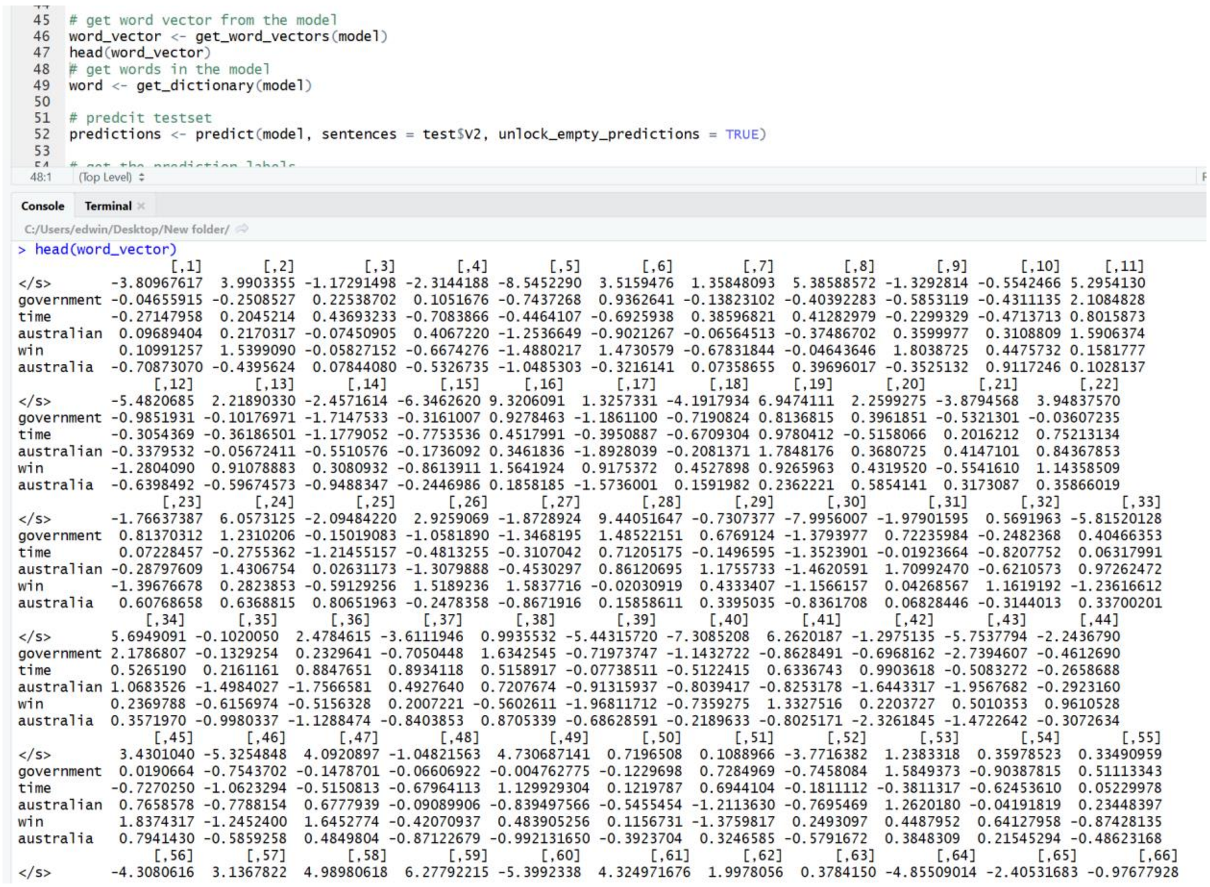
Input training file for the R code:

The input training file is named as “train_remStop_Tok_Lem_remTop4.csv” and the input test file is name as “test_Lem_Tok_removestop.csv” in the war file. Please refer section the pre-processing steps for more information.

Further explanation of the word feature and word vector of the assessment:

The word vectors and word dictionary are demonstrated in the R code as follow:

Word vector:

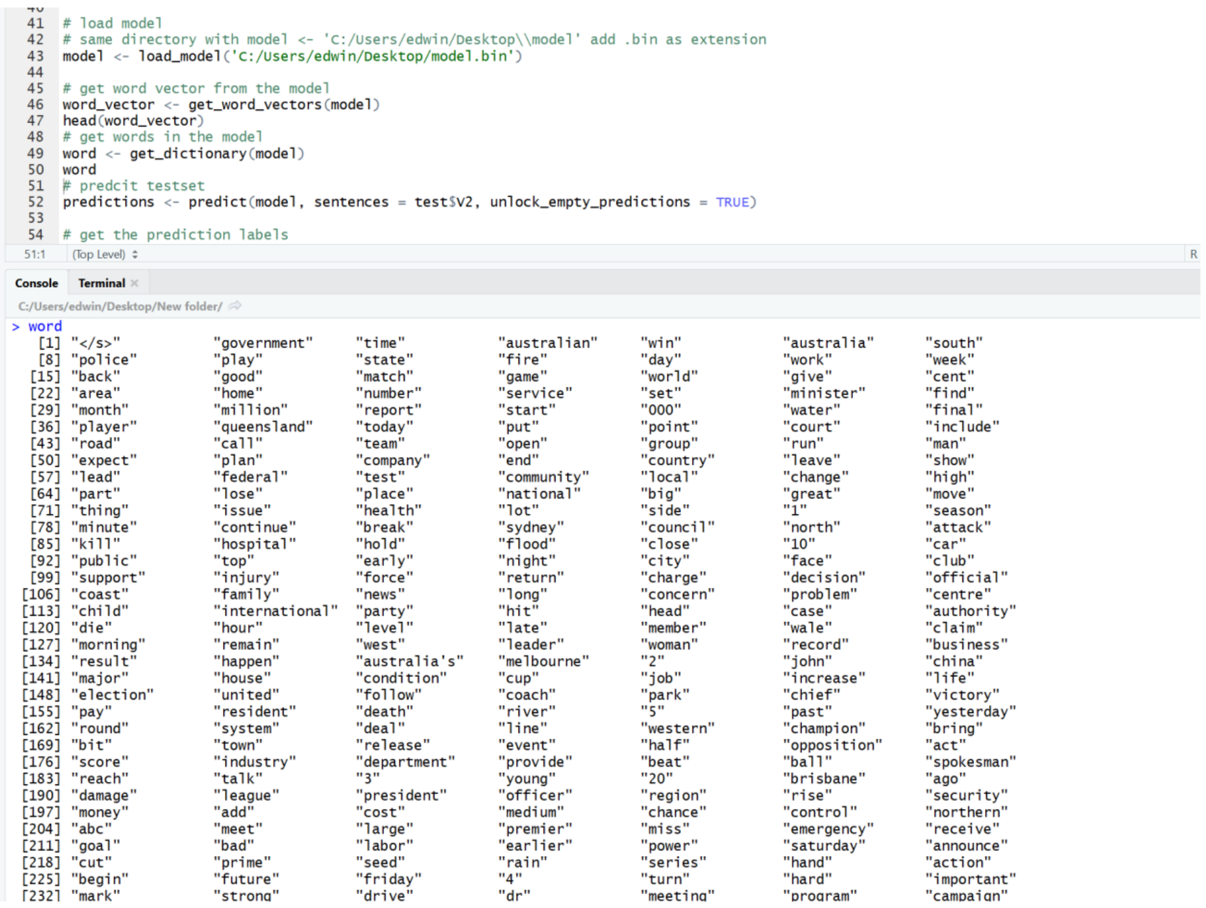


```r
# get word vector from the model
word_vector <- get_word_vectors(model)
head(word_vector)
# get words in the model
word <- get_dictionary(model)

# predcit testset
predictions <- predict(model, sentences = test$V2, unlock_empty_predictions = TRUE)
```

```
> head(word_vector)
                  [,1]       [,2]        [,3]       [,4]       [,5]       [,6]        [,7]        [,8]       [,9]      [,10]     [,11]
</s>       -3.80967617  3.9903355 -1.17291498 -2.3144188 -8.5452290  3.5159476  1.35848093  5.38588572 -1.3292814 -0.5542466 5.2954130
government -0.04655915 -0.2508527  0.22538702  0.1051676 -0.7437268  0.9362641 -0.13823102 -0.40392283 -0.5853119 -0.4311135 2.1084828
time       -0.27147958  0.2045214  0.43693233 -0.7083866 -0.4464107 -0.6925938  0.38596821  0.41282979 -0.2299329 -0.4713713 0.8015873
australian  0.09689404  0.2170317 -0.07450905  0.4067220 -1.2536649 -0.9021267 -0.06564513 -0.37486702  0.3599977  0.3108809 1.5906374
win         0.10991257  1.5399090 -0.05827152 -0.6674276 -1.4880217  1.4730579 -0.67831844 -0.04643646  1.8038725  0.4475732 0.1581777
australia  -0.70873070 -0.4395624  0.07844080 -0.5326735 -1.0485303 -0.3216141  0.07358655  0.39696017 -0.3525132  0.9117246 0.1028137
                [,12]       [,13]      [,14]      [,15]     [,16]      [,17]      [,18]     [,19]      [,20]      [,21]       [,22]
</s>       -5.4820685  2.21890330 -2.4571614 -6.3462620 9.3206091  1.3257331 -4.1917934 6.9474111  2.2599275 -3.8794568  3.94837570
government -0.9851931 -0.10176971 -1.7147533 -0.3161007 0.9278463 -1.1861100 -0.7190824 0.8136815  0.3961851 -0.5321301 -0.03607235
time       -0.3054369 -0.36186501 -1.1779052 -0.7753536 0.4517991 -0.3950887 -0.6709304 0.9780412 -0.5158066  0.2016212  0.75213134
australian -0.3379532 -0.05672411 -0.5510576 -0.1736092 0.3461836 -1.8928039 -0.2081371 1.7848176  0.3680725  0.4147101  0.84367853
win        -1.2804090  0.91078883  0.3080932 -0.8613911 1.5641924  0.9175372  0.4527898 0.9265963  0.4319520 -0.5541610  1.14358509
australia  -0.6398492 -0.59674573 -0.9488347 -0.2446986 0.1858185 -1.5736001  0.1591982 0.2362221  0.5854141  0.3173087  0.35866019
                 [,23]      [,24]       [,25]      [,26]      [,27]       [,28]      [,29]      [,30]       [,31]      [,32]       [,33]
</s>       -1.76637387  6.0573125 -2.09484220  2.9259069 -1.8728924  9.44051647 -0.7307377 -7.9956007 -1.97901595  0.5691963 -5.81520128
government  0.81370312  1.2310206 -0.15019083 -1.0581890 -1.3468195  1.48522151  0.6769124 -1.3793977  0.72235984 -0.2482368  0.40466353
time        0.07228457 -0.2755362 -1.21455157 -0.4813255 -0.3107042  0.71205175 -0.1496595 -1.3523901 -0.01923664 -0.8207752  0.06317991
australian -0.28797609  1.4306754  0.02631173 -1.3079888 -0.4530297  0.86120695  1.1755733 -1.4620591  1.70992470 -0.6210573  0.97262472
win        -1.39676678  0.2823853 -0.59129256  1.5189236  1.5837716 -0.02030919  0.4333407 -1.1566157  0.04268567  1.1619192 -1.23616612
australia   0.60768658  0.6368815  0.80651963 -0.2478358 -0.8671916  0.15858611  0.3395035 -0.8361708  0.06828446 -0.3144013  0.33700201
               [,34]      [,35]      [,36]      [,37]      [,38]       [,39]      [,40]      [,41]      [,42]      [,43]      [,44]
</s>       5.6949091 -0.1020050  2.4784615 -3.6111946  0.9935532 -5.44315720 -7.3085208  6.2620187 -1.2975135 -5.7537794 -2.2436790
government 2.1786807 -0.1329254  0.2329641 -0.7050448  1.6342545 -0.71973747 -1.1432722 -0.8628491 -0.6968162 -2.7394607 -0.4612690
time       0.5265190  0.2161161  0.8847651  0.8934118  0.5158917 -0.07738511 -0.5122415  0.6336743  0.9903618 -0.5083272 -0.2658688
australian 1.0683526 -1.4984027 -1.7566581  0.4927640  0.7207674 -0.91315937 -0.8039417 -0.8253178 -1.6443317 -1.9567682 -0.2923160
win        0.2369788 -0.6156974 -0.5156328  0.2007221 -0.5602611 -1.96811712 -0.7359275  1.3327516  0.2203727  0.5010353  0.9610528
australia  0.3571970 -0.9980337 -1.1288474 -0.8403853  0.8705339 -0.68628591 -0.2189633 -0.8025171 -2.3261845 -1.4722642 -0.3072634
                [,45]      [,46]      [,47]       [,48]        [,49]      [,50]      [,51]      [,52]      [,53]       [,54]       [,55]
</s>        3.4301040 -5.3254848  4.0920897 -1.04821563  4.730687141  0.7196508  0.1088966 -3.7716382  1.2383318  0.35978523  0.33490959
government  0.0190664 -0.7543702 -0.1478701 -0.06606922 -0.004762775 -0.1229698  0.7284969 -0.7458084  1.5849373 -0.90387815  0.51113343
time       -0.7270250 -1.0623294 -0.5150813 -0.67964113  1.129929304  0.1219787  0.6944104 -0.1811112 -0.3811317 -0.62453610  0.05229978
australian  0.7658578 -0.7788154  0.6777939 -0.09089906 -0.839497566 -0.5455454 -1.2113630 -0.7695469  1.2620180 -0.04191819  0.23448397
win         1.8374317 -1.2452400  1.6452774 -0.42070937  0.483905256  0.1156731 -1.3759817  0.2493097  0.4487952  0.64127958 -0.87428135
australia   0.7941430 -0.5859258  0.4849804 -0.87122679 -0.992131650 -0.3923704  0.3246585 -0.5791672  0.3848309  0.21545294 -0.48623168
                [,56]      [,57]      [,58]      [,59]      [,60]       [,61]     [,62]     [,63]       [,64]       [,65]       [,66]
</s>       -4.3080616  3.1367822 4.98980618 6.27792215 -5.3992338 4.324971676 1.9978056 0.3784150 -4.85509014 -2.40531683 -0.97677928
```

Word dictionary:



```r
# load model
# same directory with model <- 'C:/Users/edwin/Desktop\\model' add .bin as extension
model <- load_model('C:/Users/edwin/Desktop/model.bin')

# get word vector from the model
word_vector <- get_word_vectors(model)
head(word_vector)
# get words in the model
word <- get_dictionary(model)
word
# predcit testset
predictions <- predict(model, sentences = test$V2, unlock_empty_predictions = TRUE)

# get the prediction labels
```

51:1 (Top Level)

Console Terminal

C:/Users/edwin/Desktop/New folder/

```
> word
  [1] "</s>"          "government"    "time"          "australian"    "win"           "australia"     "south"
  [8] "police"        "play"          "state"         "fire"          "day"           "work"          "week"
 [15] "back"          "good"          "match"         "game"          "world"         "give"          "cent"
 [22] "area"          "home"          "number"        "service"       "set"           "minister"      "find"
 [29] "month"         "million"       "report"        "start"         "000"           "water"         "final"
 [36] "player"        "queensland"    "today"         "put"           "point"         "court"         "include"
 [43] "road"          "call"          "team"          "open"          "group"         "run"           "man"
 [50] "expect"        "plan"          "company"       "end"           "country"       "leave"         "show"
 [57] "lead"          "federal"       "test"          "community"     "local"         "change"        "high"
 [64] "part"          "lose"          "place"         "national"      "big"           "great"         "move"
 [71] "thing"         "issue"         "health"        "lot"           "side"          "1"             "season"
 [78] "minute"        "continue"      "break"         "sydney"        "council"       "north"         "attack"
 [85] "kill"          "hospital"      "hold"          "flood"         "close"         "10"            "car"
 [92] "public"        "top"           "early"         "night"         "city"          "face"          "club"
 [99] "support"       "injury"        "force"         "return"        "charge"        "decision"      "official"
[106] "coast"         "family"        "news"          "long"          "concern"       "problem"       "centre"
[113] "child"         "international" "party"         "hit"           "head"          "case"          "authority"
[120] "die"           "hour"          "level"         "late"          "member"        "wale"          "claim"
[127] "morning"       "remain"        "west"          "leader"        "woman"         "record"        "business"
[134] "result"        "happen"        "australia's"   "melbourne"     "2"             "john"          "china"
[141] "major"         "house"         "condition"     "cup"           "job"           "increase"      "life"
[148] "election"      "united"        "follow"        "coach"         "park"          "chief"         "victory"
[155] "pay"           "resident"      "death"         "river"         "5"             "past"          "yesterday"
[162] "round"         "system"        "deal"          "line"          "western"       "champion"      "bring"
[169] "bit"           "town"          "release"       "event"         "half"          "opposition"    "act"
[176] "score"         "industry"      "department"    "provide"       "beat"          "ball"          "spokesman"
[183] "reach"         "talk"          "3"             "young"         "20"            "brisbane"      "ago"
[190] "damage"        "league"        "president"     "officer"       "region"        "rise"          "security"
[197] "money"         "add"           "cost"          "medium"        "chance"        "control"       "northern"
[204] "abc"           "meet"          "large"         "premier"       "miss"          "emergency"     "receive"
[211] "goal"          "bad"           "labor"         "earlier"       "power"         "saturday"      "announce"
[218] "cut"           "prime"         "seed"          "rain"          "series"        "hand"          "action"
[225] "begin"         "future"        "friday"        "4"             "turn"          "hard"          "important"
[232] "mark"          "strong"        "drive"         "dr"            "meeting"       "program"       "campaign"
```

For further information, please run the R code, or interview team member.

List of R libraries used

FastText is the library created by Facebook for text classification and embeddings, by creating a vector representation for words. It was open sourced and used neural network for the embeddings. The speed of the library is one of its main features.

Available parameters from fasttext library is as follow:

```
$ ./fasttext supervised
Empty input or output path.

The following arguments are mandatory:
  -input              training file path
  -output             output file path

  The following arguments are optional:
  -verbose            verbosity level [2]

  The following arguments for the dictionary are optional:
  -minCount           minimal number of word occurrences [5]
  -minCountLabel      minimal number of label occurrences [0]
  -wordNgrams         max length of word ngram [1]
  -bucket             number of buckets [2000000]
  -minn               min length of char ngram [3]
  -maxn               max length of char ngram [6]
  -t                  sampling threshold [0.0001]
  -label              labels prefix [__label__]

  The following arguments for training are optional:
  -lr                 learning rate [0.05]
  -lrUpdateRate       change the rate of updates for the learning rate [100]
  -dim                size of word vectors [100]
  -ws                 size of the context window [5]
  -epoch              number of epochs [5]
  -neg                number of negatives sampled [5]
  -loss               loss function {ns, hs, softmax} [ns]
  -thread             number of threads [12]
  -pretrainedVectors  pretrained word vectors for supervised learning []
  -saveOutput         whether output params should be saved [0]
```

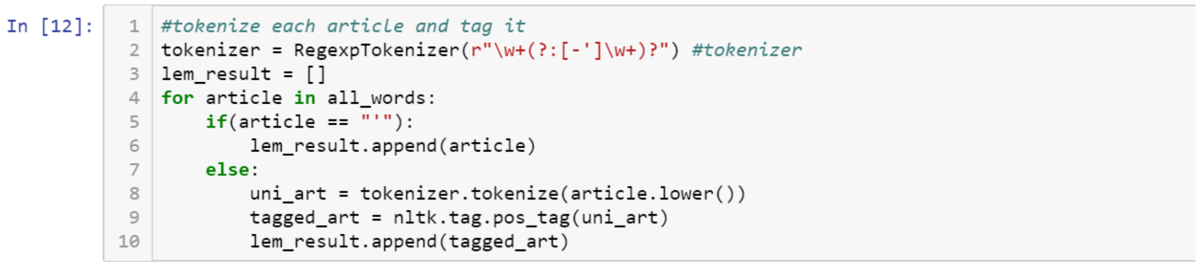
```
The following arguments for quantization are optional:
-cutoff                 number of words and ngrams to retain [0]
-retrain                finetune embeddings if a cutoff is applied [0]
-qnorm                  quantizing the norm separately [0]
-qout                   quantizing the classifier [0]
-dsub                   size of each sub-vector [2]
```

The pre-processing steps

Pre-processing Steps:

After Reading in the file in Python, the following steps are applied to both “training_docs.txt” and “testing_docs.txt”:

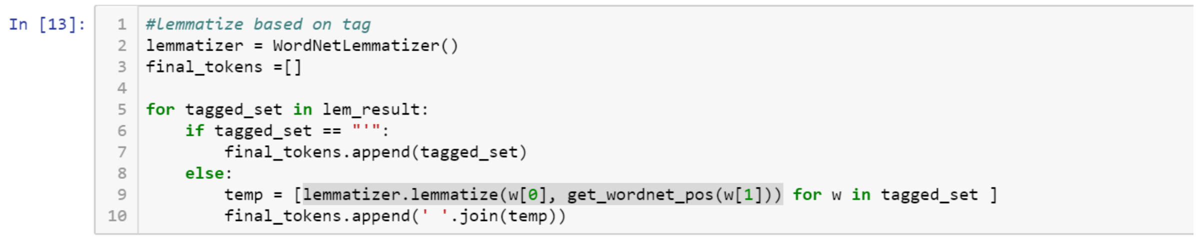
- **Case Normalization:** Since the words in each text has different cases, all the words is normalized to lower case.
- **Tokenization:** In this stage, the text is tokenized based on this regular expression, “\w+(?:[-']\w+)?”. So the special characters are omitted and only the words or numbers are kept. However, there are a few documents in the datasets with only ‘ in the text, we decided to simply keep those ‘ as is.



```
In [12]:
#tokenize each article and tag it
tokenizer = RegexpTokenizer(r"\w+(?:[-']\w+)?") #tokenizer
lem_result = []
for article in all_words:
    if(article == "'"):
        lem_result.append(article)
    else:
        uni_art = tokenizer.tokenize(article.lower())
        tagged_art = nltk.tag.pos_tag(uni_art)
        lem_result.append(tagged_art)
```

Figure 1. Case Normalization and Tokenization

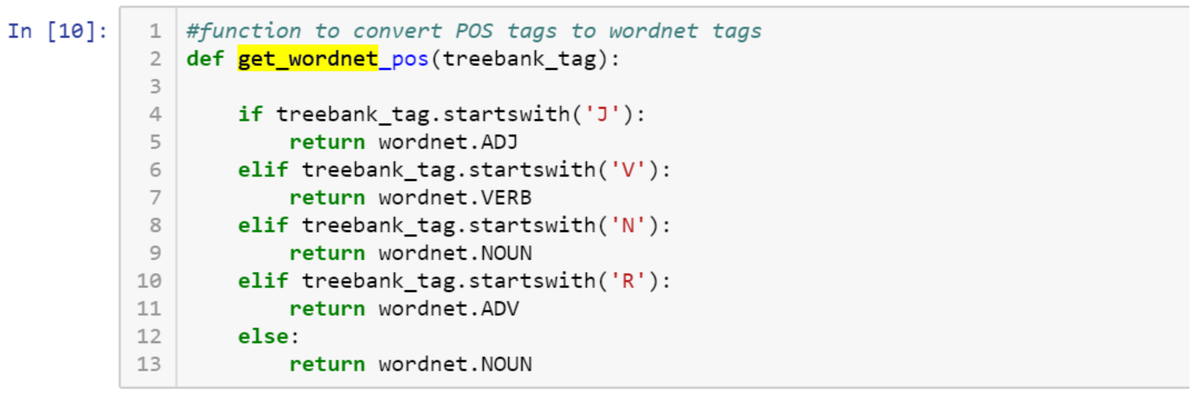
- **Lemmatization:** The tokenized texts are then lemmatized. The lemmatization is based on the POS tags to ensure the operation is done properly.



```
In [13]:
#Lemmatize based on tag
lemmatizer = WordNetLemmatizer()
final_tokens =[]

for tagged_set in lem_result:
    if tagged_set == "'":
        final_tokens.append(tagged_set)
    else:
        temp = [lemmatizer.lemmatize(w[0], get_wordnet_pos(w[1])) for w in tagged_set ]
        final_tokens.append(' '.join(temp))
```

Figure 2. Lemmatization

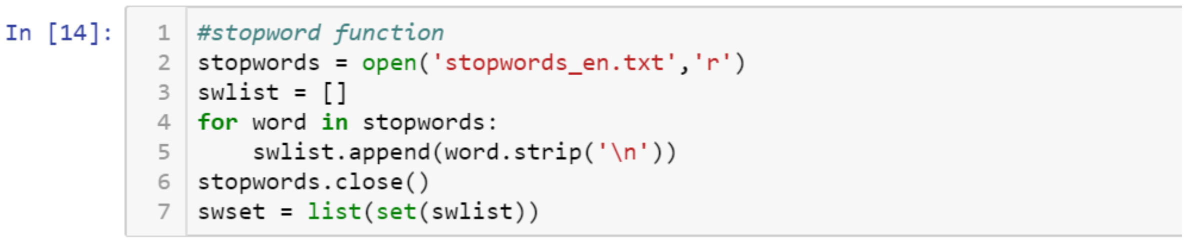


```
In [10]:
#function to convert POS tags to wordnet tags
def get_wordnet_pos(treebank_tag):

    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN
```

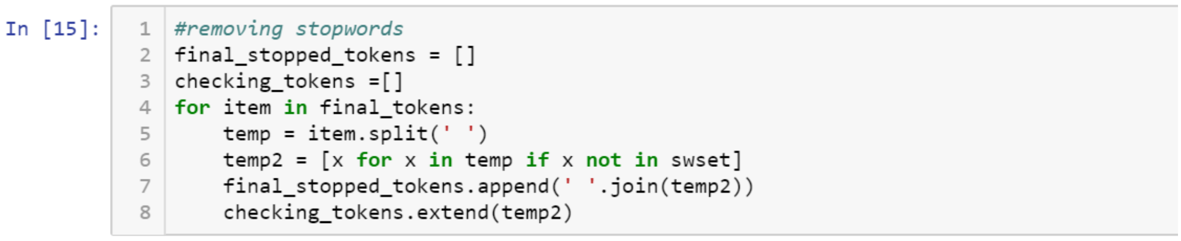
Figure 3. POS-Tagging

- **Stopwords removal:** After the tokenization and lemmatization, all the stop words are removed. We use the list of stopwords, “stopwords_en.txt”, obtained from Kevin Bouge’s website.



```
In [14]:
#stopword function
stopwords = open('stopwords_en.txt','r')
swlist = []
for word in stopwords:
    swlist.append(word.strip('\n'))
stopwords.close()
swset = list(set(swlist))
```

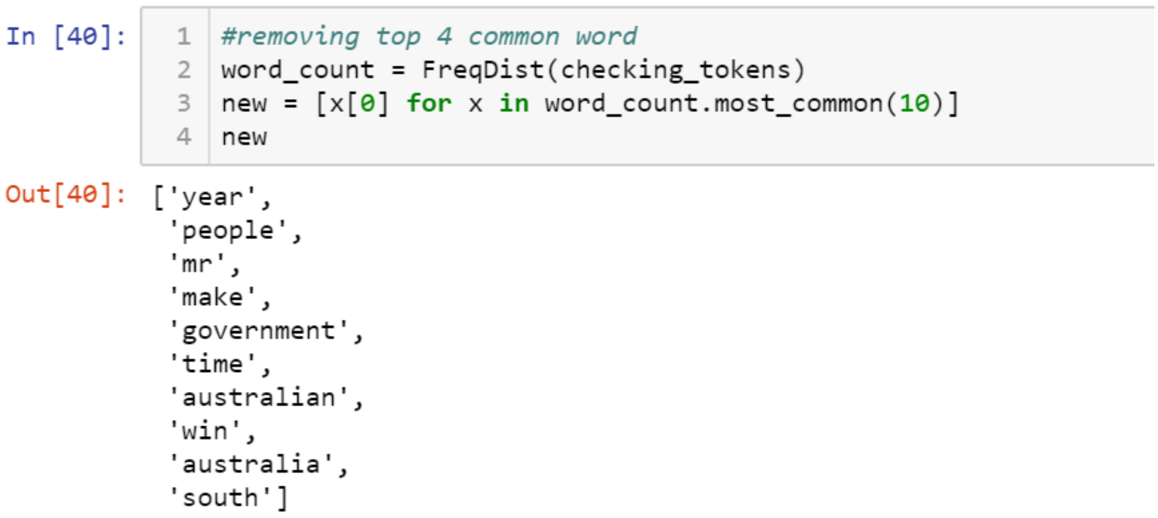
Figure 4. Stopwords



```
In [15]:
#removing stopwords
final_stopped_tokens = []
checking_tokens =[]
for item in final_tokens:
    temp = item.split(' ')
    temp2 = [x for x in temp if x not in swset]
    final_stopped_tokens.append(' '.join(temp2))
    checking_tokens.extend(temp2)
```

Figure 5. Stopwords Removal

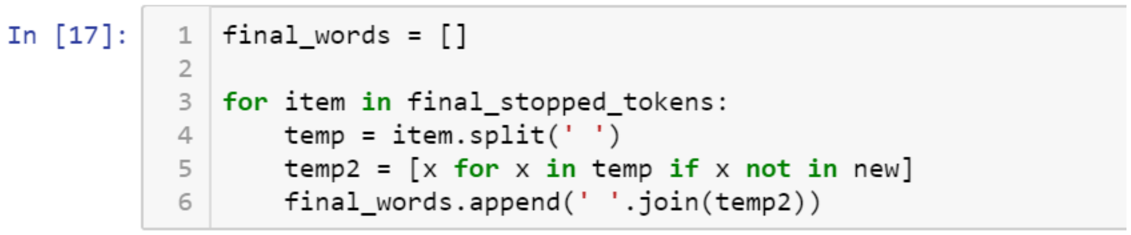
- **Removing the top 4 frequent words:** This step is only applied to the training dataset to train a better algorithm. The frequency of each word is calculated and the 4 most frequently appeared words are removed. Before we removed the top 4 frequent words, we first looked at the top 10 frequent words since those words add up to 5% of the text. When inspecting the words, we decide some of the words can be important in classifying the text. For example, Australia, Australian can give the geography of the news. Government can mean the text is about politics. If the article has win in it, it could mean it's a sport news. Yet, the top 4 words can mean anything and may confused the classifier at times. So, we decided it's best to remove the top 4 words.



```
In [40]:
#removing top 4 common word
word_count = FreqDist(checking_tokens)
new = [x[0] for x in word_count.most_common(10)]
new

Out[40]: ['year',
 'people',
 'mr',
 'make',
 'government',
 'time',
 'australian',
 'win',
 'australia',
 'south']
```

Figure 6. The top 10 frequent words in descending order


```
In [17]:
final_words = []

for item in final_stopped_tokens:
    temp = item.split(' ')
    temp2 = [x for x in temp if x not in new]
    final_words.append(' '.join(temp2))
```

Figure 7. Top 4 frequent words removal

After the above pre-processing on the text, we output csv files for both training and test datasets to use as input.

The selected features and how they generated

In fasttext, for supervise learning the library uses continuous bag of word as the feature selection strategy.

Context is represented by multiple words for a given target words. For example, we could use “cat” and “tree” as context words for “climbed” or “C1” as the target word/label.

Continuous bag of word (CBOW) to predict the next word according to the context, fasttext uses CBOW to input one article of words to predict the nearest label.

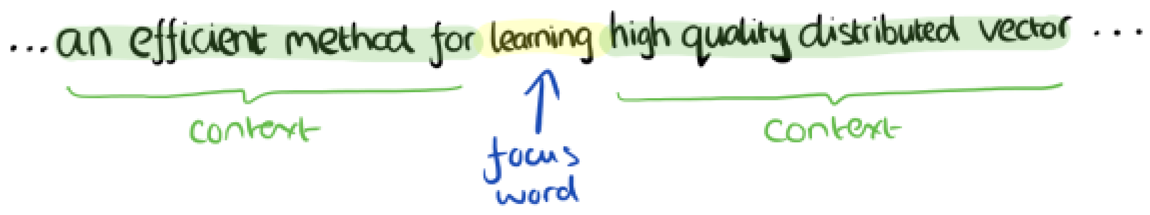

...an efficient method for learning high quality distributed vector ...
context
focus word
context

Figure 8

To predict a label, we will replace the label with focus word in figure 8, and the input context will be one article instead of the word near the prediction.

We will define the word selection by using “context_window_size”, “-ws” in the execute command.

In figure 1, the “-ws” = 4

The features we use here are the nearest windows which contain the word from the near context. The number of features depends on the size of the window.

The methodology used to develop the model/algorithm

In fasttext, the library allows us to use three loss functions to maximise the probability of each label or word. They are heritage tree, SoftMax and negative sampling. However, in this case, theoretically, negative sampling and softmax are applying similar loss function which in negative sampling using ordinary logical regression and SoftMax is multinominal logical regression.

The model of CBOW consist of three layers, input, projection and output layer.

For example:

A given (context(w), w), assume context(w) has c number of words in it front and after it. The word vector of the context word are v(context(w)$_1$) ….. v(context(w)$_{2c}$)

Heritage tree/heritage SoftMax

Input layer

As shown in the figure 9, the input layer a combination of the 2C word's vectors.

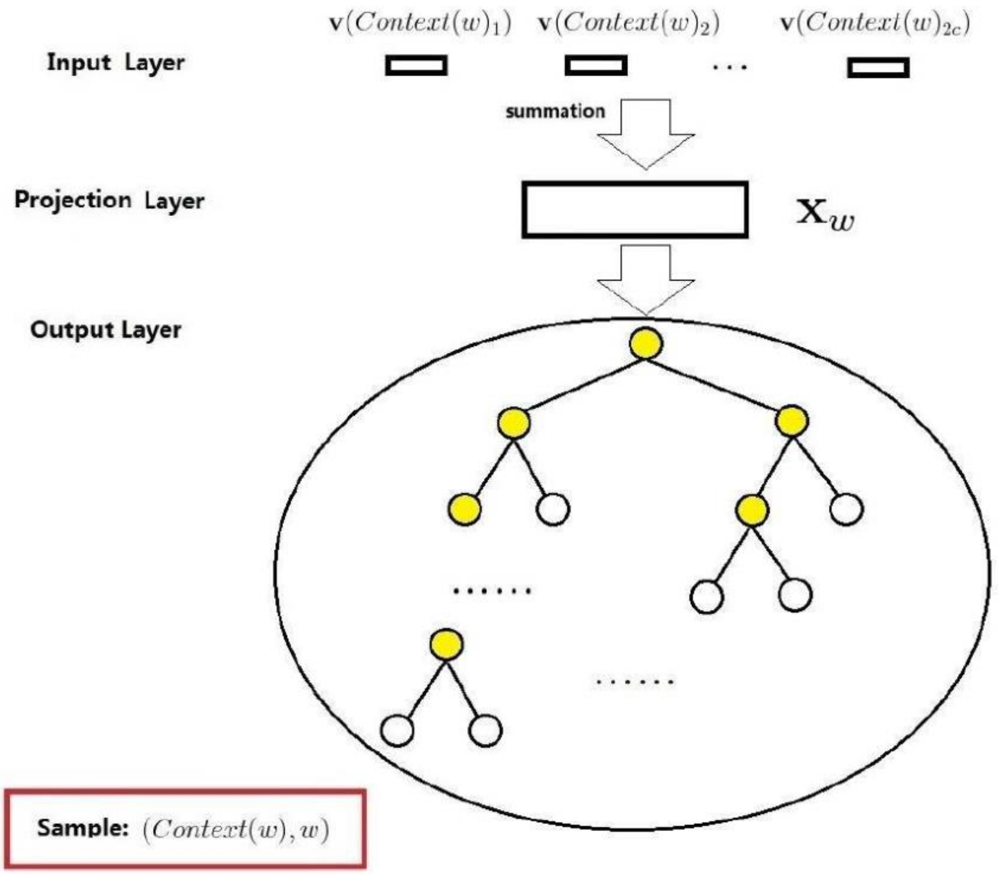

Input Layer
$\mathbf{v}(Context(w)_1)$
$\mathbf{v}(Context(w)_2)$
$\mathbf{v}(Context(w)_{2c})$
summation
Projection Layer
$\mathbf{x}_w$
Output Layer
Sample: $(Context(w), w)$

Figure 9

Projection Layer

As shown in figure 1, the input of the projection layer is the summation of the 2C word's vectors.

$$\mathbf{x}_w = \sum_{i=1}^{2c} \mathbf{v}(Context(w)_i) \in \mathbb{R}^m.$$

m is the length of the word vector.

Output Layer

In figure 1, it is using heritage tree as the loss function to produce the output. It uses the words with the smallest weight at the bottom in the article as the leaf, and each of the yellow knot is the summation of the weight for the two leaf. Eventually, the leaf with the highest weight is consider has the highest relationship with the target word or label will have the shortest path to the predicted word or label.

SoftMax

The only differ between SoftMax and heritage tree/heritage SoftMax is the loss function of the output layer.

Instead of using heritage tree to project the relationship of context and word/label, it uses SoftMax function to find the probability of each contest corresponding to the word/label.

Figure 10 shows the process of using SoftMax loss function.

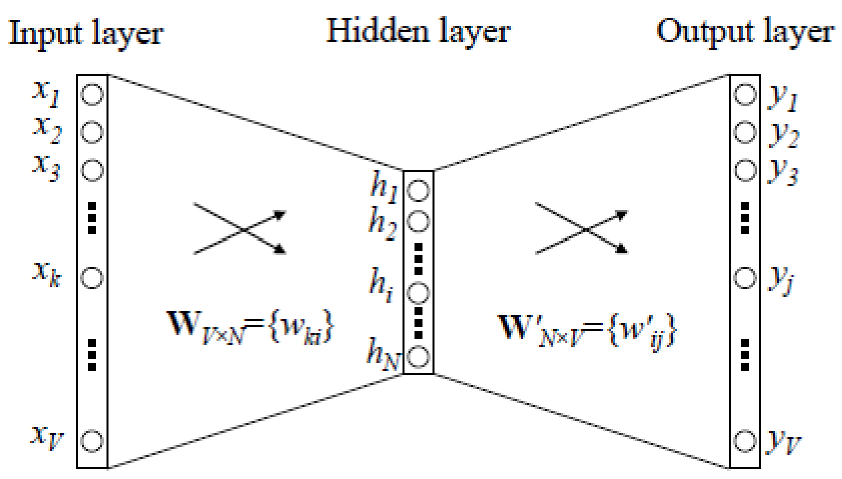


Figure 10

Negative sampling

In negative sample, for CBOW model, if known word(w)'s or label w context of context(x), we will predict 'w', therefore, for given context(x), word 'w' is a positive sample, other words are negative samples.

Not if we had a negative sample for w NEG(w)

$$L^w(\widetilde{w}) = \begin{cases} 1, & \widetilde{w} = w; \\ 0, & \widetilde{w} \neq w, \end{cases}$$

Positive sample label=1, negative sample, label=0

For a given positive sample (context(w), w), we want to maximize

$$g(w) = \prod_{u \in \{w\} \cup NEG(w)} p(u|Context(w)),$$

And

$$p(u|Context(w)) = \begin{cases} \sigma(\mathbf{x}_w^\top \theta^u), & L^w(u) = 1; \\ 1 - \sigma(\mathbf{x}_w^\top \theta^u), & L^w(u) = 0, \end{cases}$$

X_w is the sum of context(w), $\theta^u \in \mathbb{R}^m$ is the represented word u assisting vector for training.

$$g(w) = \sigma(\mathbf{x}_w^\top \theta^w) \prod_{u \in NEG(w)} \left[1 - \sigma(\mathbf{x}_w^\top \theta^u)\right],$$

$\sigma(\mathbf{x}_w^\top \theta^w)$ is the probability of w given context(w), $\sigma(\mathbf{x}_w^\top \theta^u),\ u \in NEG(w)$ is the probability of u given context(w), therefore, maximize g(w) is maximizing $\sigma(\mathbf{x}_w^\top \theta^w)$ and minimizing $\sigma(\mathbf{x}_w^\top \theta^u),\ u \in$ (maximize the positive sample, and minimize negative sample)

Therefore, for given article c $G = \prod_{w \in \mathcal{C}} g(w)$

A description of the model/algorithm used for learning

The learning algorithm to optimize all the loss function above is gradient decent.

Heritage tree/heritage SoftMax

According to the heritage tree, each leaf (word) being correctly placed is a logistic regression.

Therefore, we can use the following to represent the probability that the leaf is properly places.

$$\sigma(\mathbf{x}_w^\top \theta) = \frac{1}{1+e^{-\mathbf{x}_w^\top \theta}},$$

Not being properly placed. $1 - \sigma(\mathbf{x}_w^\top \theta),$

Where d^w_j is the distant path of Huffman tree for the word w, θ^w_{j-1} is the path of w for non-knot vector.

The conditional probability can be written as follow:

$$p(w|Context(w)) = \prod_{j=2}^{l^w} p(d_j^w|\mathbf{x}_w, \theta_{j-1}^w),$$

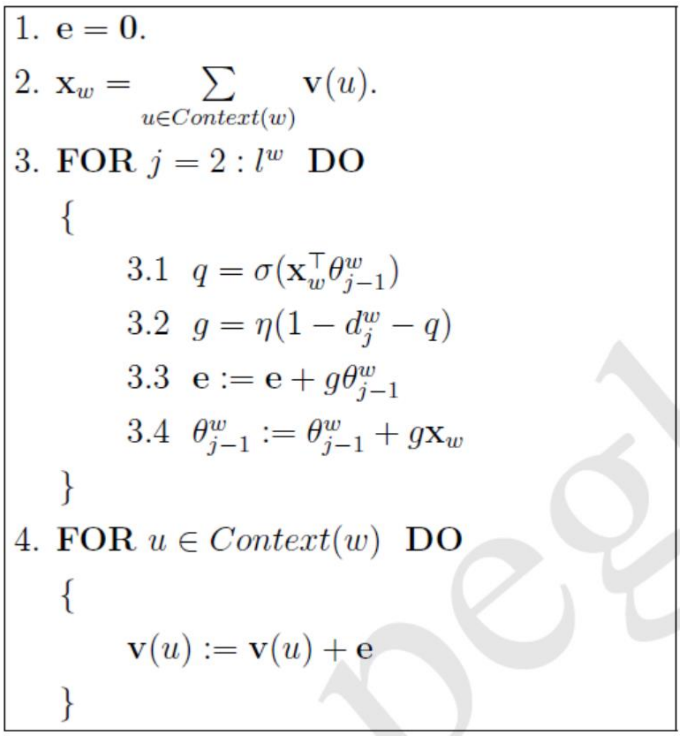
$$p(d_j^w|\mathbf{x}_w, \theta_{j-1}^w) = [\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{1-d_j^w} \cdot [1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{d_j^w}.$$

Put it into log likelihood function:

$$\begin{aligned}\mathcal{L} &= \sum_{w\in\mathcal{C}} \log \prod_{j=2}^{l^w} \left\{ [\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{1-d_j^w} \cdot [1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{d_j^w} \right\} \\ &= \sum_{w\in\mathcal{C}} \sum_{j=2}^{l^w} \left\{ (1-d_j^w)\cdot \log[\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] + d_j^w \cdot \log[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] \right\},\end{aligned}$$

And we will be using gradient decent to optimise the loss function to find the optimal place for the word (w).

Below is the algorithm for gradient decent for this loss function.



```
1. $\mathbf{e} = \mathbf{0}$.
2. $\mathbf{x}_w = \sum_{u \in Context(w)} \mathbf{v}(u)$.
3. FOR $j = 2 : l^w$ DO
   {
       3.1 $q = \sigma(\mathbf{x}_w^\top \theta_{j-1}^w)$
       3.2 $g = \eta(1 - d_j^w - q)$
       3.3 $\mathbf{e} := \mathbf{e} + g\theta_{j-1}^w$
       3.4 $\theta_{j-1}^w := \theta_{j-1}^w + g\mathbf{x}_w$
   }
4. FOR $u \in Context(w)$ DO
   {
       $\mathbf{v}(u) := \mathbf{v}(u) + \mathbf{e}$
   }
```

Figure 11

SoftMax

For SoftMax, we will use the cost function as below:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Where the probability of each word (z) /label belongs to the related context. And use gradient decent to optimize the probability for

Negative sampling

According to above, the loss function for negative sampling is written as follow:

$$\begin{aligned}
\mathcal{L} &= \log G = \log \prod_{w \in \mathcal{C}} g(w) = \sum_{w \in \mathcal{C}} \log g(w) \\
&= \sum_{w \in \mathcal{C}} \log \prod_{u \in \{w\} \cup NEG(w)} \left\{ \left[\sigma(\mathbf{x}_w^\top \theta^u)\right]^{L^w(u)} \cdot \left[1 - \sigma(\mathbf{x}_w^\top \theta^u)\right]^{1 - L^w(u)} \right\} \\
&= \sum_{w \in \mathcal{C}} \sum_{u \in \{w\} \cup NEG(w)} \left\{ L^w(u) \cdot \log\left[\sigma(\mathbf{x}_w^\top \theta^u)\right] + \left[1 - L^w(u)\right] \cdot \log\left[1 - \sigma(\mathbf{x}_w^\top \theta^u)\right] \right\}.
\end{aligned}$$

Then we will use gradient decent to optimize the loss function. To find the probability of the set of word with the related label.

A general demonstration for fasttext in supervise learning problem shown below:

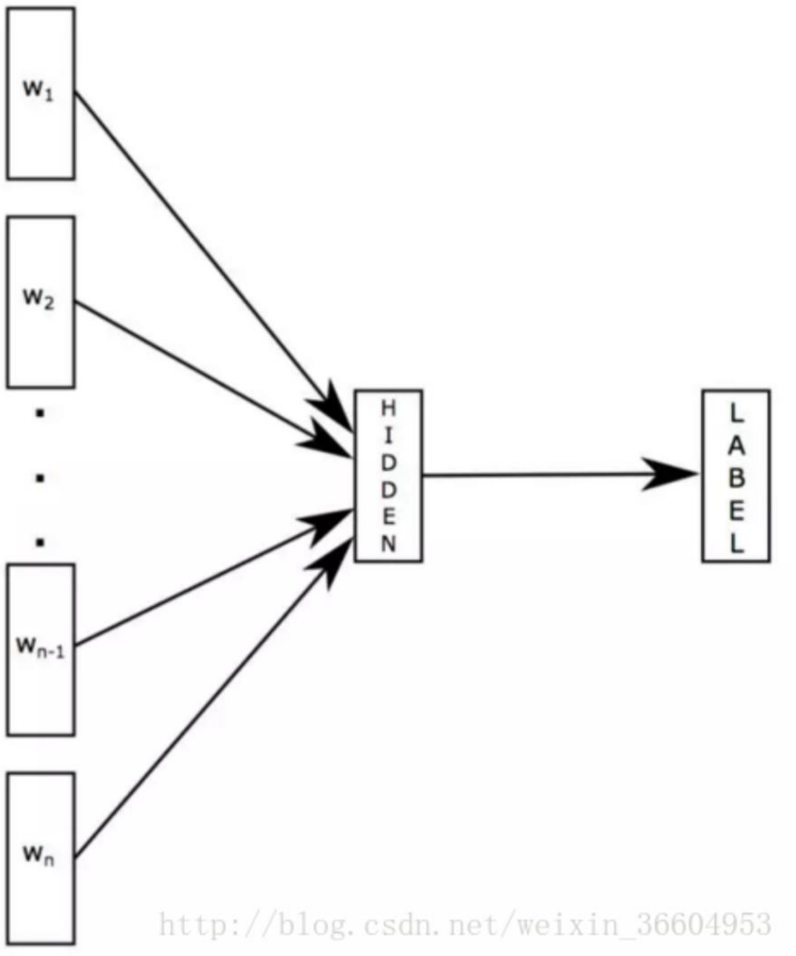


Figure 12

References

fastText 源码分析 - Helei's Tech Notes. (n.d.). Retrieved from https://heleifz.github.io/14732610572844.html

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2017). Bag of Tricks for Efficient Text Classification. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers* . doi:10.18653/v1/e17-2068

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2017). Bag of Tricks for Efficient Text Classification. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers* . doi:10.18653/v1/e17-2068

Learning word embeddings - Natural Language Processing & Word Embeddings | Coursera. (n.d.). Retrieved from https://www.coursera.org/lecture/nlp-sequence-models/learning-word-embeddings-APM5s

List of options · fastText. (n.d.). Retrieved from https://fasttext.cc/docs/en/options.html

Word2Vec Sentiment Classification with R and H2O. (n.d.). Retrieved from https://stackoverflow.com/questions/30901595/word2vec-sentiment-classification-with-r-and-h2o

word2vec 中的数学原理详解 - peghoty - 博客园. (n.d.). Retrieved from http://www.cnblogs.com/peghoty/p/3857839.html

word2vec 原理(二) 基于 Hierarchical Softmax 的模型 - 刘建平 Pinard - 博客园. (n.d.). Retrieved from http://www.cnblogs.com/pinard/p/7243513.html