode2Vec
    - Constructs graph of local and document-level co-occurrences, then applies NCE
    - Good for text retrieval, search engines
* New models
  + Shift from context-free models to contextual models based on language modelling (predicting sequences of words)
  + ELMo
    - Contextual
      * depends on surrounding words
    - Deep
      * Combine all layers of a deep pre-trained NN
    - Character-based
      * Purely character-based, allowing to use morphological clues to form robust representations for OOV tokens unseen in training
  + BERT
    - Bi-directional context
    - Deep(er) + Wider
    - Based on Words + character n-grams (subwords when OOV)
    - Seq2Seq models: Transformer
    - Extensible
* Summary of Advantages:
  + Unsupervised
  + Context is (sometimes) used
* Apps:
  + Synonym handling in search
  + Document aboutness
  + Ad serving
  + Language models
  + Machine translation
  + Sentiment analysis

# Lecture 7: Context Vectors (ELMo & BERT)

Representations of a word

* Word vector tools: Word2vec, GloVe
  + These don’t include context (semantics, syntactic behaviour, register/connotations)
* Contextual:
  + Unidirectional (ELMo)
  + Bidirectional (BERT) (words before and after for context)

Intro to Recurrent Neural Networks (RNNs)

* Memory state
* As inputs are passed to the RNN, the memory is updated to remember what it has seen
* Memory states are passed as input progresses along with the inputted word
* Cons:
  + Banishing gradient problem
    - Not good at long-distance dependencies
    - Solution: Long Short-Term Memory (LSTM) RNN

LSTM RNN:

* Gate of Forgetting
  + what past hidden state is worth keeping
* Gate of Input
  + what past hidden state will be useful to figure out the input
* Gate of Output
  + What out of past state + this current input will be useful later?

ELMo:

* Language Modelling task
  + ELMo’s use comes from its pre-training
  + Train on a large dataset of unlabelled text
  + The pre-training task is Language Modelling
  + A language Model predicts the probability of the next word given context
* High-level
  + Bidirectional LSTMs, which are later concatenated
  + Output – dense vector for 1 word
* Specifics
  + Words as character n-grams
  + Learning objective – modelling language bidirectionally
  + Once trained, weights are “frozen”, but still providing contextual info
* Results:
  + Great for all tasks (before (Feb 2018) approaches were specific for each problem)

Transformer models

* GPT (OpenAI)
* BERT (Google AI)
* GPT-2 (OpenAI)

**Transformer** Architectures (High-Level)

* Problem with RNNs: remembering
  + Even with dedicated memory cells like LSTMs and GRUs, path length between cells grows with sequence length
  + This means RNNs on their own can’t remember for very long
  + Trouble with long-term dependencies
* Motivation
  + Want parallelisation, but RNNs are inherently sequential
  + Need **attention** mechanism to deal with long-range dependencies (selectively weighting words)
    - Path length between states grows with sequence length
    - Objective: have all context words in the input selectively influence the embedding of a target word
      * ex: “butter” different with “peanut” context
    - Self-attention: all words influenced by others in the same input
    - Target word is the “query”, context words are “keys”
    - Learning context with self-attention:
      * Calculate pairs of importance
      * Sum all weighted states to get contextual embedding
      * Attention layers can be stacked
      * Simulates a deeper network
    - Batch computation of attention weights
      * Depends on application
      * Attention function computed on a set of queries simultaneously
      * Packed together into a matrix Q
      * The keys and values are also packed together into matrices K and V
      * **Scaled Dot-Product Attention**
        + .
      * **Multi-Head Attention**
        + More parameters
        + Each matrix is linearly transformed with a “dense” layer
* Overview
  + **Non-recurrent** sequence-to-sequence encoder-decoder model
    - Recurrence effect is captured by the position embedding
    - Encoder-decoder: decode what output is needed
      * ex: machine translation – encode source sentence in a language into meaning of sentence, decode into target language
  + Good for GPU parallelisation
  + **Task**: machine translation
  + **Data**: parallel sentences in each language

**BERT** (Bidirectional Encoder Representations from Transformers) Models

1. **Semi-supervised** (pre-training)
   * Training on large amounts of text (books, Wikipedia, etc)
   * Objective: Predict masked word (language modelling)
2. **Supervised** (fine-tuning)
   1. Training on a specific task with a labelled dataset

* Many different ones, depending on training
* Pre-processing
  + **Custom Tokeniser**
    - Generates embeddings for words and sub-words
      * disable -> di, ##sable
  + Custom **tags**:
    - **[CLS]** – classification task, always the first sentence element
    - **[SEP]** – separates sentences, marks end of sentence
* Just a Transformer Encoder
  + Trained using a modification on the original Language Modelling task: Masked Language Modelling
    - Hide 15% of words from the input and try to predict them
  + Inherently bidirectional since **attention** allows all words to contribute to the prediction
  + BERT uses next-sentence prediction as a secondary task
* Tasks
  + Trained on 2 tasks:
  + Masked Language Model (MLM)
    - Replace random tokens with a special [MASK] and use context to predict it
  + Next Sentence Prediction (NSP)
    - Binary classification task
    - For two adjacent sentences, set binary label to 1 or 0
* Performance
  + Improves every task

Contextual embedding summary

* Shift from context-free models to contextual models based on language modelling (predicting sequences of words)
* ELMo:
  + Contextual: representation for each word depends on previous words
  + Character-based: purely, allowing the network to use morphological clues to form robust representations for out-of-vocabulary tokens unseen in training
  + Unidirectional (inherently)
* BERT
  + Bidirectional context
  + Based on Words + character n-grams
  + Seq2Seq models: Transformer model + attention
  + Extensible: to a classifier / ranking model
    - Classifiers are leading to major breakthroughs in NLP and IR

Summary

* Best to use **pre-trained** BERT models
  + Then **fine-tune** it to task
* BERT uses the Transformer architecture
  + Transformers use “attention”, which allows words to **influence each other through learnt weights**
* Pre-trained BERT models are powerful because they can be applied to many problems
* Don’t forget objective: BERT is just fancy software to capture and leverage word relationships

# Lecture 8: Information Extraction – NER & Linking

Goals:

* Organise info so it’s useful to **people**
  + Info boxes in Wikipedia
  + Summary of facts across an entire collection of news
* Organise info so it’s useful for **machine algorithms**
  + Data analytics
  + New knowledge
  + Question answering

In general

* Goal: Discover/extract structured info from text
  + Unstructured/semi-structured (Text) to Structured (Databases aka Knowledge Graphs)
  + Structure: often Subject-Relation-Object (RDF triples)
* How: by mining lots of info from a corpus
  + “Machine reading” of text
* Segmentation + classification + clustering + association
  + Segmentation – separate text into concepts
  + Classification – classify concepts according to category
  + Association – link together concepts
  + Clustering – Linking together themes of concepts
* Clear, factual info
* Extract **entities**, **relations** between entities, larger **events** taking place

Pipeline

* Named Entity Recognition – Detect and classify entities
* Coreference resolution – Resolve pronouns to named entities
* Entity Resolution (linking)
  + Ground entities to a representation of ‘world’ knowledge
* Relation Extraction – Classify relationships into tuples

Pre-defined task vs Open Info Extraction

* Most info extraction problems focus pre-defined tasks that are well-defined
* Predefined IE tasks
  + Decide what to include and what to exclude
  + ex: find all mentions of companies in news articles, find all pairs of counties and their capitals from text
* Open Info Extraction (Open IE)
  + Does not have a predefined task and attempts to turn all text into same structured form
  + ex tool: Stanford CoreNLP

Knowledge

* Knowledge Graph
  + Nodes with directed (sometimes labelled) edges
  + Vertices – entities
  + Edges – relations
  + Useful for all kinds of applications, including search and question systems
  + Queries can become traverses on a knowledge graph
  + ex: WikiData
* Knowledge Base
  + database with tables

Applications

* Finding mentions of specific entities (person, organisation, etc)
  + Can help search functionality
* Find “facts” mentioned in text
  + Build structures knowledge bases/graphs
    - Graphs – QA, etc
    - Bases – help find info without reading lots of text

Named Entity Recognition

* named entity – object referred to by a name, also quantity, date/time, etc
* Begin-Inside-Outside (BIO) labelling – whether token starts with NE, continuation of NE, not an NE
* Often paired with task of extracting type of entity
* Supervised sequence labelling problem
  + Supervised – needs annotated data for training
* Conditional random fields (CRF)
  + Improvement on Hidden Markov Models (HMMs)
  + Approach:
    - Turn into a supervised classification problem
    - Basic features are the current token and the previously predicted label
    - Can then add custom features (e.g., part of speech, capitalisation, etc)
  + Allow modelling of more complex sequences while integrating in extra features
* BERT-based sequence labelling
  + New deep learning language models can encode each token as a dense vector
  + Use that vector and feed it into another classifier to classify each token
* Alt: Solving with a huge dictionary of terms
  + 1. Defined a list of terms and synonyms
  + 2. Use exact string matching to find them in corpus
  + Pros:
    - Don’t need annotated examples to train a classifier
    - Works fairly well for specific cases (names of drugs, etc) where words are only used in a single context
  + Cons:
    - Completely ignores context of sentence
    - Requires exhaustive list of term & synonyms – not appropriate for some purposes
* Summary of tools:
  + Supervised:
    - HMMs
      * classic method – doesn’t do well with extra features
    - CRFs
      * Treats problem like a sequential classification problem
      * Can factor in extra features, like part of speech
    - BERT-based
      * Classification problem for each token
      * Generates dense context vectors for each token
      * Use vectors as input to a classifier for different token labels

Entity Linking

* Optional for some tasks
* Entity ‘Grounding’
* Low-level NLP: everything ungrounded
  + Recall: manipulate text with parsing and PoS but no idea what it means
* IE starts (partly) to resolve this
  + From strings to things
* Deal with **entities**. Can ground to “real world” (ex: entries in Wiki)
  + ex: “red” to a red image
* Link detected entities in text to a knowledge base
* **Challenge: Ambiguity**
* Formally
  + **AKA** Named Entity Disambiguation (NED), Wikification
  + **Input**: text doc D of words and a list of entity mentions
  + **Output**: list of mention-entity pairs where each entity is an entry in a **knowledge base** (KB)
* Challenges:
  + Nested entity mentions
    - Four nested entities
    - 3 layers deep
  + Slang entity mentions
    - ex: Obamessiah, Nobama
    - 1 solution: Entity linking with Urban Dictionary (Dalton, 2013)
* As info retrieval (IR)
  + Given a mention, perform search over the KB to find candidates
    - Simple string name matching heuristics
    - Retrieval algorithm (BM25, etc)
  + Optionally, re-rank candidates
    - Possibly ‘collectively’ with evidence from all mentions in D
* Using synonyms from the KB
  + Goal: Linking text to entities in a KB. Many KBs track synonyms for each term
  + But:
    - 2 KB entities may have the same synonym
    - What if the text matches no synonyms?
* Dense entity representations
  + Zero-shot Entity Linking with Dense Entity Retrieval
    - Learn a representation for each entity from text descriptions (cache these)
    - Perform approximate “fuzzy” retrieval in dense vector space, score candidates

NLP Pipeline

* Tokenisation and lemmatisation
  + Not as straightforward
* Part-of-speech tagging
  + Identify nouns, verbs, etc
* Parsing (dependency)
  + Diagramming sentences
* **Named Entity Recognition**
  + Detect and classify entities
* **Entity Linking (optional)**
  + Resolve strings to entries in a KG
* Relation Extraction
  + Classify relationships between entities
* Coreference resolution
  + Resolve pronouns to named entities

# Lecture 9: Relation Extraction & Coreference Resolution

Relations between entities

Motivation

1. Information extractions can extract facts from text. Often triples – 2 entities and a relationship.
2. If you collect enough triples, you have a knowledge graph/base.

Example:

|  |  |  |  |
| --- | --- | --- | --- |
| Entity 1 | Relation | Entity 2 | Easy/Hard for algorithm? |
| Obama | born on | August 4, 1961 | Easy |
| Obama | born in (location) | Kapiolani Medical Center for Women and Children | Medium |
| Obama | born in (country) | Honolulu, Hawaii |  |
| He | is | president | Easy |
| president | born outside | states | Hard |
| He | born to | American mother |  |
| He | born to | Kenyan father |  |
| she | of | English descent |  |
| mother | is | Ann Dunham |  |
| mother/Ann | born in | Wichita, Kansas |  |

## Relation Extraction

1. Parses text into structures relations to populate a database/knowledge base
2. Resource Description Framework (RDF) triples
   1. <subject> <relation> <object> [optional context]
3. Why is it difficult?
   1. Linguistic variability
      1. Entity recognition
      2. Word embeddings?
   2. Entity Ambiguity
      1. Apple – food or organisation?
      2. Entity recognition
      3. Entity resolution
   3. Implicit Relations
   4. Complex language with many clauses, long list of qualifiers, negations, etc

Relations

1. Binary relations
2. Non-binary
   1. ex: (company) appointed (person) as (position)
   2. Can be decomposed into binary

Example Knowledge Bases

1. WorNet: low-level semantic relations
   1. Hypernymy: “is a kind of”
   2. Meronymy: “subsists of”
2. Freebase / Satori / Wikidata / Google Knowledge Graph
   1. Films, celebrities, etc
3. SemMedDB – medical knowledge

How to extract:

1. Basic task
2. Find mentions of entities
3. Extract relations from the context of the mentions of the entities
4. Formulations:
   1. Rule-based (ex: Hearst patterns)
   2. Supervised (extract & score)
   3. Semi-supervised (bootstrapping)
   4. Unsupervised (clustering of patterns)
5. Named Entities aren’t enough
6. Richer relations using Rules
   1. Intuition: relations often hold between specific entities
      1. located-in (organisation, location)
      2. founded (person, org)
      3. cures (drug, disease)
   2. Start with Named Entity tags to help extraction
7. ReVerb: Relation Extraction from Verbs
   1. Paper
   2. Open information extraction – extract everything possible

Supervised Relation Extraction

1. Challenges:
   1. What type of task?
   2. What learning model?
   3. What features?
      1. ex:
2. Distant Supervision
   1. Because getting annotated data is costly and when 2 known associated entities are in a sentence, they are often discussed with the known relation, use an existing knowledge base (ex: WikiData) to label a lot of text data and use this as training data
   2. Data – silver standard (noisy)

Unsupervised: BERT

1. Matching the blanks: Distributional Similarity for Relation Learning
2. Result: Large-scale knowledge bases
   1. Facts that can be represented
   2. Applications:
      1. Monitor changes in the world (fun/profit)
      2. Run structured database queries (data analytics)
      3. Answer questions

Tools:

1. Entity resolution (linking)
   1. Spacy, TagMe, DbPedia Spotlight, etc
2. Relation extraction
   1. Stanford CoreNLP, DeepDive, Stanford MIMLE-RE, UMass Factorie
   2. Custom packages

## Coreference Resolution

1. Goal: Resolve different **mentions** of the same **entity**
   1. Mention – single “chunk” (often noun-phrase)
   2. Entity – grounded “thing”
2. Pronoun refers to what?

Motivation:

1. Information extraction
2. Machine Translation
3. Question answering systems
4. Chatbots and dialogue systems
5. Other semantic NLP tasks (summarisation, etc)

Coreference Process:

1. Detect the mentions (“easy”)
   1. Can be nested
2. Cluster (distinguish) the mentions (hard)

Mention detection

1. Mention – span of text referring to some entity
2. Kinds:
   1. Pronouns
      1. I, your, it, she, him, etc
      2. Use a PoS tagger
   2. Named entities
      1. People, places, etc
      2. NER tagger
   3. Noun phrases
      1. a dog, the big fluffy cat stuck in the tree
      2. Use a parser
3. Not as easy as it seems:
   1. “**it** is sunny”, “**every** student”, “**no** student”, “the best donut in the world”, “100 miles”
4. Linguistically:
   1. Coreference:
      1. Anaphora – expression depends on antecedent
         1. “Sally arrived, but nobody saw her”
      2. Cataphora – expression depends in postcedent
         1. “Before her arrival, nobody saw Sally”
   2. Other types of reference:
      1. Exophora – external context (may be unresolved)
         1. “He was standing over **there**”

Possible features:

1. Person/Number/Gender agreement
2. Semantic compatibility
3. Certain syntactic constraints
4. More recently mentioned entities preferred for references
5. Grammatical Role
6. Parallelism

Models:

1. Rule-based (pronominal anaphora resolution)
   1. Hobbs (1978) proposes an algorithm that searches parse trees for antecedents of a pronoun
   2. Algorithm:
      1. Starting at the NP node immediately dominating the pronoun
      2. Search previous trees, in order of recency, left-to-right, breadth-first
      3. Looking for the first match of the correct gender and number (male-female/singular-plural)
   3. Fails hard cases:
      1. These require nuanced semantic understanding of the world
      2. ex: The city councilmen refused the demonstrators a permit because they advocated/feared violence.
         1. **Winograd schema**
         2. Changing verb changes coreference
   4. Commentary:
      1. Hobbs in Lingua
2. Pairwise coreference model
   1. Task: Given a mention and earlier mentions, classify whether it refers to each earlier entity or not given the surrounding context
   2. Binary classification problem
   3. Create a **coreference chain** as a collection of **pairwise mention links**
   4. Make **independent pairwise** decisions
   5. **Reconcile** them in some deterministic way (ex: transitivity/greedy partitioning)
      1. A single mistake can lead to huge (incorrect) clusters
3. Coreference clustering
   1. Start with each mention in its own singleton cluster
   2. Merge a pair of clusters at each step
   3. Use a model to score which cluster merges are good
4. Neural
   1. Merge clusters
   2. Mention Pairs -> Mention-Pair Representations -> Cluster-Pair Representation -> Score

Evaluation

1. Many different metrics: MUC, CEAF, LEA, B-CUBED, BLANC
   1. Often report average over different metrics
2. B-Cubed – for each mention, compute precision & recall
   1. Precision - % of elements in a hypothesised reference chain that are in the true reference chain
   2. Recall - % of elements in a true reference chain that are in the hypothesised reference chain
   3. Overall precision & recall are the average of per-chain precision & recall
3. Optimising chain-chain pairings is a hard problem
   1. In the computational NP-hard sense – a greedy alignment is used

In practice – Spacy Coreference

1. Rule-based mention-extraction
2. Scorer is a neural mention ranking model
3. Still has lots of heuristics:
   1. greediness
   2. max\_dist
   3. max\_dist\_match
   4. blacklist

Progress:

1. 2010 – English 55, Chinese 50
2. 2017 – English ??, Chinese 63

Summary

1. Coreference is a useful, challenging, and linguistically interesting task
   1. Hard examples require reasoning
2. Variety of systems
   1. Rule-based
   2. Machine learning-based
   3. Neural network-based
3. Systems getting better rapidly, largely due to better neural models
4. Overall, results are still not excellent

# Lecture 10: Ethics, Explainability & Revision

Concerns

* Ethical
  + AI is being integrated into almost everything, including tools of warfare
  + There have been protests by tech workers about AI use in defence systems
  + ex: Job applications
    - What do you use as training/evaluation data?
      * Candidates that were finally hired. Bias present
    - Hiring has **massive biases** (ex: recruiters hire people that are from a similar background to them – often white males)
      * ML could focus on applicant names, gender, etc
    - ML methods can identify **correlation** and **not causation**
      * Can easily focus on unimportant characteristics, especially with a small dataset
    - Even data that is **seemingly filtered** for identifiable info can have correlated info
      * ex: name/location of university may correlate with ethnicity
  + Bias
    - Word and context vectors encode meaning from the text they are trained on
    - All text has biases, but internet text is *particularly* biased (most Internet users – young white males)
    - Paper – On the Dangers of Stochastic Parrots
      * Big data doesn’t guarantee diversity
        + Predominantly white & male
      * Encoded bias [Netflix doc]
      * Environmental/future use concerns
      * “Unknown” input data
        + How to document?
      * Chasing leaderboards isn’t helpful
      * They’re just parrots
      * “Are ever larger LMs inevitable or necessary? What costs are associated with this research direction and what should we consider before pursuing it?”
* Monetary
  + The monetary cost of a big model
    - estimated cost for GPT-3: 4.6 million USD
    - estimated time for GPT-3: 355 GPU years
  + Cost has implications for **who can use**/train a system
* Environmental
  + Amount of CO2
  + Training a language model uses a lot of energy and has environmental implications

Explainability

* The move from features to data
  + Evolution of AI models from feature-driven to data-driven
    - Feature-driven
      * Relied on human perceived abstract representations of the data
      * Hence, more clarity about what happens and how
      * Easy to include/exclude features based on intuition
    - Data-driven
      * Relies on machine-generated abstractions, e.g., 1D convolution for text, 2D convolution for images, etc
      * How do we control the predictions?
      * How do we convince others, e.g., think about convincing a medical person about an automated diagnosis
* Issues of trust
  + If users don’t trust a model/prediction, they won’t use it
  + Notions of trust:
    - Trusting a **prediction**, i.e., whether a user trusts an individual prediction sufficiently to take some action based on it
    - Trusting a **model**, i.e., whether the user trusts a model to behave in reasonable ways if deployed
  + Both are directly impacted by how much the human understands a model’s behaviour, as opposed to seeing it as a black box
* Explanations:
  + Prediction-level
    - Explain the predictions of any classifier (/regression model) by approximating it locally with an *interpretable* model
    - The explanations are usually in the form of *importance weights* assigned to different features
  + Model-level
    - Obtain per-instance *explanations*/feature lengths and then choose a set of representative instances
  + Different explanations for different predictions over the same instance
    - Recall that the objective depends on the predicted class labels
    - Hence, the explanation weights can be different for different predicted labels
* Transforming data to explanations with LIME
  + Objective: Estimate the (soft attention) weights of each feature of this instance
  + Points sampled around **neighbourhood**
  + Fit a simple (linear) classifier on this subset
  + This simple classifier *approximates* the behaviour of the complex decision boundary locally
  + Explain the current point with the parameters of this linear classifier
* Understanding how BERT works
  + Lots of applications require detailed Explainability
    - New language models (ex: Bert) seem to have some impressive capabilities, but we don’t fully understand how some of these capabilities work
  + BERT learns a traditional NLP pipeline
    - BERT models contain many layers, and it appears that each layer learns how to process text similar to the approach learnt earlier
    - ex: PoS tagging, constituency parsing, dependency parsing, entity tagging, semantic role labelling, coreference resolution, semantic proto-roles, relation classification
  + BERT seems to encode some knowledge
    - Could language models be used instead of DBs to store knowledge?
  + Language models will “hallucinate” knowledge
    - A language model will always guess missing words, even if it’s creating factually incorrect text
  + Representing language by dense vectors is a little “odd”
    - Dense vectors are a very powerful tool, but representing meaning by numbers feels strange
      * How do you deal with documents of various length?
      * Maybe we should keep other models in mind
    - If all research is focused on using dense vectors, maybe we’re putting our eggs in one basket. The field of NLP may change dramatically in the future

Revision

* Perplexity
  + Language models estimate the probability of a token sequence
    - The probability is very valuable for lots of applications:
      * Speech recognition
        + “I sea see shells” or “I see sea shells”?
      * Handwriting recognition
        + ex: “7 like chocolate” or “I like chocolate”?
      * Auto-complete
  + Good models give higher probabilities for valid text
  + Probabilities on a per-token basis
    - Calculating P for a large section of text quickly creates tiny probabilities. Even for a very likely text, the probability will be miniscule
    - For language data, we calculate average probabilities for the N tokens in our data
  + Measuring a model using valid language
    - Perplexity – measurement of how good the prediction is
      * Reciprocal of average probability of tokens in some held-back text
      * Low probability:
        + Text – unlikely
        + Perplexity – high
      * High probability
        + Text – likely
        + Perplexity – low (desired)
  + Generalising with valid and invalid data
    - Normally, we measure LMs using some test text that we know is valid
    - Hypothetically, we could also use text that we know is invalid, which we would want to have low probability
    - Remember, these are averaged probabilities across tokens
  + Relation to cross-entropy
    - Perplexity = 2 ^ cross-entropy
    - Cross-entropy – average negative log-likelihood of the test data
      * cross-entropy = -log2 p(test data) / N
    - perplexity = N / p(test data)
  + Core idea of perplexity for LMs
    - We want a language model that gives a **high probability** for our known valid text
    - Therefore, **low perplexity**
* Information Extraction
  + Methods included:
    - **Named Entity Recognition**
      * Could use predefined dictionary of terms or classifier system
      * **Entity linking** can ground a mention of an entity to a knowledge base
        + From strings to things
      * **Coreference** can help spot mentions that are talking about the same thing
    - **Relation Extraction**
      * 2+ entities are discussed in the same text. How are they associated?
      * Could track co-mentions (surprisingly powerful)
      * Or train relation extraction system that can distinguish different relations
  + Apps:
    - Useful for searching documents
    - Can enable humans to avoid reading lots of docs
    - Structured knowledge is invaluable for:
      * Search
      * Question answering and conversational agents (Alexa, Siri, etc)
* Context Vectors
  + Static Word Vectors
    - Similar/related words should be more similar vectors
  + Solution: Encode the words in context of where appear in text
  + ex pre-trained context vectors: ELMo and BERT