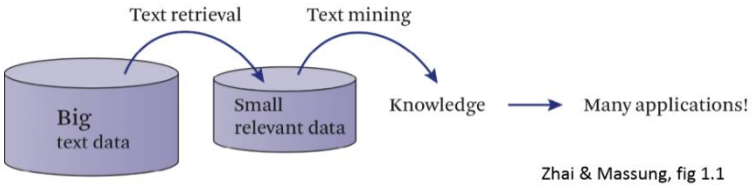
Lecture 1 – Introduction

Text Mining is automatic extraction of knowledge (structured) from text (unstructured).

Text mining is a form of data mining, using many of the same learning methods (classification and clustering). However, text data begins unstructured and requires text-specific processing.

Text mining (TM) and information retrieval (IR) are related disciplines, with IR often being the first step in the TM process. First retrieve documents using IR, then extract and structure the relevant information using TM.



Zhai & Massung, fig 1.1

Text mining and NLP
Natural Language Processing (NLP) is often used for information extraction. Not all NLP tasks are also TM tasks (machine translation, speech recognition), but some parts are used in TM.

The text mining pipeline
1. Retrieve documents from collection
2. Pre-process documents
3. Extract structured information

Types of text processing tasks
Three types of text mining tasks:
1. Text classification/clustering: assign a category or cluster per document (newspaper articles)
2. Sequence labelling: assign a category per word in a text (label person names, places (NER))
3. Text-to-text generation: input is text, output is text (summary, translation, etc.)

Challenges of text data
1. Text data is unstructured, or at best semi-structured
2. Text data can be multilingual
3. Text data is noisy. Spelling or OCR errors, as well as typography may give challenges in processing.
4. Language is infinite. A new document in your collection is likely to add new terms, which also adds new dimensions. The number of new words rapidly increases when the corpus (= your collection of texts) is small and will continue increasing indefinitely, but at a slower rate for large corpus (called Heaps' law).
5. Text data is sparse (as a classification object, not as a sequence)

Text as a classification object
When classifying documents, we represent the text as a **bag of words (BoW)**. In the traditional BoW model, the word order, punctuation, sentence and paragraph borders are not relevant. From a BoW, we construct a vector representation for each word. These words are used as features. Because there are only a few of all words in the corpus occurring in a given document, text vectors are sparse vectors.

Zipf's law
Given a text collection, the frequency of any word is inversely proportional to its rank in the frequency table. In English, the top four most frequent words are about 10-15% of all word occurrences. The top 50 words are 35-40% of all occurrences.

Text as a sequence
If we want to extract knowledge from a text, rather than simply classify it, the word order, punctuation, and capitalisation do matter. For example, you must use them to extract birth dates and birth places from a text.

Evaluation of text mining methods

Extrinsic evaluation
When you evaluate the complete application, so judging the automatic application as a human and seeing if humans are helped/satisfied by the results.

Intrinsic evaluation
When you evaluate just the components. For this, you need **ground truth labels** to see what the machine should have come up with. These labels are human-assigned labels in the data.

Evaluation metrics
- Accuracy = the percentage of correct classifications made by the model.
- Precision = the proportion of the assigned labels that are correct.
- Recall = the proportion of the relevant labels that were assigned.

Say A is the set of labels assigned by an algorithm, and T is the set of true labels.

$$\text{Precision} = \frac{|A \cap T|}{|A|} \qquad \text{Recall} = \frac{|A \cap T|}{|T|}$$

Example:
In the case of spam classification, we either classify emails as spam or non-spam. The accuracy would be the proportion of messages that is correctly labelled. But, there are two ways the label can be wrong: a spam message is said to be non-spam, or a non-spam message is classified as spam.
Precision (of the spam class) measures the proportion of the messages in the spam box were actually spam.
Recall (of the spam class) measures the proportion of the true spam messages that were correctly put in the spam box.

Lecture 2 – Preprocessing

From raw text to clean text
All text needs clean-up of some kind. OCR (scanned documents), but also html, Word documents.
Scanned and born-digital PDFs may contain:
- Photos, tables, graphics
- Layout or design information
- Disclaimers
- Headers, footers
- Column or page breaks
- OCR errors
- Character encoding errors

You can also encounter semi-structured text, text with markup such as HTML, XML, .docx).

Markup is meta-information that is clearly distinguishable from the textual content. In the case of XML and JSON, markup often provides useful information in text processing.

Character encoding
The way that a computer displays text so that humans can understand.
ASCII is character encoding that translates a binary string to a character. It is a 7-bit encoding based on the English alphabet. However, other parts of the world use alphabets other than English alphabet on which ASCII is based.
Unicode is a universal standard for all writing systems, containing more than 100,000 characters. For maximum compatibility, it is best to use Unicode.

Data cleaning in practice
The first steps in the text mining process are:

1. Digitalisation
2. Data conversion
3. Cleaning

These three necessary steps are time consuming, error prone and often complicated.
You can use regular expressions to find and count patterns in a file or string, or extract the matched patterns.

Tokenization and sentence splitting

A **token** is an instance of a word or term occurring in a document.
A **term** is a token when it is used as a feature, generally in normalized form (so lowercased, etc.).

Tokenization
The process of splitting a text into tokens. It is not as straight forward as removing punctuation and splitting on whitespaces.

Stop words
Stop words are extremely common words that do not carry any content (such as *a, an, are, and*, etc.). Eliminating stop words is a technique commonly used in IR and text classification.

> **Advantages**
> It reduces dimensionality (the number of features)
> It removes the noise of highly frequent words, improving your model
>
> **Disadvantages**
> Sometimes, stop words do carry important information
> A phrase needs stop words to make sense.

So, remove stop words for large datasets and long documents, but keep them in small datasets and short documents.

Sentence splitting
Many times, you can easily split sentences because they end on either ! ? . followed by a whitespace. However, this doesn't work in the case of abbreviations, names that include punctuation, sentences without punctuation such as titles, or line endings inside sentences (occurring in PDF conversions).

Lemmatization and stemming

It can be useful to normalize specific word forms to the same term, for example having just ‘bicycle’ for all occurrences of both ‘bicycle’ and ‘bicycles’. This reduces the number of features and generalizes better, especially in small datasets. There are two types of basic word forms, lemma and stem. We almost always prefer lemmas over stems. Stemming can be effective for very small collections.

Lemma
Is the dictionary form of a word. For a verb, this is the infinitive. So ‘thinks’, ‘thinking’, ‘thought’ all become ‘think’. For nouns, we use the singular form, so ‘mice’ becomes ‘mouse’.

Stem
Is the portion of a word that is common to a set of (inflected) forms when all affixes are removed, and is not further analysable into meaningful elements. It is the part of a word that never changes even when morphologically inflected, so it doesn’t have to be an existing word.
For example, ‘comput' for forms ‘computer’, ‘computing’, ‘computers’, ‘compute’.

Edit distance

You can use the edit distance to measure string similarity, which is useful for spelling correction and normalisation. For example, ‘graffe’ can mean ‘giraffe’ (differs by one letter), or ‘grail’ or ‘graf’ (more letters). The correct word is the word with the lowest difference.

Levenshtein distance
The minimum edit distance between two strings is defined as the minimum number of editing operations (insertion, deletion, substitution) needed to transform one string into the other. In Levenshtein distance, these operations all have a cost of one. The copy operation has cost 0.

Levenshtein algorithm
The algorithm uses a dynamic programming approach to calculate the least number of edits necessary to modify one string into the other string. It uses a m by n matrix with in each cell the Levenshtein distance between the m-character prefix of the one word and the n-prefix of the other word. If you fill the matrix in from upper left to lower right, the number in the lower right corner is the Levenshtein distance between the two words.

		a	n		a	c	t
	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
	2	1	1	1	2	3	4
c	3	2	2	2	2	2	3
a	4	3	3	3	2	3	3
t	5	4	4	4	3	3	3

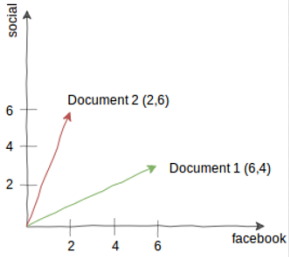
Suzan Verberne 2019

Lecture 3 – Vector Semantics

Distributional hypothesis
By Z. Harris in 1954. The context of a word defines its meaning, and words that occur in similar contexts tend to be similar.

Vector Space Model (VSM)
A way to represent documents and queries in a vector space where the dimensions are the words. There are two main problems with the VSM:

	doc1	doc2
facebook	6	2
social	3	6



- Because there are many ways to refer to the same object (bicycle and bike), we have poor recall.
- Poor precision because many words have more than one distinct meaning.

The alternative is to represent documents not by words but by:
- Topics: **topic modelling**
- Concepts: **word embeddings**

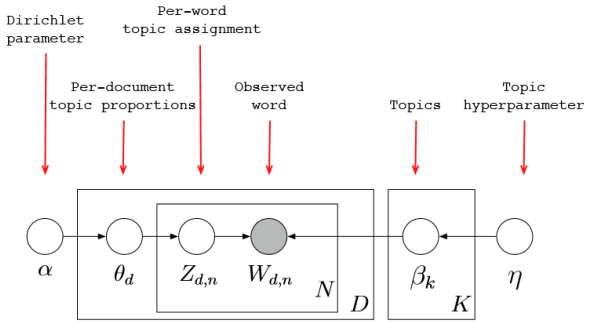
Topic Modelling

Assumes that each document consists of a mixture of topics, and each topic consists of a collection of words. It is often used for corpus exploration.
Three goals:
1. Uncover the hidden topical patterns in the collection
2. Annotate the documents according to those topics
3. Use the annotations to organise, summarise and search the texts.

It is an unsupervised technique, because topic labels are not given. The number of topics needs to be pre-specified.

Latent Dirichlet Allocation (LDA)
Is the most used topic modelling technique. The idea is that documents exhibit multiple topics, with each document being a random mixture of corpus-wide topics, and each word drawn from one of those topics. As we only observe the documents, our goal is to infer the underlying topic structure.

The goal of LDA is to learn β, the topic models. We only observe the words W. So, we start with random probability distributions of words in topics and of topics in documents.
α is the Dirichlet parameter that defines the distribution.
η is the number of topics.
We update the probability distributions while we observe the words in the documents, and keep doing this until β converges or the maximum number of epochs has been reached.



Each piece of the structure is a random variable.

The Dirichlet distribution is a continuous multivariate probability distribution. LDA works better than earlier topic modelling techniques due to the fact that it:
- Better generalises to new documents
- The dataset servers as training data for the Dirichlet distribution of document-topic distributions
- It is easy to sample the Dirichlet distribution and move forward from there if we have not seen a document.

LDA works (it puts 'topical' words together) because word probabilities are maximized by dividing the words among the topics. This results in finding clusters of co-occuring words. The Dirichlet on the topic proportions penalises a document for using many topics, which leads to sets of terms that more tightly co-occur.

Challenges with LDA is that you have to pre set the number of topics. It needs a large dataset to give stable results, and it's non-deterministic because the randomly initiate the clustering. Also, interpreting the output (the topics) is left to the user.

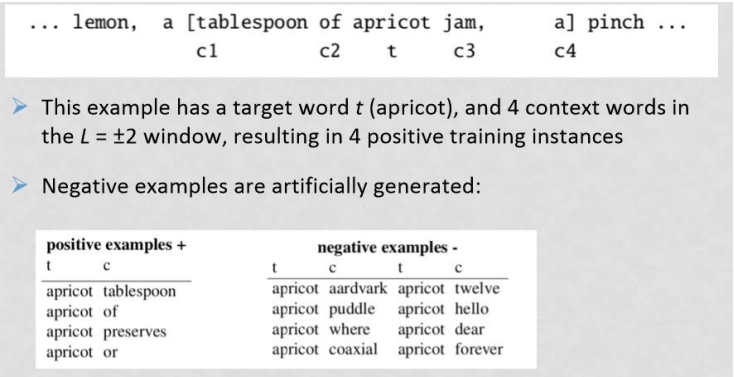
Evaluation of topic modelling
Word intrusion: given 5 high probability topic words and one random word, can you find the intruder?
Topic intrusion: given a document and 5 high probability topics and one random topic, can you find the intruder?

Word embeddings

As mentioned, the Vector Space Model is not great with its sparsity. It treats words as discrete atomic symbols, leading to arbitrary encodings that provide no useful information about the relations between words.
Word embeddings are dense representations of words, as opposed to the sparse vectors. They embed (represent) words in a continuous vector space, which is relatively low-dimensional, containing only 100 to 400 dimensions as opposed to 10000s in a VSM. Semantically and syntactically similar words are mapped to nearby points (following the **distributional hypothesis**).

Word2Vec
Is a predictive model for learning word embeddings from raw text. It works by training a classifier on a binary prediction task (ex. does word *w* show up near word *bicycle*). Then, we don't care about the outcome of the task but use the learned classifier **weights** as the word embeddings.
It starts with a large collection (1 million words), followed by extracting the vocabulary (say, 10000 terms). It's goal is to represent each of these 10,000 terms as a dense, lower-dimensional vector, typically 100-400 dimensions.

Training Word2Vec is rather simple. It uses binary classification of words in the text, by treating the target word and a neighbouring context word as positive examples, and randomly sampling other words to get negative samples. A classifier is trained to distinguish the two cases, and the weights of the classifier are used as embeddings. The classifier used is a neural network with one hidden layer. This hidden layer uses logistic sigmoid functions as activation functions. Then, the regression weights are used as the embeddings.



The outcome of the classification determines whether we adjust the current word vector. Gradually, the vectors converge to sensible embeddings for words.

Word2Vec has several advantages:
- It scales well
- Pre-trained word embeddings can be used in other research, for entirely different tasks
- Support incremental training

Word2Vec use cases
Find the similarity of two words.
Find the most similar word for a given word.
Find the word that doesn't match the others in a string.

It can be used to uncover knowledge about natural language, learning semantic and semantic relations. (A is to B as C is to ??)
vector(king) – vector(man) + vector(woman) = vector(queen)
This can also be used for syntactic relations.

Can also be used to improve NLP applications, for example in sentence completion and text prediction.

Lastly, it can improve TM applications, as it can be used in document similarity, sentiment classifications.

Deep Neural Networks on text data take word embeddings as low-level representations of words.

Lecture 4 – Text Categorization

Different kinds of categorization
- Binary classification (a text either is or isn't a certain category)
- Multi-class classification (a text is one of X categories)
- Multi-label classification (a text can be multiple of X categories)

The text categorization process

What is needed for text classification?
1. A definition of the task
2. Example data
3. Pre-processing
4. Feature extraction
5. Classifier learning
6. Evaluation

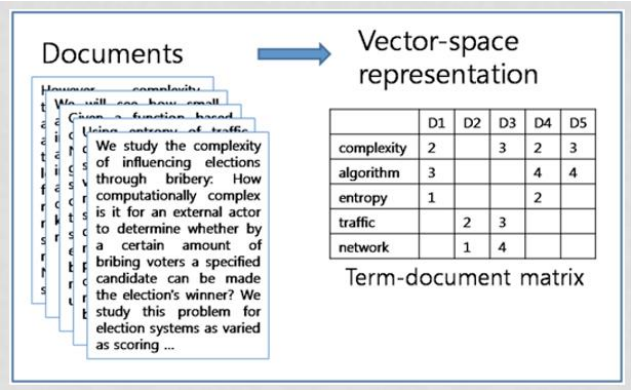
A definition of the task
What is the text unit (document)? Is it complete documents, sections, sentences?
What are the categories? Spam/no spam, language, sentiment, topic, …?

Pre-processing
We at least have to use tokenization. Most applications use lowercasing and punctuation removal. We can also remove stopwords, and apply lemmatisation or stemming. It is also possible to add phrases as features ('not good'). These can be bigrams or trigrams or noun phrases.

We can also create lower level features: character k-grams. A 4-gram of the string "you will. To" would be: you_ ou_w u_wi _wil will ill. ll._ l._T ._To

Feature selection
Then, we have to decide on vocabulary size, so we must select our features. The goals here are to reduce the dimensions and reduce overfitting. We can use either **global term selection** or **local term selection**. In global term selection, we include or exclude words based on their overall term

frequency in the document. In local term selection, we score each term by a scoring function that captures its degree of correlation with each class it occurs in.
Only the top-n terms in a document are then used for classifier training.



	D1	D2	D3	D4	D5
complexity	2		3	2	3
algorithm	3			4	4
entropy	1			2	
traffic		2	3		
network		1	4		

Once we have defined the terms (features) in our vocabulary, we can assign weights (feature values). We can construct a term-document matrix to quickly see our vocabulary and documents.

Term weighting
There is three types of term weights we can use.

Term frequency (tf)
The term count $tc_{t,d}$ of term t in document d is the number of times that t occurs in d. The raw term count is not what you want to use, because a document with tc=10 is more relevant than a document with tc=1, but not ten times more relevant. So, we use the log frequency.

$$tf_{t,d} = \begin{cases} 1 + \log_{10} tc_{t,d} & \text{if } tc_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Inverse document frequency (idf)
Intuition tells us that the most frequent terms are not very informative.
If we have df_t as the document frequency, so the number of documents that t occurs in, df_t is an inverse measure of the informativeness of term t. N is the number of documents we have.

$$idf_t = \log_{10} \frac{N}{df_t}$$

Tf-idf
Defined as tf * idf

Examples
We have a collection of 1 million documents. The term t, computer, occurs in 100 documents. What is its idf?
$idf_{computer}$ = log(N/100) -> log(1000000/100) = 4

We have a document with 10 times the word 'computer'. What is the tf-idf for this word?
$Tf_{computer}$ = 1 + log(10) = 2
$Idf_{computer}$ = 4
$Tf\text{-}idf_{computer}$ = 2*4 = 8

Classifier learning
Estimators well-suited for text data:
- Naïve Bayes
- Support Vector Machines
- Stochastic Gradient Descent

Naïve Bayes

Is a learning and classification method based on probability theory, using prior probability of each category given no information about an item. The classification produces a posterior probability distribution over the possible categories given a description of an item.

It is based on Bayes' theorem:

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)}$$

Prior

Evidence

Task it to classify a document d based on its feature representation X(d): $\{t_1, t_2, \ldots, t_k\}$

Learning the model

For learning, it uses the frequencies in the training data.

P(c) = $|D_c|$ / $|D|$, or the number of documents for class c / the total number of documents.

P(d | c) = P(t_1, t_2, …, t_k | c), where P(t | c) =

When we have done P(t | c) for each term t, we can multiply them to get P(d | c).

$$\frac{Tr_{ct}}{\sum_{t' \in V} Tr_{ct'}}$$

the number of occurrences of t in training documents from class c

total number of term occurrences in training documents from class c

Doc id	Content	Class
1	request urgent interest urgent	Spam
2	assistance low interest deposit	Spam
3	symposium defence june	No spam
4	siks symposium deadline june	No spam
5	registration assistance symposium deadline	?

- P(spam) = 2/4
- P(registration|spam) = 0/8
- P(assistance|spam) = 1/8
- P(symposium|spam) = 0/8
- P(deadline|spam)=0/8
- P(no spam) = 2/4
- P(registration|no spam) = 0/7
- P(assistance|no spam) = 0/7
- P(symposium|no spam) = 2/7
- P(deadline|no spam)=1/7

This example shows P(c) for c = spam in the first two lines.

The other lines are P(t|c), with c = spam and t being one of the terms occurring in the four annotated documents.

Now that we have calculated this, we can calculate P(d | c) for d = document 5 and c = spam and c = no spam.

P(registration assistance symposium deadline | spam) =

0/8 x 1/8 x 0/8 x 0/8 x 2/4 = 0

P(registration assistance symposium deadline | no spam) =

0/7 x 0/7 x 2/7 x 1/7 = 0

We see that the probability for both classes is zero, which of course should be impossible because we can't classify it with P(class) for both classes being zero. We always get this for term-class classifications that did not occur in the training data. So, we apply add-one smoothing (LaPlace smoothing). This is a uniform prior, the assumption that each term occurs one additional time for each class.

This turns P(t|c) into $P(t \mid c) = \frac{T_{ct}+1}{\sum_{t' \in V}(T_{ct'}+1)} = \frac{T_{ct}+1}{(\sum_{t' \in V} T_{ct'})+|V|}$

So, our P(t|c) becomes

- P(spam) = 2/4
- P(registration|spam) = (0+1)/(8+11)
- P(assistance|spam) = (1+1)/(8+11)
- P(symposium|spam) = (0+1)/(8+11)
- P(deadline|spam) = (0+1)/(8+11)

- P(no spam) = 2/4
- P(registration|no spam) = (0+1)/(7+11)
- P(assistance|no spam) = (0+1)/(7+11)
- P(symposium|no spam) = (2+1)/(7+11)
- P(deadline|no spam) =(1+1/7+11)

We do + 11 because there are 11 unique words in our documents.
Now, we get the following probabilities:

- P(spam|registration assistance symposium deadline) = $7.67 * 10^{-6}$
- P(no spam|registration assistance symposium deadline) = $2.86 * 10^{-5}$

The probability for no spam is higher than the one for spam, so we classify (registration assistance symposium deadline) as no spam.

Naïve Bayes assumptions
Conditional Independence Assumption: features are independent of each other given the class.
Positional Independence Assumption: the conditional probabilities of a term are the same, independent of the position in the document.

Naïve Bayes accuracy
Classifications by NB are usually fairly accurate. However, due to the inadequacy of the Conditional Independence Assumption, the actual posterior-probability numerical estimates are not.
NB is a good baseline for text classification.

Classifier Evaluation

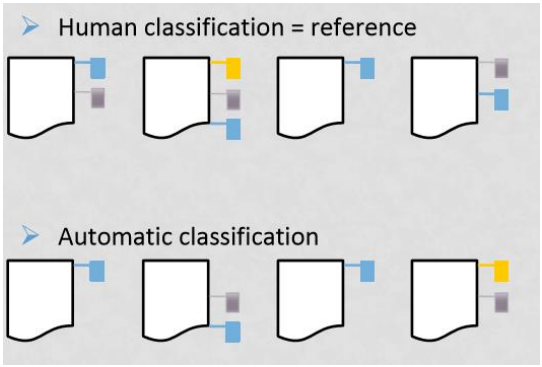
We use a held-out test set for evaluation of a classifier. This prevents overfitting on the training set and makes for a fair evaluation on unseen data. We do this by doing a train-test split (normally 80-20). In the case of small datasets, we can use cross validation.

Due to unbalanced classes, accuracy is not a good metric to evaluate a classifier, as high accuracy in one class may mean low accuracy in the other class. It is also possible we are more interested in the correctness of the labels rather than the completeness of the labels.

Precision and recall

Recall: there are 8 true categories (assigned by humans)
the classifier assigned 5 of them
recall = 5/8

Precision: the classifier assigned 6 categories
5 of them match human categories
Precision: 5/6



F1 is the mean of precision and recall:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

Lecture 5 – Data Collection

In supervised learning, we need example data for training and evaluation. To gather this data, we can either choose to use existing labelled data or to create our own labelled data.

Existing labelled data

This is a benchmark dataset created by someone else, maybe using existing expert annotations or labelled user-generated content. A benchmark dataset is used to evaluate and compare methods, often created in the context of shared tasks.

Advantages

- High quality
- Reusable
- Allows you to compare your results to others

Disadvantages

- Not available for every problem.

Existing expert classifications

These are labels that were added to items by humans, but not originally created for training automatic tools. For example, keywords in digital libraries were not intended for use by automatic tools, but for people using the library. Another example is the international patent classification system. It contains millions of patents manually classified in a hierarchical classification system by patent experts.

Advantages

High quality, potentially large and often freely available

Disadvantages
Not available for every specific problem, and not always directly suitable for training classifiers.

Labelled user-generated content
For example, hashtags on twitter (to detect sarcasm), or scores in product reviews (to learn sentiments and opinion), or likes on Facebook comments to learn to detect relevant texts.

Advantages
Potentially large, human created, freely available

Disadvantages
Noisy, possibly low-quality, indirect signal

Create labelled data

1. Make a sample of items
2. Define a set of categories
3. Write annotation guidelines V1
4. Test and revise guidelines with new annotators until the guidelines are sufficiently clear.
   The task should be clearly defined, but not too trivial.
5. Choose humans (experts or crowdsourced?)
6. Compare the labels by different annotators to estimate the reliability of the data (inter-rater agreement)

You need at least hundreds of examples per category, the more the better. The more difficult the problem, the more examples you need.

Crowdsourcing
Useful for tasks that humans are better at than computers, and where no experts are needed (no domain-specific knowledge). Most used platform: Amazon Mechanical Turk.

The main challenge when crowdsourcing is quality control. First rule is not to pay too little.
Have a check in the task set-up to make sure they pay attention. Always say that their annotations are compared to expert annotations (even when they're not).
Use the **inter-rate agreement** to evaluate the reliability of the data.

Inter-rate agreement
Two human classifiers never fully agree. So, always have part of the example data labelled by 2 or 3 raters and compute the inter-rater agreement to know the reliability of the example data (and the difficulty of the classification task). The measure for inter-rate agreement is Cohen's Kappa.

$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)}$$

Pr(a) = the actual (measured) agreement, the percentage they agreed on.
Pr(e) = the expected (chance) agreement based on the probabilities of occurrence of each of the values.

Example:

Pr(a) = (20+15) / (20+5+10+15) = 0.7

Pr(e) =

A1 says yes to 25 items, 0.5 of all items

A2 says yes to 30 items, 0.6 of all items

Pr(e, yes) = 0.5 x 0.6 = 0.3

Pr(e, no) = 0.5 x 0.4 = 0.2

Pr(e) = 0.3 + 0.2 = 0.5

K = (0.7 – 0.5)/(1 – 0.5) = 0.2/0.5 = 0.4

Agreement table		A2	
		Yes	No
A1	Yes	20	5
	No	10	15

Interpretation of Kappa

- < 0: no agreement
- 0–0.20: slight agreement
- 0.21–0.40: fair agreement
- 0.41–0.60: moderate agreement
- 0.61–0.80: substantial agreement
- 0.81–1: almost perfect agreement

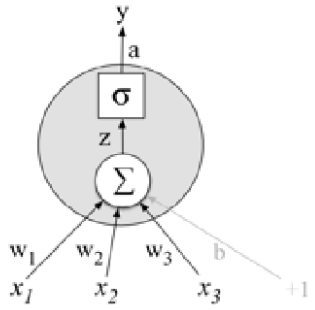
Lecture 6 – Neural NLP and Transfer Learning

The difference between neural network methods and traditional models:

- Traditional classifiers use hand-derived features
- Neural networks take **raw data as input** and learn to induce features as part of the process of learning to classify.

Neural networks share much of the same mathematics as logistic regression, but they are much more powerful because:

- Multiple nodes = multiple functions = non-linearity
- Multiple layers = multiple abstractions over the input data
- A minimal neural network can be shown to learn any function



The figure shows a single unit with one function for three input variables. σ is the non-linear activation function (sigmoid function). The most used activation function is ReLU (Rectifier Linear Unit): y = max(x, 0).

Neural networks learn the weights automatically, using supervised learning with the true labels. Hidden layers learn to form useful representations by learning the weights that optimize the output classification.

Feedforward networks

A feedforward network is a multilayer neural network in which the units are connected with no cycles. The outputs from units in each layer are passed to units in the next layer, and no outputs are passed back to lower layers. So, simple feedforward networks have three kinds of nodes: input, hidden, and output. In standard architecture, each layer is fully connected, so each unit in each layer takes as input the outputs from all units in the previous layer, meaning each hidden unit sums over all the input units.

Feedforward network as classifier
For binary classification, there is a single output node (yes/no). y is the probability positive output.
In multi-class classification, there is one output node for each category. y is the probability of that category.

So, the output layer gives us a probability distribution. Here, the output of classifier z is transformed to a probability distribution using the *softmax* function.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}} \quad 1 \le i \le d$$

d=dimensionality
z = [0.5,3.6,-1.2]
softmax(z) = [0.043,0.949,0.0078]

Training neural networks
Feedforward neural networks are trained using supervised learning with input x and correct output y. The system produces ŷ, an estimation of the true y.
The goal of the training procedure is to learn parameters W[i] and b[i] for each layer i that make ŷ for each training observation as close as possible to the real y. For this, we need two things:

1. Cross-entropy loss: a loss function that models the distance between ŷ and y.
2. Gradient descent algorithm: an algorithm to find the paramters that minimize the loss function.

Cross-entropy loss: $$L_{CE}(\hat{y}, y) = -\sum_{i=1}^{C} y_i \log \hat{y}_i$$

The challenge in training a feedforward network is that we have to optimize the weight parameters in all layers of the network, even though we only calculate the loss at the very end of the network. The solution for this is the **error backpropagation algorithm**. This is an iterative, recursive and efficient method for computing the weights updates to improve the network.
We also **dropout** to prevent overfitting. This is randomly dropping some units and their connections from the network during training.

The parameters of a neural network are the weights W and biases b, which are learned by gradient descent.
Hyperparameters are parameters of the architecture and learning setup. Ex: number of layers, dropout rate, activation function, number of hidden nodes per layer, learning rate η, etc. These are chosen by the algorithm designer, and tuned on a development set rather than set by gradient descent learning on the training set.

Neural Language Models
Language modelling is a word prediction task. We can use the traditional n-gram model (given the previous n words, predict the next word) or use a neural language model, a feedforward neural network that takes a representation of the previous n words and outputs a probability distribution over possible next words. Neural language models are able to handle much longer histories, and they can generalize over contexts of similar words.
The language model is the resulting embeddings matrix, which can be used as representation in other tasks.

Transfer Learning

We can learn feature representations (the word meanings) on a large text collection, for example Wikipedia, and use the resulting representations to learn to classify or label text with a much smaller labelled training set.

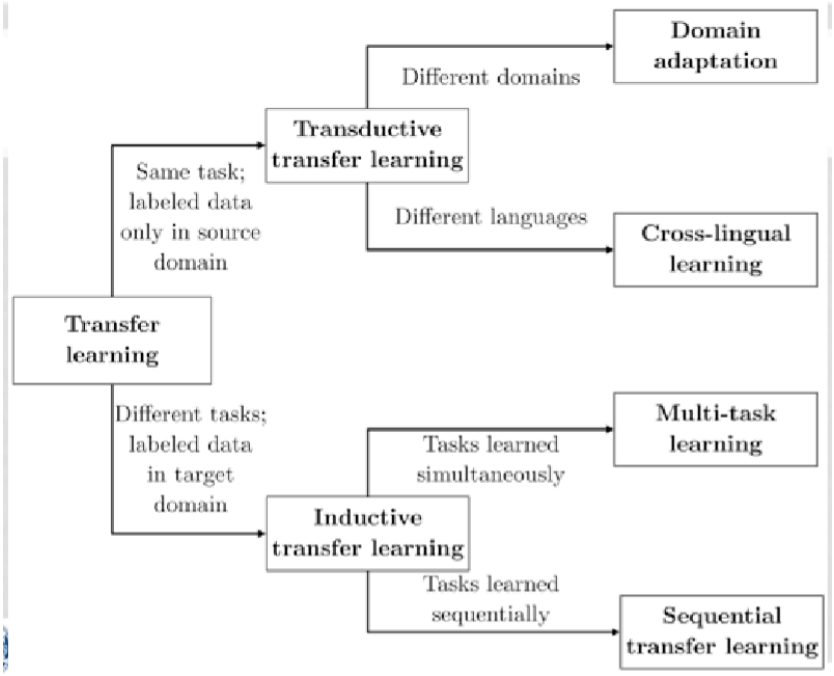
Transfer learning with neural language models
Inductive transfer learning is transferring the knowledge from pretrained language models to any text mining tasks. When you want to transfer to the target task, there are two options:

1. Direct use (feature extraction): do not change the pretrained weights, and just train a linear classification on top of the pretrained representations.
2. Fine-tuning: change the pretrained weights. Use them only as the initialisation for parameters of the downstream model. The whole pretrained architecture is then trained during the adaptation phase.

We want to avoid overwriting useful pretrained information (called catastrophic forgetting). The more parameters we choose to train again, the slower the training will be. Fine tuning can of course give better results, but often requires more extensive hyperparameter searching.

Types of transfer learning
The target task is often a low-resource task.

1. Sequential transfer learning. If related tasks are available, we can fine-tune our model first on a related task with more data before fine-tuning on the target task.
2. Multi-task fine-tuning. Alternatively, we can fine-tune the model jointly on related task together with the target task. This related task can be an unsupervised auxiliary task.

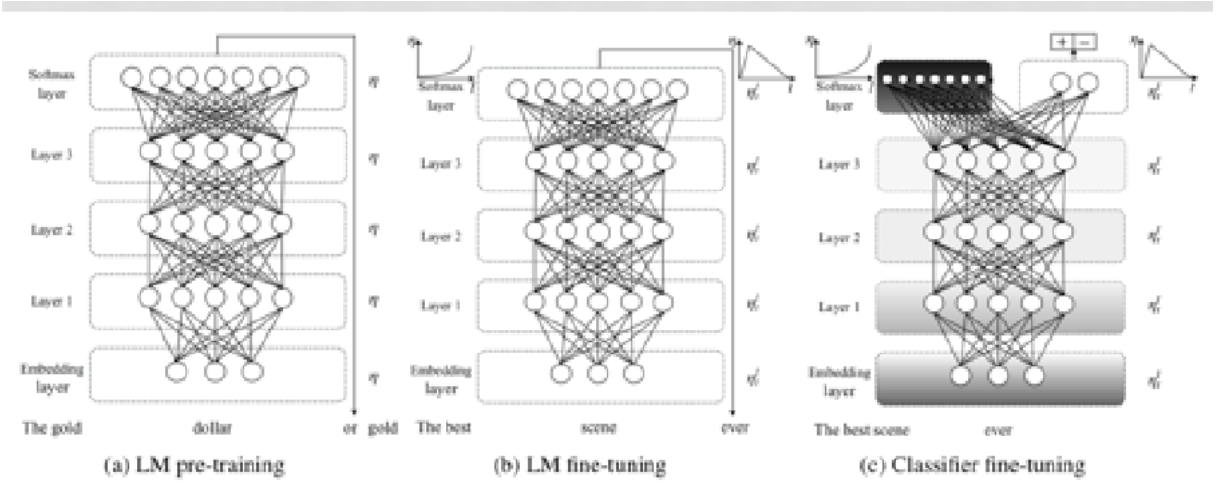


State of the art models

ULMFIT (Universal Language Model fine tuning)
Three stages:

1. Language model is trained on general-domain corpus to capture general features of the language in different layers

2. The full language model is fine-tuned on target task data using discriminative fine-tuning '*Disc*' and slanted triangular learning rates (STLR) to learn task specific features.
3. The classifier is fine-tuned on the target task using gradual unfreezing, '*Disc*' and STLR to preserve low level representations and adapt high-level ones.



(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

BERT

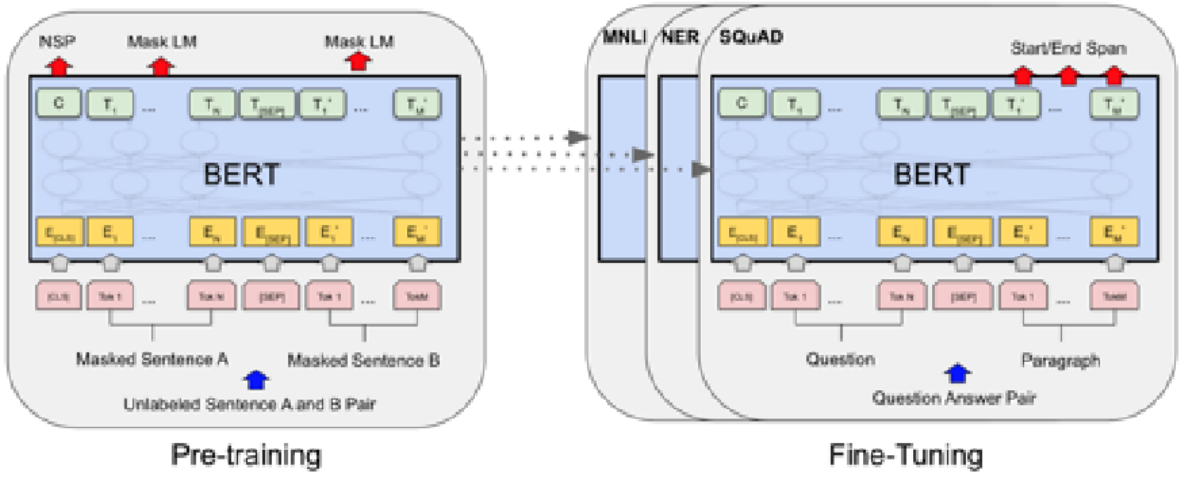


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

Intuition on language modelling tasks:

- Predicting randomly masked word in context (bi-directional), useful to capture the meaning of words.
- Next-sentence classification to capture the relationship between sentences.

BERT can achieve state-of-the-art results on a large range of tasks in a large range of domains. Its pretrained models can be easily fine-tuned.

Challenges of state-of-the-art models

They need heavy pre-training, and need dropout or other forms of regularisation to prevent overfitting. Parameter freezing/updating/forgetting

Optimising hyperparameters on the development set takes time, so you can adopt them from the pre-training task. They are difficult to explain or interpret, an attention layer can help with this.

Because of these challenges, traditional methods are also still commonly used in classification tasks.

Lecture 7 – Information Extraction

The general goal of information extraction from text is to discover structured information from unstructured or semi-structured text.
Example applications:

- Identifying and linking biomedical entities from patents to existing knowledge bases
- Finding and obfuscating person names in confidential documents
- Identifying company names and information from economic newspapers
- Advanced search problems.

Information extraction tasks:

- Named Entity Recognition (NER)
- Relation extraction

Named Entity Recognition

A named entity is a sequence of words that designates some real-world entity, typically a name.

- General types, such as persons, organisations, locations
- Extended types, dates, times, monetary values, percentages.
- Domain-specific types, biomedical entities.

Challenges of NER

- Ambiguity of segmentation: what is an entity and what is not an entity? What are the boundaries of an entity?
- Type ambiguity: JFK can refer to a person, an airport, schools, bridges etc.
- Shift of meaning: President of the US means Obama, then Trump, etc. depending on time.

Recognizing entities
The easy approach is to use a list of names and simply label them in the text. However, we would need to add all kinds of variants of a name, the list is never done, and entities are typically multi-word phrases so we have to deal with boundaries.
So, NER is really a sequence-labelling problem. It is a word by word sequence labelling task, in which the assigned tags capture both the boundary and the type.

Sequence labelling
A sequence is a sentence, of which a word is an element. We label this with the entity type, with one label per word. In IOB tagging, we tag each word beginning (B), inside (I) an entity or outside (O) an entity.

Methods for NER

1. Feature-based NER
2. Neural-network-based NER
3. Rule-based NER

Feature based NER

A supervised learning task where each word is represented by a feature vector with information about the word and its context. We need training data in the form of IOB labelled texts.

Part of speech

Useful information for feature based NER. We want information on the word type and neighbouring types.

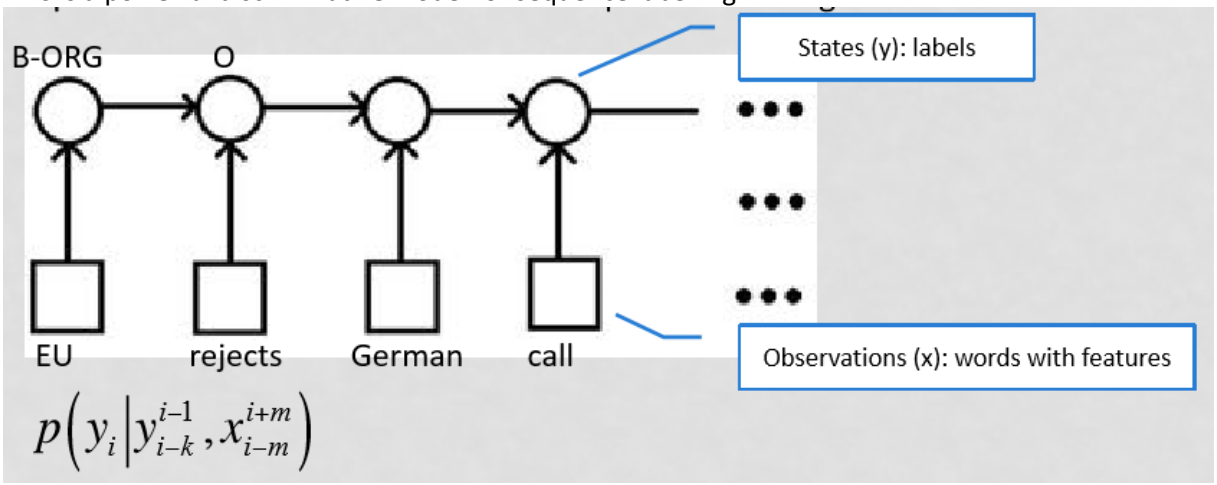
- Noun, verb, adjective, adverb
- Pronoun, preposition, conjunction, determiner

Gazetteer and word shape

A gazetteer is a list of (place) names or name list containing common first and last names.
Word shapes are features used to represent the abstract letter pattern of the word by mapping lower-case letters to x, uppercase to X, numbers to d and retaining punctuation.

Maximum Entropy Markov models

This is a powerful discriminative model for sequence labelling.



As the figure shows, it uses the IOB label of previous words as well as the word features. The formula captures the following: the probability of the current state y_i given the previous states y, the current observation x_i, the previous observations x_{i-m} to x_{i-1} and the next observations x_{i+1} to x_{i+m}

Conditional Random Fields

An indirect graphical model, which derives its power from bidirectionality. At each time step, instead of computing a probability for each tag, it includes features and predicted labels of words in future time steps. This optimises the sequence as a whole. The probability of the best sequence is computed using the Viterbi algorithm.

Neural models for NER

The most commonly used neural sequence model for NER is bi-LSTM, a form of recurrent neural networks. LSTM stands for Long-short term memory. An RNN is a network that contains a cycle within its connections. This is similar to a normal feedforward network, but with an addition: a set of weights U that connect the hidden layer from the previous time step to the current hidden layer. These weights U determine how the network should make us of past context in calculating the output for the current input. These connection weights are trained using the usual backpropagation.

Bi-LSTMs

Word and character embeddings are computed for word w_i and the context words, and passed through a bidirectional LSTM, whose outputs are concatenated to produce a single output layer at position *i*. We could then go to the softmax layer to choose tag t_i, but this is insufficient for NER, because there are strong constraints for neighbouring tokens. So, we use a CRF layer on top of the biLSTM output.

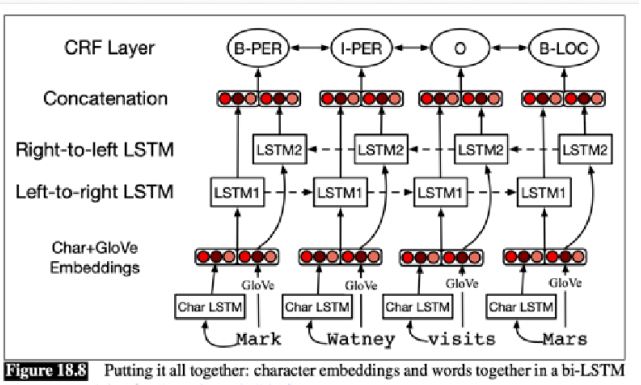


Figure 18.8 Putting it all together: character embeddings and words together in a bi-LSTM sequence model. After Lample et al. (2016).

Rule based NER

Academic research uses machine learned sequence models, but commercial applications for NER often still use lists and rules, complemented with a little bit of supervised learning.

Approach:

1. Use high-precision rules to tag unambiguous entity mentions.
2. Search for substring matches of the previously detected names
3. Consult application-specific name lists to identify likely named entity mentions from the given domain.
4. Apply probabilistic sequence labelling techniques that make use of the tags from previous stages as additional features.

Relation Extraction

Resource Description Framework (RDF) can be used to turn facts (Golden Gate Park is in San Francisco) into a metalanguage. An RDF is a tuple of entity-relation-entity, called a subject-predicate-object expression:

Subject	**predicate**	**object**
GoldenGatePark	location	SanFrancisco

Several methods for relation extraction:

1. Pattern-based
2. Feature-based (supervised learning)
3. Bootstrapping
4. Distant supervision

Feature-based learning is the most reliable, but it requires labelled data. If we don't have that, we must fall back on the other options.

Pattern-based relation extraction

Example sentence: "Agar is a substance prepared from a mixture of red algae, ***such as*** Gelidium, for laboratory or industrial use."

The pattern here is: NP0 *such as* NP1. (An NP is a Noun Phrase, a phrase with a noun as syntactic head). This generalises to a hyponym(NP1, NP0).

Advantages are high-precision and the ability to be tailored to specific domains. However, it suffers from low recall and is not generalisable.

Feature-based relation extraction
A supervised learning task that assumes two entities and one relation, with the relation verbalised in one sentence. It sees extraction as a classification problem:

1. For each pair of entities in a sentence,
2. determine whether or not they have a relationship
3. and if they do, what the relation is.

Features could be:

- co-occurrence frequencies
- entity features (words, entity type)
- lexical contextual features (ex. “founded”)
- syntactic contextual features (ex. SUBJ – “founded” – OBJ)
- background knowledge (entity clusters from a large embeddings model)

Bootstrapping

1. Start with a set of known pairs, for example *medicine - side effect*.
   Flomax – fever
   Flomax – cough
   Etc. etc.
2. Search for co-occurrences of these items in a large text collection (“Flomax may cause fever”, “Side effects of Flomax, including fever”, etc.)
3. Extract generalised patterns (Hearst patterns) such as X may cause Y, Side effects of X, including Y.
4. Filter for unreliable patterns
5. Use the reliable patterns to find more pairs of entities.

Distant supervision

1. Start with a large, manually created knowledge base
2. Find occurrences of pairs of related entities from the database in sentences. This rests on the assumption that if two entities participate in a relation, any sentence that contains these entities expresses that relation.
3. Define features over the sentences expressing the relation (using supervised learning)
4. Apply these features to sentences with yet unconnected other entities in order to find new relations.

Evaluation

You need manually annotated documents as the ground truth, best is to use benchmark datasets because they are ready-to-use and well edited. They are often created in the context of conferences and workshops.
There are two criteria for correctly identified named entities:

- Correct entity identification (boundaries)
- Correct entity classification (types)

We can calculate Precision and Recall using human labels as the ground truth.

F = the set of entities found by the algorithm
T = the set of true entities according to humans

$$\text{Precision} = \frac{|F \cap T|}{|F|} \qquad \text{Recall} = \frac{|F \cap T|}{|T|}$$

Lecture 8 – Summarization

Typology
Purpose:
- Informative vs. indicative
- Generic vs. user-oriented
- General purpose vs. domain-specific

Input: single document vs. multi-document
Output: extract vs. abstract.

Single document vs. multi-document
Single document summarisation takes one text and turns it into a single, shorter text.
Examples: news articles, scientific articles, minutes.
Multi-document summarisation takes multiple texts and turns them into a single, shorter text containing information on all the documents.
Examples: output of a search engine, multiple news articles on a single topic, email messages in a single thread.

Extract vs. abstract
An extract is a summary composed completely of material from the source, while an abstract is a summary that contains material not originally in the source, for example paraphrases.

Summarization systems

How do summarisation systems work?
Extractive summarization
Select the most important sentences. This is a classification/ranking task.
- Classification: for each sentence, select it (yes/no)
- Ranking: assign a score to each sentence, then select the top *k* sentences.

This is an easy, reliable method, but limited in terms of fluency and it requires fixes after sentence selection.

Abstractive summarization
Learns a text-to-text transformation model. Uses pairs of longer and shorter texts as training data. Sequence-to-sequence models learn a mapping between an input sequence and an output sequence. This gives a more natural and fluent result, but requires a lot of training data and risk untrue content.

Baseline summarisation system
Lead-3 baseline simply takes the first three sentences from the document, often only slightly outperformed by state-of-the-art models.

Sentence selection methods
Unsupervised:
- Centrality based
- Graph-based

Supervised:
- Feature-based
- Neural-network based

Centrality-based sentence selection
Measure the cosine similarity between each sentence and the document, either using the sparse vector space with words as dimensions or the dense vector space using embeddings. Then, select the sentences with the highest similarity (these are the most representative).

Graph-based methods
Sees each sentence as a node in the graph, with an edge between sentences. The weigh of this edge is the similarity between the two sentences. This results in a weighted graph, on which you can apply a ranking function to rank the nodes in the graph based on their weighted connected edges.

Sentence selection as a supervised learning task
Feature engineering + classifier (ex. SVM)
Features can be the position in the document, the word count, word lengths, frequencies, punctuation, etc.
Neural sentence selection (CNN, RNN)
Use low-level representations as input features. Models are trained as word-or-sentence classifiers that predict whether a fragment should be included in the summary. This approach requires a lot of training data.

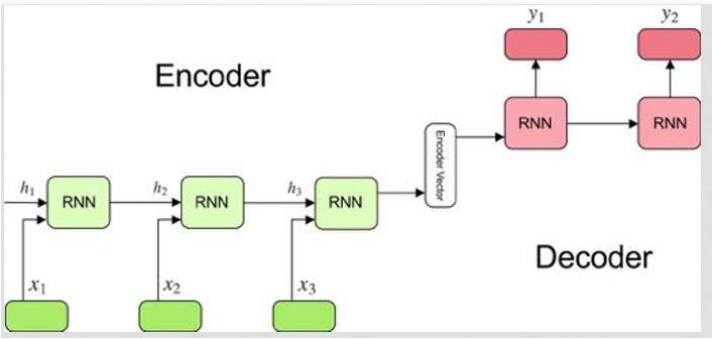
Problems with sentence selection
Selecting sentences that contain unresolved references to sentences that are not in the summary or not explicitly included in the document is a problem.
You might need some fixes after sentence selection, such as sentence ordering, sentence revision, sentence fusion or sentence compression.

Abstractive summarisation
Is summarisation as a translation task, translating a long text into a shorter text. Uses recurrent neural networks (LSTMs or GRU-RNNs).

Recurrent Neural Networks for summarisation
Need a large amount of training data. They train a sequence-to-sequence model to learn the translation from a longer to a shorter sentence with the same meaning.

Encoder – decoder model
Used to map sequences of different lengths to each other.

The encoder vector is the final hidden state from the encoder part of the model. It contains informative representations for all input elements to help the decoder make accurate predictions.



Challenges of summarisation
- Task subjectivity/ambiguity
- Training data bias
- Evaluation

Task subjectivity/ambiguity
We can use human reference summaries, and use Cohen's Kappa to measure inter-rater agreement. Given all the sentences in the original text, we look if they were selected or not (1 or 0) and compute Cohen's Kappa. However, it is possible that raters might select different sentences with the same meaning, which is not considered by Cohen's Kappa.

Training data bias
Most used benchmark sets for training and evaluation of summarisation models are based on news data. In newspaper articles, the most important information is in the first paragraph, which explains why Lead-3 is so strong. In other domains than newspapers, this characteristic does not apply.

Evaluation

Two approaches:
- Intrinsic: test the summariser on its own task, by comparing to reference summaries or asking judges.
- Extrinsic: test the summariser within another task. Use it as speed gain in a human task and ask experts about its utility.

Intrinsic evaluation
*Compare to reference summaries (*Extractive summarisation)
Calculate precision and recall against human reference data, using a set of selected sentences by the system and a set of true sentences in the reference summary.

$$\text{precision} = \frac{|\text{selected} \cap \text{true}|}{|\text{selected}|} \qquad \text{recall} = \frac{|\text{selected} \cap \text{true}|}{|\text{true}|}$$

*Compare to reference summaries (*Abstractive summarisation)
Compute the overlap with a human reference summary using the ROUGE (Recall Oriented Understudy for Gisting Evaluation) package. It measures the quality of a summary by comparing it to reference summaries. Its metrics count the number of overlapping units between system summary and reference summary.
ROUGE-N is the proportion of n-grams from the reference summary that occur in the automatically created summary.

$$\text{ROUGE-N} = \frac{\sum_{S \in \{\text{Refs}\}} \sum_{\text{gram}_n \in S} \text{count}_{match}(\text{gram}_n)}{\sum_{S \in \{\text{Refs}\}} \sum_{\text{gram}_n \in S} \text{count}(\text{gram}_n)}$$

n is the length of the n-gram, $\text{count}_{\text{match}}(\text{gram}_n)$ is the number of n-grams co-occuring in the tested summary and set of reference summaries. The number of n-grams in the denominator increases as we add more reference summaries. The numerator sums over all reference summaries, effectively giving more weight to matching n-grams occurring in multiple references.

Example
Reference summary: "police killed the gunman"
Summary A: "police kill the gunman"
Summary B: "the gunman kill police"
Using ROUGE-2:
A: police, police kill, kill the, the gunman 3/5 matching
B: the, the gunman, gunman kill, kill police 1/5 matching
So, summary A is better than summary B.

Asking human judges
Rate a summary on:
- Relevance: selection of important content from the source
- Consistency: factual alignment between summary and source
- Fluency: quality of individual sentences
- Coherence: collective quality of all sentences

Multiple judges, can also use blind side-by-side comparison between machine and reference summary.

Challenges in abstractive evaluation
ROUGE often has weak correlation with human judges. However, human judges have strong correlation between individual relevance and fluency scores.

Extrinsic evaluation
Give users a task to complete where they need to use a text or its summary, for example judging if a particular document is relevant to a topic of interest.
Concluded that automatic text summarisation is very effective in relevance assessment tasks on news articles.

Lecture 9 – Sentiment Analysis

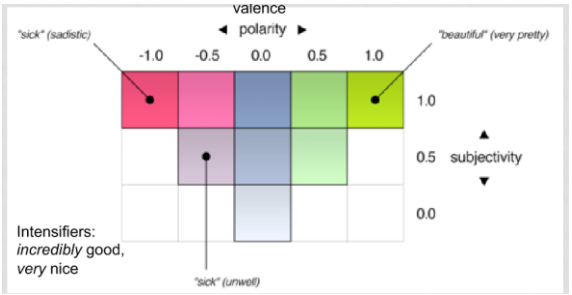
Objective sensors capture the world in an objective manner, for example on video. Subjective sensors, like text, first perceive and then express (by a person) the real world. The resulting text is therefore subjective and loaded with opinions.
Sentiments expressions are a subset of subjective language use. Not every subjective sentence ("I think he went home.") necessarily contains a sentiment.
Typical sentiment questions might be:
- What is the opinion of a person or organisation on a given topic.
- Aggregating positive and negative opinions on a particular entity.
- Detecting how opinions change over time
- Comparing entities.

Levels of sentiment analysis
- Document level: give a sentiment score for a complete product or tweet.
- Sentence level: subjectivity classification, distinguishing between objective and subjective sentences.
- Entity and Aspect level: relate the sentiment to features of product, event or entity.

Annotation on the document level

The most basic common labelling is negative, positive, neutral. However, you could also do objective or subjective, emotions, or ordinal scales. Using ordinal scales, mislabelling a -2 as a -1 is a smaller mistake than labelling a -2 as a +2.

Polarity/subjectivity/intensity scale



Annotation on the entity/aspect level

Simple Document analysis focuses on finding an opinion pair (D, S), where D is document and S is sentiment score.
When doing aspect-based sentiment analysis, you attempt to find a quintuple (E, A, S, H, C)

- E: entity
- A: aspect or feature of E
- S: sentiment
- H: holder of opinion
- C: context

Example: John Doe, 8 November 2018: "I bought an iPhone a few days ago. It was such a nice phone. The touch screen was really cool. The voice quality was clear too. Although the battery life was not long, that is ok for me."
(iphone, touch screen, really cool, John Doe, 8 November 2019)
Extracting E, H and C is a matter of NER or Event Detection, or Time Recognition.
Extracting A is Information Extraction / aspect categorisation.
Both E and A are domain dependent and can therefore be challenging to extract.

Challenges of sentiment analysis

1. Sentiment words do not always express a sentiment.
2. Sentiment words are ambiguous, context- and domain dependent
3. Sarcasm is difficult.
4. Objective sentences can also express sentiments. (As just factual observations.)

Evaluation

		Actual		
		Pos	Neu	Neg
Predicted	Pos	PP	PU	PN
	Neu	UP	UU	UN
	Neg	NP	NU	NN

For three labels
Given three labels: positive, negative, neutral

- Precision_{pos} = PP / PP+PU+PN
- Recall_{pos} PP / PP+UP+NP
- $F_1^{pos} = 2 \times (\text{Precision}_{pos} \times \text{Recall}_{pos})/(P_{pos} + R_{pos})$

Average F1 is only calculated on the positive and negative labels.

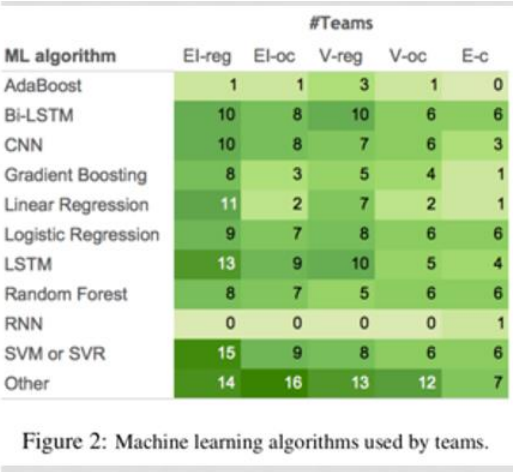
In case of regression
Root Mean Squared Error (RMSE)
And use Pearson/Spearman correlation coefficient *r*

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

State of the art – tweets
SemEval is a sentiment analysis benchmark competition on Twitter data. They run several tasks, emotion from tweets is one of them. For each emotion, they selected 50-100 terms that were associated with that emotion at different intensity levels. Then, they retrieve tweets from Twitter that contained these terms. Below, the different tasks and sizes of datasets are shown. On the next page, a table showing the most used features and methods is displayed.

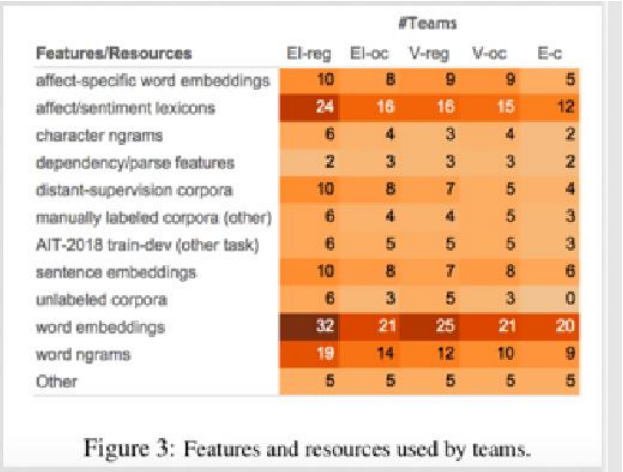
- 1. emotion intensity regression
- 2. emotion intensity ordinal classification
- 3. valence (sentiment) regression
- 4. valence ordinal classification
- 5. emotion classification

Dataset	Train	Dev	Test	Total
English				
E-c	6,838	886	3,259	10,983
EI-reg, EI-oc				
anger	1,701	388	1,002	3,091
fear	2,252	389	986	3,627
joy	1,616	290	1,105	3,011
sadness	1,533	397	975	2,905
V-reg, V-oc	1,181	449	937	2,567



ML algorithm	#Teams				
	EI-reg	EI-oc	V-reg	V-oc	E-c
AdaBoost	1	1	3	1	0
Bi-LSTM	10	8	10	6	6
CNN	10	8	7	6	3
Gradient Boosting	8	3	5	4	1
Linear Regression	11	2	7	2	1
Logistic Regression	9	7	8	6	6
LSTM	13	9	10	5	4
Random Forest	8	7	5	6	6
RNN	0	0	0	0	1
SVM or SVR	15	9	8	6	6
Other	14	16	13	12	7

Figure 2: Machine learning algorithms used by teams.



Features/Resources	#Teams				
	EI-reg	EI-oc	V-reg	V-oc	E-c
affect-specific word embeddings	10	8	9	9	5
affect/sentiment lexicons	24	16	16	15	12
character ngrams	6	4	3	4	2
dependency/parse features	2	3	3	3	2
distant-supervision corpora	10	8	7	5	4
manually labeled corpora (other)	6	4	4	5	3
AIT-2018 train-dev (other task)	6	5	5	5	3
sentence embeddings	10	8	7	8	6
unlabeled corpora	6	3	5	3	0
word embeddings	32	21	25	21	20
word ngrams	19	14	12	10	9
Other	5	5	5	5	5

Figure 3: Features and resources used by teams.

Lecture 10 – Biomedical Text Mining

There is a large amount of scientific literature in the biomedical domain. It is needed for systematic reviews, building knowledge bases of relations between biomedical entities, building research hypotheses and extracting metadata. There are a lot of articles being added to literature databases.

Scientific literature is growing exponentially, but also a lot of experimental data is growing fast. The goals of biomedical text mining are to assist the expert in finding the needed information, called Interactive Knowledge Discovery. It consists of assisting research in finding, evaluating and interpreting scientific literature.

Biomedical research questions that can be answered using TM tools

- Gene/protein/disease extraction
- Drug-target discovery
- Adverse events
- Knowledge discovery (and predictive models) from electronic health records.
- Extracting metadata for large experimental datasets.

Gene/protein/disease extraction
Interactive TM methods for annotation and interpretation of literature. The goal is to create an overview of the biological processes that are associated with sets of genes and proteins. It links to underlying literature, allowing for interpretation of the results.
Example tool: GNormPlus, a tool for tagging genes, gene families and protein domains.

Drug target discovery
Target discovery is understanding the role of a gene in the development of a disease. Genes that play a pivotal role in these processes are possible drug target candidates.

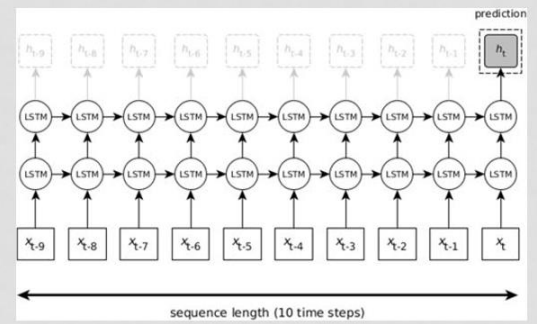
Adverse events
Mining adverse events from medical case reports. There is an Adverse Drug Effect (ADE) benchmark corpus, consisting of nearly 3000 case reports manually annotated with 5063 drugs and 5776 conditions, plus an ontology of adverse events.

Electronic health records

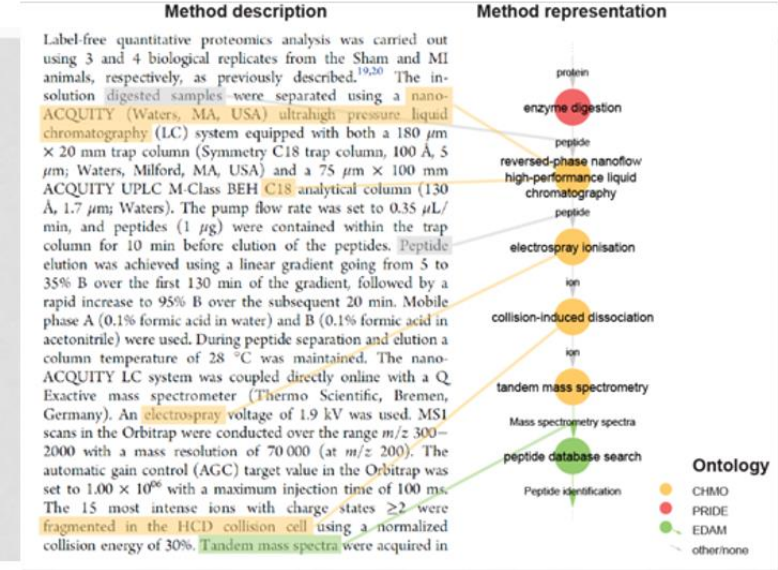
EHRs can be a useful resource for:

- Info on side effects
- Discovering drug interactions
- Discovering adverse effects
- Revealing unknown disease correlations
- Predictive models for diagnoses or events.

Predictive models can be trained on EHRs, for example predicting the chance of end-of-life occurring during a specific month.

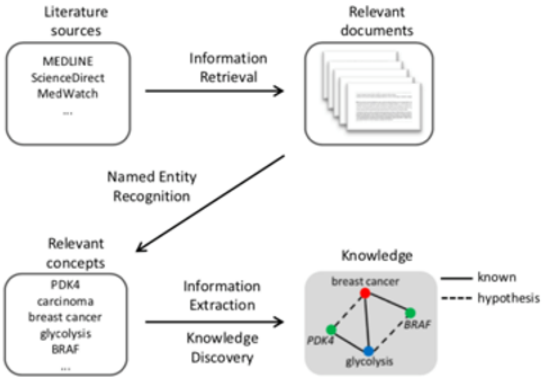


Metadata extraction for large datasets



BIO-TM modules

1. Information Retrieval
2. NER and ontology linking
3. Relation Extraction
4. Knowledge Discovery
5. Visualisation



Information retrieval

Most used IR system is PubMed, with the underlying database MEDLINE. It has full text papers and annotated abstracts with Medical Subject Heading (MeSH) terms.

TM systems sometimes have advanced query options, called ‘concepts’. This is organising similar keywords such as synonyms and alternative names into one concept based on a controlled vocabulary and subsequently incorporating all keywords of the same concept into the query.

Named entity recognition

Identifying biomedical entities in retrieved documents. Entities are highlighted and linked to the specific concept in a controlled vocabulary (thesaurus or ontology). The Unified Medical Language System integrates and distributes key terminology, classification and coding standards and associated

resources to promote creation of more effective and interoperable biomedical information systems and services, including EHRs.

One of the biggest challenges in bio-NER is the recognition of genes and protein names in texts. They are often described using different names and symbols, with symbols and names shared between multiple genes.

Unsupervised NER is linking terms to concepts in an ontology, with some open to the public. Many TM applications have implemented ontologies into their workflows to structure their search strategy, visualise and categorise the search results.

Supervised NER is machine learning, based on *hidden markov models*, *maximum entropy markov models* and *conditional random fields*. They of course need training data. They can be used to identify chemical entities in text, or be combined with rule-based and lexical methods to identify organism names in a text. Also aid in improving clinical decision making by extracting cancer staging information from EHRs.

Relation extraction – Co-occurrence based methods
Assumes that two concepts that often occur together in the same text are related. Can use several statistics for co-occurrence frequencies:

- Actual number of co-occurrences.
- Expected number of co-occurrences based on the frequencies of both entities.
- Statistical test to decide if co-occurrence is statistically significant (ex. Chi Square)

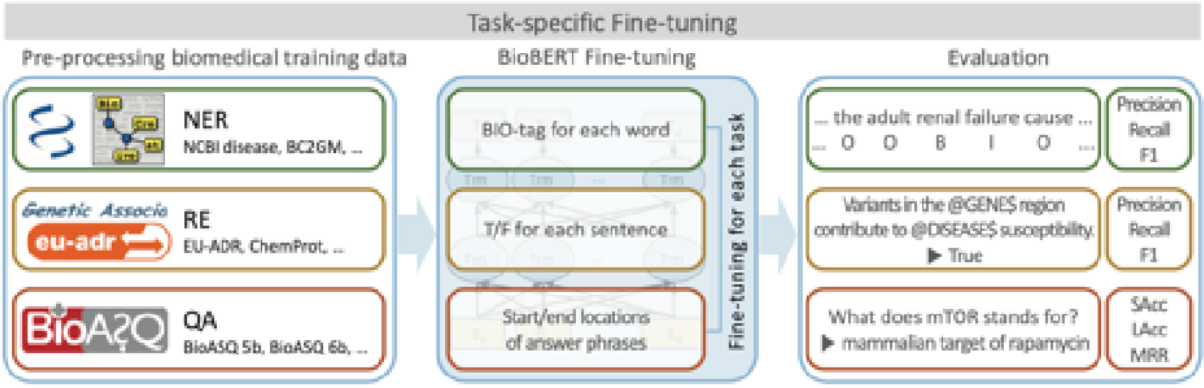
Relation extraction – NLP based methods
Phrase based and able to detect triples in text. Triples provide information about the type of relationship between two concepts. These methods often have a higher precision than co-occurrence based methods but a lower recall. MEDIE is a tool that uses NLP to detect relationships between two given concepts. (ex. “What causes colon cancer?”)
In practice, hybrid methods are used, with co-occurrence methods used to detect relations between concepts followed by NLP methods to establish the nature of the relation.

Knowledge discovery
Literature-based knowledge following the ABC principle. “A and C are never mentioned together, but both are mentioned with B, so A and C are indirectly related.”
The ABC principle is conceptually simple, but not trivial to implement. New predictions made must always be statistically validated.

Visualisation

- Using networks/graphs
- Using word clouds

State of the art: bioBERT
BERT trained with PubMed and PMC, biomedical domain corpora instead of the normal Wikipedia and BooksCorpus in normal BERT.



Lecture 11 – Authorship attribution

Authorship attribution involves identifying the original author of a text. In the typical problem, a text of unknown authorship is assigned to one candidate author, given a set of candidate authors for whom text samples of undisputed authorship are available. From a machine learning point of view, it can be viewed as a multiclass, single-label text categorisation task using textual features.

Definitions
- Verification: is this text written by author A?
- Identification: by which author is this text written, A, B or C?
- Author profiling: what age/gender is the author? Can we detect personal traits?

Authors as generators of a text
To identify authors by their texts, you want to find constant elements in their texts. You need a stylistic fingerprint of the author, as the content, topic, goal, and genre of available texts may all differ.
Human stylome hypothesis: Authors can be distinguished by measuring specific properties of their writings, their stylome as it were.
For this, you often use clues in function words, such as pronouns, determiners, prepositions and conjunctives.

Application areas
- Literary studies: helps resolve authorship disputes, able to link pseudonyms to writers and allows you to analyse collaborative writing.
- Plagiarism detection
- Forensic linguistics: to verify authorship (of text messages etc.)

History
Attempting to define features for quantifying a writing style is part of a line of research called stylometry. Started in the 1990s on small numbers of entire books with only 2 or 3 possible authors. From the 2000s, NLP, IR and ML use rose:
- NLP: access to linguistic abstraction levels, such as Part of Speech
- IR: to handle large amounts of texts
- ML: to train classifiers and develop experimental methodology

Text classification vs. authorship attribution

Differences in style-based and topic-based text classification:

- Topic based: content words are the most important, they carry semantic information
- Style based: function words are most important (articles, prepositions, pronouns etc.). These are common words, found to be among the best features to discriminate between authors.

Features

1. Lexical features: consider a text as a mere sequence of word-tokens.
2. Character features: consider a text as a mere sequence of characters.
3. Syntactic features: requires linguistic analysis
4. Application-specific features: can be defined only in certain text domains or languages.

1. Lexical features

Split the text in tokens (words, numbers, punctuation). This can of course be applied to any language and any corpus with no additional requirements except the availability of a tokenizer. There is two types of lexical features:

- Bag of words features (top-k): use the most frequent words themselves as (sparse) features.
- Aggregated features: use counts over lexical characters. The first authorship attribution attempts were based on simple measures such as sentence length counts and word length counts. The **vocabulary richness** functions are attempts to quantify the diversity of a vocabulary of a text. Typical example: **type-token ratio** V/N, where V is the size of the vocabulary and N is the total number of tokens of the text.

2. Character features

Split the text in characters, and use character n-grams or counts as features. This approach is language- and domain independent, without any NLP necessary. N-grams offer a solution when tokenisation is difficult, and character features have been proven quite useful to quantify the writing style.

Aggregated character-level features:

- Alphabetic characters count
- Digit characters count
- Upper and lowercase characters count
- Punctuation marks count
- Whitespace count
- Etc.

3. Syntactic features

Idea is that authors tend to unconsciously use similar syntactic patterns. Therefore, syntactic information is considered a more reliable authorial fingerprint as opposed to lexical information. Disadvantages are that it requires robust and accurate NLP, being a language dependent procedure. Also, parser errors may lead to noisy data.

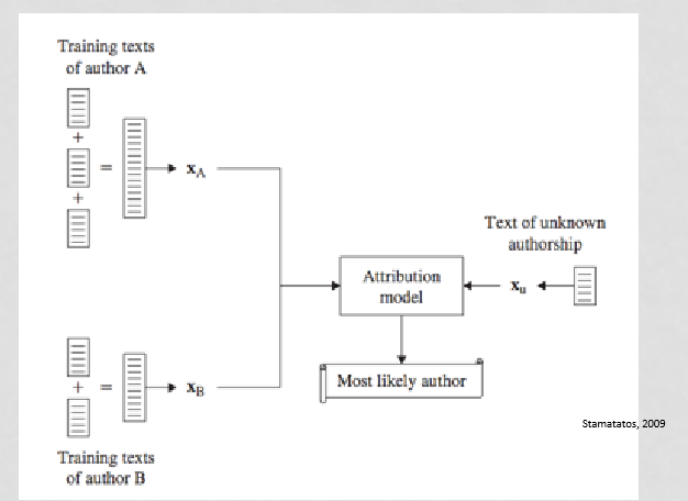
4. Application specific features

Application specific measures allow you to better represent the nuances of style in a given text domain, for example e-mail, text or forum messages. Structural features:

- Use of greetings and farewells
- Emoji use
- Indentation use
- Paragraph length
- Etc.

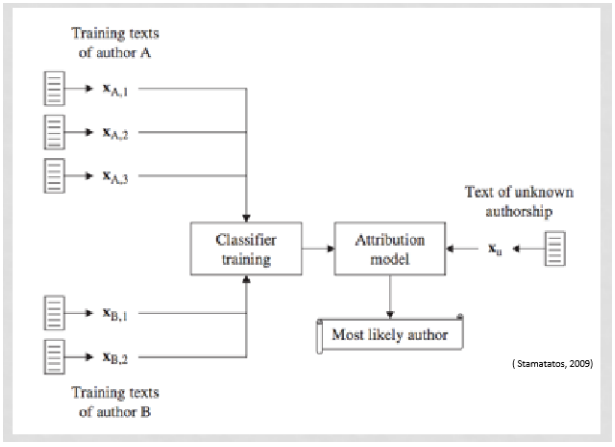
Learning methods

Authorship identification methods:
- Profile-based approaches
- Instance-based approaches
- Hybrid approaches



Profile-based methods
-The ZIP method concatenates text by author a, and turns this into a compressed file. Unseen text y is added to each profile a, and the compression algorithm compresses a + y. The difference in bit-wise size between a and a + y then indicates the similarity between the unseen text y for each author. The author with the lowest difference is probably the author of y.

-Common n-grams (CNG) extracts all character n-grams from all texts of each author, and creates an author profile of the most frequent n-grams. It has two important parameters:
- Profile size L: how many strings constitute the profile (best between 1000 and 5000)
- Character n-gram length n, best between 3 and 5.



Instance-based methods
-Vector space methods extract features and represent texts as feature vectors. Then, compute cosine distance between vectors or use any ML algorithm suitable for text categorisation.

-Unmasking method builds a classifier for each unseen text in order to discriminate it from the training texts of each candidate author. Then, iteratively remove a predefined amount of the most important features for each classifier and measure the drop in accuracy. In the beginning, all classifiers have more or less the same, very high accuracy. After a few iterations, the accuracy of the classifier that discriminates between the unseen text and the true author would be too low while the accuracy of the other classifiers would remain relatively high. This is because the differences between the unseen text and the other authors are evident, so by removing a few features, the accuracy is not affected dramatically.

Hybrid approaches
Methods that use both profile-based and instance-based approaches. For example, all training text samples are represented separately in a vector space model, and the representation vectors for the texts of each author are feature-wisely averaged and used to produce a single profile vector for each author (called the centroid). Then, the distance of the profile of an unseen text from the profile of each author is calculated.

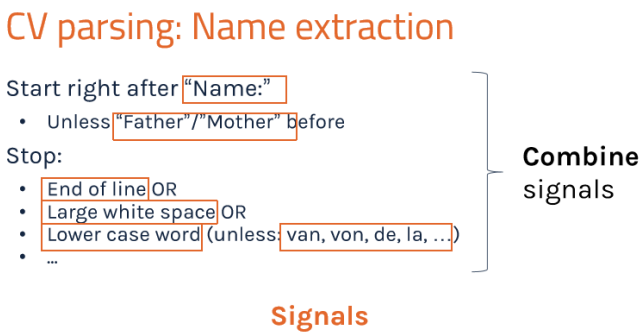
Lecture 12 – Guest lecture TextKernel

Company using AI to match people and jobs. The problem they experience and solve is that of a semantic gap between CVs and job ads. This gap is normally solved by recruiters or job boards. The semantic gap includes multilinguality (Java developer == java ontwikkelaar) and semantic understanding of documents (a job ad for someone with a PhD shouldn't confuse a reference to someone with a PhD in your CV.)

Semantic recruitment
Consists of several parts:

- CV/vacancy/query understanding. and mapping to the ontology. — This is understanding documents, so parsing them
- Search engine. and jobs. — Developing a flexible search engine to match CVs
- Domain knowledge. — Building and maintaining the ontology means understanding concepts (job titles, degrees, etc.) and relationships, and dealing with multilinguality.

CV parsing
Cannot be done rule based, as rules get complicated quickly because they have to accommodate exceptions. Also, their coverage is limited.
The signals used in rule parsing are not 100% sure. So, using machine learning, we can estimate the quality of signals and combine multiple signals. Sequence labelling (PoS, NER) also helps extracting names.

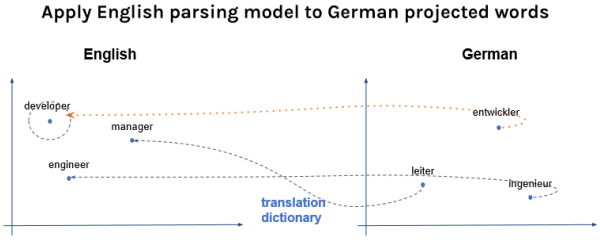


Extraction
The typical ML pipeline looks like this:

- Detection of CV pages (in case of scans)
- Section segmentation (education, work experience, personal info etc.)
- Item segmentation
- Phrase extraction

Parsing CVs and jobs
TextKernel uses ML since 2001, to combine signals (email, typical experience words) using Hidden Markov Models, CRF to parse CVs and jobs. Since 2014, they also apply deep learning, which means people are no longer needed to detect signals. Deep learning uses word2vec to map words similar to CEO, for example. It uses RNNs for parsing CVs.
They saw very large increases when moving from CRF and HMM to the deep learning approach.

Universal CV parsing
Best is to create one model to parse multiple languages, as CVs are written similarly across languages and countries. The approach is using a translation dictionary to be able to apply the English parsing model to German projected words.

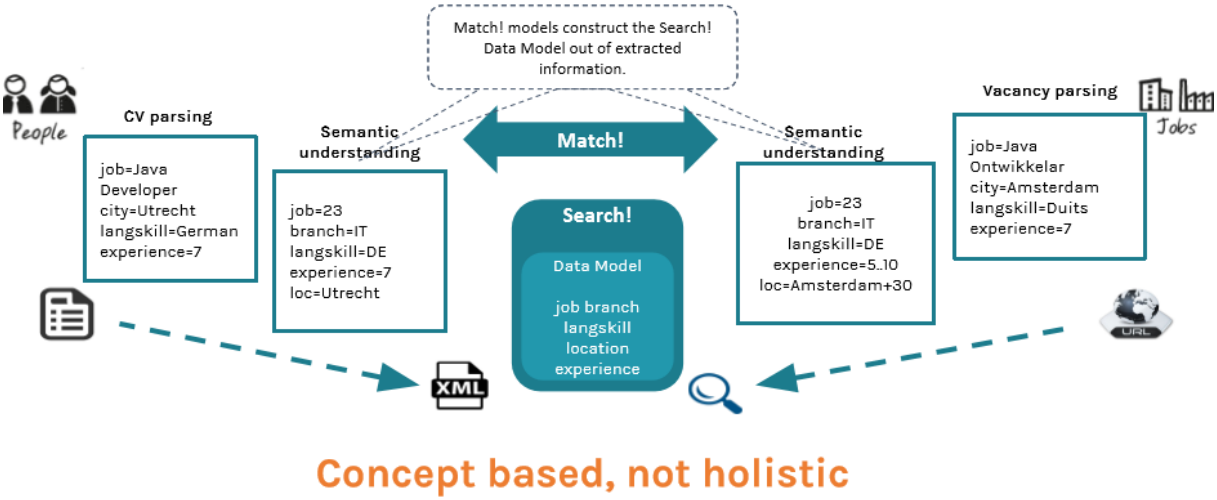


For this, they would apply transfer learning. They found that transfer learning works (even zero-shot parsing, using only English words). Introducing more German data led to a better performance.

The translation dictionary is very important in case you don't have a lot of training data. Pay attention to domain words, frequent words and a size from 5k to 10k words.

Deep Learning Match

Their system matching CVs to vacancies.



Semantic fingerprints

Works by creating vectors for CVs and jobs and projecting them in a Vector Space Model. Relevant CVs will be close to a job.

These semantic fingerprints can also be applied to find similar candidates, merge or deduplicate similar job postings, and allow for better profession clustering as it is based on more than just the job title.

It can also be used to find alternative job recommendations that match a CV, and offer a 'next job' advice to employees.

Lecture 13 – Course Summary

Models	Resources
Tasks	Evaluation
Applications	Domains

Tasks

1. Document retrieval
2. Pre-processing
3. Exploration (topic modelling)
4. Classification
5. Information extraction (summarisation)
   - NER
   - Relationship extraction

Pre-processing
Cleaning files (encoding, regex, spelling correction), followed by the linguistic pipeline containing tokenization, stop word removal, lemmatization/stemming, POS tagging.

Classification
- Multiclass vs multilabel
- Feature selection
- Term weighting: tf-idf

Information extraction
-Named Entity Recognition
- Segmentation and classification
- Sequence labelling task with IOB labels
- Rule-based or feature-based or neural-network based
- Uses dictionaries for help

-**Relation extraction**
- Co-occurrences or patterns or neural classification
- Bootstrapping & distant supervision

Summarisation
Extractive summarisation: sentence classification
Abstractive summarisation: sequence-to-sequence
Challenges: needed training data and how to evaluate.
For intrinsic summarisation, evaluation is a comparison against a ground truth (human). Metrics used are just accuracy, precision & recall and F1, and ROUGE (for abstractive).
For extrinsic, judge the effectiveness in context with other tasks.

Models

Classification
- Vector Space Model and Bag of words
- Dimensionality reduction
- Machine learning methods:
    - NB
    - SVM
    - Feedforward neural networks

Sequence labelling
- Maximum Entropy Markov models
- CRF
- Bi-LSTMs

Sequence-to-sequence
Encoder-decoder models.

Embeddings
Neural language models: traditional word2vec and others, or transformer based (BERT and others)
They attempt to create low dimensional (200), dense vectors from high dimensional (10000) sparse vectors. Are pre-trained as word/sentence prediction task. The hidden layer used in the neural network has the dimensionality of the embeddings used.

Transfer learning
Pre-trained neural language models allow transfer learning. Inductive transfer learning is transferring the knowledge from pretrained language models to any text mining task. This is the current state-of-the-art for almost all text mining tasks. It helps when you don't have a lot of labelled examples.

Resources
-**Labelled data**
Necessary for training and evaluating task-specific models, typically small (100s or 1000s of examples). Use supervised learning. Obtain labelled data from benchmark data, experts or crowdsourcing.
-**Unlabelled data**
For pre-training language models. Can be general or domain-specific. Often very large, with millions of words.

Applications

Sentiment analysis
- Classification or ordinal regression or linear regression.
- Extraction: aspect based sentiment analysis (E, A, S, H, C)

Authorship attribution
Classification/clustering

Domains
- Social media analytics can use: classification, extraction, sentiment
- E-commerce can use: sentiment classification/extraction, ontologies
- Bio medical and clinical applications can use: classification/retrieval, extraction
- Humanities, history can use: preprocessing, classification/retrieval, extraction.

Text mining in practice

Criteria:
- Understand the user: what is the purpose?
- Adaptability and understandability
- Fairness
- Accuracy
- Implementation in commercial software

Understand the user
User requirements:
- Recall or precision?
- How much time does the user want to spend?
- Does the software make decisions or just give suggestions?

Adaptability and understandability
Rule based approaches are often used in commercial applications, whereas the research community uses dominantly machine learning.
Explainability is very important in recommender systems in order to convince the user. It has become much more challenging with the use of neural models. You have to be able to explain the predictions of your model to the expert user. Current research focuses on attention layers and explaining sentiment analysis.

Fairness
You must avoid prejudice caused by biased algorithms. In the example of job/CV selection, it is important that we do not develop a model biased for gender, religion or race. We can easily just exclude gender feature, but when adding resumes to the model, we still include gender-biased terms (docente, medewerkster) to the model.

Accuracy
Use only accurate measures, or at least have the accuracy transparent to the user. Of course, ML is not perfect and should only be used to make suggestions. We do not want ML to generate false evidence.

Commercial software
When using packages, licensing should allow for commercial re-use given modifications. Otherwise, the company will have to re-implement the whole package. For example, GNU is the general public license, an open source license but can only be re-used in software that is also open source.