1. Introduction of Word2Vec
2. Introduction of CBOW Model
3. Introduction of Skip gram Model
4. Introduction of SoftMax function
5. Mathematical derivation of CBOW Model
6. Mathematical derivation of Skip gram Model
7. Case Study: Movie Review Sentiment analysis using word2vec

More detail @ https://www.youtube.com/ShashiSAS

Word2Vec

Word2Vec is one of the most popular technique to learn word embeddings using shallow neural network. It was developed by Tomas Mikolov in 2013 at Google.

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

Word2vec represents words in vector space representation. Words are represented in the form of vectors and placement is done in such a way that similar meaning words appear together and dissimilar words are located far away. This is also termed as a semantic relationship. Neural networks do not understand text instead they understand only numbers. Word Embedding provides a way to convert text to a numeric vector.

Male-Female

Verb tense

Why do we need them?

Consider the following similar sentences:

1. *Have a good day*
2. *Have a great day.*

They hardly have different meaning. If we construct an exhaustive vocabulary (let's call it V), it would have V = {Have, a, good, great, day}.

More detail @ https://www.youtube.com/ShashiSAS

Now, let us create a one-hot encoded vector for each of these words in V. Length of our one-hot encoded vector would be equal to the size of V (=5). We would have a vector of zeros except for the element at the index representing the corresponding word in the vocabulary. That particular element would be one. The encodings below would explain this better.

Have = [1,0,0,0,0];a=[0,1,0,0,0] ; good=[0,0,1,0,0] ;great=[0,0,0,1,0] ; day=[0,0,0,0,1]

If we try to visualize these encodings, we can think of a 5-dimensional space, where each word occupies one of the dimensions and has nothing to do with the rest (no projection along the other dimensions). This means 'good' and 'great' are as different as 'day' and 'have', which is not true.

Our objective is to have words with similar context occupy close spatial positions. Mathematically, the cosine of the angle between such vectors should be close to 1, i.e. angle close to 0.

Word2Vec is a method to construct such an embedding. It can be obtained using two methods:

1. Common Bag of Words (CBOW)
2. Skip Gram

***CBOW Model**:* This method takes the context of each word as the input and tries to predict the word corresponding to the context. Consider our example: *Have a great day*.

Let the input to the Neural Network be the word, great. Notice that here we are trying to predict a target word (day) using a single context input word great. More specifically, we use the one hot encoding of the input word and measure the output error compared to one hot encoding of the target word (day). In the process of predicting the target word, we learn the vector representation of the target word.

Let us look deeper into the actual architecture.

Skip-Gram: The input is the target word, while the outputs are the words surrounding the target words. It attempts to predict the immediate neighbours of a given word. It looks one word ahead and one behind, or two ahead, or some other variation. We use the word we want to model as our input X, and the surrounding words as target output Y.

Let us look deeper into the actual architecture.

SoftMax function: -

Softmax function also known as soft argmax or normalized exponential function. This is a function that takes as input a vector of K real number and normalize into probability distribution consisting K probability proportional to the exponentials of the input number.

1. Prior to applying softmax, some vector components could be negative, or greater than one.
2. Negative input will be converted into non-negative values.
3. Each output will be in the interval (0,1).
4. As the denominator in each Softmax computation is the same,the value become proportional to each other, this normalization sure that together they sum to 1.

```
>>> import numpy as np
>>> a = [1.0, 2.0, 3.0, 4.0, 1.0, 2.0, 3.0]
>>> np.exp(a) / np.sum(np.exp(a))
array([0.02364054, 0.06426166, 0.1746813, 0.474833, 0.02364054,
       0.06426166, 0.1746813])
```

Continuous bag of words (CBOW)

(Simple CBOW model with one word (C=1) in the Context)

weight of neuron(w) between Input Layer & hidden layer

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}_{V\times N}$$

weight of neuron(w') between hidden & output Layer.

$$W' = \begin{bmatrix} w'_{11} & w'_{12} & w'_{13} & w'_{14} \\ w'_{21} & w'_{22} & w'_{23} & w'_{24} \\ w'_{31} & w'_{32} & w'_{33} & w'_{34} \end{bmatrix}_{N\times V}$$

Model

→ In this model 'X' represent the one-hot encoded vector corresponding to the input word in the training instance. $\{X_1, \dots X_c\}$ for a word window of size C

→ $\{Y_1 \dots Y_c\}$ are the probability distribution vector

→ The $(V \times N)$ matrix 'W' is the weight between the input layer and hidden layer whose ith row represents the weights corresponding to the ith word in the vocabulary.

Each neuron should active only for particular type of inputs and not all inputs.

Therefore, by propogation forward we see how how well our model is behaving and find the error. If we find out our model has error, we back propogate and use a form of gradient descent to update new values of the weights (w, & w'). then we will again forward propagate to see how well those weights (w, w') are performing and then will backward propagate to update the weight. This will go on untill we reach some minima error value and optimal weights.

forward propogation :- In neural networks, forward propagate to get the output and compare it with the real value to get the error

Backward propogation :- To minimize the error, we propagate backwards by finding derivative of error with respect to each weights and then subtracting this value from the weight value.

After getting optimal weight of neuron to we predict the target word. output will provide the probability distribution scores correspondance to the word.

"This boy is ____."

Forward Propagation: Weight calculation (w) :- Input Layer to hidden Layer

Assume

$V = 4$

$N = 3$

$C = 1$

$$h_1 = w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4$$

$$h_2 = w_{12}x_2 + w_{22}x_2 + w_{32}x_3 + w_{42}x_4$$

$$h_3 = w_{13}x_3 + w_{23}x_2 + w_{33}x_3 + w_{43}x_4$$

$$\underset{N=3}{\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}} = \underset{N \times N}{\begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \end{bmatrix}} \underset{V \times C}{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}$$

4×1

$$\boxed{h = w^T x} \qquad (1)$$

Forward Calculation: Weight Calculation (w'): Hidden to O/P Layer.

Output Layer

$$u_1 = w'_{11} h_1 + w'_{21} h_2 + w'_{31} h_3$$

$$u_2 = w'_{12} h_1 + w'_{22} h_2 + w'_{32} h_3$$

$$u_3 = w'_{13} h_1 + w'_{23} h_2 + w'_{33} h_3$$

$$u_4 = w'_{14} h_1 + w'_{24} h_2 + w'_{34} h_4$$

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} w'_{11} & w'_{21} & w'_{31} \\ w'_{12} & w'_{22} & w'_{32} \\ w'_{13} & w'_{23} & w'_{33} \\ w'_{14} & w'_{24} & w'_{34} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

$$\boxed{u = w'^T h} \longrightarrow (2)$$

More detail @ https://www.youtube.com/ShashiSAS

output layer

Softmax function:

$$f(x_i) = \frac{e^{x_j}}{\sum_{i=1}^{V} e^{x_i}}$$

$$Y_1 = \frac{e^{u_1}}{e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4}}$$

$$Y_2 = \frac{e^{u_2}}{e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4}}$$

The softmax output for the word w_j with respect to the given input context word w_I can be given by conditional probability, in terms of exponentials, i.e.

$$\boxed{P(w_j / w_I) = Y_j = \frac{e^{u_j}}{\sum_{j=1}^{V} e^{u_j}}} \quad \text{— (3)}$$

Error Calculation

Assumption:- Let 'w_o' is the actual output word, w_I is the given input context words and 'V' is the size of the input context.

The objective is to maximize the conditional probability of observing the actual output word 'w_o', given the input context words 'w_I', thus the loss function can be defined as:

$$\max P(w_O/w_I) = \max(y_{j^*}) = \max(\log(y_{j^*}))$$

where, j^* is the ~~actual~~ index of the actual output word in the output layer.

Example:- If 4th word represent the o/p

$$E(y_4) = \log P(w_{y4}/w_I) = \log_e\left(\frac{e^{u_4}}{e^{u_1}+e^{u_2}+e^{u_3}+e^{u_4}}\right)$$

$$E(y_4) = u_4 - \log(e^{u_1}+e^{u_2}+e^{u_3}+e^{u_4})$$

As, we want to minimize the error, so.

$$E = -\log P(w_O/w_I)$$

$$E(y_4) = \log(e^{u_1}+e^{u_2}+e^{u_3}+e^{u_4}) - u_4$$

we can generalize it as:-

$$\boxed{E = \log\sum_{j=1}^{V} e^{u_j} - u_{j^*}} \quad \text{--- (4)}$$

It is the special case of cross-entropy measurement b/w two probabilistic distribution u_j and u_{j^*}

taking derivative E w.r.to u_4, we have

$$\frac{dE(y_4)}{d(u_4)} = \frac{e^{u_4}}{e^{u_1}+e^{u_2}+e^{u_3}+e^{u_4}} - \frac{d(u_4)}{d(u_4)}$$

$$\boxed{\frac{d\log x}{dx} = \frac{1}{x} \qquad \frac{de^x}{dx} = e^x}$$

$$\frac{dE(y_4)}{d(u_4)} = y_4 - 1$$

we can generalize the above derivation.

$$\boxed{\frac{dE}{du_j} = \frac{d(\log\sum_{j=1}^{V} e^{u_j} - u_{j^*})}{du_j} = y_j - t_j = e_j} \quad \text{--- (5)}$$

Note:- $t_j = 1$, if $(j = j^*)$; else $t_j = 0$

Back-Propagation: output to hidden layer (update ω')

In this back-propagation we, update the weight of all neurons from hidden layer to output layer.

Step 1: Taking gradient 'E' w.r.t; ω'_{11}

$$\frac{dE(Y_1)}{d\omega'_{11}} = \frac{dE(Y_1)}{du_1} \times \frac{du_1}{d\omega'_{11}}$$

$$\frac{dE(Y_1)}{du_1} = (Y_1 - 0) = e_1 \Rightarrow \text{from eqn (5)}$$

from eqn (2)

$$\frac{du_1}{d\omega'_{11}} = \frac{d(\omega'_{11}h_1 + \omega'_{21}h_2 + \omega'_{31}h_3)}{d\omega'_{11}} = h_1 \quad \left[u_1 = \omega'_{11}h_1 + \omega'_{21}h_2 + \omega'_{31}h_3\right]$$

$$\therefore \frac{dE(Y_1)}{d\omega'_{11}} = \frac{dE(Y_1)}{du_1} \times \frac{du_1}{d\omega'_{11}}$$

$$\boxed{\frac{dE(Y_1)}{d\omega'_{11}} = e_1 \cdot h_1} \quad \text{——— (6)}$$

Step 2: Updating the weight of ω'_{11}

$$(\omega'_{11})^{new} = \omega'_{11} - \eta \frac{dE(Y_1)}{d\omega'_{11}} = \omega'_{11} - \eta(e_1 \cdot h_1)$$

Similarly, we can update the weight of $\omega'_{12}, \omega'_{13}, \ldots \omega'_{34}$

$$\boxed{(\omega'_{ij})^{new} = (\omega_{ij})^{old} - \eta\, e_j \cdot h_i} \quad \text{——— (7)}$$

where, η = learning rate

$e_j = Y_j - t_j$

More detail @ https://www.youtube.com/ShashiSAS

Back-Propagation: Hidden to Input layer (update ω)

In this back propagation we, update the weight of all neurons from hidden layer to output layer.

Step 1: Taking gradient 'E' w.r.t, ω_{11}

$$\frac{dE}{d\omega_{11}} = \frac{dE}{dh_1} \times \frac{dh_1}{d\omega_{11}}$$

$$\frac{dE}{dh_1} = \left(\frac{dE}{du_1} \times \frac{du_1}{dh_1}\right) + \left(\frac{dE}{du_2} \times \frac{du_2}{dh_1}\right) + \left(\frac{dE}{du_3} \times \frac{du_3}{dh_1}\right) + \left(\frac{dE}{du_4} \times \frac{du_4}{dh_1}\right)$$

$$\frac{dE}{dh_1} = (e_1 \cdot \omega'_{11}) + (e_2 \omega'_{12}) + (e_3 \omega'_{13}) + (e_4 \omega'_{14})$$

$$= \sum_{i=1}^{4} e_j \omega'_{ij} = EH_i \quad \text{— (9)}$$

$$\frac{dh_1}{d\omega_{11}} = \frac{d(\omega_{11} x_1 + \omega_{21} x_2 + \omega_{31} x_3 + \omega_{41} x_4)}{d\omega_{11}} = x_1 \quad \text{— (6)}$$

$$\frac{\partial E}{\partial \omega_{11}} = \frac{dE}{dh_1} \times \frac{dh_1}{d\omega_{11}}$$

$$= \sum_{i=1}^{4} e_j \omega'_{11} \cdot x_1 = EH_1 \cdot x_1$$

$$\boxed{\therefore \frac{\partial E}{\partial \omega_{ki}} = EH_i \cdot x_k} \quad \text{— (7)}$$

Step 2: Updating the weight of ω_{11}

$$(\omega_{11})^{new} = (\omega_{11})^{old} - \eta \frac{dE}{d\omega_{11}} = (\omega_{11})^{old} - \eta EH_1 x_1$$

$$\boxed{(\omega_{ij})^{new} = (\omega_{ij})^{old} - \eta EH_i \cdot x_i} \quad \text{— (8)}$$

where,

η = learning rate

$EH_i = \sum_{i=1}^{4} e_j \omega'_{ij}$

CBOW Model

CBOW model with only one word in the Context and multiple Context

More detail @ https://www.youtube.com/ShashiSAS

Skip-Gram Model

More detail @ https://www.youtube.com/ShashiSAS

The input of the skip-gram model is a single word w_I and the output is the words in w_I's context $\{w_{o_1}, w_{o_2}, \ldots w_{o_c}\}$ defined by a word window of size c.

for example:-

Consider the sentence "I drove my car to the store".

O/P ← Input word → O/P

? ? ? car ? ? ? ⇒ find nearest word of i/p word.

A Potential training instance could be the word "car" as an input and the words {"I", "drove", "my", "to", "the", "store"} as outputs.

All of these words are one-hot encoded meaning they are vector of length V (the size of the vocabulary) with a value 1 at the index ~~corr~~ corresponding to the word and zero (0) in all other indexes.

In this model X represents the one-hot encoded vector corresponding to the input word in the training instance and $\{Y_1, Y_2 \ldots Y_c\}$ are the one-hot encoded vectors corresponding to the output words in the training instance. The $V \times N$ matrix W is the weight matrix between the input layer and hidden layer whose i^{th} row represents the weights corresponding to the ith word in the vocabulary. Each output word vector also has an associated $N \times V$ out matrix W'. There is also a hidden layer consisting of N nodes.

Weight b/w Input & hidden layer

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}$$

Weight b/w hidden & O/P layer.

$$W' = \begin{bmatrix} w'_{11} & w'_{12} & w'_{13} & w'_{14} \\ w'_{21} & w'_{22} & w'_{23} & w'_{24} \\ w'_{31} & w'_{32} & w'_{33} & w'_{34} \end{bmatrix}$$

Forward Propagation: weight calculation

$$\xrightarrow{w'^T h} \boxed{u} \longrightarrow \boxed{\text{softmax}} \longrightarrow y$$

Input Layer to hidden Layer

$$h_1 = w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4$$
$$h_2 = w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + w_{42}x_4$$
$$h_3 = w_{13}x_1 + w_{23}x_2 + w_{33}x_3 + w_{43}x_4$$

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$\boxed{h = w^T x} \quad \text{—— (1)}$$

Hidden Layer to output Layer

$$u_{11} = w'_{11}h_1 + w'_{21}h_2 + w'_{31}h_3$$
$$u_{12} = w'_{12}h_1 + w'_{22}h_2 + w'_{32}h_3$$
$$u_{13} = w'_{13}h_1 + w'_{23}h_2 + w'_{33}h_3$$
$$u_{14} = w'_{14}h_1 + w'_{24}h_2 + w'_{34}h_3$$

$$\begin{bmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{14} \end{bmatrix} = \begin{bmatrix} w'_{11} & w'_{21} & w'_{31} \\ w'_{12} & w'_{22} & w'_{32} \\ w'_{13} & w'_{23} & w'_{33} \\ w'_{14} & w'_{24} & w'_{34} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

$$\boxed{u = w'^T h} \quad \text{—— (2)}$$

$$\xrightarrow{w'^T \cdot h} \boxed{\Sigma} \longrightarrow u$$

Forward Propagation: Calculating softmax output

$$u \longrightarrow \boxed{\text{Softmax} \quad \frac{e^{u_j}}{\sum^{V} e^{u_i}}} \longrightarrow y$$

$$y_{11} = \frac{e^{u_{11}}}{e^{u_{11}} + e^{u_{12}} + e^{u_{13}} + e^{u_{14}}}$$

$$y_{12} = \frac{e^{u_{12}}}{e^{u_{11}} + e^{u_{12}} + e^{u_{13}} + e^{u_{14}}}$$

The softmax output for the word w_j with respect to the given output word w_I can be given by conditional probability, in terms of exponentials, i.e

$$P(w_j / w_I) = y_j = \frac{e^{u_j}}{\sum_{j=1}^{V} e^{u_j}} \quad \text{—— (3)}$$

Error Calculation

Assumption: Let, $\{w_{1j}, w_{2j}, w_{3j} \dots w_{cj}\}$ are actual output words, for the given single input word w_I. where, w_{1j} is the jth word of 1st context window,

The Training objective is to maximize the conditional probability of observing the actual output words, $\{w_{1j}, w_{2j}, w_{3j} \dots w_{cj}\}$ given the single input context word w_I. Additionally, we want to minimize the error, so the loss function can be defined by conditional probability, in term of exponentials, i.e

$$E = -\log P(w_{1j}, w_{2j}, \dots w_{cj} \mid w_I) = -\log \prod_{c=1}^{C} \left[\frac{e^{u_{cj}}}{e^{u_{11}} + e^{u_{12}} + \dots + e^{u_{1J}}} \right]$$

$$E = -\log \prod_{c=1}^{C} \left[\frac{e^{u_{cj}}}{\sum_{J=1}^{J} e^{u_{1J}}} \right]$$

$$\boxed{E = C \cdot \log \sum_{j=1}^{J} e^{u_J} - \sum_{c=1}^{C} u_{cJ^*}} \quad \text{--- (4)}$$

where, C = total context window

J = vocabulary size

Now, taking derivative, w.r.t,

$$\frac{dE}{du_{cj}} = \frac{e^{u_{cj}}}{\sum_{j=1}^{J} e^{u_{cJ}}} - Z_{cj} = Y_{cj} - Z_{cj} \quad \text{--- (5)}$$

where,

Y_{cj} = out score of jth word of cth context window

$Z_{cj} = 1$, if the jth word of cth context window is the actual output word, otherwise, we put $Z_{cj} = 0$

Back-Propagation : Output to Hidden Layer (update w')

In this back-propagation phase we, update the weight of all neurons from hidden layer to output layer. ($w'_{11}, w'_{12} \ldots$)

Step 1: Taking gradient of E' w.r.t, w'_{11}

$$\frac{dE(Y_1)}{dw'_{11}} = \sum_{c=1}^{K} \frac{dE(Y_{11})}{du_{c1}} \times \frac{du_{c1}}{dw'_{11}}$$

from eq^n (5)

$$\sum_{c=1}^{c} \frac{dE(Y_1)}{du_{c1}} = \sum_{c=1}^{c} (Y_{c1} - \bar{Z}_{c1}) = e_1$$

$$\sum_{i=1}^{c} \frac{du_{c1}}{dw'_{11}} = \sum_{c=1}^{c} \frac{d(w'_{11} h_1 + w'_{21} h_2 + w'_{31} h_3)}{dw_{11}} = h_1$$

$$\boxed{\therefore \frac{dE(Y_1)}{dw'_{11}} = \frac{dE(Y_1)}{du_1} \times \frac{du_1}{dw'_{11}} = e_1 h_1} \quad \text{—— (6)}$$

Step 2: Updating the weight of w'_{11}

$$(w'_{11})^{new} = (w'_{11})^{old} - \eta \frac{dE(Y_1)}{dw'_{11}}$$

$$\left[\frac{dE}{dw'_{ij}} = \sum_{c=1}^{c} \frac{dE}{du_{cj}} \times \frac{du_{cj}}{dw'_{ij}} = \sum_{c=1}^{c} (Y_{cj} - Z_{cj}) \cdot h_i \right]$$

$$(w'_{11})^{new} = (w'_{11})^{old} - \eta (e_1 \cdot h_1)$$

$$\cancel{(w'_{ij})^{new} = (w_{ij})^{old} - \eta (e_j h_i)} \quad \text{—— (7)}$$

$$\boxed{(w'_{ij})^{new} = (w'_{ij})^{old} - \eta \sum_{c=1}^{c} (Y_{cj} - Z_{cj}) \cdot h_i} \quad \text{—— (7)}$$

Back-Propagation: Hidden to Input Layer (update w)

In this back-propagation phase we update the weight of all neuron from hidden layer to output layer.

Step 1: Taking gradient of 'E' w.r.to h_1

$$\frac{dE}{dh_1} = \sum_{j=1}^{Y} \frac{dE}{du_{c1}} \times \frac{du_{c1}}{dh_1} = \sum_{j=1}^{J} \sum_{c=1}^{C} (Y_{cj} - Z_{cj}) \cdot \omega'_{1j}$$

Step 2: Calculating gradient of 'E', w.r.to. ω_{11}

$$\frac{dE}{d\omega_{11}} = \frac{dE}{dh_1} \times \frac{dh_1}{d\omega_{11}} = \sum_{j=1}^{Y} \sum_{c=1}^{C} (Y_{cj} - Z_{cj}) \cdot \omega'_{1j} \times (X_1)$$

Step 3: Updating the weight of ω_{11}

$$(\omega_{11})^{new} = \omega_{11} - \eta \frac{dE}{d\omega_{11}}$$

$$= \omega_{11} - \eta \left(\sum_{j=1}^{J} \sum_{c=1}^{C} (Y_{cj} - Z_{cj}) \cdot \omega'_{1j} \times (X_1) \right)$$

$$\boxed{(\omega_{ij})^{new} = (\omega_{ij})^{old} - \eta \left(\sum_{j=1}^{J} \sum_{c=1}^{C} (Y_{cj} - Z_{cj}) \cdot \omega'_{ij} \times (X_i) \right)} \quad (8)$$

More detail @ https://www.youtube.com/ShashiSAS

Model

forward Propagation →

$$\text{Input } (x) \times \text{weight 1 } (\omega) = h = \omega^T x \; * \; \text{weight 2 } (\omega') = u = \omega'^T h \rightarrow \frac{e^{u_j}}{\sum e^{u_j}} \rightarrow y_j = \frac{e^{u_j}}{\sum_{j=1}^{V} e^{u_j}}$$

$$(\omega_{ij})^{new} = (\omega_{ij})^{old} - \eta \left[\sum_{J=1}^{J} \sum_{c=1}^{C} (Y_{cj} - Z_{cj}) \cdot \omega'_{ij} \times x_i \right]$$

$$(\omega'_{ij})^{new} = (\omega'_{ij})^{old} - \eta \sum_{c=1}^{C} (Y_{cj} - Z_{cj}) \cdot h_i$$

$$E = C \log \sum_{j=1}^{J} e^{u_J} - \sum_{c=1}^{C} u_{cJ}$$

← Backward propagation

Skip-gram

More detail @ https://www.youtube.com/ShashiSAS

In [1]:
```
#Install gensim and bokeh
#!pip install gensim
#!pip install bokeh
```

More detail @ https://www.youtube.com/ShashiSAS

Dataset : https://code.google.com/archive/p/word2vec/

Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases.

In [1]:
```
Path=r'C:\Users\Arun\Desktop\Nipu\GoogleNews-vectors-negative300.bin'
```

In [3]:
```
import gensim
#train word2vec model
# Load Google's pre-trained Word2Vec model.
model = gensim.models.KeyedVectors.load_word2vec_format(Path, binary=True)
#Vocabulary size
words = list(model.wv.vocab)
print('Here is the Vocabulary Size.. %d' % len(words))
```

```
Here is the Vocabulary Size.. 3000000
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: DeprecationWarning: Call to deprecated `wv` (Attr
ibute will be removed in 4.0.0, use self instead).
```

Word2vec makes each word a vector. We are using the 300-number vector, which can be seen for the word "king".

In [17]:
```
w = model['king']
print("Array length of Word 'King':-",len(w))
```

```
Array length of Word 'King':- 300
```

In [19]:
```
print("'King' 300 dimension array values:-")
print(w)
```

```
'King' 300 dimension array values:-
[ 1.25976562e-01  2.97851562e-02  8.60595703e-03  1.39648438e-01
 -2.56347656e-02 -3.61328125e-02  1.11816406e-01 -1.98242188e-01
  5.12695312e-02  3.63281250e-01 -2.42187500e-01 -3.02734375e-01
 -1.77734375e-01 -2.49023438e-02 -1.67968750e-01 -1.69921875e-01
  3.46679688e-02  5.21850586e-03  4.63867188e-02  1.28906250e-01
  1.36718750e-01  1.12792969e-01  5.95703125e-02  1.36718750e-01
  1.01074219e-01 -1.76757812e-01 -2.51953125e-01  5.98144531e-02
  3.41796875e-01 -3.11279297e-02  1.04492188e-01  6.17675781e-02
  1.24511719e-01  4.00390625e-01 -3.22265625e-01  8.39843750e-02
  3.90625000e-02  5.85937500e-03  7.03125000e-02  1.72851562e-01
  1.38671875e-01 -2.31445312e-01  2.83203125e-01  1.42578125e-01
  3.41796875e-01 -2.39257812e-02 -1.09863281e-01  3.32031250e-02
 -5.46875000e-02  1.53198242e-02 -1.62109375e-01  1.58203125e-01
 -2.59765625e-01  2.01416016e-02 -1.63085938e-01  1.35803223e-03
 -1.44531250e-01 -5.68847656e-02  4.29687500e-02 -2.46582031e-02
  1.85546875e-01  4.47265625e-01  9.58251953e-03  1.31835938e-01
  9.66328125e-02 -1.85546875e-01 -1.00097656e-01 -1.33789062e-01
 -1.25000000e-01  2.83203125e-01  1.23046875e-01  5.32226562e-02
 -1.77734375e-01  8.59375000e-02 -2.18506859e-02  2.05078125e-02
 -1.39648438e-01  2.51464844e-02  1.38671875e-01 -1.05468750e-01
  1.38671875e-01  8.88671875e-02 -7.51953125e-02 -2.13623047e-02
  1.72851562e-01  4.63867188e-02 -2.65625000e-01  8.91113281e-03
  1.49414062e-01  3.78417969e-02  2.38281250e-01 -1.24511719e-01
 -2.17773438e-01 -1.81640625e-01  2.97851562e-02  5.71289062e-02
 -2.89306641e-02  1.24511719e-02  9.66796875e-02 -2.31445312e-01
  5.81054688e-02  6.68945312e-02  7.08007812e-02 -3.08593750e-01
 -2.14843750e-01  1.45507812e-01 -4.27734375e-01 -9.39941406e-03
  1.54296875e-01 -7.66601562e-02  2.89062500e-01  2.77343750e-01
 -4.86373901e-04 -1.36718750e-01  3.24218750e-01 -2.46093750e-01
 -3.03649902e-03 -2.11914062e-01  1.25000000e-01  2.69531250e-01
  2.04101562e-01  8.25195312e-02 -2.01171875e-01 -1.60156250e-01
 -3.78417969e-02 -1.20117188e-01  1.15234375e-01 -4.10156250e-02
 -3.95507812e-02 -8.98437500e-02  6.34765625e-03  2.03125000e-01
  1.86523438e-01  2.73437500e-01  6.29882812e-02  1.41601562e-01
 -9.81445312e-02  1.38671875e-01  1.82617188e-01  1.73828125e-01
  1.73828125e-01 -2.37304688e-01  1.78710938e-01  6.34765625e-02
  2.36328125e-01 -2.08984375e-01  8.74023438e-02 -1.66015625e-01
 -7.91015625e-02  2.43164062e-01 -8.88671875e-02  1.26953125e-01
 -2.16796875e-01 -1.73828125e-01 -3.59375000e-01 -8.25195312e-02
 -6.49414062e-02  5.07812500e-02  1.35742188e-01 -7.47070312e-02
 -1.64062500e-01  1.15356445e-02  4.45312500e-01 -2.15820312e-01
 -1.13281125e-01 -1.92382812e-01  1.78094438e-01 -1.25000000e-01
  2.65502930e-03  1.92382812e-01 -1.74804688e-01  1.39648438e-01
  2.92968750e-01  1.13281250e-01  5.95703125e-02 -6.39648438e-02
  9.96093750e-02 -2.72216797e-02  1.96533203e-02  4.27246094e-02
 -2.46093750e-01  6.39648438e-02 -2.25585938e-01 -1.68945312e-01
  2.89916992e-03  8.20312500e-02  3.41796875e-01  4.32128906e-02
  1.32812500e-01  1.42578125e-01  7.61718750e-02  5.98144531e-02
 -1.19140625e-01  2.74658203e-03 -6.29882812e-02 -2.72216797e-02
 -4.82177734e-03 -8.20312500e-02 -2.49023438e-02 -4.00390625e-01
 -1.06933594e-01  4.24804688e-02  7.76367188e-02 -1.16699219e-01
  7.37304688e-02 -9.22851562e-02  1.07910156e-01  1.58203125e-01
  4.24804688e-02  1.26953125e-01  3.61328125e-02  2.67578125e-01
 -1.01074219e-01 -3.02734375e-01 -5.76171875e-02  5.05371094e-02
  5.26428223e-04 -2.07031250e-01 -1.38671875e-01 -8.97216797e-03
 -2.78320312e-02 -1.41601562e-01  2.07031250e-01 -1.58203125e-01
  1.27929688e-01  1.49414062e-01 -2.24609375e-02 -8.44726562e-02
 -1.22558594e-01  2.15820312e-01 -2.13867188e-01  3.12500000e-01
 -3.73046875e-01  4.08935547e-03  1.07421875e-01  1.06933594e-01
  7.32421875e-02  8.97216797e-03 -3.88183594e-02 -1.29882812e-01
  1.49414062e-01 -2.14843750e-01 -1.83868408e-03  9.91210938e-02
  1.57226562e-01 -1.14257812e-01 -2.05078125e-01  9.91210938e-02
  3.69140625e-01 -1.97265625e-01  3.54003906e-02  1.09375000e-01
  1.31835938e-01  1.66992188e-01  2.35351562e-01  1.04980469e-01
 -4.96093750e-01 -1.64062500e-01 -1.56250000e-01  5.22460938e-02
  1.03027344e-01  2.43164062e-01 -1.88476562e-01  5.07812500e-02
 -9.37500000e-02 -6.68945312e-02  2.27050781e-02  7.61718750e-02
  2.89062500e-01  3.10546875e-01 -5.37109375e-02  2.28515625e-01
  2.51464844e-02  6.78710938e-02 -1.21093750e-01 -2.15820312e-01
 -2.73437500e-01 -3.07617188e-02 -3.37890625e-01  1.53320312e-01
  2.33398438e-01 -2.08007812e-01  3.73046875e-01  8.20312500e-02
  2.51953125e-01 -7.61718750e-02 -4.66308594e-02 -2.23388672e-02
  2.99072266e-02 -5.93261719e-02 -4.66918945e-03 -2.44140625e-01
 -2.09960938e-01 -2.87109375e-01 -4.54101562e-02 -1.77734375e-01
 -2.79296875e-01 -8.59375000e-02  9.13085938e-02  2.51953125e-01]
```

The following code shows which words are most similar to the given one.

In [7]:
```
model.most_similar('king')
```

Out[7]:
```
[('kings', 0.7138046026229858),
 ('queen', 0.6510957479476629),
 ('monarch', 0.6413194537162781),
 ('crown_prince', 0.6204220056533813),
 ('prince', 0.6159993418118474),
 ('sultan', 0.5864823460578918),
 ('ruler', 0.5797567367553711),
 ('princes', 0.5646551847457886),
 ('Prince_Paras', 0.5432944297790527),
 ('throne', 0.5422104597091675)]
```

Kings,queen,monarch,crown_prince,sultan...etc these words are similar to word 'king'

In [8]:
```
model.most_similar('queen')
```

Out[8]:
```
[('queens', 0.7399442791938782),
 ('princess', 0.7070531249046326),
 ('king', 0.6510956883430481),
 ('monarch', 0.6383601427078247),
 ('very_pampered_McElhatton', 0.6357026696205139),
 ('Queen', 0.6163408160209656),
 ('NYC_anglophiles_aflutter', 0.6060680150985718),
 ('Queen_Consort', 0.5923796296611969),
 ('princesses', 0.5908075571060181),
 ('royal', 0.5637185573577881)]
```

queen,princess,king,monarch,queen...etc these words are similar to word 'queen'

The following code shows the similarity between two words.

In [9]:
```
model.similarity('king', 'queen')
```

Out[9]:
```
0.6510957
```

65% have probability both have similar words

The code below shows the distance between two words.

In [10]:
```
import numpy as np

w1 = model['king']
w2 = model['queen']

dist = np.linalg.norm(w1-w2)

print("Distance Between King & queen:-",dist)
```

```
Distance Between King & queen:- 2.4796925
```

This shows the classic word2vec equation of queen = (king - man) + female

In [11]:
```
model.most_similar(positive=['woman', 'king'], negative=['man'])
```

Out[11]:
```
[('queen', 0.7118192315101624),
 ('monarch', 0.6189674735669275),
 ('princess', 0.5902431011199951),
 ('crown_prince', 0.5499466097174072),
 ('prince', 0.5377321243286133),
 ('kings', 0.5236844420443044),
 ('Queen_Consort', 0.5235946178436279),
 ('queens', 0.5181133747710083),
 ('sultan', 0.5098593235015869),
 ('monarchy', 0.5087411599702454)]
```

71% have probability 'queen' is the correct value of equation " queen = (king - man) + female"

The following code shows which item does not belong with the others.

In [12]:
```
model.doesnt_match("man king queen apple".split())
```

```
C:\ProgramData\Anaconda3\lib\site-packages\gensim\models\keyedvectors.py:877: FutureWarning: arrays to stack must be
passed as a "sequence" type such as list or tuple. Support for non-sequence iterables such as generators is deprecate
d as of NumPy 1.16 and will raise an error in the future.
  vectors = vstack(self.word_vec(word, use_norm=True) for word in used_words).astype(REAL)
```

Out[12]:
```
'apple'
```

Visualization of vocabalary in 2-D Space

In [53]:
```
# Importing bokeh libraries for showing how words of similar context are grouped together
import bokeh.plotting as bp
import pandas as pd
from bokeh.models import HoverTool, BoxSelectTool,LabelSet,ColumnDataSource
from bokeh.plotting import figure, show, output_notebook
voc_size=3000
#Defining the chart
output_notebook()
plot_chart = bp.figure(plot_width=700, plot_height=600, title="A map/plot of 3000 word vectors",
    tools="pan,wheel_zoom,box_zoom,reset,hover",
    x_axis_type=None, y_axis_type=None, min_border=1)

#Extracting the list of word vectors, limiting to 1000, each is of 200 dimensions
word_vectors = [model[w] for w in list(model.wv.vocab.keys())[:voc_size]]

# Reducing dimensionality by converting the vectors to 2d vectors
from sklearn.manifold import TSNE
tsne_model = TSNE(n_components=2, verbose=1, random_state=0)
tsne_w2v = tsne_model.fit_transform(word_vectors)

# Storing data in a dataframe
tsne_df = pd.DataFrame(tsne_w2v, columns=['x', 'y'])
tsne_df['words'] = list(model.wv.vocab.keys())[:voc_size]

# Corresponding word appears when you hover on the data point.
plot_chart.scatter(x='x', y='y',source=tsne_df)
hover = plot_chart.select(dict(type=HoverTool))
hover.tooltips={("word": "@words"}
show(plot_chart)
```

```
BokehJS 2.0.1 successfully loaded.
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: DeprecationWarning: Call to deprecated `wv` (Att
ribute will be removed in 4.0.0, use self instead).

[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 3000 samples in 0.424s...
[t-SNE] Computed neighbors for 3000 samples in 9.688s...
[t-SNE] Computed conditional probabilities for sample 1000 / 3000
[t-SNE] Computed conditional probabilities for sample 2000 / 3000
[t-SNE] Computed conditional probabilities for sample 3000 / 3000
[t-SNE] Mean sigma: 0.883493
[t-SNE] KL divergence after 250 iterations with early exaggeration: 110.179993
[t-SNE] KL divergence after 1000 iterations: 2.198179
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:23: DeprecationWarning: Call to deprecated `wv` (Att
ribute will be removed in 4.0.0, use self instead).
```

More detail @ https://www.youtube.com/ShashiSAS

A map/plot of 3000 word vectors

In [ ]:

More detail @ https://www.youtube.com/ShashiSAS

Import Input data

Pre-processing

1.Remove Number
2.Replace punctuations with a white space
3. Tokenization
4. Remove All Stop Word
5. Lemmatization with POS tag

Word Embedding

Convert text to vector using Word2Vec

Split Dataset in Train and Test

Model

For Classification apply SVM,Logistic Regression,Neural Network

Analysis on Model (Accurecy,AUC,F-score)

Pre-Processing of text

Word Embedding /Feature Extraction

Model

Movie Review Analysis using Word2Vec

In [1]:
```
#install pandas,matplotlib,numpy,nltk,gensim,scikit-learn,bokeh package
#!pip install pandas
#!pip install matplotlib
#!pip install numpy
#!pip install nltk
#!pip install gensim
#!pip install scikit-learn
#!pip install bokeh
#!pip install tensorflow
#!pip install keras
```

More detail @ https://www.youtube.com/ShashiSAS

In [2]:
```
#Import Data
import pandas as pd
df = pd.read_csv('movie-Review.csv')
```

In [3]:
```
import numpy as np
np.random.seed(500)
```

In [4]:
```
class_count=df.groupby('class').count()
print(class_count)
import matplotlib.pyplot as plt
plt.bar(class_count.index.values, class_count['text'])
plt.xlabel('Review Sentiments')
plt.ylabel('Number of Review')
plt.show()
```

```
       text
class
Neg    1000
Pos    1000
```

In [5]:
```
#Remove number
import re # import all Regular expression functions
df['text']=[re.sub('\d','', i)for i in df['text']]
df.head(10)
```

Out[5]:

	class	text
0	Pos	films adapted from comic books have had plent...
1	Pos	every now and then a movie comes along from a...
2	Pos	you ve got mail works alot better than it des...
3	Pos	jaws is a rare film that grabs your atte...
4	Pos	moviemaking is a lot like being the general m...
5	Pos	on june a self taught idealistic ye...
6	Pos	apparently director tony kaye had a major b...
7	Pos	one of my colleagues was surprised when i tol...
8	Pos	after bloody clashes and independence won l...
9	Pos	the american action film has been slowly drow...

In [6]:
```
# Replace punctuations with a white space
import string
df['text']=[re.sub('[%s]' % re.escape(string.punctuation), ' ', i) for i in df['text']]
df.head(10)
```

Out[6]:

	class	text
0	Pos	films adapted from comic books have had plent...
1	Pos	every now and then a movie comes along from a...
2	Pos	you ve got mail works alot better than it des...
3	Pos	jaws is a rare film that grabs your atte...
4	Pos	moviemaking is a lot like being the general m...
5	Pos	on june a self taught idealistic ye...
6	Pos	apparently director tony kaye had a major b...
7	Pos	one of my colleagues was surprised when i tol...
8	Pos	after bloody clashes and independence won l...
9	Pos	the american action film has been slowly drow...

In [7]:
```
#Convert into in lower case
df['text']=[i.lower() for i in df['text']]
```

More detail @ https://www.youtube.com/ShashiSAS

In [8]:
```
# import pandas as pd
import pandas as pd
#Word Tokenization
import nltk # import package for tokenization
#nltk.download('punkt') # download all spporting function /files for NLTK package
from nltk.tokenize import word_tokenize
df['text_wt'] = [word_tokenize(i) for i in df['text']]
df.head()
```

Out[8]:

	class	text	text_wt
0	Pos	films adapted from comic books have had plent...	[films, adapted, from, comic, books, have, had...
1	Pos	every now and then a movie comes along from a...	[every, now, and, then, a, movie, comes, along...
2	Pos	you ve got mail works alot better than it des...	[you, ve, got, mail, works, alot, better, than...
3	Pos	jaws is a rare film that grabs your atte...	[jaws, is, a, rare, film, that, grabs, your, a...
4	Pos	moviemaking is a lot like being the general m...	[moviemaking, is, a, lot, like, being, the, ge...

More detail @ https://www.youtube.com/ShashiSAS

In [9]:
```
#To show the stop words
#nltk.download('stopwords') #download Stopwords
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
#Remove All Stop Word
df['text_SW'] = [[i for i in j if not i in stop_words] for j in df['text_wt']]# remove the word which is aviable in stopword libr
df.head()
```

Out[9]:

	class	text	text_wt	text_SW
0	Pos	films adapted from comic books have had plent...	[films, adapted, from, comic, books, have, had...	[films, adapted, comic, books, plenty, success...
1	Pos	every now and then a movie comes along from a...	[every, now, and, then, a, movie, comes, along...	[every, movie, comes, along, suspect, studio, ...
2	Pos	you ve got mail works alot better than it des...	[you, ve, got, mail, works, alot, better, than...	[got, mail, works, alot, better, deserves, ord...
3	Pos	jaws is a rare film that grabs your atte...	[jaws, is, a, rare, film, that, grabs, your, a...	[jaws, rare, film, grabs, attention, shows, si...
4	Pos	moviemaking is a lot like being the general m...	[moviemaking, is, a, lot, like, being, the, ge...	[moviemaking, lot, like, general, manager, nfl...

In [10]:
```
#nltk.download('tagsets')
#nltk.help.upenn_tagset()# tagset documentation
#nltk.download('wordnet')
from collections import defaultdict #Default Dictionary is imported from collections
from nltk.corpus import wordnet as wn #the corpus reader wordnet is imported.
from nltk.tag import pos_tag
# WordNetLemmatizer requires Pos tags to understand if the word is noun or verb or adjective etc.
#By default it is set to Noun
tag_map = defaultdict(lambda : wn.NOUN) #Dictionary is created where pos_tag (first letter) are the key values
tag_map['J'] = wn.ADJ                   #whose values are mapped with the value
tag_map['V'] = wn.VERB                  #from wordnet dictionary. We have taken the only first letter as
tag_map['R'] = wn.ADV
# we will use it later in the loop.
#tag_map
```

In [11]:
```
#Lemmatization
from nltk.stem import WordNetLemmatizer
 # Initializing WordNetLemmatizer()
lemmatizer = WordNetLemmatizer()

df['lemma']=[[lemmatizer.lemmatize(word,tag_map[tag[0]]) for word ,tag in pos_tag(i)] for i in df['text_SW']]
df.head()
```

Out[11]:

	class	text	text_wt	text_SW	lemma
0	Pos	films adapted from comic books have had plent...	[films, adapted, from, comic, books, have, had...	[films, adapted, comic, books, plenty, success...	[film, adapt, comic, book, plenty, success, wh...
1	Pos	every now and then a movie comes along from a...	[every, now, and, then, a, movie, comes, along...	[every, movie, comes, along, suspect, studio, ...	[every, movie, come, along, suspect, studio, e...
2	Pos	you ve got mail works alot better than it des...	[you, ve, got, mail, works, alot, better, than...	[got, mail, works, alot, better, deserves, ord...	[get, mail, work, alot, good, deserves, order,...
3	Pos	jaws is a rare film that grabs your atte...	[jaws, is, a, rare, film, that, grabs, your, a...	[jaws, rare, film, grabs, attention, shows, si...	[jaw, rare, film, grab, attention, show, singl...
4	Pos	moviemaking is a lot like being the general m...	[moviemaking, is, a, lot, like, being, the, ge...	[moviemaking, lot, like, general, manager, nfl...	[moviemaking, lot, like, general, manager, nfl...

In [12]:
```
df['lemma2']= df['lemma'].apply(lambda x: ' '.join(x))
```

In [13]:
```
df['lemma2'].head()
```

Out[13]:
```
0    film adapt comic book plenty success whether s...
1    every movie come along suspect studio every in...
2    get mail work alot good deserves order make fi...
3    jaw rare film grab attention show single image...
4    moviemaking lot like general manager nfl team ...
Name: lemma2, dtype: object
```

More detail @ https://www.youtube.com/ShashiSAS

class gensim.models.word2vec.Word2Vec(sentences=None, corpus_file=None, size=100, alpha=0.025, window=5, min_count=5, max_vocab_size=None, sample=0.001, seed=1, workers=3, min_alpha=0.0001, sg=0, hs=0, negative=5, ns_exponent=0.75, cbow_mean=1, hashfxn=, iter=5, null_word=0, trim_rule=None, sorted_vocab=1, batch_words=10000, compute_loss=False, callbacks=(), max_final_vocab=None)

Min_count: It will ignore all the words with a total frequency lower than this. Size: It tells the dimensionality of the word vectors. Workers: These are the threads to train the model Window: Maximum distance between the current and predicted word within a sentence. Sg: It is a training algorithm and 1 for skip-gram and 0 for a Continuous bag of words. Hs: If this is 1 then we are using hierarchical softmax for training and if 0 then negative sampling is used. Alpha: Initial learning rate

In [55]:
```
#Train a Word2Vec model using Gensim
import gensim
Min_count=1      #It will ignore all the words with a total frequency lower than this
Size = 200      #It tells the dimensionality of the word vectors
Workers=4        #These are the threads to train the model
Window=5         #Maximum distance between the current and predicted word within a sentence
#train word2vec model
model = gensim.models.Word2Vec(sentences = df['lemma'], size = Size,window=Window, workers = Workers, min_count = Min_count)
#Vocabulary size
words = list(model.wv.vocab)
print('Here is the Vocabulary Size.. %d' % len(words))
```

```
Here is the Vocabulary Size.. 32325
```

In [56]:
```
# This will print the most similar words  present in the model
model.most_similar("film")
```

```
C:\Users\shashi.singh\.conda\envs\NLP\lib\site-packages\ipykernel_launcher.py:2: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.mo
st_similar() instead).
```

Out[56]:
```
[('movie', 0.9535752534866333),
 ('picture', 0.9248700141906738),
 ('flick', 0.9101689457893372),
 ('comedy', 0.9081454277038574),
 ('moment', 0.9040921926498413),
 ('respected', 0.8991982936859131),
 ('thriller', 0.8961208462715149),
 ('credit', 0.8950402140617371),
 ('genre', 0.8940645456314087),
 ('original', 0.8918867707252502)]
```

More detail @ https://www.youtube.com/ShashiSAS

In [59]:
```
# Importing bokeh libraries for showing how words of similar context are grouped together
import bokeh.plotting as bp
from bokeh.models import HoverTool, BoxSelectTool
from bokeh.plotting import figure, show, output_notebook
voc_size=3000
#Defining the chart
output_notebook()
plot_chart = bp.figure(plot_width=700, plot_height=600, title="A map/plot of 3000 word vectors",
    tools="pan,wheel_zoom,box_zoom,reset,hover",
    x_axis_type=None, y_axis_type=None, min_border=1)

#Extracting the list of word vectors, limiting to 1000, each is of 200 dimensions
word_vectors = [model[w] for w in list(model.wv.vocab.keys())[:voc_size]]

# Reducing dimensionality by converting the vectors to 2d vectors
from sklearn.manifold import TSNE
tsne_model = TSNE(n_components=2, verbose=1, random_state=0)
tsne_w2v = tsne_model.fit_transform(word_vectors)

# Storing data in a dataframe
tsne_df = pd.DataFrame(tsne_w2v, columns=['x', 'y'])
tsne_df['words'] = list(model.wv.vocab.keys())[:voc_size]

# Corresponding word appears when you hover on the data point.
plot_chart.scatter(x='x', y='y', source=tsne_df)
hover = plot_chart.select(dict(type=HoverTool))
hover.tooltips={"word": "@words"}
show(plot_chart)
```

(https://bokeh.org) BokehJS 2.0.2 successfully loaded.

```
C:\Users\shashi.singh\.conda\envs\NLP\lib\site-packages\ipykernel_launcher.py:13: DeprecationWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__
getitem__() instead).
  del sys.path[0]

[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 3000 samples in 0.141s...
[t-SNE] Computed neighbors for 3000 samples in 1.640s...
[t-SNE] Computed conditional probabilities for sample 1000 / 3000
[t-SNE] Computed conditional probabilities for sample 2000 / 3000
[t-SNE] Computed conditional probabilities for sample 3000 / 3000
[t-SNE] Mean sigma: 0.052408
[t-SNE] KL divergence after 250 iterations with early exaggeration: 64.855522
[t-SNE] KL divergence after 1000 iterations: 0.888756
```

More detail @ https://www.youtube.com/ShashiSAS

Next we have to build word vectors for input text in order to average the value of all word vectors using the following function:

In [17]:
```
import numpy as np
#Build word vector set by using the average value of all word vectors , then scale
def buildWordVector(text, size):
    vec = np.zeros(size).reshape((1, size)) #As word vectors are of zero length size value(i.e 300)
    count = 0 # no. of words with a valid vector in the sentence/review
    for word in text: #for each word in a sentence/review
        try:
            vec += model[word].reshape((1, size))
            count += 1.
        except KeyError:
            continue
    if count != 0:
        vec /= count
    return vec
```

Scaling moves our data set is part of the process of standardization where we move our dataset into a gaussian distribution with a mean of zero, meaning that values above the mean will be positive, and those below the mean will be negative. Many ML models require scaled datasets to perform effectively, especially those with many features (like text classifiers).

In [18]:
```
from sklearn.preprocessing import scale
vecs = np.concatenate([buildWordVector(z, Size) for z in df['lemma']])
#print("Before Scaling:",vecs[1:2])
vecs = scale(vecs)
#print("After Scaling:",vecs[1:2])
```

```
C:\Users\shashi.singh\.conda\envs\NLP\lib\site-packages\ipykernel_launcher.py:8: DeprecationWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__g
etitem__() instead).
```

In [19]:
```
print("Dimension of vector :",vecs.shape)
```

```
Dimension of vector : (2000, 200)
```

In [20]:
```
#Convert Target varible in numerical form using label encoder method
from sklearn.preprocessing import LabelEncoder
Encoder = LabelEncoder()
df['class2'] = Encoder.fit_transform(df['class'])
print(df['class2'].head())
```

```
0    1
1    1
2    1
3    1
4    1
Name: class2, dtype: int32
```

In [51]:
```
#Split dataset in Train and Test:
#test_size=0.2 (Dataset split ,Test dataset 20% Train dataset =80%)
#random_state=342 :Providing a value to random state will be helpful in reproducing the same values
                   #in the split when you re-run the program
from sklearn.model_selection import train_test_split
Train_X, Test_X, Train_Y, Test_Y = train_test_split(vecs,df['class2'],test_size=0.2,random_state=342)
```

More detail @ https://www.youtube.com/ShashiSAS

In [22]:
```
# Classifier - Algorithm - SVM
# fit the training dataset on the classifier
from sklearn import model_selection,  svm
from sklearn.metrics import accuracy_score
SVM = svm.SVC(C=1.0, kernel='linear')
SVM.fit(Train_X,Train_Y)
# predict the labels on validation dataset
predictions_SVM = SVM.predict(Test_X)
# Use accuracy_score function to get the accuracy
print("SVM Accuracy Score -> ",round(accuracy_score(predictions_SVM, Test_Y)*100,2),"%")
```

```
SVM Accuracy Score ->  77.25 %
```

In [23]:
```
# Logistic Regression Model:-
# 1. import
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
# 2. instantiate a logistic regression model
logreg = LogisticRegression()
# 3. train the model using X_train_dtm
%time logreg.fit(Train_X, Train_Y)#4. make class predictions for Test_X
predictions_log = logreg.predict(Test_X)

print(" ------ Confusion Matrix-----[TN FP  FN TP]")

print(metrics.confusion_matrix(predictions_log, Test_Y))
print(metrics.classification_report(predictions_log, Test_Y))
# Use accuracy_score function to get the accuracy
print("Logistic Regression Accuracy Score -> ",round(accuracy_score(predictions_log, Test_Y)*100,2),"%")
print("Logistic Regression Area under curve -> ",round(metrics.roc_auc_score(predictions_log, Test_Y),2))
```

```
Wall time: 85.8 ms
 ------ Confusion Matrix-----[TN FP  FN TP]
[[153  45]
 [ 54 148]]
              precision    recall  f1-score   support

           0       0.74      0.77      0.76       198
           1       0.77      0.73      0.75       202

    accuracy                           0.75       400
   macro avg       0.75      0.75      0.75       400
weighted avg       0.75      0.75      0.75       400

Logistic Regression Accuracy Score ->  75.25 %
Logistic Regression Area under curve ->  0.75

C:\Users\shashi.singh\.conda\envs\NLP\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

More detail @ https://www.youtube.com/ShashiSAS

Neural Network

In [53]:
```
import tensorflow as tf
import keras.layers as layers
from keras.models import Model

from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.layers import Input,Embedding,Dense,Flatten
from sklearn.metrics import accuracy_score,classification_report
from sklearn.metrics import f1_score

epochs = 25
batch_size = 1024
loss = "binary_crossentropy"
optimizer = "adam"
metrics = ["accuracy"]

from keras import models

callbacks = [EarlyStopping(monitor='val_loss', patience=2),
            ModelCheckpoint(filepath='best_model.h5', monitor='val_loss', save_best_only=True)]

# Build neural network
model = models.Sequential()
model.add(Dense(512, activation='relu', input_shape=(200,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss=loss,optimizer=optimizer,metrics= metrics)
model.fit(Train_X,Train_Y,epochs=epochs,batch_size=batch_size,callbacks=callbacks,validation_data=(Test_X,Test_Y))


predictions = model.predict(Test_X)
predictions = [0 if i<0.5 else 1 for i in predictions]

f1_score(Test_Y, predictions)
print("Accuracy: ",accuracy_score(Test_Y,predictions)*100)
```

```
Train on 1600 samples, validate on 400 samples
Epoch 1/25
1600/1600 [==============================] - 0s 125us/step - loss: 0.7340 - accuracy: 0.5381 - val_loss: 0.7026 - val_accuracy: 0.5800
Epoch 2/25
1600/1600 [==============================] - 0s 54us/step - loss: 0.6749 - accuracy: 0.5981 - val_loss: 0.7168 - val_accuracy: 0.5550
Epoch 3/25
1600/1600 [==============================] - 0s 53us/step - loss: 0.6818 - accuracy: 0.5819 - val_loss: 0.6450 - val_accuracy: 0.6225
Epoch 4/25
1600/1600 [==============================] - 0s 48us/step - loss: 0.6263 - accuracy: 0.6506 - val_loss: 0.6922 - val_accuracy: 0.5975
Epoch 5/25
1600/1600 [==============================] - 0s 50us/step - loss: 0.6342 - accuracy: 0.6456 - val_loss: 0.6477 - val_accuracy: 0.6125
Accuracy:  61.25000000000001
```

More detail @ https://www.youtube.com/ShashiSAS

References:

1. https://arxiv.org/pdf/1310.4546.pdf
2. https://arxiv.org/pdf/1411.2738.pdf
3. https://radimrehurek.com/gensim/models/word2vec.html
4. https://towardsdatascience.com/an-implementation-guide-to-word2vec-using-numpy-and-google-sheets-13445eebd281
5. http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture01-wordvecs1.pdf
6. https://ai-leader.com/
7. https://en.wikipedia.org/wiki/Word2vec
8. https://deeplearning4j.konduit.ai/language-processing/word2vec#introduction-to-word-2-vec
9. https://towardsdatascience.com/word-embedding-with-word2vec-and-fasttext-a209c1d3e12c
10. http://alexminnaar.com/2015/04/12/word2vec-tutorial-skipgram.html
11. https://poloclub.github.io/cnn-explainer/