NLP

Machine learning

Language Model

Tokenization

Chat-GPT

RegEx

Second Year Project

Lecture 1

Welcome ♥

OSEMN Data Science is Osemn (Awesome) ♥
Obtain, Scrub, Explore, Model & iNterpret :)
A lot of data is unstructured! (free text)

HPC High Performance Computer at ITU is good for intense tasks, which might be beneficial for this course

Project One of:
Hand-in a week early for feedback!
1. Named entity recognition NER the easier
2. Relation extraction

NER Determine entities in text: Person, Organisation &...
RE Relations between 2 entities (f.ex "used for")

Introduction

Semantics The study of meaning in a word/sentence
Can often be very ambiguous

Ambiguity
Lexical a word has several meanings
Syntactic relational; who does the action?
Semantic ambiguity of meaning!
Pragmatics contextual ambiguity

Tokenization We divide a string into its parts!
i.e. words etc.

RegEx in python
```
import re
re.compile('<pattern>').split(<text>)  splits on pattern
                       .findall(<text>)  finds all matching patterns
```

Robs Opinion: find words is risky because input disappears!

Lecture 2

Command Line

Architecture

<Command> -<option> <value> <input>

Useful

cat print contents

head/tail (-n 2) prints first/last 2 lines

wc counts words (Line Words Chars)

less interactive reading (/<search> for search)

| do several actions in a row

> write to file (

Text Processing

(manual ☺)

man <command>

grep <regexp>

-i ignore capitalisation -o match only regexp (not line)

-n return line numbers -c count lines that match

-v invert

sort

-n numeric -r reverse

Uniq remove adjacent duplicates

-c count number of words

cut/paste extract columns / paste columns

-d -f

tr replaces characters

sed "s;<FIND>;<REPLACE>;g"

(<FIND>: ↙ regex/word; <REPLACE>: ↙ word; . ↖ any character)

-i in-place edit

Experimental Setup

Data Splits

Train → Dev (Same as Train → Val)

stratified?

We might want to avoid f.ex. same user in both data sets, since we want to optimize for new users! Time, Document, Domain, writer

Overfit

we overfit model to training data

we overfit design decisions to validation data

Random Splits

NOT k-fold or train-val (These are standard splits)

instead we (1) shuffle (2) split ... This can be beneficial, BUT NOT BEST PRACTICE (stratified is better)

why? due to stratified ensuring small correlation between train & test data ☺

Baselines

(1) Majority classifier (always votes the most common)

(2) Simple classifier (f.ex. logistic regression)

Metrics

Choose a metric that reflects the goal

Common Metrics: Accuracy, Recall, Precision, F1

Accuracy → Balanced Classes

Recall, Precision, F1 → classes → can be macro-averaged

Reporting

Report only answering of research question

- hyperparameters in appendix

Reproducibility

Results should be similar with similar setups

Replicability

Results should be the same with same code/data

Statistical Significance

H_0: M(A)-M(B)=0

H_A: M(A)-M(B)>0

Bootstrap; how surprising is it, that model A outperforms model B?

we want to prove M(A)>M(B) by one-sided hypothesis test for H_A: M(A)-M(B)>0

(1) bootstrap n (10.000) random datasets

(2) empirically create distribution for metric (diff. in metric A-B)

(3) calculate p-value for observed metric for A

ASO; Almost Stochastic Order (see implementation in slides) ← link to

Ground Truth

Collection

Top-down: what data do we want?
Bottom-up: what data can we get?
Sample should be representative!!!
Be aware of bias & limitations
Sampling bias
Non-response bias

Annotation

→ Guideline → annotate → evaluate →

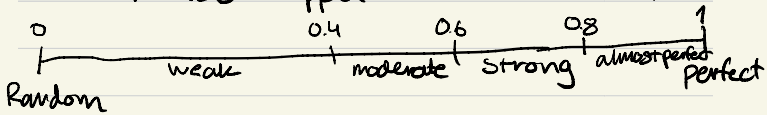
Gold standard → ground truth annotation
how well do humans mimick gold standard?
There is a tradeoff between having guidelines that produce similar results for a variety of annotators and correctness!
Guidelines goal, labels, what is annotated & difficult cases!

P_o: observed aggreement
P_e: expected agreement

$$K = \frac{P_o - P_e}{1 - P_e}$$

Inter-annotator agreement

Cohen's Kappa: chance-corrected accuracy
Fleiss Kappa: more than 2 annotators



Dataset Statement

9 Q's to answer!

Sequence Labelling

POS Tagging — A type of sequence labelling
We use wordnet to find syntactic types for words... we want to disambiguate ambiguos cases.
Penn Treebank 52 tags
Universal POS tagging 17 tags → works for most languages!

Classes
- Closed class: classes that are relatively fixed → no additions
- Open Class: frequently changing / new words are often added!

English Web Treebank — weblogs, newsgroups

- **Noun** — simple test; "the" can be put in front!
- **Verb**
- **Pronouns** — refers to someone / something
- **Adposition** — word used to describe relation

NLP Basic Classifiers

Predictor — the function that maps $X \to Y$
$Y = f(X) + \epsilon \quad \hat{Y} = f(X)$

simple NLP tasks
* Spam or not spam?
* sequence labelling (POS)

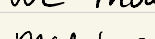
Feature types — "High-level" engineered features
"low-level" directly derived from raw data

Bag-of-Words — We only care what is present, and ignore context!

Reduce Vocabulary — lemmatisation reduce words to base form
stemming approx. lemma by removing endings

N-gram — sequence of n words in a text.

Generative **Bayes Naive** — optimises 0-1 loss (lower bound error)
Independence between features is assumed, thus $p(x, y) \approx p(y) \prod_k p(x_k | y)$

Conditional **hmm... Sigmoid** — we model $p(y|x)$ directly!
model p of $y=1$ 
Sigmoid linear model: $\sigma(wx+b)$ is also logistic regression

Regularisation — l_2 $\lambda \sum_i w_i^2$ (lasso!) add penalty to loss

Language Models

Language Model — We want to predict the next word in a sentence! $P(\text{word} \mid \text{history})$
useful for spelling corrections / suggestions

Markov Assumption — We assume independence, thus being able to do the following simplification;

N-grams — We consider the last $n-1$ words when predicting the next word in a sentence!

$P(w \mid h)$ $= \text{count}(h \circ w) / \text{count}(h)$

Baseline $P(W) = \frac{1}{|\text{vocab}|}$ (all words have the same P)

Intrinsic — We "win" if we predict words in the actual corpus!

Extrinsic

Perplexity $= P(w_1, \ldots, w_N)^{-\frac{1}{N}}$ (inverse P of sequence)
the lower the better! $\approx$ avg. #choices

Zipf's Law 

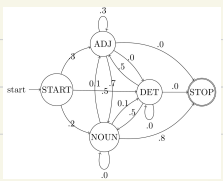
Words not in vocabulary get 0 probability
few words are frequent, most words are rare!
Problem: underestimating the P of unseen words
+ Replace unseen words with <oov> token
feature engineering → + maybe even categories of oov tokens?
+ Smoothing: move probability mass to oov tokens (Out Of Vocabulary)
+ add-1 smoothing $P(w \mid h) = \frac{\text{count}(h,w)+1}{\text{count}(h)+|\text{vocab}|}$

Interpolation

Back-off smoothing: we combine our n-gram probabilities with (n-1)-grams... Linear combinations!

$$P(w|w_1,w_2,w_3) = \beta_1(P(w|w_1,w_2,w_3)) + \beta_2(P(w|w_2,w_3)) + \beta_3(P(w|w_3)) + \beta_4(P(w))$$

Markov Chains



Assumes dependency between labels of concurrent words.

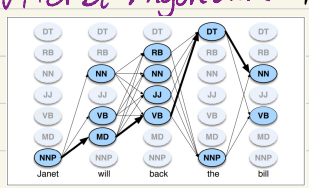
Transition Probability: the P of a label x_2 following the label x_1. It is a graph!

Hidden Markov Model

In real life we do not know the ground truth transition probabilities! We look at $p(w|t)$ (word | tag)

Evolution of Decoding

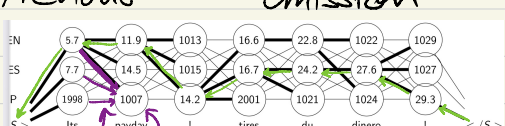
(1) all labels get the same label
(2) all tokens individually get most likely label
(3) Viterbi Algorithm find most likely path!



Follow example from slides :)

Probabilities will be:

$$\underbrace{P(\text{history})}_{\text{Previous}} \cdot \underbrace{p(\text{word}|\text{label})}_{\text{emission}} \cdot \underbrace{P(\text{label}|\text{label}_{n-1})}_{\text{transition}}$$



we backtrack to find most likely sequence

this number is the minimum negative log-probabilities of these edges... the THICK line is the chosen path!

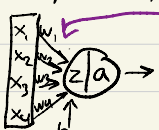
Words in Neural Nets

Deep Learning The future! (Describes NNs that are deep)

Word Representation Instead of Binary (Bag-Of-Words) with CountVectorizer, we can represent words as vectors consisting of d floats

FFNN Layered, Fully Connected

A neuron

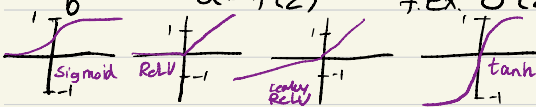


$$z = \vec{w} \cdot \vec{x} + b$$

$$a = f(z)$$

f.ex. $\sigma(z)$

Activations



With words

$$I = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{bmatrix} \quad W = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \quad z = Iw + b = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} \quad a(z) = \begin{bmatrix} a(z_1) \\ a(z_2) \\ a(z_3) \\ a(z_4) \end{bmatrix} \rightarrow$$

$I1$ $I2$

Word Embedding Save vector of floats for each word in matrix $\mathbf{E}$. Dense Continuous features

How to Stardardize NNs need same shape of input, but we might have different lengths... We aggregate! f.ex. average!

How to find w-vectors?

Right now; We initialize $\mathbf{E}$ at random, and update the floats like all other weights in the network; values will shift depending on the task.

next week; new method!

CRF Layer Is like putting Viterbi on top of a neural network, BUT! without emmission... or something"

Emmission probabilities come from neural net!

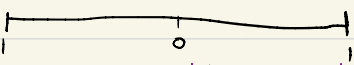
Vector Semantics

Representing Meaning In NLP we use thing like WordNet where we have synsets;
Collections of synonymous in meaning

Hypernums We find the higher-order word describing the meaning of the word.
BUT! Thesaurus is subjective and human maintained

Sparse One-hot-Encoding of features ⇒ Individual word-vectors will be orthogonal even when the meaning is similar!

Cosine Similarity $\cos_{uv} = \frac{\vec{u} \cdot \vec{v}}{|u||v|}$ we map the cosine similarity to the meaning;



But how in practice? opposites unrelated similar

Dense we encode features of a word, instead of the word itself ⇒ d (#features)
The dimensionality d is a hyperparameter

Co-occurence What other words does word X appear with
Matrix: MxM symmetric
hyperparameter: window size around word
Normalize PMI (mutual information) tf-idf to make comparable

Word2vec Framework for learning context-based word probabilities... adjust word vectors to max. P

Methods CBOW Sum context → output word

Skip gram word → project context

We do not care about order, just about words being close to each other! (within window)

RNNs & neural LM

Language is discrete; There is no continuous path between 2 entities

N-gram LM

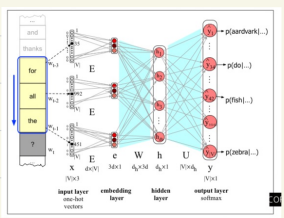
$$p(w_1, w_2, \ldots, w_N) = p(w_1) \cdot p(w_2|w_1) \cdot p(w_3|w_2, w_1) \cdot p(w_4|w_3, w_2, w_1) \cdots p(w_N|w_{N-1}, \ldots, w_2, w_1) = p(w_1) \prod_{i=1}^{N} p(w_i|w_1, \ldots, w_{i-1})$$

probability of word sequence

Markov assumption: we assume independence from everything but last (N-1) words $\approx P$

Smoothing due to sparsity (zipf's law)
Kneser-Ney, Witten-Bell, Laplace

Neural LM



FFNN → continuous-space word embeddings
→ similar idea to N-gram, but NN

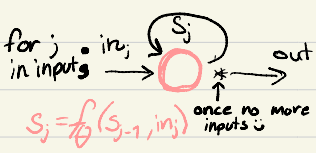
→ Input: $\mathbf{x}_i \in \{0,1\}^{|V|}$ (one-hot encoding)
Output: $\hat{\mathbf{y}} \in (0,1)^{|V|}$ (probability distribution)

$$\mathbf{e} = [\mathbf{E}\mathbf{x}_{t-3}; \mathbf{E}\mathbf{x}_{t-2}; \mathbf{E}\mathbf{x}_{t-1}]$$
$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{e} + \mathbf{b})$$
$$\mathbf{z} = \mathbf{U}\mathbf{h}$$
$$\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{z})$$

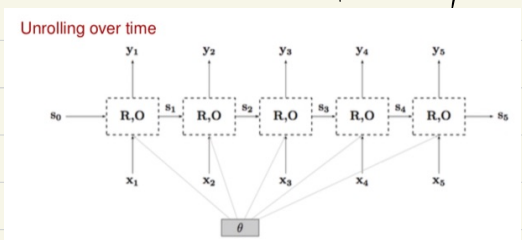
Recurrent Neural Nets

Can proces unbounded input, and retains internal state across steps ü

2 components



memory s Memory of state
function f update function for state
The cell that takes input, and is run as a cycle until input is processed



Advanced RNNs

RNN math

Back to RNNs... gimme dimensions ♥

$$x_t = \mathbf{E}w_t$$
$$h_t = g(\mathbf{V}x_t + \mathbf{U}h_{t-1})$$
$$\hat{y}_t = \mathbf{W}h_t$$

$\mathbf{E} \in \mathbb{R}^{i \times |V|}$
$\mathbf{V} \in \mathbb{R}^{h \times i}$
$\mathbf{W} \in \mathbb{R}^{|V| \times h}$
$\mathbf{U} \in \mathbb{R}^{h \times h}$

$|V|$: vocabulary size
e : embedding size
h : hidden layer size

Observation: Matrix $\in \mathbb{R}^{M \times N}$ can map a vector from dim. $\mathbb{R}^N \rightarrow \mathbb{R}^M$

Use cases

Sequence tagging/transducer, Classification/acceptor
Sequence generator (auto-regressive model / LM)

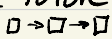
Regularization

How do we avoid over fitting a NN?

Drop-out by some probability p we drop a feature completely (set it to 0), ONLY during training (at test time we rescale by $1-p$)

Deepening

It can be beneficial to know the future!

Bi-directional RNN's we combine → and ← 
Will improve performance :)

Stacked RNN's the hidden states of 1 layers is used in the next; 

Gradients

During backpropagation we might encounter vanishing or exploding gradients... a very real problem for deep models :(

GRU

Gated recurrent unit (keep track of memory and hidden states seperately!)
at every step we consider overwriting memory c

LSTM

Long Short-Term Memory

Contextualized LMs

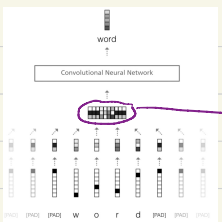
Contextualized Embeddings Attention is a model that uses a weighted sum of recurrent hidden states

But! we are missing context... a word might have several meanings

ELMo

Embeddings from Language Models

each token representation is dependent on the entire input sentence :)



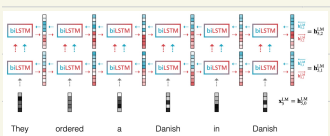
ELMo's Diet: characters !! (in this paper at least)

(1) assign each character an embedding
(2) CNN (filtering (several different size) pooling)
(3) Feed Forward with convolutional output
(4) Output new dense word-representation

How to capture context?

ELMo uses Long Short-Term Memory

We use a bi-LSTM (twice!) and combine each direction's hidden states for a token as the new contextualized embedding



Training ELMo!

Goal: predict word on basis of context ♥

Assumption: tokens always appear in sequence

BERT

4 steps

Bidirectional Encoder Representations from Transformers

Normalization uses NFD to normalize input - irreversible

Tokenization as defined earlier in course

Subword Segmentation look at how likely characters are to appear together, merge most likely characters together into "new" character (f.ex. 'th')

subword examples: {a, b, c, ab, ac, bc, abc}

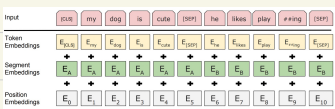
The goal might be to locate morphemes

Special Tokens
[SEP] - end of sentence/segment
[CLS] - beginnig of document

Input

we sum embeddings on 3 levels;

- Subword/Token
- Segment
- Position

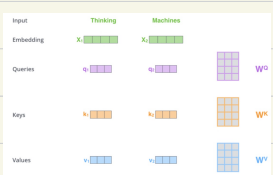


BERT Encoder

12 transformers (heads) each has;

Self-attention

Query: Learn what to focus on from current word
Key: How important as context
Value: Representation of subword (used as output)



$$q_i = W^q x_i$$
$$k_i = W^k x_i$$
$$v_i = W^v x_i$$

Multi-head

Instead of a single self-attention layer, we give the input to n heads, the flatten the outputs → hopefully learning different things

ROBERTA

Larger batches & learning rates + more data

Bias in NLP

Bias is often un-intended but *not* necessarily *un-beneficial*... For some tasks it's good!

Disparity
- **Outcome** outcomes are biased (ex. women in kitchen, men on skis)
- **Error** error occurrence is biased

Sources
- **Data** who/where does data come from?
- **Annotation** who annotates/validates data?
- **Embeddings** are ther embeddings encoding bias?
- **Models** is the model prone to a specific error?
- **Designer** what design desicions where made?

Data Include demographic information in encoding embeddings or resample to be more representative!

Annotation Train and evalcate annotators. Realize and support that some things are ambiguous!

Embeddings We can try to do interventions but there is no magic/consistently good fix :'

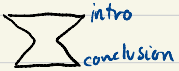
Models Models often *amplify* problematic traing data...

Rob's writing tips!

Anatomy

Abstract, Introduction
Related work, Data, Analysis, Method, Results
Experiments, Results, Discussion, Related work
Conclusion, References

Structure

Funnel-like structure 

Good ideas

Abstract write as the very last thing!
Introduction task description → end on main idea
start with an example!
Related Work describe relation and relevant methodology!
Methodology whole pipeline + motivation for choices
Experiments Data + Setup + Results
Results Compare to 3 baselines!
Analysis what did we improve on? why? how?

Project

27/3 Baseline predictions
29/3 or 3/4 project presentation
12/4 project proposal
19/5 draft hand-in
26/5 final hand-in