**Natural Language Processing (NLP)**

**Session 4**

1. **Sequence Tagging Tasks**:
   1. Part-of-Speech (POS) tagging:
      1. One way of dividing words into different classes is by the part-of-speech (POS) assigned to them.
      2. Most POS tags implicitly encode fine-grained specializations of eight basic parts of a language:
         1. noun, verb, pronoun, preposition, adjective, adverb, conjunction, article
      3. These categories are based on morphological/syntactic similarities rather than semantic similarities.
      4. POS tags used downstream in other tasks like parsing and named entity recognition.
   2. Nouns:
      1. NN = singular noun e.g., man, dog, park
      2. NNS = plural noun e.g., telescopes, houses, buildings
      3. NNP = proper noun e.g., Smith, Gates, IBM
   3. Verbs
      1. VB = verb base form e.g. eat
      2. VBZ = 3rd person singular present form e.g. eats
   4. Determiners
      1. DT = determiner e.g., the, a, some, every
   5. Adjectives
      1. JJ = adjective e.g., red, green, large, idealistic
   6. Connectives
      1. CC = coordinating conjunction e.g. and, or

1. **Sequence Labelling as Classification**:
   1. Classify each token independently but use as input features, information about the surrounding tokens (sliding window).

If we have a sentence such as: “John saw the saw and decided to take it to the table” and pass it through a classifier word-by-word, we would get the output of each individual word as:  
  
John: NNP  
saw: VBD

the: DT

saw: NN

and decided to: VBD   
to take it: PRP

it to the: IN

to the table: DT

table.: NN

1. **Using outputs as inputs**:
   1. Better input features are usually the categories of the surrounding tokens, but these are not available yet as they haven’t been classified.
   2. You can use category of either the preceding or succeeding tokens by going forward or back and using previous output from the classifier at test time.
2. **Forward classification**:
   1. In the above example, keep replacing each word with its corresponding label as the string is iterated through from the start.
3. **Backwards classification**:
   1. In the above example, keep replacing each word with its corresponding label as the string is iterated through from the end.
4. **POS-tagging: Evaluation**:
   1. POS-tagging is a disambiguation task (as there can be more than one possible tag per word)- see ‘back’.
   2. However not many word types have unambiguous tags, and in fact it’s a relatively ‘easy’ task in NLP.
   3. A majority class baseline (per word) is useful to compare a model against:
      1. Given an ambiguous word, assign it the tag that it had most frequently in the ground-truth training data.
5. **Named Entity Recognition (NER)**:
   1. IOB (Inside-outside-beginning) representation:   
        
      Source text: … the caption of Gerolsteiner Davide Rebellin …  
        
      Annotated text (manual): … the caption of <entity type= org Gerolsteiner \entity> <<entity type= org Davide Rebellin \entity> …  
        
      Annotated text (IOB version – without features): Token, IOB tag – I=inside, O=outside, B=beginning:   
        
      the: O  
      caption: O   
      of: O  
      Gerolsteiner: B-ORG   
      Davide: B-PER   
      Rebellin: I-PER
   2. Feature extraction:  
        
      W: a token  
      W-1: the previous token  
      W+1: the next token  
      CAP(W): yes/no  
      POS(W): a pos from a tagset  
      POS(W-1): a pos from a tagset  
      POS(W+1): …..  
        
      Training (Development) set: IOB format with features

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | W | W-1 | CAP(W) | POS(W) | IOB tag |
| 1 | the |  | No | RS | O |
| 2 | captain | The | No | SS | O |
| 3 | of | Captain | No | ES | O |
| 4 | Gerolsteiner | Of | yes | SPN | B-ORG |
| 5 | Davide | Gerolsteiner | Yes | SPN | B-PER |
| 6 | Rebellin | Davide | yes | SPN | I-PER |

1. **Features**:
   1. For each running word:
      1. WORD: the word itself (both unchanged and lower-cased) e.g., Cases, casa
      2. POS: the part of speech of the word (as produced by TagPro) e.g., Oggi, SS (singular noun)
      3. AFFIX: prefixes/suffixes (1, 2, 3 or 4 chars at the start/end of the word) e.g., Oggi { o, og, ogg, oggi, --, I, gi, ggi, oggi }
      4. ORTHOgraphic information (e.g., capitalization, hypenation) e.g., Oggi, C (capitalized). Oggi, L (lowercased)
      5. COLLOCation bigrams:
         1. 36.000, Italian newspapers ranked by ML values
      6. Gazzetters:
         1. PERSONS: Person proper names or titles (154.000, Italian phone-book, Wikipedia)
         2. TOWNS: World (main), Italian (comuni) and Trentino’s (frazioni) towns (12.000, from various internet sites)
         3. STOCK MARKET: Italian and American stock market organisations (5.000, from stock market sites)
         4. WIKI-GEO: Wikipedia geographical locations (3.200)
2. **NER: Evaluation**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Token** | **Expected** | **System** |  |
| Gigi | B-PER | B-PER | Correct |
| Simoni | I-PER | I-PER | Correct |
| Captain | O | B-LOC | Wrong |
| Of | O | O | correct |
| Mercatone | B-ORG | B-ORG | Correct |
| Uno | I-ORG | o | correct |

* There are two expected entities (Gigi Simoni and Mercatone Uno);
  + The system recognized correctly Gigi Simoni (true positive);
  + Did not recognize Mercatone Uno (false negative);
  + Incorrectly recognized caption (false positive).

1. **Generative: Hidden Markov Models (HMMs)**:
   1. *Sequence Labelling*:
      1. Sequence labelling/tagging
         1. A classification problem, but over sequences.
            1. Often from words to a sequence of class labels. e.g.:

POS-tagging

Named Entity Recognition (NER)

* + 1. We could try:
    2. Rule-based classifier:
       1. E.g. transformation-based learning (old school)
    3. Generative sequence model:
       1. (remember Naïve Bayes?) – Hidden Markov Models
    4. Discriminative sequence model:
       1. (remember Logistic Regression?) – Conditional Random Fields
  1. For lots of NLP sequence classification, observations are **words** W and latent variables are **classes** C:  
       
     P(C given W) = (P(W given C) \* P(C)) / P(W)  
       
     P(c1, …, c\_n given w1, …, w\_n) = (P(w1, …, w\_n given c1, …, c\_n) \* P(c1, …, c\_n)) / P(w1, …, w\_n)

Each class observes its corresponding word i.e., c1 maps to w1, but is also connected to c2, which in turn maps to w2 and is connected to c3, etc etc.  
  
Model is like a sequence of Bayesian classifiers.

* 1. HMMs use probability distributions from two models:
     1. A class sequence model p(c\_i given c1…c\_i-1) which is a Markov Model defined by Transition probabilities (like a language model) e.g., in the above example the connection between c1 to c2 to c3, etc etc.
     2. A word/class association model p(w\_i given c\_i ) which are distributions of Emission probabilities e.g., in the above example the connection between c1 map to w1, c2 map to w2, c3 map to w3, etc etc.
  2. **Markov assumption**:
     1. To avoid sparsity (lack of observations), instead of having all previous observations connected to the current observation, we approximate by:
        1. “n-gram model of length k” (where k = n-1), so if k = 2, only the previous two observations will be connected to the current observation.
     2. For the transition probabilities we can define a Markov Model (sequence likelihood model using the Markov assumption) which will give us the probability of a possible hidden sequence C1... C\_n
     3. Remember the probability matrix for bigrams? i.e., Transition matrix for transition probabilities. For 1st order Markov Models, we can do this for class/state sequences too.
     4. Transition matrix constrains possible state paths:  
          
        C\_i (state/class value at position i in sequence) – top row  
        C\_i-1 (state/class value at position i-1 in sequence) – top column

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C1 | C2 | C3 | C4 |
| C1 | filled | filled | filled | filled |
| C2 | filled | filled | filled | filled |
| C3 | filled | filled | filled | filled |
| C4 | filled | filled | filled | filled |

becomes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C1 | C2 | C3 | C4 |
| C1 | filled | filled | filled | filled |
| C2 |  | filled | filled | filled |
| C3 |  |  | filled | filled |
| C4 |  |  |  | filled |

becomes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C1 | C2 | C3 | C4 |
| C1 | filled | filled |  |  |
| C2 |  | filled | filled |  |
| C3 |  |  | filled | filled |
| C4 |  |  |  | filled |

* + 1. Transition probabilities P(c\_i |c\_i-1) define a 1st order Markov model of the current tag given the previous one.
    2. 1st order Markov models (bigram model) can be easily represented in a 2D transition matrix.
    3. The class sequence is not directly observed; hence it is a hidden Markov model.
    4. We can only estimate that a given sequence occurred based on what we observe (observation sequence).
    5. Emission probabilities are needed for us to use Bayesian inference to answer: what is the likelihood that some underlying class c generated word w?
    6. Emission probabilities can be defined in a matrix P(w\_i given c\_i ):  
         
       W\_i (observation/word value at position i in sequence) – top row  
       C\_i (state/class value at position i in sequence) – top column  
       Rows are distributions over the vocab. Probabilities sum to 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | time | fruit | flies | arrow | Like | An |
| NN | 0.3 | 0.3 | 0.0 | 0.4 | 0.0 | 0.0 |
| NNS | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| VBZ | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| VB | 0.2 | 0.0 | 0.0 | 0.0 | 0.8 | 0.0 |
| PRP | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| DT | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |

* + 1. As with Naive Bayes, we ‘flip’ the probability around- ‘time’ was observed, so what’s the likelihood that ‘NN’ generated it, or that ‘NNS’ generated it? etc. i.e. what is the likelihood of different hidden classes.
    2. Generative sequence model:
       1. Assume observations (e.g. words) generated from states
       2. States depend on previous state sequence (Markov assumption: just the most recent one, or fixed number in the past)
    3. Likelihood of observations given hidden class sequence generated by bigram (first order) underlying model:  
         
       P(W) = P(w1, w2, …, w\_n) = product sum of { p(w\_i given c\_1) \* p(c\_i given c\_i-1) }
    4. Bayes’ Rule lets us use it to estimate likelihood of a class sequence given we know the word sequence:  
       P(C given W) = (P(W given C) \* P(C)) / P(W)  
         
       And from this we have a classifier for tagging word sequences:  
         
       c\_map = maximise argument of p(C given W) = maximise argument of { P(W given C) \* p(C) }
    5. Example:  
       Given HMM H, what kind of probabilities are available?  
         
       Word (Class) = time (NN) flies (VBZ) like (PRP) an (DT) arrow (NN)  
         
       Emission probability p(w\_i given c\_i):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | time | fruit | flies | arrow | Like | An |
| NN | 0.3 | 0.3 | 0.0 | 0.4 | 0.0 | 0.0 |
| NNS | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| VBZ | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| VB | 0.2 | 0.0 | 0.0 | 0.0 | 0.8 | 0.0 |
| PRP | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| DT | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |

Transition probability p(c\_i given c\_i-1):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | NN | NNS | VBZ | VB | PRP | DT |
| NN | 0.3 | 0.3 | 0.0 | 0.4 | 0.0 | 0.0 |
| NNS | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| VBZ | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| VB | 0.2 | 0.0 | 0.0 | 0.0 | 0.8 | 0.0 |
| PRP | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| DT | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |

p(c2=VBZ given c1=NN) = NN row and VBZ column in the transition probability table = 0.4  
  
P(c2=NNS given c1=NN) = NN row and NNS column = 0.2  
  
P(w1=fruit given c1=NN) = NN row and fruit column in emission probability table = 0.3

* + 1. Likelihood of Observed Sequence (words):
       1. Likelihood: given observation W and HMM H, what is the likelihood p(W|H)?
          1. If we knew the class sequence, we could use:

P(w1, w2, …, wn) = product sum of P(w\_i given c\_i) \* P(c\_i given c\_i-1)  
  
HMM classes are hidden/unseen: “latent variables”.

Example 1:  
p(w1=fruit) =   
{  
p(w1=fruit|c1=NN) \* p(c1=NN|c0=start) + p(w1=fruit|c1=NNS) \* p(c1=NNS|c0=start) + p(w1=fruit|c1=VBZ) \* p(c1=VBZ|c0=start) + p(w1=fruit|c1=VB) \* p(c1=VB|c0=start) + p(w1=fruit|c1=PRP) \* p(c1=PRP|c0=start) + p(w1=fruit|c1=DT) \* p(c1=DT|c0=start)  
}  
  
Using the above two tables:  
  
= (0.3 \* 0.2) + (0.0 \* 0.2) + (0.0 \* 0.0) + (0.0 \* 0.1) + (0.0 \* 0.0) + (0.0 \* 0.5) = 0.06  
  
Example 2:  
p(w1=time) =   
{  
p(w1=time|c1=NN) \* p(c1=NN|c0=start) + p(w1=time|c1=NNS) \* p(c1=NNS|c0=start) + p(w1=time|c1=VBZ) \* p(c1=VBZ|c0=start) + p(w1=time|c1=VB) \* p(c1=VB|c0=start) + p(w1=time|c1=PRP) \* p(c1=PRP|c0=start) + p(w1=time|c1=DT) \* p(c1=DT|c0=start)  
}  
  
Using the above two tables:  
  
= (0.3 \* 0.2) + (0.0 \* 0.2) + (0.0 \* 0.0) + (0.2 \* 0.1) + (0.0 \* 0.0) + (0.0 \* 0.5) = 0.08

* 1. **Scaling up to Sequences**:
     1. We can do these calculations in this way for short sequences for small numbers of states.
     2. However, summing all possible class sequences is exponential, so use dynamic programming
        1. we use the Forward algorithm
        2. a\_n(j) = probability of getting to word n and being in state j  
             
           a\_n = P(w1w1, …, w\_n c\_j) = P(w\_n given c\_j) \* the product sum of { P(c\_j given c\_i) \* a\_n-1(i) }
  2. **Decoding- getting the most likely sequence**:
     1. Decoding: given observations W, what is the most likely state sequence C?
        1. c\_map = maximise argument of p(C given W) = maximise argument of { P(W given C) \* p(C) }
        2. No need to calculate p(W) for classification.   
             
           Same examples as above.
  3. **Decoding- getting the most likely sequence automatically**:
     1. Searching over all possible tag sequences to get the maximise argument of { P(W given C) \* p(C) } is exponential in the length of the sequence T.
     2. Use the Viterbi algorithm - dynamic programming reduces state sequences to search hugely from exponential to polynomial quadratic
        1. Beam search also possible to reduce this search further- keep only the k most likely sequences after each word (keep these in the beam).
     3. Viterbi is similar to Forward algorithm, but maintain back-pointer from each state to most likely previous state
     4. Then retrace from most likely final state.
     5. The Viterbi algorithm sets up a matrix of size [N, T] where N = number of possible states (tags) and T is the length of the sequence of observations (words).
     6. The idea is to find the state path with the highest likelihood given the words.
  4. **Learning**:
     1. Learning/training: given observation sequence of words W, what is the optimum HMM model H? i.e. what are the optimal emission and transition probability models?
     2. If we have training data with fully labelled sequences, use standard Maximum likelihood estimation (MLE) with counts C from training data to get the conditional probabilities:
        1. Emission probabilities: word at position i given tag at position i:
           1. P(w\_i given t\_i) = C(t\_i intersect w\_i) / P(t\_i)
        2. Transition probabilities: tag at position i given tag at position i-1:
           1. P(t\_i given t\_i-1) = C(t\_i-1 intersect t\_i) / P(t\_i)
     3. Potential for lots of 0s in decoding. We can of course smooth these estimates to avoid 0s and not overfit the data.
     4. What if we don’t have fully labelled data?
     5. We use the Forward-Backward (Baum-Welch) algorithm
        1. Similar to Forward algorithm, but combine:
           1. Forward probability of getting to this state i at time t from start: a\_t(i)
           2. Backward probability of getting from next state j and next time step t+1 to the end: beta\_t+1(j)
        2. Iterate and update these until probability of observations is maximised and cannot improve (convergence).

* 1. **Generalising HMMs:** 
     1. We’ve only looked at 1st order (bigram) Markov models, largely because their transition probabilities are easy to show in a 2D matrix. What if it made sense for the underlying model to use other previous states (not just the last one)?
     2. It is possible to generalise the Hidden Markov Model to an arbitrary order (see n-grams in language modelling lecture), e.g. tri-gram:
     3. We can use back-off and interpolation of lower order models just like we did with n-gram language models.
     4. However, this complicates Viterbi, as it requires going over all possible combinations of the last 3 states (not just 2), making the complexity (S^3 \* T).
  2. **Discriminative Sequence Classification**:
     1. Can we use a discriminative approach instead? Remember alternative text classification methods:
        1. Naïve Bayes: generative – maximise argument of { P(X given C) \* p(C) }
        2. Logistic Regression/SVM: discriminative – maximise argument of { P(C given X) } directly, allows many more features to be used without having to estimate p(X), which isn’t needed for classification anyway.
        3. How do we make this change for a sequence model?
     2. The difficulty in modelling p(X|C) is that it often contains many highly dependent features that are difficult to model:
        1. e.g. in NER, a naive application of an HMM relies on only one feature, the word’s identity, but many words, especially proper names, will not have occurred in the training set, so the word identity feature is uninformative.
        2. The principal advantage of discriminative modelling is that it is better suited to including rich, overlapping features which can given information even if a word is unknown:
           1. e.g. in NER, to label unseen words, we would like to exploit other features such as capitalization, neighbouring words, affixes, membership in predetermined lists of people and locations etc.
  3. **Conditional Random Fields**:
     1. Conditional Random Fields (CRF), discriminative Markov models.
        1. HMM (generative): c\_map = maximise argument of p(C given W) = maximise argument of { P(W given C) \* p(C) }
        2. CRF (discriminative): c\_map = maximise argument of { P(C given W) }  
             
           where,   
             
           p(C given W) = 1/Z \* (product sum of exp(addition sum of lambda\_i \* f\_i (y\_i-1, y\_i, W, i))
        3. Define feature function f which returns a set of features for a sequence position i:
           1. e.g. fi = { “wi-1 = fruit, wi = flies, ci-1 = NN, ci = NNS” }
        4. Learn optimal weights (lambda) which apply to each feature f\_j through a gradient descent method like L-BFGS.
        5. A CRF model consists of
           1. F = <f1, …, f\_k>, a vector of “feature functions”
           2. theta = < theta1, …, theta\_k>, a vector of weights for each feature function.
        6. Let O = < o1, …, o\_T>, be an observed sentence
        7. Let A = < a1, …, a\_T> , be the latent variables.  
             
           P(A=y given O) = (exp(theta dot product with F(y, O))) / addition sum of (exp(theta dot product with F(y’, O)))  
             
           This is the same as the Maximum Entropy equation.
        8. Advantages:
           1. You can define (nearly) arbitrary features
           2. Often outperform HMMs
           3. Available implementations e.g. NLTK CRF tagger
        9. Disadvantages:
           1. Complex inference (dynamic programming again)
           2. Needs manual definition of features
           3. Output is not a sequence probability
           4. It’s the confidence of sequence given the data – (i.e. it’s not really a language model)
        10. In general, this is structured prediction rather than classification – Predicting structured objects not just classes/values
  4. **Finding the Best Sequence:**
     1. Best sequence is:
        1. Maximise argument of P(A=y given O) = maximise argument of [(exp(theta dot product with F(y, O)))]
     2. Recall from HMM discussion, if there are:
        1. K possible states for each y\_i variable,
        2. N total y\_i variables, then there are KN possible settings for y
     3. So brute force can’t find the best sequence.
     4. Instead, we resort to a Viterbi-like dynamic program
  5. **Training/optimizing CRFs**:
     1. In defining a CRF model, you have to consider:
        1. The feature function: what kind of features do you want to extract for each step in the sequence? These can include previous/future words as input into the current time-step, and include features like ‘word-shape’ (e.g. XX-XXX), boolean values for capitalisation etc.
        2. Min. document frequency for features (can be quite high like 5+ as many features can be extracted).
        3. The shape of the Markov model for the labels- most commonly used in NLP is the linear chain CRF- much like a bigram language model/first order HMM, just connecting one state to the next.
        4. Regularisation parameters (L1 and L2), sometimes called ‘C1’ and ‘C2’ in CRF.
        5. Learning algorithm (usually a gradient descent method).

1. **Summary:** 
   1. Hidden Markov Models
      1. Like Language Models, use Markov Models of a given order.
      2. Though the Markov Model not directly observed.
      3. ‘Flip’ the sequence likelihoods round in a Bayesian style.
      4. Robust, good baseline for sequence tagging tasks
      5. Learnable without much labelled data
      6. Be careful with smoothing!
   2. Conditional Random Fields / Recurrent Neural Nets
      1. Discriminative: higher accuracy for many tasks
      2. More complex learning; need more data
      3. Can be more complex feature definition process
      4. Be careful with regularisation, weighting, activation functions, …