CS-626 Course Assignments

Samkit Shah (143050040)
Eeshan Malhotra (14305R001)
Sanjeev Muralikrishnan (14305R002)
Ankith MS (143059007)

Assignments

- POS Tagging Using HMMs
- Verb Group Parsing (CYK)
- Parse Tree Interconversion
  - Dependency to Constituency Tree
  - Constituency to Dependency Tree
- POS Tagging Using Neural Networks

Corpus Statistics

Length of the Corpus: 1,161,192

Total Tag Count(raw): 472

Tag Count (Removing non-POS tags): 190

Total Word Count: 56,057

Pre-processing

- The corpus has some tags which contain information about the contextual usage of words, but does not directly relate to the POS
- -HL, -TL, -NC, FW-
- We have created a set of matrices discarding this information

Unknown word Handling

Method 1 - Global prior

Unknown words are handled by assigning prior emission probability as the cumulative probability of each tag. These will be driven largely by Transition probabilities.

Rule based Augmentation: Bias capitalised words towards proper nouns

Final Method (Suffix)

Varying-length suffixes are considered, and P(tag|suffix) is considered as proxy to P(tag|word).Took suffixes of lengths 4,3,2,1 with backoff strategy applied if higher-length suffix is not found. If no suffix is known, fall-back to Method1

Lemmatizing and Stemming

- Lemmatizing
  - Words are split in to Lemma+Suffix in training data
  - For input test sentence, the same procedure is applied
  - Accuracy of model does not improve

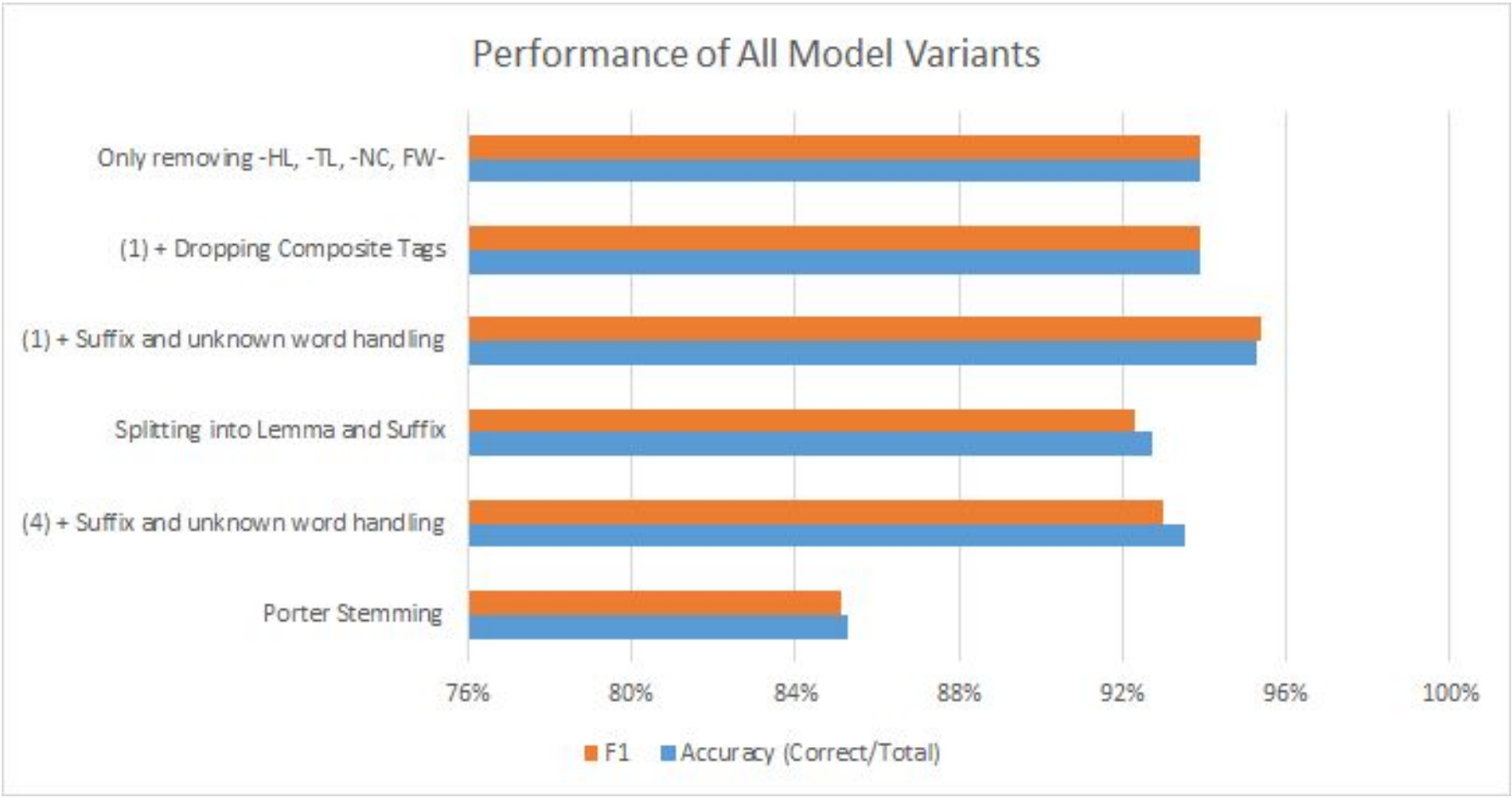
- Stemming
  - Using Porter stemming, words are replaced by their stems
  - Since stemmed root often contains insufficient information about POS, accuracy goes down
    Eg: I play cricket & I played cricket get stemmed the same, but POS tags are distinct

Performance

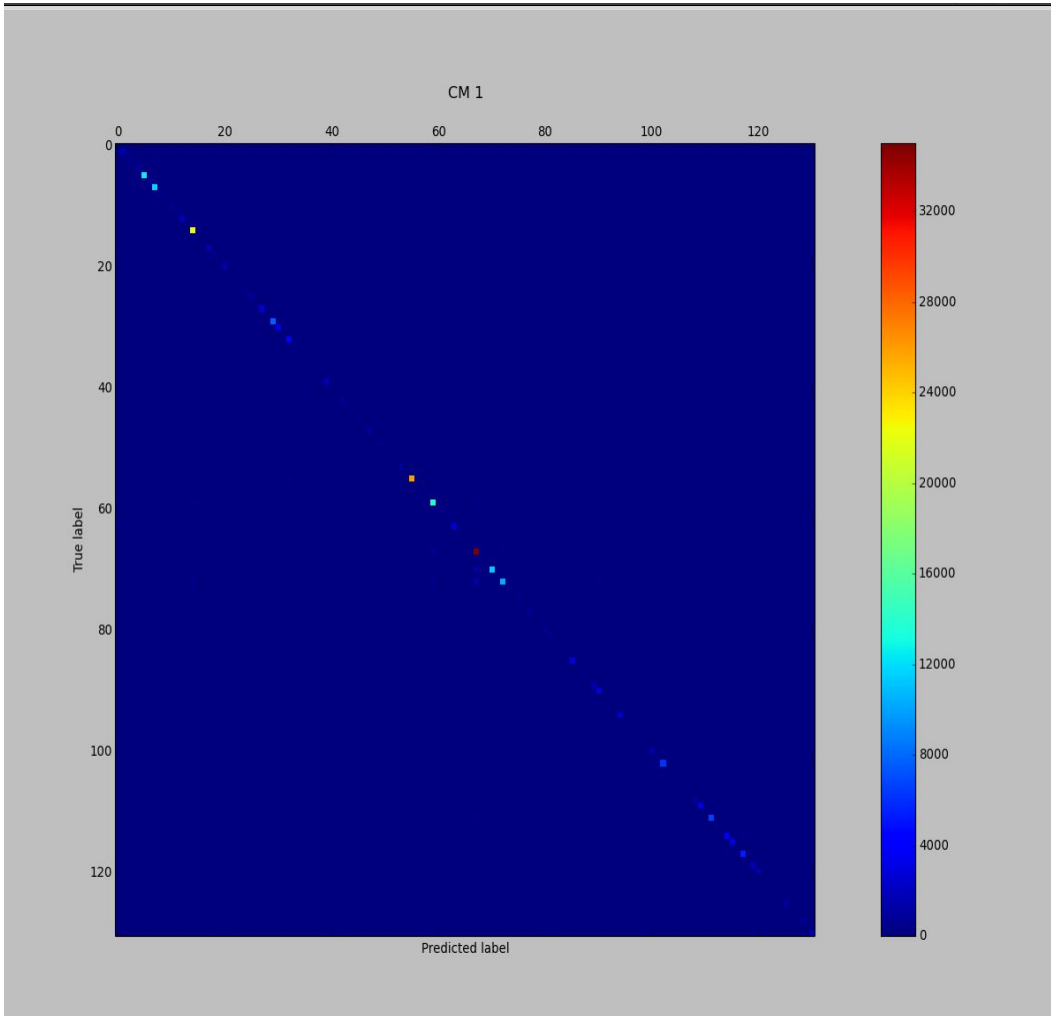
Using 5-fold cross-validation, different models produced performance as:

	Model	Accuracy (Correct/Total)	Precision	Recall	F1
1	Only removing -HL, -TL, -NC, FW-	93.90%	0.939	0.939	0.939
2	(1) + Dropping Composite Tags	93.89%	0.939	0.938	0.939
3	**(1) + Suffix and unknown word handling**	**95.26%**	**0.961**	**0.945**	**0.954**
4	Splitting into Lemma and Suffix	92.70%	0.928	0.918	0.923
5	(4) + Suffix and unknown word handling	93.50%	0.933	0.928	0.930
6	Porter Stemming	85.29%	0.886	0.818	0.851

Performance



Confusion Matrix



Conclusions

- Removing composite tags boosts performance, but also significantly improves computation speed. Since Viterbi complexity is O(<word-count>*<tag set size>^{2}) , speed up of 30x achieved compared to full composite-tag tagging

- Using only lemma or stem reduces performance significantly. In fact, lowest performance was achieved using only stemmed information

- The best model performance was achieved with smart handling of unknown words, using suffix information. (Accuracy: 95.26%)

Assignment 2

CYK Parsing

About the Grammar

Most of the grammar is derived from the the basic structure of:

VG -> (MOD) (PERF) (PROG) (PASS) VERB

However, not all combinations of the above are valid. Therefore, those rules are encoded explicitly.

We have also tried to handle most exceptional cases

About the Grammar

Auxiliary verbs, and perfect forms are also grouped according to syntactic and semantic rules.

Eg:

could be giving vs can be giving

could have been eating vs could has been eating

So, ‘could’ & ‘can’ need to be treated differently; and ‘have’ and ‘has’ need to be treated differently.

About the Grammar

- Particles and modals are accommodated wherever appropriate
- The Brown corpus was scraped and unique terminals productions for different verb forms were extracted
- To convert the grammar to CNF, we wrote a parser that allowed the manual writing of the grammar to be much more simplified

Algorithm

- The CYK algorithm was implemented to load the grammar once and parse multiple verb groups

- Ambiguous verb groups are also handled
  - VG -> A B; A -> X; B-> Y Z
  - VG -> C D; C -> X Y; D -> Z

Limitations

- Semantic attachment preferences
  - do not be gone
  - do not be given

  - wanted to eat
  - slept to eat

These distinctions need semantic information to be captured

Limitations

- Phrasal verb
  - will take off
  - will like off

  - may give in
  - may dance in

Requires all phrasal verbs and prepositions to be pre-determined

Limitations

- Verb Form Occurrences in Corpus
  - Since list of VB, VBN, … has been picked from the Brown corpus, verbs that do not occur in the expected form (or at all) will be treated as incorrect
  - could have been electrocuted

Conclusions

- Since testing is manual, it is difficult to estimate accuracy exactly, but both precision and recall were found to be high
- Grammar performs well at syntactic level, but no semantic information can be captured
- In these cases, the bias is towards accepting sentences with syntactically correct structure

Assignment 3

Parse Tree Interconversion

Motivation For Converting C2D

- Transition based Dependency Parsing algorithms run in linear time compared to cubic time in constituent parsing.
- Dependency Parse Tree are more helpful in languages which do not have strong word order.

Motivation For Converting D2C

- Constituency Parse Trees are relatively harder to annotate
- DPTs are easier to understand and to teach to people without a linguistic background
- We can construct phase structure corpus, by first building dependency structure corpus and then converting it

Constituent Parse Tree to Dependency Parse Tree

CPT to DPT

- Rule-based approach for ‘propagation’ of head in constituent parse tree
- Find head of every single constituent in CPT in bottom up manner
- Relationship between adjacent nodes established on this basis

CPT to DPT

- base case
  for X -> a
  head(X) = a
- recursive case
  for X -> Y_1 Y_2 … Y_n
  h=head(X -> Y_1 Y_2 … Y_n)
  Then, head(X) = head(Y_h)

(Source: Collins, 1999)

Assigning Dependency Labels

- Heuristics based approach derived from careful analysis of several constituent Treebanks
- Due to Marcus et al., 1993; Nielsen et al., 2010; Weischedel et al., 2011; Verspoor et al., 2012

Limitations

- Wh- Movement

  What did they try to do?

Expected	Obtained
['ROOT-0', 'did2']	['ROOT-0', 'did2']
['did2', 'What1']	['did2', 'What1']
['did2', 'they3']	['did2', 'they3']
['did2', 'try4']	['did2', 'try4']
['try4', 'to5']	['try4', 'to5']
['to5', 'do6']	['to5', 'do6']

Limitations

- Incorrectly parsed constituent trees (correctly parsed with revised tree)

  A hearing is scheduled on the issue today

Stanford parser's constituency parsing

```
ROOT(
  (S
    (NP (DT A) (NN hearing))
    (VP (VBZ is)
      (VP (VBN scheduled)
        (PP (IN on)
          (NP (DT the) (NN issue)))
        (NP (NN today)))))))
```

Dependency tree obtained

```
['ROOT-0', 'is3']
['is3', 'hearing2']
['hearing2', 'A1']
['is3', 'scheduled4']
['scheduled4', 'on5']
['on5', 'issue7']
['issue7', 'the6']
['scheduled4', 'today8']
```

Possible Improvements

- Handle discontinuities like topicalization, *wh*-fronting, scrambling, and extraposition using complex rules.

  Topicalization: Topicalization is a mechanism of syntax that establishes an expression as the sentence or clause topic: in English, by having it appear at the front of the sentence or clause

  E.g. I don't like **that thing**

  That thing I don't like

  ***Wh*-movement**: Sentences or clauses containing a wh-word show a special word order that has the *wh*-word (or phrase containing the *wh*-word) appearing at the front of the sentence or clause

  Extraposition: Mechanism of the syntax which alters word order

  E.g. Someone who we don't know **left a message**
  Someone **left a message** who we don't know.

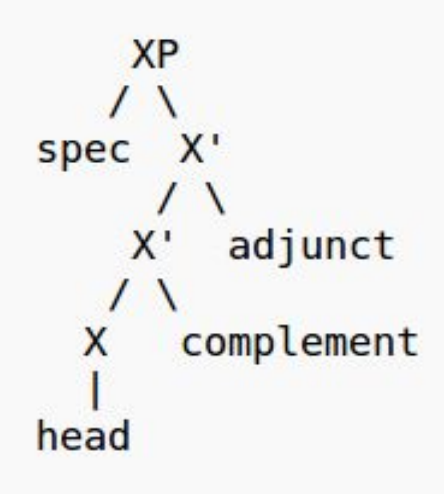
Dependency Parse Tree to Constituency Parse Tree

DPT to CPT

- Conversion based on X-bar theory which is a generic specification of language syntactic structure.
- The conversion aims to generate a phrase structure tree compatible with X-bar theory rules.

X-bar Theory

- A general notation for all phase structure
- For English word order:
  - Specifier Rule:
    - XP -> (specifier) X’ | X’ (specifier)
  - Modifier Rule:
    - X’ -> (adjunct) X’ | X’ (adjunct)
  - Argument/Complement Rule:
    - X’ -> X’ (complement...)
      | (complement…) X’



Resultant tree for one head

Categorization of Dependency Relationships

- Relationships in Dependency structure can be categorized in three categories:
  - Specifier
  - Argument/Complement
  - Modifier/Adjunct

- We manually classified the relations in the Stanford parser (about 50 of them)

Categorization

- Argument/Complement
  - A word, phrase or clause that is necessary to complete the meaning of a given expression. completes the meaning intended by head. Ex. agent

- Modifier
  - An optional element in phrase structure or cla structure that modifies the head. Can be remo from sentence without affecting grammatical correctness. Ex. adjectival modifier, appositio modifier, subj

Categorization

- Specifier

  A word, phrase or clause which qualifies (determines) the head. Non recursive. Ex. determiner, predeterminer

Example

She writes good novels.

writes - root

(she) - specifier of “writes”

(good novels) - argument of “writes”

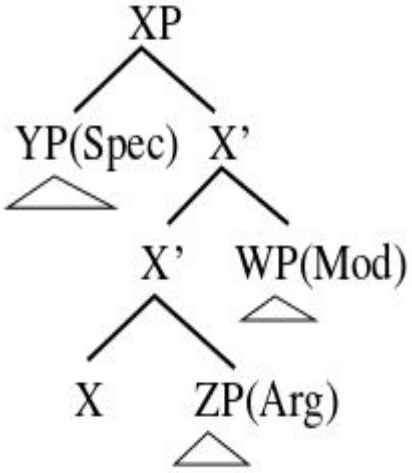
(good) - modifier of “novels”

Conversion algorithm

- Identify relationships from the given dependency tree.
- For each relationship between two words, create a new subtree capturing that relationship. Or, create a branch on the existing tree.
- Create or modify trees such that it remains compatible with X-bar theory. For that, do the manipulations to achieve the below structure for each head word and it's dependents:

Here, X is the syntactic category of the head word.

X = NN, VB, JJ etc



Algorithm

- Based on the type of relationship being captured - specifier, modifier, complement - attach sub-tree of dependent to appropriate node of the sub-tree of head (for that dependent)
  - Arguments attach 2 levels above the head.
  - Modifiers attach 3 levels above the head.
  - Specifiers attach 4 levels above the head.
- This will result in different sub-trees for each sub-phrase of the given sentence. Follow the above attachment rules until all dependency relations have been accounted for.
- Resultant tree is the phrase structure tree. This is a heuristic that results in phrase structure tree compatible with X-bar, which is known to be a correct specification.

Limitations

- The algorithm uses some rather inflexible generalizations that may not always be correct.
- These generalizations ignore all contextual information such as relative position of words, the syntactic category of words etc.
- For example, it states that modifiers always attach 3 levels above head and specifiers attach 4 levels above head. But this actually depends on the position of the words relative to each other and the head.
    - This was an example of generalization of “attachment positions”
- Similarly, the algorithm generalizes “number of projections of each category” and “number of projections for all dependents”

References

- Collins, Michael, *Head-Driven Statistical Models for Natural Language Processing*, PhD Thesis, University of Pennsylvania, 1999
- Choi, Jinho D. and Palmer, Martha, *Guidelines for the Clear Style Constituent to Dependency Conversion*, University of Colorado Boulder Institute of Cognitive Science, 2012
- Fei Xia and Palmer, Martha, *Converting dependency structures to phrase structures.*

Assignment 4

POS tagging using Neural Networks

Setup

- A word x word co-occurrence matrix was extracted from the Brown corpus
- Each row contains the number of contexts in which the row-word and column-word co-occur
- To prevent distortion by varying overall frequencies of words, we normalized each row by dividing by its sum

Setup

- These rows work as the word vectors for single words,

  $$\text{vec}(w)$$

- To predict tag for w_i, we use vectors made of

  $$\{ \text{vec}(w_{i-1}), \text{vec}(w_i), \text{vec}(w_{i+1}) \}$$

- This adds information about context to the neural network

Setup

- The output is encoded as the one-hot vector, indicating the actual tag, t_i of the target word

  [0,0,0,1,0,0,0,....0,0]

- The final tag is predicted for each word using an argmax computation on activations of the last layer

Training Phase

- We used a three-layer neural network
- #neurons in input layer = 3 * length of vector + 1 bias unit = 2878
- #neurons in output layer = #tags
- For the hidden layer, we tuned the parameter by trying multiple values of neurons. Bias unit was added as expected.
- Final model has 100 neurons in the hidden layer.

Training Phase

- Parameters for learning rate and momentum were also added
- Adaptive parameters are used so learning rate decreases in later iterations
- The optimal starting values were tuned using trial and error. Starting Values used:
  - Learning Rate: 0.5
  - Momentum: 0.0.4

Testing Phase

- Again, we generated the combined vector

$$\{ \text{vec}(w_{i-1}), \text{vec}(w_i), \text{vec}(w_{i+1}) \}$$

for each set of three consecutive words in the test sentence

- This was passed as input to the final trained neural network

Performance

- Because of small size of input data, but large number of weights to be learned, the algorithm gets stuck in local minima almost always. The NN ends up giving the same tag to every word. which achieves a trivial accuracy of 16%

- Using Schmid's method* works considerably better. We were able to get

  Accuracy ~ 85%

*Description in Appendix

Performance

In comparison, an HMM model using Viterbi algorithm, trained with a similar corpus size gave performance of (5-fold cross validation):

Accuracy: 69.39%

Precision: 0.68

Recall: 0.69

F1: 0.68

Learnings and Conclusions

- Because number of parameters to be estimated is very high, without large amount of data, neural networks cannot perform very well
- Adding more neurons to the hidden layer improves performance up to a limit, but increases training time significantly
- The learning rate parameters must be tuned well for good performance

Overall Assignments' Conclusions

- In most open-ended research, there are often multiple methodologies to solve a problem. This is even more so in NLP, where there are multiple options at each stage: framing, modeling, tuning, …

- It is a hard task to decide between these options. The best approach is to use empirical results. But this is time consuming, so intelligent judgments are critical

Appendix A

Schmid's Method (POS tagging using ANNs)

Setup (Input)

- For each word, we generated the vector containing probability

$$P(t|w)$$

- To predict tag for w_i, we use vectors made of

$$\{ P(t|w_{i-1}), P(t|w_i), P(t|w_{i+1}) \}$$

- This adds information about context to the neural network

Setup (Output)

- This is the same as used in word vector implementation
- The output is encoded as the one-hot vector, indicating the actual tag, t_i of the target word

  [0,0,0,1,0,0,0,....0,0]

- The final tag is predicted for each word using an argmax computation on activations of the last layer

Setup

- We used a three-layer neural network
- #neurons in input layer = 3 * #tags + 1 bias unit
- #neurons in output layer = #tags
- For the hidden layer, we tuned the parameter by trying multiple values of neurons. Bias unit was added as expected.
- Final model has 10 neurons in the hidden layer.

Testing Phase

- Again, we generated the combined vector

$$\{ P(t|w_{i-1}), P(t|w_i), P(t|w_{i+1}) \}$$

for each set of three consecutive words in the test sentence

- This was passed as input to the final trained neural network

Performance

- After optimally tuning the parameters of the neural network, we were able to get

  Accuracy ~ 85 %

Thank You