Mukundhan, Vaishnavi

Lee, Senyung

Word Sense Disambiguation Classifier

Target word: *driver*

* Sense 1: a device (e.g., *You have to install an audio driver to your computer*.)
* Sense 2: driver as a person (e.g., *The bus driver was singing as he drives*.)

Files for training:

* Driver\_Technology.txt (from <https://en.wikipedia.org/wiki/Driver_circuit> and

<https://en.wikipedia.org/wiki/Device_driver>),

* Driving.txt (from <https://en.wikipedia.org/wiki/Driving>, <https://en.wikipedia.org/wiki/Chauffeur>, <https://en.wikipedia.org/wiki/Motorman>)

Files for testing:

* video\_drivers.txt (from <http://www.tomshardware.com/forum/230027-33-what-install-video-drivers>)
* truck\_driver.txt (from <http://www.alltrucking.com/faq/first-year-truck-driver-salary>)

Python file for feature extraction:

* wsd.py

**Phase 1. Making a classifier using a Naïve Bayesian approach**

Using Python 2.7, we implemented a classifier that categorizes a given text as referring to one or the other meaning of the target word, *driver*. The steps we took are as follows:

1. Create two corpora of text, one for Sense 1 (technology) and one for Sense 2 (person)
2. Create two lists: one with all words in the first corpus, and one with all words in the second corpus
3. POS-Tag the words in the two lists using NLTK
4. Filter the two lists. From the lists,

1) remove stop words,

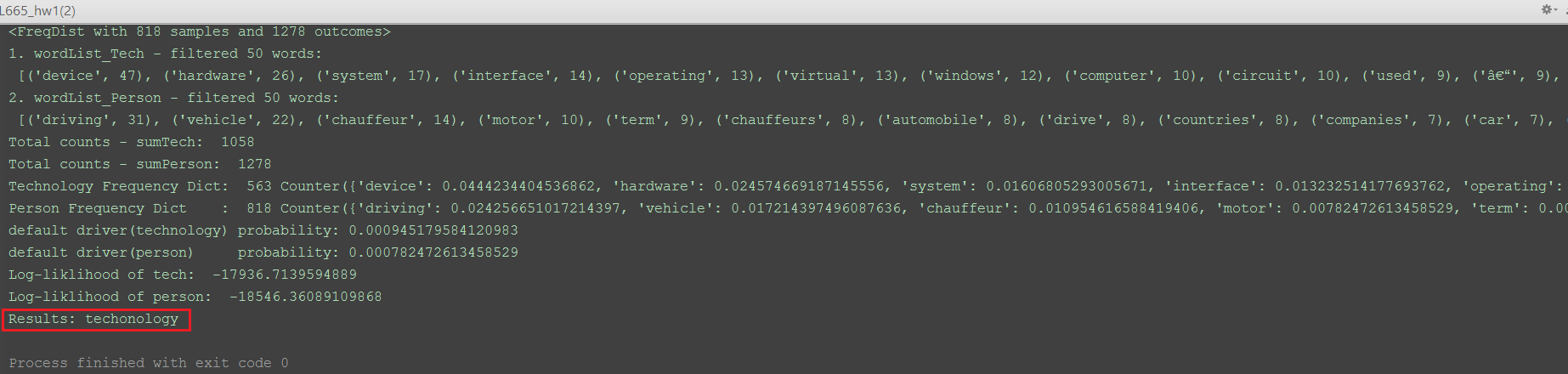
2) manually add more stop words into the NLTK stop words list (e.g., ‘a’, ‘the’, ‘:’)

3) remove any numbers (numeric characters) because numbers themselves are not informative

4) remove the words that occur in both corpora

Now the two lists only contain open-class words (which we deem informative for word sense disambiguation) that occur only in one of the two corpora.

1. Create two frequency profiles using the two filtered lists. Our frequency profiles are lists of tuples in decreasing order of the raw frequency of each word.
2. Take the most frequent 1000 words from each of the two lists. (This step was added later to see if the accuracy improves.)
3. Add the word-length into the feature set (This step was added later to see if the accuracy improves.)
4. Create a model for training. The model includes three (or four) features and a category: word, POS, (length of the word,) probability of the word occurring in a text that used the word in a certain sense, and category
5. The created Naïve Bayesian classifier tells whether the word *driver* in a given test text was used in Sense 1 or Sense 2.
6. We tested this classifier on an unseen text file *video\_drivers.txt*. The classifier correctly classified it as “technology” as follows.

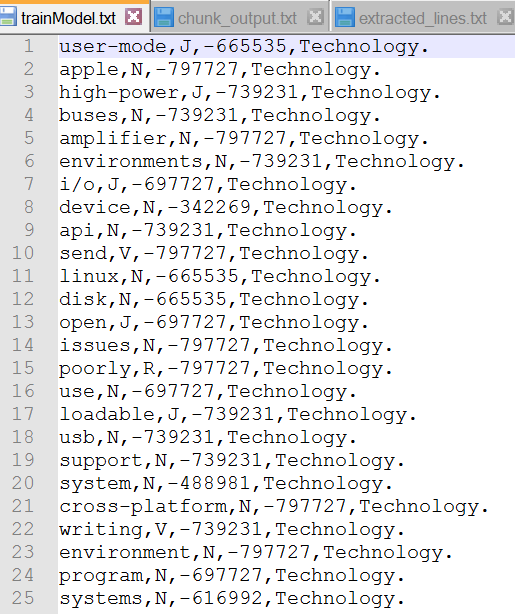


11. After finishing making a classifier in Python, we extended the program so that it gets an input text file and converts it into a model that can be fed into timbl (C4.5 format). The last element of each line in the test model is the category determined by the classifier.

Note: We decided not to lemmatize the words in the corpora because a word such as *driving* gives information that the lemma *drive* is being used to describe the act of driving. If this word is lemmatized to *drive*, then it loses its distinctive meaning of *driving* and the ambiguity is increased.

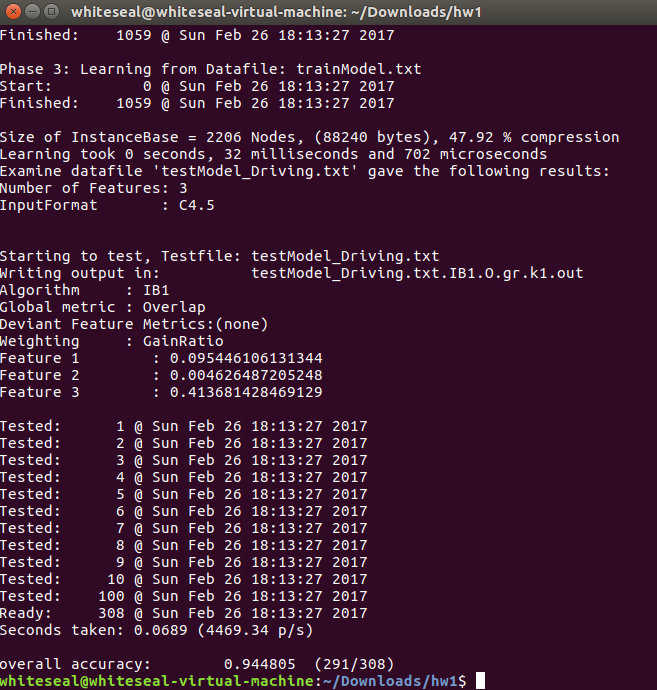
**Phase 2. Training and testing the model in TiMBL**

Timbl was installed on Ubuntu Linux 16.10 using a virtual machine on a laptop with Windows 10. Following the Timbl manual, we used C4.5 format for input data for Timbl, as shown in the screenshot below. One left one shows the model with three features (word, POS, probability), and the right one shows the model with four features (‘length of the word’ added).



First, using the training model with three features, we tested the classifier by TiMBL on an unseen text file *truck\_driver.txt* (from www.alltrucking.com/faq/first-year-truck-driver-salary). The test file included the features we extracted (word, POS, and probability of the word occurring in the text) and its category determined by the Naïve Bayesian classifier we created in Phase 1.

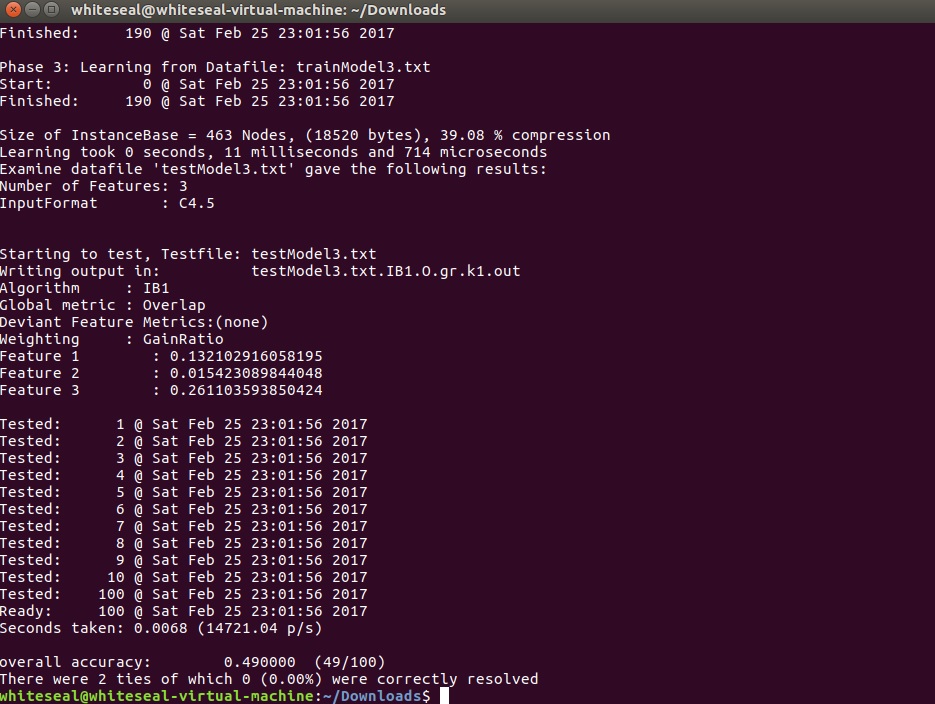
Initially, the accuracy on Timbl ranged from 41% to 56%. But after several trials, we extensively removed stop words, any non-informative function words (numbers, modals, etc), and the words that occur in both corpora, the accuracy reached 94%.



The operation log is as follows:

|  |
| --- |
| whiteseal@whiteseal-virtual-machine:~/Downloads/hw1$ timbl -f trainModel.txt -t  testModel\_Driving.txtTiMBL 6.4.8 (c) CLST/ILK/CLIPS 1998 - 2016. Tilburg Memory Based Learner Centre for Language and Speech Technology, Radboud University Induction of Linguistic Knowledge Research Group, Tilburg University CLiPS Computational Linguistics Group, University of Antwerp Sun Feb 26 18:13:26 2017  Examine datafile 'trainModel.txt' gave the following results: Number of Features: 3 InputFormat       : C4.5  Phase 1: Reading Datafile: trainModel.txt Start:          0 @ Sun Feb 26 18:13:26 2017 Finished:    1059 @ Sun Feb 26 18:13:27 2017 Calculating Entropy         Sun Feb 26 18:13:27 2017 Lines of data     : 1059 DB Entropy        : 0.95908894 Number of Classes : 2  Feats    Vals    InfoGain    GainRatio     1   1059    0.95908894    0.095446106     2     10    0.0079490035    0.0046264872     3     25    0.95908894    0.41368143  Preparation took 0 seconds, 24 milliseconds and 930 microseconds Feature Permutation based on GainRatio/Values : < 3, 2, 1 > Phase 2: Building multi index on Datafile: trainModel.txt Start:          0 @ Sun Feb 26 18:13:27 2017 Finished:    1059 @ Sun Feb 26 18:13:27 2017  Phase 3: Learning from Datafile: trainModel.txt Start:          0 @ Sun Feb 26 18:13:27 2017 Finished:    1059 @ Sun Feb 26 18:13:27 2017  Size of InstanceBase = 2206 Nodes, (88240 bytes), 47.92 % compression Learning took 0 seconds, 32 milliseconds and 702 microseconds Examine datafile 'testModel\_Driving.txt' gave the following results: Number of Features: 3 InputFormat       : C4.5   Starting to test, Testfile: testModel\_Driving.txt Writing output in:          testModel\_Driving.txt.IB1.O.gr.k1.out Algorithm     : IB1 Global metric : Overlap Deviant Feature Metrics:(none) Weighting     : GainRatio Feature 1     : 0.095446106131344 Feature 2     : 0.004626487205248 Feature 3     : 0.413681428469129  Tested:      1 @ Sun Feb 26 18:13:27 2017 Tested:      2 @ Sun Feb 26 18:13:27 2017 Tested:      3 @ Sun Feb 26 18:13:27 2017 Tested:      4 @ Sun Feb 26 18:13:27 2017 Tested:      5 @ Sun Feb 26 18:13:27 2017 Tested:      6 @ Sun Feb 26 18:13:27 2017 Tested:      7 @ Sun Feb 26 18:13:27 2017 Tested:      8 @ Sun Feb 26 18:13:27 2017 Tested:      9 @ Sun Feb 26 18:13:27 2017 Tested:     10 @ Sun Feb 26 18:13:27 2017 Tested:    100 @ Sun Feb 26 18:13:27 2017 Ready:     308 @ Sun Feb 26 18:13:27 2017 Seconds taken: 0.0689 (4469.34 p/s)  overall accuracy:        0.944805  (291/308) |

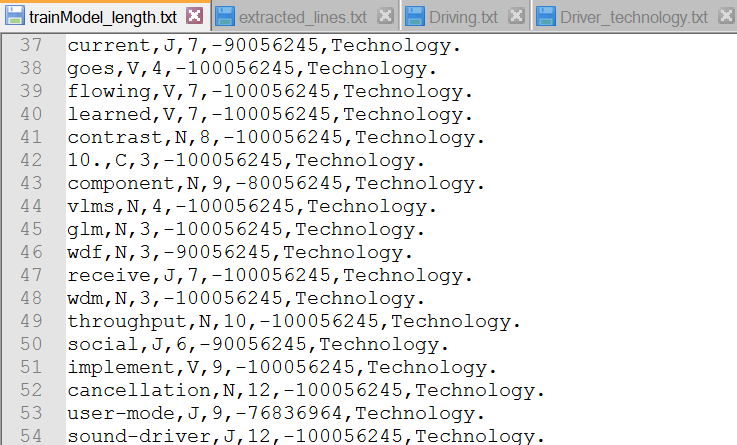
When we ran the TiMBL classifier on another unseen text, *video\_drivers.txt* (from [www.tomshardware.com/forum/230027-33-what-install-video-drivers](http://www.tomshardware.com/forum/230027-33-what-install-video-drivers)), the accuracy did not reach as high as the text used for driving. Before cleaning data, the accuracy was around 26%, and after extensively cleaning the data, the accuracy reached 49%. The results are as follows:



**Phase 3. Discussion**

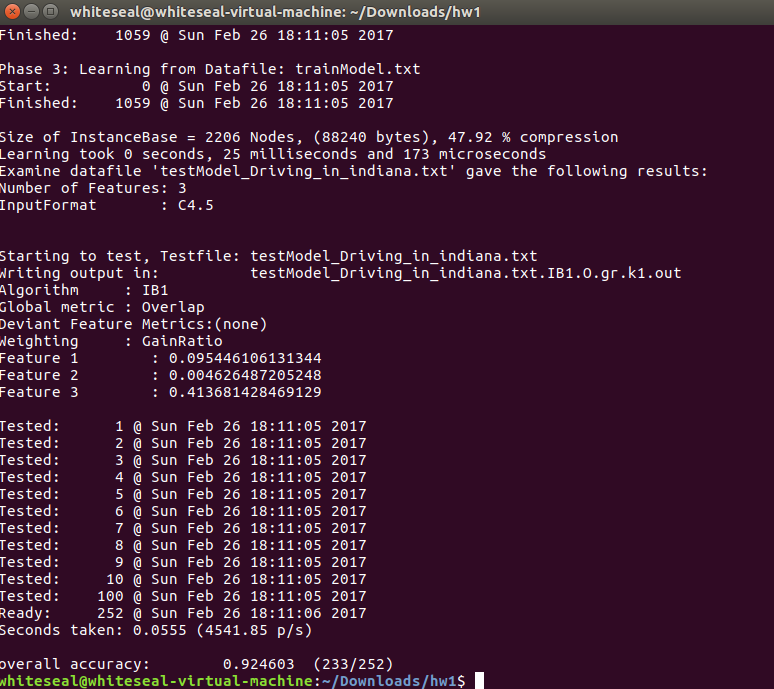
Initially, the Naïve Bayesian classifier we made in Phase 1 did not exclude the words that occur in both corpora, and it did not classify testing data accurately. In addition, at first when making the Naïve Bayesian classifier, we only used the 100 most frequently occurring open-class words in each of the two corpora. This was because we initially thought that the words that only occur one time in a corpus do not give much information. Later we removed all non-informative words (e.g., *can, may, one, two, the, a(n)*) and words that occur in both corpora, and we obtained much cleaner word list with selected features. After taking this measure, we decided to use the 1000 most frequently occurring open-class words, and the accuracy improved.

Just to see if we can improve the accuracy with one more feature, we added one more feature in our Python feature extractor, the length of each word. The resulting file from the feature extractor has one more column of feature, as follows:



When we added the word-length feature, the accuracy on Timbl was not consistent. The accuracy ranged from 26% (on “video\_drivers.txt”) to 100% (“truck\_driver.txt”). Since adding the word-length feature did not yield consistent results, we decided to take this feature out.

We wanted to see how our classifier works on another unseen data, so we fed a new file “driving\_in\_indiana.text” as a test file (taken from <http://www.in.gov/bmv/2332.htm>). The classifier correctly categorized it as *person* and created a feature model from this text. When we ran this test model on Timbl, the accuracy was 92%, as can be seen in the image below:



We also tried to include the WordNet hypernyms() function in NLTK, but after consideration, we decided not to use it. This is because a word that could be common in the *technology* corpus such as *program* had *plans* as the hypernym based on the first element of synset in WordNet. That is, if we use hypernym as a feature, all the occurrences of *program* will have a feature *plans* as a hypernymwhich can dramatically increase the ambiguity of the target term *driver*. Since selecting an appropriate word sense using WordNet and deducing a correct hypernym is another level of word sense disambiguation, apart from deducing the word sense of *driver*, we decided not to include the hypernym feature.

Overall, while the accuracy on Timbl ranged from 56% to 92% when we used a test model file based on person-related text (truck\_driver.txt), the accuracy on Timbl ranged from 26% to 49% when we used a test model file based on technology-related text (video\_drivers.txt). Thus, it is possible that the classifier we implemented is more likely to classify any given text as *person* over *technology*. We believe that perhaps the fact that the *technology* corpus used for training had much fewer words (“Driver\_Technology.txt”, around 1686 words) than the *person* corpus (“Driving.txt, ”around 2202 words) may have factored in on the accuracy of classifier because the *person* corpus had more information to use than the *technology* corpus. Using bigger corpora for training data might improve the accuracy of our classifier.