Simple Natural Processing Techniques

Name: Vishnu Asutosh Dasu

Affiliation: Manipal Institute of Technology,Second Semester, CSE

Location: Manipal, Karnataka

Email: vishnu98dasu@gmail.com

Phone: +91-9845567071

GitHub: vdasu

Mentor: Yash Kumar Lal

Description:

Implementation of fundamental NLP techniques and algorithms along with detailed analysis. These include POS, language modelling and representation. Popular libraries in python that provide a platform to implement NLP techniques include NLTK and spaCy, which can be used for baselining and testing purposes.
NLP is a very important domain in computer sciences. The goal of NLP is to facilitate seamless interaction between humans and computers in languages spoken by humans. A few good examples that utilise NLP techniques would be TTS(Text-to-Speech) systems and intelligent personal assistants like Siri, Cortana, etc.

Implementation:

The language that I've decided to work on is English and the programming language of preference is Python3 and the tagset I've chosen is the Penn Treebank. For this project, I've decided to built the POS tagger using Hidden Markov Models. The sequence of tags is then determined by the Viterbi Algorithm.
The Penn Treebank contains 2,499 articles from the Wall Street Journal, amassed over a period of three years. The tagset contains 36 tags, which would be sufficient to implement a basic POS tagger using the Viterbi algorithm. The time complexity of the Viterbi algorithm is given by $O(nS^2)$ where n is the length of the sentence and S is the size of the tagset.
Using larger and elaborate tagsets would significantly increase computational time as the time complexity is proportional to the square of the size of the tagset. The Penn Treebank was also

one of the first large scale treebanks to be implemented and is the default tagset used by popular NLP libraries like NLTK.
Choosing an appropriate tagset depends on the what the tagged text would be used for. More elaborate and intricate tagsets like the Brown tagset can be used depending on the requirements of the project.
For this project, I've decided to use the Penn Treebank as it is one of the most widely used tagsets for the English language thereby making the process of baselining as easier task. Later on, I plan to add support for further tagsets once the implementation of the Viterbi algorithm has been perfected.

Word Tokenizer:

The initial step before designing a POS tagger is to implement a word tokenizer. This can be done easily in Python using regular expressions. Below is a naive implementation of a simple word tokenizer.

```
>>> import re
>>> text = "Hello, my name is Vishnu."
>>> re.findall(r"\w+|[^\w\s]",text)
['Hello', ',', 'my', 'name', 'is', 'Vishnu', '.']
```

Once the text has been tokenized it can be passed into the trained POS tagger.

Language Modelling:

Consider a finite set of vocabulary in the English language, α = {the,dog,barks,at,cat}. We can then define a set of strings β which contain words from α. Some examples of sentences in β include "*the%", "*the barks%", "*the dog barks%", "*the cat at%", etc. The special character '*' denotes the start of the sentence and '%' denotes the end of the sentence.
The task at hand with Language Modelling is to define a probability distribution P, such that

$$\sum_{x \in \beta} P(x) = 1, P(x) \geq 0 \forall x \in \beta$$

.
The probability is computed with reference to a training set that appropriately chosen.
A good Language Model assigns high probability to sentences that are more likely to occur in the language. For example, in the sample set, the sentence "the dog barks" should be assigned a higher probability as compared to the other sentences.

Simplify and reduce computational resources required to compute the joint probability distribution:

A Markov model is a stochastic process that satisfies the Markov property. To put it explicitly, a process satisfies a Markov property if it is conditional (conditional probability is the measure of

probability of an event given that another event has already occurred) on its present state and independent of its future and past states.
Consider three random A, B variables and C. The joint probability of these events can be computed as follows,
$P(A, B, C) = P(A) \times P(B|A) \times P(C|A, B)$. This expression follows directly from the definition of conditional probabilities. Scaling up to to n terms,

$$P(X_1 = x_1, ...X_n = x_n) = P(X_1 = x_1) \prod_{i=2}^{n} P(X_i = x_i | X_1 = x_1, ...X_{i-1} = x_{i-1})$$.

The above expression for the joint probability of n events is the exact value. The downside is, however, it is computationally intensive. The probability of joint events can be approximated by using Markov's assumption.

$$P(X_1 = x_1, ...X_n = x_n)P \approx P(X_1 = x_1) \prod_{i=2}^{n} P(X_i = x_i | X_{i-1} = x_{i-1})$$

The above expression is known as the First-Order Markov Process as the conditional probability of each term depends only on one term before it. If the approximation is made such that each term depends on k terms ($1 \leq k \leq n$) before it, it is known as the k^{th} Order Markov Process. Naturally, larger the value of k, better the approximation.

Design a Second-order Markov Language model:

A Trigram Language Model consists of the following components:

- A finite set of vocabulary α
- A set of parameters $q(z|x, y)$, for each trigram x, y, z (which a basically a set of three words), such that $z \in \alpha \bigcup \{\%\}$ and $x, y \in \alpha \bigcup \{*\}$. Here '*' and '%' denote the start and end of a sentence respectively.

The probability of the occurrence of a sequence of words can then be evaluated using Second-Order Markov Model as follows,
Given a sentence $x_1 \ldots x_n \forall x \in \alpha \bigcup \{\%\}$ and $x_n = \%$, the probability of this sentence according to the Trigram Model is given by,

$$P(x_1 \ldots x_n) = \prod_{i=1}^{n} q(x_i | x_{i-1}, x_{i-2})$$ where $x_{-1} = x_0 = *$.

POS Tagging:

As the name suggests, POS Tagging classifies words based on their grammatical properties. For example, given the sentence "My name is Vishnu" a POS tagger would classify the words as follows: My (Determiner), name(Noun), is(Verb), Vishnu(Noun).

To sum up, given a sentence $x_1 x_2 \dots x_n, \forall x \in \alpha$ the task of the POS tagger is to map each word in the sentence to a set tags $y_1 y_2 \dots y_n, \forall y \in S$ where S is the set of all POS tags. An example of such a set would be the Penn Treebank.

Defining the joint probability distribution overall possible words and tags:

Consider an input sentence $x = x_1 x_2 \dots x_n, \forall x \in \alpha$and set of tags $y = y_1 y_2 \dots y_n, \forall y \in S$.
The task at hand is to use a HMM to define the following:
$P(x_1, \dots, x_n, y_1, \dots, y_n)$, which is the joint distribution of the sequence of words and the corresponding tags. This is a Generative Model where we try to learn a joint distribution from the training examples.
The most likely sequence of tags for a given sentence can be obtained by varying y to obtain the maximum joint probability distribution P . This is denoted as:
$arg \max_{y} P(x_1, \dots, x_n, y_1, \dots, y_n)$

Compute the joint probability sequence as a Second-Order Markov process:

Given a sentence $x = x_1 x_2 \dots x_n, \forall x \in \alpha$ and a set of tags $y = y_1 y_2 \dots y_n, \forall y \in S$, the joint probability of the words in the sentence and the tags can be defined as follows:

$$P(x, y) = \prod_{i=1}^{n+1} Q(y_i | y_{i-1}, y_{i-2}) \prod_{i=1}^{n} R(x_i | y_i)$$.

Here, the tags corresponding to x_0 and x_1 are taken as $*$, which denotes the start of the sentence. The tag y_{n+1} is taken as $\%$ which denotes the end of the sentence.
The above joint probability distribution can be proved easily using conditional probability.
$P(A, B) = P(A|B).P(B)$. Here A corresponds to sentence x and B corresponds to the set of tags y. The expression $P(B)$ is simplified and expressed as a Second-Order Markov process.
The task at hand is to estimate Q and R so that the joint distribution P. This can be done with the help of the Viterbi algorithm.

Maximizing the joint probability distribution:

The problem that has to be dealt with is maximize the joint probability distribution P.

P takes the form $P(x,y) = \prod_{i=1}^{n+1} Q(y_i|y_{i-1}, y_{i-2}) \prod_{i=1}^{n} R(x_i|y_i)$, where $y_0 = y_{-1} = *$ and $y_{n+1} = \%$.

One possible approach would be to iterate through all the possible tags and find the sequence which maximises P. The time complexity of such a solution would be $O(s^n)$, where s is the size of the tag set and n is the number of words in the sentence. This is very inefficient as the complexity increases exponentially with increase in the number of words in the sentence.

One possible way to tackle this problem is to use dynamic programming. At the each stage, we compute the maximum probability of the tag sequence at a given position in the sentence. This is in turn depends on the choice of tags of the previous two words as the probability sequence has been approximated using a Trigram model. The base case of the algorithm returns 1 as at the first two words of the sentence are the start characters, which are already known and fixed. The tags that maximize the joint probability sequence is stored at each stage and finally the entire tag sequence returned.

Given below is the pseudo-code of the Viterbi Algorithm:

```
Input: A sentence $x = x_1 \dots x_n$, $Q(s|u,v)$ and $R(x|s)$
Base Case: $\Psi(0,*,*) = 1$
Algorithm: For $k \in \{1 \dots n\}$:
        For $u, v \in S$:
            $\Psi(k,u,v) = \max_w(\Psi(k-1,w,u).Q(v|u,w).R(x_k|v))$
            $T(k,u,v) = arg\max_w(\Psi(k-1,w,u).Q(v|u,w.R(x_k|v))$
        $y_{n-1}, y_n = arg\max_{u,v}(\Psi(n|u,v).Q(\%|u,v))$
        For $k \in \{1..(n-2)\}$:
            $y_k = T(k+2, y_{k+1}, y_{k+2})$
    Return $\{y_1 \dots y_n\}$
```

Legend: $*$ - Start character

$\%$ - Stop character

x - Input sentence

y - Set of output tags

S - Set of all tags

u, v, w - Tags that belong to S

n - Number of words in x

Ψ - Maximum probability of the tag sequence ending in tags u, v at the position k

T - Arguments of Ψ

Q, R are components of the joint probability distribution P

Challenges:

Implementation of the word tokenizer is a fairly easy task and can be accomplished in a short duration of time. Although I believe I have a fair understanding of Language Modelling, Markov Process and the Viterbi algorithm the implementation of the POS tagger is going to be a mammoth task. All these concepts and ideas I haven't learned before as I'm new to NLP. A thorough revision a couple more times should solidify these concepts and the implementation would still take some time, but hopefully be smooth sailing.

Timeline:

During the coding period, I can guarantee to dedicate at least 35 hours a week to the project.

June1 - June4, 2017:

- Interaction with students and mentors.
- Analyse the coding standards to be followed and expectations of mentors.

June5 - June8, 2017:

- Brush up basic concepts of Probability and Statistics
- Analyse the PennBank tagset
- Determine the word tokenizer to be designed

June9 - June14, 2017:

- Basic implementation of the word tokenizer

June14 - June21, 2017:

- Compare the word tokenizer to existing implementations
- Improve the word tokenizer built
- Write documentation of code written so far

June21 - June25, 2017:

- Plan implementation of POS tagger
- Begin work on the Viterbi algorithm
- Test the algorithm on small training data ie small sentences and custom tagset

June26 - June30, 2017:

- Midterm evaluations

July1 - July12, 2017:

- Implement a fully working POS tagger
- Testing on the complete dataset

July13 - July16, 2017:

- Baselining with existing implementations (NLTK)
- Improve the POS tagger as much as possible
- Write documentation for the code written so far

July17 - July20, 2017:

- Final Evaluations

Background:

Cultivating interest in data science and statistics. Aspire to become a data scientist one day. I do have prior experience in Machine Learning and NLP. I'm currently doing two courses on ML: Stanford ML course, Coursera and Udacity's Intro to Machine Learning (my solutions to these courses can be found on my github). I'm also working on Dr. Bhargav Bhatkalkar's research project which aims to accurately predict macular degeneration disorder so as to initiate treatment from an early (my contributions to the project can be found on my github).

References:

1. https://nlp.stanford.edu/software/tagger.shtml
2. http://www.nltk.org/book/ch01.html
3. http://www.phontron.com/slides/nlp-programming-en-04-hmm.pdf
4. https://nlp.stanford.edu/courses/cs224n/2010/reports/parawira.pdf
5. http://partofspeech.org/
6. https://www.youtube.com/user/afigfigueira/playlists?sort=dd&shelf_id=5&view=50
7. https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

8. http://www.comp.leeds.ac.uk/amalgam/tagsets/upenn.html
9. http://cl.indiana.edu/~md7/09/645/slides/07-pos-tagging/07-pos-tagging.pdf
10. https://web.archive.org/web/19970614160127/http://www.cis.upenn.edu:80/~treebank/
11. Discrete Mathematics and Its Applications [7th Edition] by Kenneth H. Rosen