**Natural Language Processing (NLP)**

**Session 9**

1. Documents as Vectors:

The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in (column) document:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | As You Like It | Twelfth Night | Julius Caesar | Henry V |
| Battle | 1 | 1 | 8 | 15 |
| Soldier | 2 | 2 | 12 | 36 |
| Fool | 37 | 58 | 1 | 5 |
| Clown | 5 | 117 | 0 | 0 |

* 1. Documents (plays) are column vectors:
     1. As You like It: [1, 2, 37, 5]
     2. Twelfth Night: [1, 2, 58, 117]
     3. Julius Caesar: [8,12, 1, 0]
     4. Henry V: [15,36,5,0]
  2. A [x, y] plot of the above-mentioned vectors with an arrow originating from [0, 0] to the co-ordinate points of the vectors:
     1. Henry [5,15]
     2. Julius Caesar [1,8]
     3. As You Like It [37, 1]
     4. Twelfth Night [56, 1]
  3. Similarity:
     1. Cos(theta) = (A dot B) / (|A| \* |B|) = (sum of (A\_i\*B\_i)) / (sum of(A\_i^2) \* sum of (B\_i^2))
  4. Row vectors in a matrix whose columns are also words – co-occurrences in context e.g. in a k-word window.

1. Weighting terms in Vectors
   1. Raw counts not an effective vector representation for word/term document and word/term-context matrices.
   2. Lots of unimportant words occur frequently (stop words).
   3. Important words occur very infrequently.
   4. We need a method to give importance to words that occur in a few documents, whilst minimising effect of non-important words.
2. For documents:   
     
   w\_{t, d} = tf\_{t, d} \* idf\_t  
   1. TF-IDF TF/IDF: initiated for creating document vectors
   2. TF: frequency of the word in the document (Luhn 1975),
   3. IDF: inverse document frequency (Sparck Jones, 1972).
   4. TF the term frequency – more usually its log, with 1 added to counts (Laplace smoothed) to avoid the log of 0:   
        
      tf\_{t, d} = log\_10(count(t+d)+1)
   5. IDF is defined using the fraction N/df\_t, where N is the total number of documents in the collection, and df\_t is the number of documents in which term t occurs.
   6. Since N is high, idf (and tf) usually adjusted with a log function, e.g. :  
        
      idf\_t = log\_10(N / df\_t)
3. Example of effect:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | As You Like It | Twelfth Night | Julius Caesar | Henry V |
| Battle | 1 | 0 | 7 | 13 |
| Soldier | 114 | 80 | 62 | 89 |
| Fool | 36 | 58 | 1 | 4 |
| Clown | 20 | 15 | 2 | 3 |

|  |  |  |
| --- | --- | --- |
| **Word** | **df** | **Idf** |
| Romeo | 1 | 1.57 |
| Salad | 2 | 1.27 |
| Falstaff | 4 | 0.967 |
| Forest | 12 | 0.489 |
| Battle | 21 | 0.246 |
| Wit | 34 | 0.037 |
| Fool | 36 | 0.012 |
| Good | 37 | 0 |
| sweet | 37 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | As You Like It | Twelfth Night | Julius Caesar | Henry V |
| Battle | 0.074 | 0 | 0.22 | 0.28 |
| Soldier | 0 | 0 | 0 | 0 |
| Fool | 0.019 | 0.021 | 0.0036 | 0.0083 |
| Clown | 0.049 | 0.044 | 0.028 | 0.022 |

A tf-idf weighted matrix for four words in four Shakespeare plays, using the counts in Fig. 6.3. For example, the 0.049 value for wit in As you like it if the product of tf = log\_10(20+1) = 1.322 and idf = 0.037 (N=37). note that the idf weighting has eliminated the importance of the ubiquitous word good and vastly reduced the impact of the almost-ubiquitous word fool.

1. For words: PPMI:
   1. PPMI\_a(w,c) = maximise (log\_2 (P(w,c) /P(w)\*P(c) , 0)
   2. “How much more strongly are w and c associated than we would expect by chance?”
   3. Given a word-context matrix with W rows and C columns:
      1. F\_{ij} is the number of times word w\_i occurred in the context of word c\_j.
         1. P(w\_i, c\_j) = (f\_{ij}) / (double sum of (f\_{ij}))
         2. P(w\_i, c\_j) = ( sum of C (f\_{ij})) / (double sum of (f\_{ij}))
         3. P(w\_i, c\_j) = ( sum of W (f\_{ij})) / (double sum of (f\_{ij}))
2. Example of effect:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | computer | data | result | pie | sugar | Count(w) |
| Cherry | 2 | 8 | 9 | 442 | 25 | 486 |
| Strawberry | 0 | 0 | 1 | 60 | 19 | 80 |
| Digital | 1670 | 1683 | 85 | 5 | 4 | 3447 |
| information | 3325 | 3982 | 378 | 5 | 13 | 7703 |
| Count(context) | 4997 | 5673 | 473 | 512 | 61 |  |

Co-occurrence counts for four words in 5 contexts in the Wikipedia corpus, together with the marginals, pretending for the purpose of this calculation that no other words/contexts matter.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | computer | data | result | pie | sugar |
| Cherry | 0 | 0 | 0 | 4.38 | 3.30 |
| Strawberry | 0 | 0 | 0 | 4.10 | 5.51 |
| Digital | 0.18 | 0.01 | 0 | 0 | 0 |
| information | 0.02 | 0.09 | 0.28 | 0 | 0 |

The PPMI matrix showing the association between words and context words, computed from the counts in the first image. Note that most of the 0 PPMI values are ones that had a negative PMI; for example PMI(cherry, computer) = -6.7, meaning that cherry and computer co-occur on Wikipedia less often than we would expect by chance, and with PPMI we replace negative values by zero.  
  
And we also often reduce dimensionality via e.g. SVD or PCA. 10,000s of dimensions ! 100s of dimensions. Faster computation, less storage. Smoothe

1. Dense vectors: Word Embeddings
   1. Word-context (term-term) and word-document vectors represent a word as a sparse, long vector with dimensions corresponding to words in the vocabulary or documents in a collection.
   2. We can also use a more powerful word representation: embeddings, which are short dense vectors (every value is present, and can be negative).
   3. These are often learned by neural network/deep learning models.
2. Sparse versus dense vectors
   1. Why dense vectors?
      1. Short vectors may be easier to use as features in machine learning (fewer weights to tune)
      2. Dense vectors may generalise better than explicit counts
      3. Dense vectors may do better at capturing synonymy:
         1. car and automobile are synonyms; but are distinct dimensions
         2. a word with car as a neighbour and a word with automobile as a neighbour should be similar, but aren’t
      4. In practice, they work better.
3. Neural methods
   1. So far we are estimating vectors based on co-occurrence counts
   2. An alternative: learn vectors that are good at predicting co-occurrence
      1. e.g. word2Vec
   3. Popular method: Skipgram
      1. train a classifier to predict whether target word t is likely to occur close to context word c or not
      2. Use “negative sampling”: train with real vs randomly sampled pairs
      3. The weights of this classifier are used as the word vectors
4. Skipgram (used in word2vec)
   1. “The man who passes the sentence should swing the sword” - Ned Stark

|  |  |  |
| --- | --- | --- |
| Sliding window (size=5) | Target word | Context |
| [The man who] | The | Man, who |
| [The man who passes] | Man | The, who, passes |
| [man who passes the] | Who | The, man, passes, the |
| [man who passes the sentence] | Passes | Man, who, the, sentence |
| … | … | … |
| [sentence should swing the sword] | Swing | Sentence, should, the sword |
| [should swing the sword] | the | Should, swing, sword |
| [swing the sword] | sword | Swing, the |

1. Aside: training by gradient descent
   1. Initialise weights randomly
   2. Measure the loss (cost)
   3. Determine the gradient of the loss
      1. (partial derivative wrt. each weight)
   4. Move each weight in the direction that reduces loss
      1. gradient descent
   5. Converge to optimal w
2. Semantic regularities:
   1. Graph of Countries (points) and Cities (points also) linked together by arrows.
3. Analogies:
   1. apple – tree = grape – vine
   2. apple – tree + vine = grape
   3. tree – apple + grape = vine
   4. So we can try to solve:
      1. apple – tree + vine = X
      2. “tree is to apple as vine is to what?”
      3. tree – apple + grape = X
      4. “apple is to tree as grape is to what?”
4. Sentence meanings
   1. Sequence: man bites dog dog bites man
   2. Syntax:
      1. (S (NP man) (VP (V bites) (NP dog)))
      2. (S (NP dog) (VP (V bites) (NP man)))
   3. Semantics:
      1. bite(man,dog)
      2. bite(dog,man)
      3. bite(e) & biter(e,man) & bitten(e,dog)
      4. bite(e) & biter(e,dog) & bitten(e,man)
5. Compositional Semantics
   1. We have learnt how to build vectors for words and for documents.
   2. What about for sentences?
      1. Neither of methods used for words or documents can be directly applied to get sentence vectors.
         1. We cannot use the method used for document vectors, since words do repeat in a document but very rarely in a sentence.
         2. We cannot use the method used for word vectors, since one can collect frequency data for words but not for sentences, as they very rarely repeat.
   3. Compositionality
      1. We need to be able to build sentence representations
      2. … e.g. to assign a natural language sentence to its canonical logical form
      3. … out of the representations of its components (words)
      4. This is a new take on the old problem of compositionality of meaning, studied for a long time in traditional semantics (i.e. the meaning of a sentence is determined by those of its parts), where, going away from simple approaches assuming bags of words, we want to maintain the semantic information of interest.
   4. Naïve way 3:
      1. Vampires kill men = men kill vampires
      2. We lose sequence/structure information
6. Compositionality
   1. We need to be able to build sentence representations
   2. … e.g. to assign a natural language sentence to its canonical logical form
   3. We need to know about the components (words) and how they relate to each other
   4. syntax: expression (a) ``Kipling serves vegetarian food”
   5. semantics: expression (b) Serves(Kipling, Vegetarian(food))
   6. How do we get from (a) to (b)?
   7. We need to map syntax to semantics:  
        
      Input 🡪 Parser 🡪 build tree 🡪 Semantic analyzer 🡪 output semantic representations
   8. Classical semantic composition:
      1. Tree:
      2. A tree with left node as NP and right node as VP. NP has one node connected in the next level Proper Noun which is connected to John which is connected to J. VP has Verb and NP as left and right nodes. Verb has one node “eats” which is connected to “lambda y. lambda x. eat (x,y)” and NP has one node “Tofu” which is connected to node labelled “T”.
      3. The Semantic analyzer labels the root node S as “lambda x. eat(x, T)(J) = eat (J, T)“. The right node of the root node “VP” is represented as “lambda y. lambda x. eat(x, y)(J) = lambda x. eat (x, T)”
   9. We need a vector-space equivalent …
   10. For a model of sentence meaning for vector space semantics, we need a procedure which, given vectors for each word in a phrase or sentence, combines the vectors in some way to produce a single vector representing the meaning of the sentence where sentences with similar meanings are close together in the space (Clark 2015).
   11. Of great practical interest for natural language queries: man killed dog should not return similar sentences close to the meaning man killed by dog. Language not just a bag of words.
7. Traditional syntax>semantics mapping:
   1. Syntactic CFG Rule: S 🡪 NP VP
   2. Semantics: {VP.sem(NP.sem)}
   3. This means: to get the semantic LF for S, apply the semantics of VP (VP.sem) to that of NP (NP.sem)
   4. Similarly for the rest of the rules below:

|  |  |
| --- | --- |
| VP 🡪 Verb NP | {Verb.sem(NP.sem)} |
| Verb 🡪 serves | {lambda x in e, y I sa(e, serving) intersect Server(e, y) intersect served (e, x)} |
| NP 🡪 ProperNoun | {ProperNoun.sem} |

Key questions for vector space semantics:   
1) What is the semantic type corresponding to a particular syntactic type?   
2) How can the vector for e.g. transitive verbs be combined with vectors of a subject and object to give a vector in the sentence space?

1. Compositional Distributional Semantics:
   1. Logical types:
      1. A 🡪 curvy(A)
      2. X/Y 🡪 X dot Y
      3. X\Y 🡪 Y dot X
      4. X/Y X 🡺 X 🡪 X dot Y y 🡺 X
      5. Y X\Y 🡺 X 🡪 Y Y dot X 🡺 X
   2. Vectors:
      1. A 🡪 cuvry(A)
         1. Curvy(A) = {e\_i}\_i in T\_i = sum of(C\_i\*e\_i)
      2. A/B 🡪 curvy(A) dot curvy(B):
         1. Curvy(A)={e\_i}\_i, Curvy(B)={e\_j}\_j
         2. Curvy(A) dot Curvy(B) in T\_{ij} = sum of (C\_{ij} \* (e\_i dot e\_j))
   3. Cubes:
      1. A//(B/C) 🡪 curvy(A) dot (curvy(b) dot curvy(C))
         1. Curvy(A)={e\_i}\_i, Curvy(B)={e\_j}\_j, Curvy(C)={e\_k}\_k
         2. Curvy(A) dot Curvy(B) dot Curvy(C) in T\_{ij} = sum of (C\_{ijk} \* (e\_i dot e\_j dot e\_k))
   4. Matrix Multiplication:
      1. A/B B 🡺 A 🡪 Curvy(A) dot Curvy(B), Curvy(B) 🡺 Curvy(A)
   5. Tensor Contraction:
      1. T\_{ij} \* T\_{j} (tensor) 🡺 (contract) T\_i
      2. (sum of (C\_ij \* e\_i dot e\_j)) \* (sum of (C\_j \* e\_j)) = sum of (C\_ij\*C\_j\*e\_i <e\_j given e\_i>)
2. Structural Vector Combination Operations:
   1. Syntax 🡪 Semantics (structure preserving map)
   2. Pregroup 🡪 Vectors Spaces (strongly monoidal functor)
   3. Lambek Calculus 🡪 Vectors Spaces
   4. CCG 🡪 Vectors Spaces (case by case analysis)
   5. Dialogue 🡪 Vectors spaces (Dynamic Syntax)
   6. Alternatives
      1. There’s more than one way to do this!
      2. Map type depends on grammar type
      3. Still an active research area …
      4. You just need to know that it’s possible and remember the basic idea!
      5. More common: vector sum/average
      6. More common: learn the function directly usually with a neural network
3. Application/Evaluation
   1. Intrinsic evaluation (test quality of word meanings against some gold standard marked up by human annotators):
      1. Word-level: Synonym, antonym, hypernym, hyponym detection/ judgement.
      2. Word-level task but in context: Word-sense disambiguation (WSD), the task of determining the correct sense of a word in context. Multiple-choice synonym questions.
      3. Sentence-level tasks: Disambiguation, Similarity judgements, entailment.
   2. Extrinsic evaluation (I.e. how much do these vector representations help to improve performance when used as input to other NLP tasks?)
      1. Text classification, sentiment analysis, sequence tagging, parsing, NL inference etc.
4. Word sense disambiguation:
   1. Say we’ve got an ambiguous verb like ‘draw’.
   2. It has multiple senses/meanings.
   3. One sense it that is is more like ‘pulled’ as in pulled (out) a ceremonial sword.
   4. The other is that it is more like attract as in ‘attract attention’.
   5. WSD as a task is to decide which sense is being employed in a sentence. Concretely, to determine which of the following pairs are closer together in terms of meaning, i.e. are most similar
5. Evaluating Vector Models
   1. WordSim-353 : a set of of ratings for 353 noun pairs.
   2. SimLex-999 : quantifies similarity (cup, mug) rather than relatedness (cup, coffee), and includes adjective, noun and verb pairs.
   3. TOEFL dataset : a set of 80 questions, each consisting of a target word with 4 additional word choices; the task is to
   4. choose which is the correct synonym.
   5. Stanford Contextual Word Similarity (SCWS) : human similarity ratings for 2,003 pairs of words, but in their sentential context.
   6. Word2Vec Dataset: a set of patterns “a is to b as c is to d”. Given a, b, and c, the task is to find d. The words are in certain semantic relationships with each other. For example Athens is to Greece as Oslo is to ? …. Norway.: Mikolov et al. 2013.