NETWORK EMBEDDING on TEMPORAL DATA

AND FUTURE LINK PREDICTION

Using Unsupervised Learning Algorithms

**Sai Kumar Naik Buckya**

**National Institute of Technology Manipur**

**Under the guidance of :**

**Dr. Sanasam Ranbir singh**

**Associate Professor**

**Dept. of Computer Science and Engineering**

**Indian Institute of Technology Guwahati.**

**Mentor :**

**Akash Anil**



May 2019-July 2019

Acknowledgment

I would like to thank my guide Dr. Sanasam Ranbir Singh (Associate Professor), of Computer Science and Engineering, IIT Guwahati, for accepting me as an intern and my mentor Mr.Akash Anil(Ph.D.) , not only a mentor but also a wonderful teacher. In this 2 months journey, an everyday 1 hour meeting with our mentor, we learnt a lot of new things that are related theoretical and technical knowledge of this project as well as some important lessons of life, time and flow Management,teamwork and motivation. Thank you so much sir Akash Anil, we are fortunate to have a wonderful mentor like you. I would also thank the Management of the college and OSINT Lab for providing the server and lab facilities for the project. I would also like to thank my fellow interns and friends Ashish Ranjan and Samyak Raj for their support and with whose help I could complete the project in the stipulated time. At last, I would like to thank my parents.

**Table of contents** Page no

1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2 Background study

2.1 Machine Learning and types . . . . . . . . . . . . . . . . . . . . 5

2.2 Neural Networks. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2.3 Word2Vec Algorithms

2.2.1 Skip-Gram. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.2.2 CBOW . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.4 Activation Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

3 Experimental Analysis

3.1 Experimental Dataset . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

3.2 Experimental Analysis:

3.2.1 Dataset Preprocessing . . . . . . . . . . . . . . . . . . . . . . 20

3.2.2 Corpus Generation: Time Aware user embeddings 21

3.2.3 Embeddings for nodes . . . . . . . . . . . . . . . . . . . . . . . 30

3.2.4 Edge feature generation . . . . . . . . . . . . . . . . . . . . . . . 30

3.2.5 Evluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

4 Model Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32

5 Results

5.1 HyperParameters 33

5.2 Total Number of Results 34

5.3 Comparison of Performance between kernel and Baseline 35

6 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36

7 References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38

**1 Introduction :**

Network embedding methods aim at learning a low-dimensional latent representation of nodes in a network. These representations can be used as features for a wide range of tasks on graphs such as classification, clustering, link prediction, and visualization.

The main goal of this paper is to introduce unsupervised techniques that can be used to generate the embedding vectors to the nodes and edges present in temporal data from huge data sets with thousands of nodes and edges, and with lakhs of edges between nodes along with the timestamp. And apply classification algorithms to classify and predict future links or edges between nodes.

**2 Background Study**

**2.1 Machine Learning:**

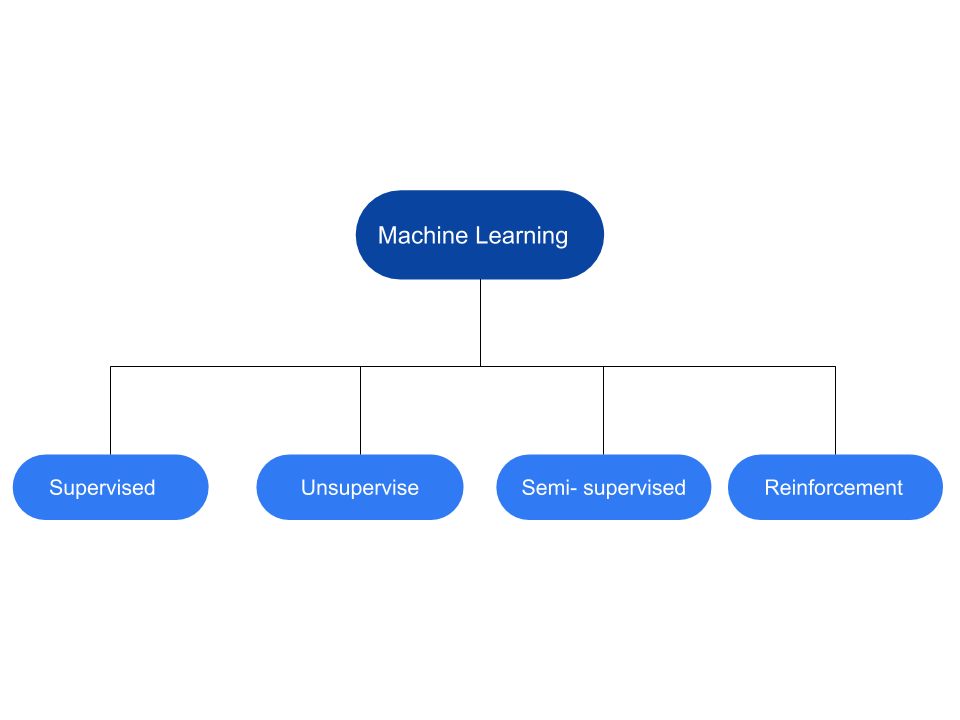
Machine Learning may be defined as the study of algorithms scientifically and statistical models which are used by the computer systems in order to perform a certain or specific task effectively and efficiently without using explicit instructions, relying on patterns and other resources.

Machine Learning may be termed as a subset of Artificial Intelligence(AI).

Machine Learning algorithms are classified on the basis of the type of data they input and output and the type of task that they are actually required to solve.

Types of Machine Learning Algorithms:

1. Supervised
2. Unsupervised
3. Semi-supervised
4. Reinforcement



**2.1.1** **Supervised Learning**

These are those class of algorithms which build a mathematical model of a

certain set of data that contains both the inputs as well as the desired outputs. The corresponding data is known as training, and consists of a certain or specific set of training examples and each training example has actually one or more inputs and a desired output or label**.**

**Example :-** Linear Regression, Logistic Regression, Naive Bayes Classifier , SVM, Decision Tree, Neural Networks etc., .

**2.1.2**  **Un-Supervised Learning**

These are the class of algorithms which actually take a set of data containing only inputs, and find a certain or specific structure in the corresponding data, like grouping or clustering of data points. Thus the algorithms learn from the test data that has not been labeled, classified or categorized.

Ex: Clustering

**2.1.3 Semi-Supervised Learning**

These are those class of machine learning tasks and techniques that also make suitable use of unlabeled data for training – typically a pretty small amount of labeled data with a pretty large amount of unlabeled data. These type of machine learning algorithms lie between unsupervised learning (without any labeled training data) and ,supervised learning (with completely labeled training data).

**2.1.4 Reinforcement Learning**

These are those class of machine learning models which actually make a certain sequence of decisions. The agent actually learns to achieve a goal in an uncertain, and quite a potentially complex environment. In reinforcement learning, the artificial intelligence lead to face a game-like situation. The computer employs trial and error to come up with a solution to the problem. In order to get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the corresponding actions it performs. Its goal is to maximize the total reward.

**2.2 Artificial Neural Networks**

**Artificial Neural Networks** abbreviated as **ANN** may be defined as that branch of Artificial Intelligence and area of computer science, which is related to making computers behave and perform more intelligently. Artificial Neural Networks process the data so given and exhibit some intelligence and they behaves exhibiting intelligence in such a way like pattern recognition, Learning and generalization.

A certain or specific ANN is basically modeled like the human brain where neurons are connected ,exhibiting complex patterns to process data from the senses, establish memories and control the movement and coordination of body parts. An Artificial Neural Network is a system based on the operation of biological neural networks or it is also defined as an effort of biological neural system.

**2.3 Word2Vec Algorithms**

Word2Vec may be defined as a certain or specific group of models that are actually used to represent each of the corresponding words in a large text as a vector in a space of dimensionality N(which we basically use to call as features).

There are basically two popular models which are used for the Word2Vec algorithms:--

1.SkipGram Model

2.CBOW[Continuous Bag Of Words] Model

2.3.1 ***SkipGram Model:--***

The main idea behind the SkipGram model can be explained as follows:--

It actually takes each of the corresponding words in a large corpus of words(the word so taken is termed as the focus word) and also takes one by one the corresponding words which are surrounding it within a chosen ‘window size’ which is then fed to a neural network which trains and then it actually predicts the probability of each of the corresponding words to indeed appear in the chosen window around the respective focus word.

**To be more precise,** after choosing a certain window size the SkipGram Model actually predicts the surrounding context words for the respective **focus word**.

Let us consider an example sentence as:--

“**A thing of beauty is a joy forever**”

Now let us consider the following Source Texts:--

**1.**

|  |  |  |
| --- | --- | --- |
| A | thing | of |

{window = 3}

{focus word = “**A**” }

The corresponding training samples are:--

a)- (A,thing)

b)- (A,of)

**2.**

|  |  |  |  |
| --- | --- | --- | --- |
| A | thing | of | beauty |

{window = 4}

{focus word = “**thing**” }

The corresponding training samples are:--

a)- (thing, A)

b)- (thing, of)

c)-(thing, beauty)

**3.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | thing | of | beauty | is |

{window = 5}

{focus word = “**of**” }

The corresponding training samples are:--

a)- (of, A)

b)- (of, thing)

c)-(of, beauty)

d)-(of, is)

**4.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | thing | of | beauty | is | a |

{window = 6}

{focus word = “**beauty**” }

The corresponding training samples are:--

a)- (beauty, A)

b)- (beauty, thing)

c)- (beauty, of)

d)- (beauty, is)

e)- (beauty, a)

***Model:--***

We can not just feed a neural network with some pairs of words so we actually find a breakthrough way to do this.

So we need to find a way to represent these words mathematically so that the network can process them.

One way we can use to do the same is that we can create a vocabulary of all the words in our text and then to encode each of our words as a vector of the same dimensions as that of our vocabulary.

So we will correspondingly have a vector with all 0s and a single 1 which represents corresponding word in the vocabulary.

Such a technique is popularly known as the ***one-hot encoding.***

***Example:--*** If we have a vocabulary consisting of the words as:

{“A”,”thing”,”of”,”beauty”,”is”,”a”,”joy”,”forever”}

The corresponding one-hot encoding of “joy” will have vector as :

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

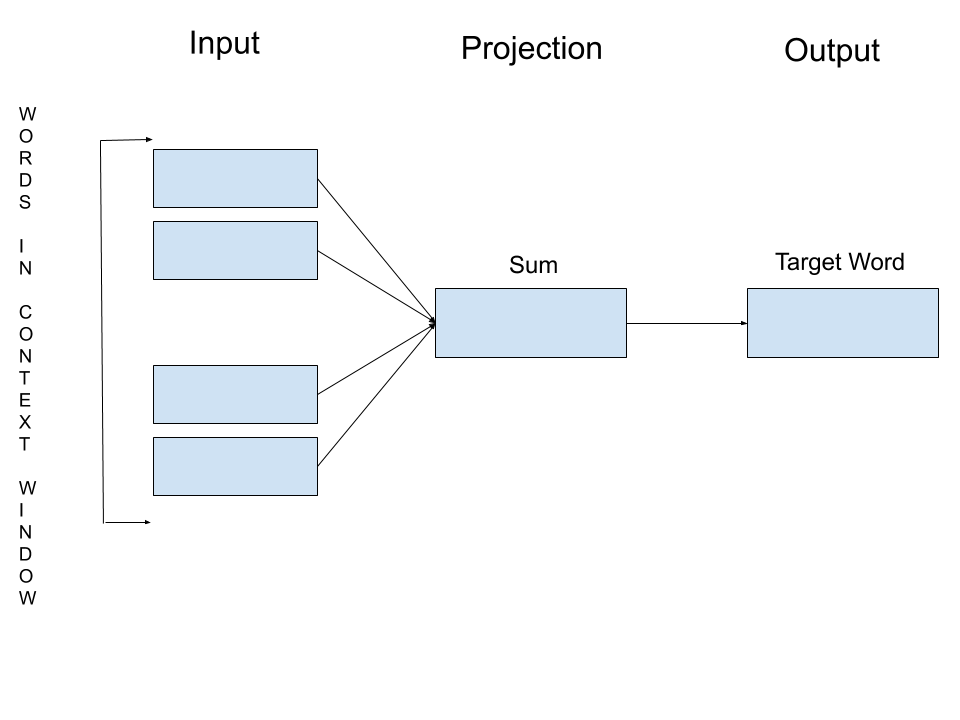
Now since our input is ready with us so we will use it to feed our ***“2 layer neural network”.***

It will actually process it in the second layer(output layer) and output a probability for each of the corresponding words in our vocabulary to appear in a randomly chosen position around the focus word(but still inside the window).

***CBOW[Continuous Bag Of Words] Model:--***

The CBOW[Continuous Bag of Words] is basically that Word2Vec model which actually tries to predict the current word as target(also termed as the center word) based on the words surrounding the corresponding target word ,in a certain or specific window size (also termed as source context words).

For Example , consider a simple sentence, “A thing of beauty is a joy forever”, we can have pairs of (context\_window(as per the window\_size), word\_as\_target) where if we consider a context window of size 2, we can have training samples like ([A, of], thing), ([thing, beauty], of), ([a, forever], joy) and so on in the same manner with different window sizes.



\*\*In our experiment we have basically used the SkipGram model

**2.4 Activation Functions:**

In simple words activation functions are mathematical equations that determine the output of a layer of neural network. This function is also used to decide that a particular neuron should be activated or not, based on whether that neuron’s input is relevant for the prediction of the model or not. Another advantage of these functions is that they help in normalizing the output of neurons to a range between 1 and 0 or between -1 and 1.

In a neural network first we multiply the input of each neuron with the corresponding weight and add the bias term.Now if we do not apply the activation function then this output is a result of linear function.We give this output as input to the next layer, this process will apply for every layer of the neural network and output of final layer will also be the result of linear function. Linear functions has very limited complexity.They can not learn the complex functional mappings from the given data and so the neural network will not learn the complex features of the data. Here the activation function comes into the picture,they provide non-linearity to the neural network model. They give the ability to the neural network to learn the complex and complicated things from the data.Data which consists of audio files,video files or images have complex features and without using the activation functions it is not possible to train the neural network for these data.

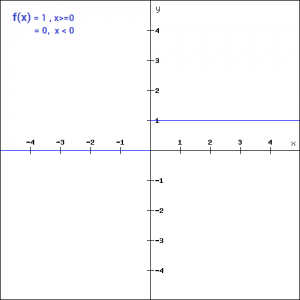
It is important to note that activation function must be differentiable.During the backpropagation of neural network we compute the gradient of activation function and it is supplied with the error to update the weight and biases.

**Types of Activation Functions:-**

**1.Binary step Function**

It is mathematically defined as-

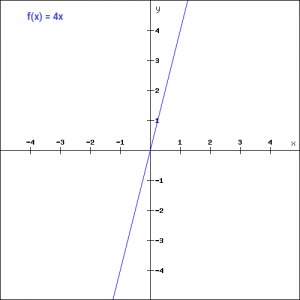
and



The main disadvantage of this function is that it’s differentiation is 0 and so it is not so useful in backpropagation.

**2.Linear Function**

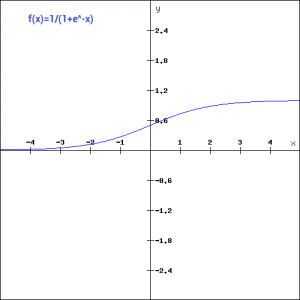
It is mathematically defined as-



This solves the problem of differentiation being zero (in binary step function ) but it’s differentiation is a constant term.Hence it is also not so useful as gradient will be same every time we do the backpropagation and error will not be improved.

**3.Sigmoid Function**

Sigmoid is one of the most widely used activation function.it is defined

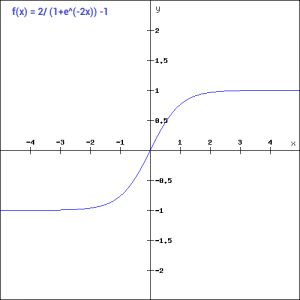


The biggest advantage of this function is that it is a nonlinear function and so it provides non-linearity to the model.It’s differentiation contains the term x. This means that during backpropagation we can easily use this function and weights and biases can be accordingly updated.The main disadvantage of this function is vanishing gradient. becomes flat beyond the region +3 and -3 on x-axis.This means that change in gradient is very small for corresponding change in in this region.Gradient is approaching zero and model is not learning well.Another disadvantage is that the sigmoid function value range is from 0 to 1 only.So sigmoid function is not symmetric around the origin and the values calculated are all positive and every time only positive values will be given to the neuron of next layer.

**4.Tanh Function**

It is similar to the sigmoid function and mathematically defined as-

or (In terms of Sigmoid Function )

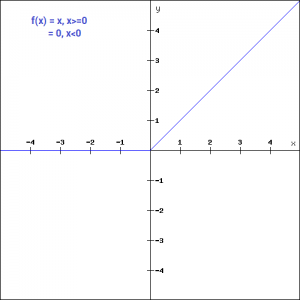


Tanh activation function solves the problem of all calculated value being positive.But all other properties of Tanh function is same as the Sigmoid function and it also suffers the problem of vanishing gradient.

**5.ReLU Function**

ReLU stands for Rectified Linear Unit.It is the most widely used activation function.

It is mathematically defined as-



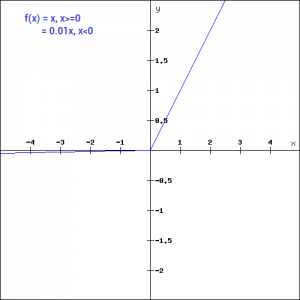
The main advantage of using the ReLU function over other activation functions is that unlike other activation functions does not activate all the neurons at the same time. If the input to the ReLU function is less than or equal to zero the neuron does not get activated. So at a time only few neurons will be activated and computation.for the network will be easy.

But on the other side there are also disadvantages of the function.

As for negative region of gradient becomes zero and so weight is not updated during backpropagation.This leads to the problem of dead neurons which never gets activated.

**6.Leaky ReLU**

Leaky ReLU function is improved version of ReLU function.It is mathematically defined as-

  
Here we have replaced the horizontal-line of negative region with a non horizontal-line and it solves the problem of gradient being zero in negative region and so dead neurons problem also got vanished.

**7.Softmax Function**

Softmax function is used when we are dealing with classification problems and we have to classify the output into more than 2 categories.It gives the probability of the input being in a particular class. It is mathematically defined as-

It’s main advantage is that it is able to handle the multiple classes.It is used typically in output layer of the neural network to classify the input in a particular class based on probability.

**3 Experimental setups and Analysis**

**3.1 Experimental Dataset**

This project uses Protein-Protein interaction dataset , a collection of Protein-Protein interactions along with the date of the discovery of each interaction.

Here proteins are considered as nodes, their biological interactions are considered as edges and interaction discovery date as edge’s timestamp.

There are total 16,458 nodes(Proteins) and 144,033 edges(interactions)

And the graph of this data is neither weighted nor directed.

This dataset consists of timestamp including day, month and year. Here we are working in a yearly granularity from 1970 to 2015.

Below is the descriptive table of the dataset,

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data set | Weighted | Directed | Nodes | Edges | Diameter | Date |
| PPI | - | - | 16,458 | 144033 | 10 | 1970-2015 |

This data is downloaded from the following source link <https://github.com/urielsinger/PPI>

**3.2 Experimental Analysis:**

**3.2.1 Dataset Preprocessing**

* In this PPI dataset there are total 16,458 nodes(Proteins) and 144,033 edges(interactions) having timestamp from 1970 to 2015.
* In the first step,the above data set is divided into training data and testing data at the ratio of 80:20.
* For train\_positive data set as the yearly granularity starts from 1970, the threshold start year is set as 1970 and the threshold year is 2013.
* Training\_positive data has 15,150 nodes and 1,15,555 edges along with their corresponding timestamps including day, month and year.
* In the second step, for testing data the threshold start year is set as 2014 and threshold year is 2015.
* Testing\_positive data consists of 5303 nodes and 11273 edges along with their corresponding timestamps including day, month and year.
* We create test and train raw datasets from our datasets,and we remove the node-pairs from test raw dataset which are also in train raw dataset.
* So the number of these such node-pairs(edges) came out to be 28478.
* We created train positive and train negative datasets as well as test positive and test negative datasets.
* The overall train dataset(including positive and negative) has edges and 1,15,555overall test dataset(including positive and negative) has **22546** edges.

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | No. of Nodes | No. of Edges | Time Granularity |
| Training | 15,150 | 1,15,555 | 1970-2013 |
| Testing | 5303 | 11,273 | 2014-2015 |

**3.2.2 Corpus Generation : *Time Aware user Embeddings***

In Natural Language processing, we can generate embeddings for words present in corpus using some state of the art techniques like **Word2Vec** .

But in this project we are dealing with nodes and their corresponding timestamps. To generate the embeddings for nodes we use the latest state of the art technique **Node2Vec.**

(reference : <https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf>)

Here in  **Node2Vec**  we are using Random walk algorithm for traversing across the graph created by the nodes and Alias sampling method is used to perform temporal walks for each node.

(reference: [https://hips.seas.harvard.edu/blog/2013/03/03/the-alias-method-efficient-sampling-with-many-discrete-outcomes/ )](https://hips.seas.harvard.edu/blog/2013/03/03/the-alias-method-efficient-sampling-with-many-discrete-outcomes/)

The hyperparameters used are:--

a) - **Sampling length(l)** ,which is taken as 100 in the experiment.

Sampling length usually tells us the length of the corresponding random walk.

b) - **Number of Samples(n) ,** which is taken from the set ,{1,2,5,10,20,30}.

Number of samples tells us the number of times ,a random walk starts from a given node.

The embedding results to be so obtained has a dimension of ,**128**(d = 128).

c) -**Sigma ,** which is basically a parameter for a kernel which is to be fixed according to our chunking.The value for sigma must be usually greater than or equal to the chunking which comes out to be in this case.

So ,sigma is taken as , **16,100**.

* The chunking is decided from the formula(which can be varied for getting better results):--

In this data set

threshold\_year = 2013

year = the current year that a data hold.

month = {1,2,3,4,5,6,7,8,9,10,11,12} **[as per the dataset]**

day = Day

* The value of *W*decreasesas the timestamp values moves towards the threshold year 2013.
* The Max value of *W* obtained using the above formula is
* The Min value is

As the value *Sigma* should be greater than or equal to the value of *W* ,

So the *Sigma* value is set as 16,100.

Here we use **six kernel functions** as **time decay functions**. The five time decay functions from [Reference :- <https://dl.acm.org/citation.cfm?id=3133060>**{Go to PDF}(section 3.2)**] along with a baseline function.

1. Gaussian Kernel ,
2. Triangle Kernel ,
3. Cosine(Hamming) Kernel ,
4. Circle Kernel ,
5. Passage Kernel and
6. Baseline **[in which the time decay is simply taken as 1(as per our experiment)]**

***Kernels:***

The value of time decay function increases as *w* decreases. Which means nodes with recent edges are given with more weights (priority).

For PPI Training dataset the values of time decay function using

Kernel equations along with their graphs are given below :

**Kernel 0:**

0 otherwise

For PPI Training dataset the values of time decay function using

Kernel 0 is shown in below graph:

1. **Gaussian Kernel :-**

0 otherwise

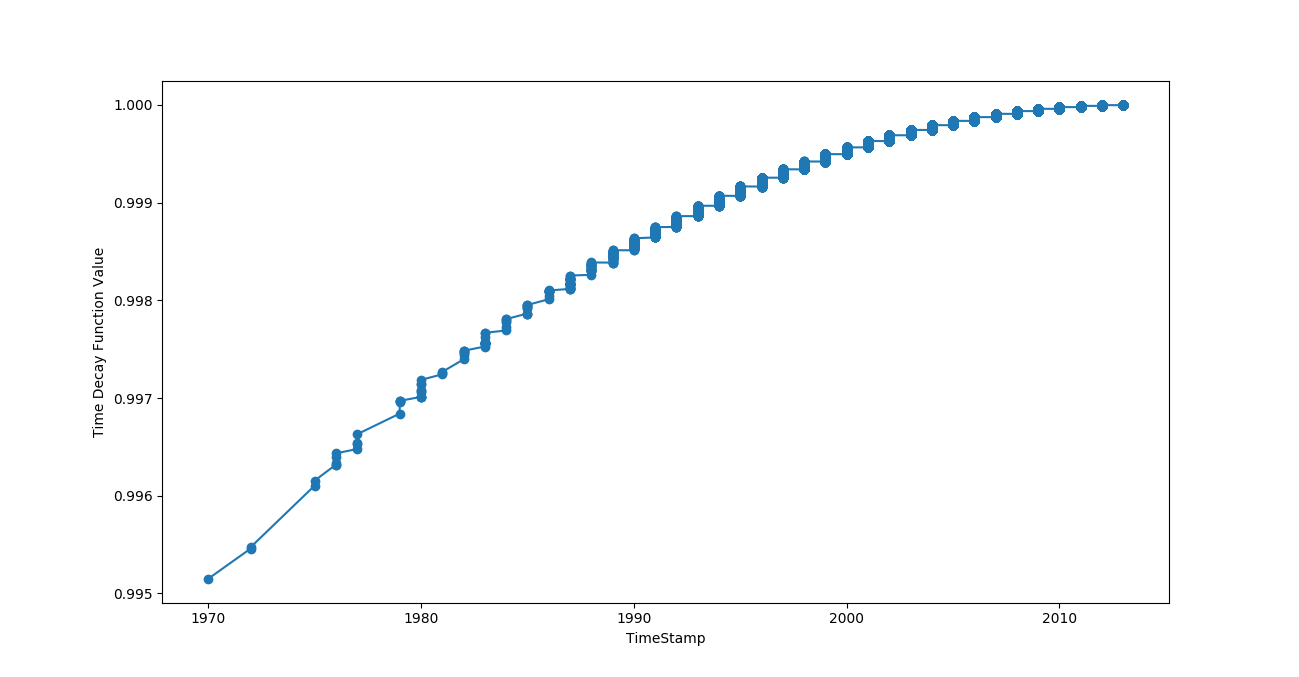


Fig: 3.2.2. 1 Timestamp Vs Time Decay function values using kernel 1

**2. Triangle Kernel :-**

0 otherwise

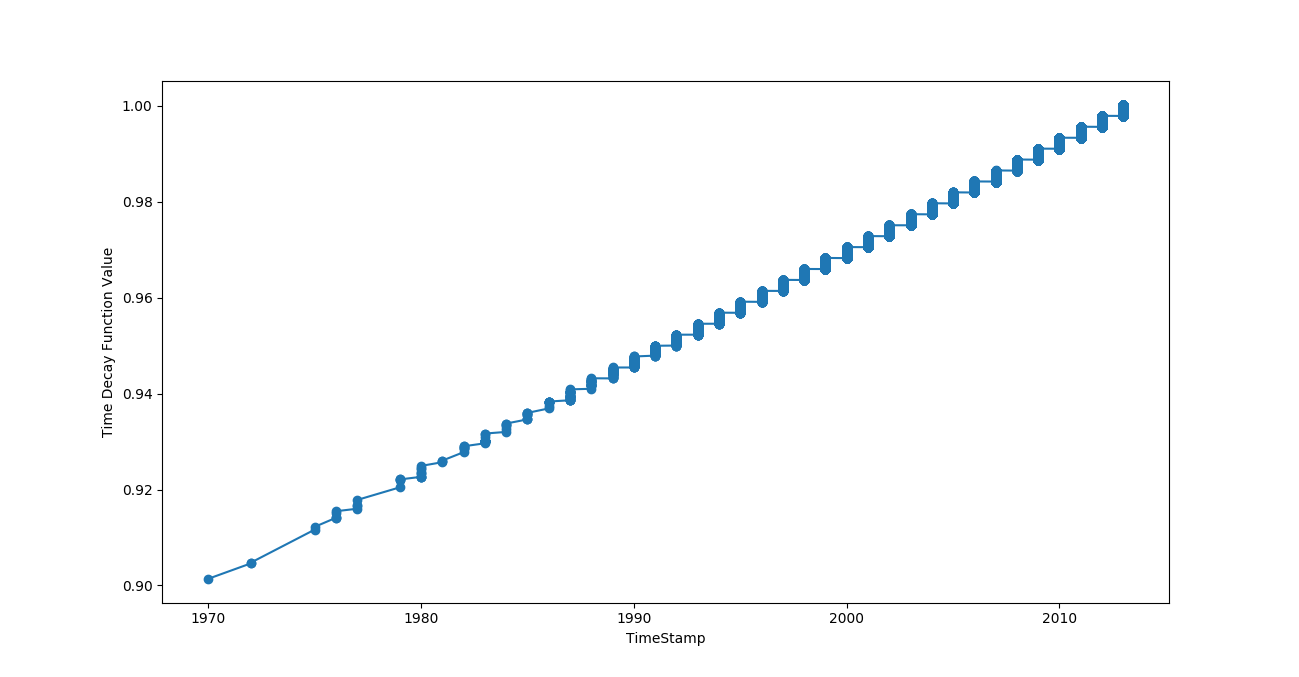


Fig: 3.2.2. 2 Timestamp Vs Time Decay function values using kernel 2.

**3. Cosine(Hamming) kernel :-**

0 otherwise

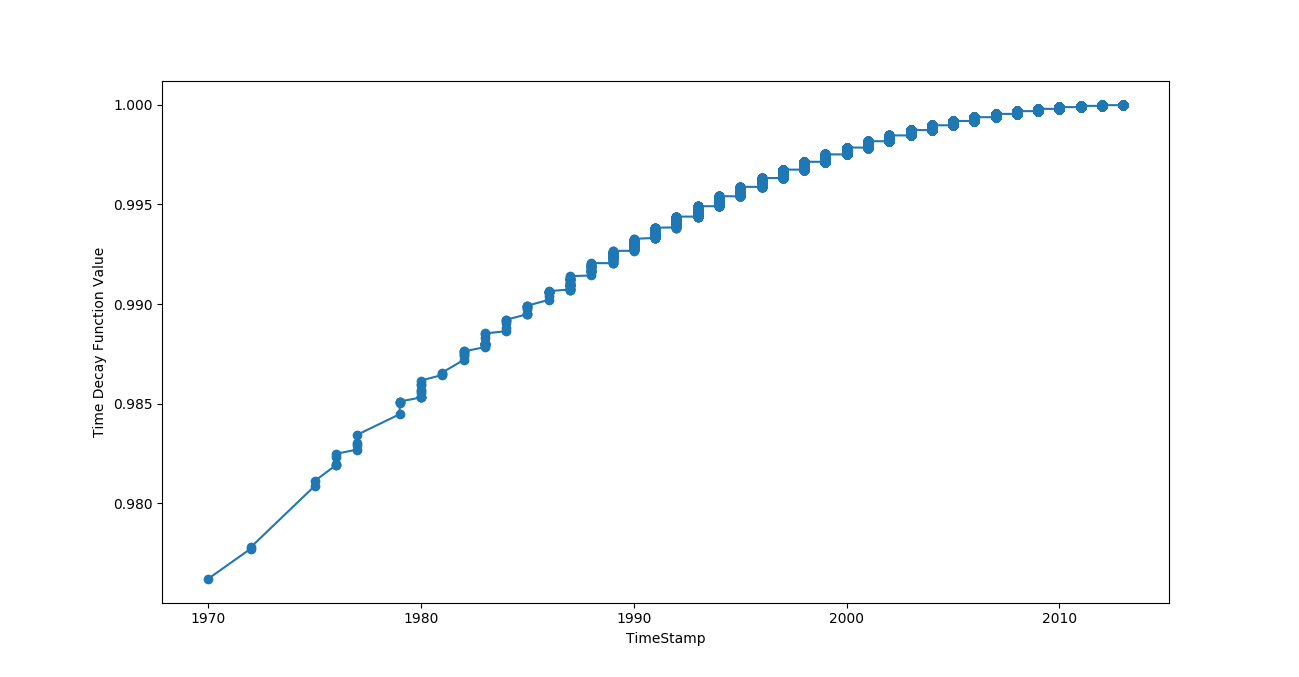


Fig: 3.2.2. 3 Timestamp Vs Time Decay function values using kernel 3

**4. Circle Kernel :-**

0 otherwise

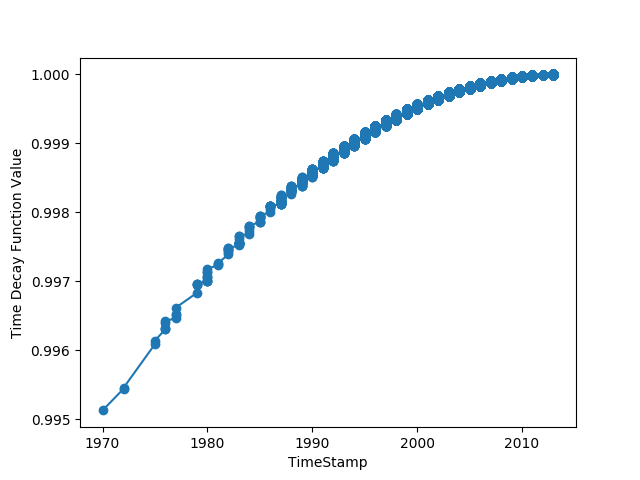


Fig: 3.2.2. 4 Timestamp Vs Time Decay function values using kernel 4

**5. Passage Kernel :-**

0 otherwise

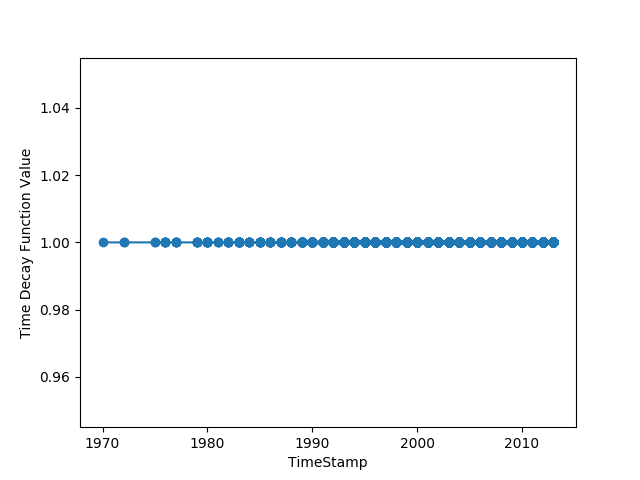
￼

Fig: 3.2.2. 5 Timestamp Vs Time Decay function values using kernel 5.

**6. Baseline Kernel :-**

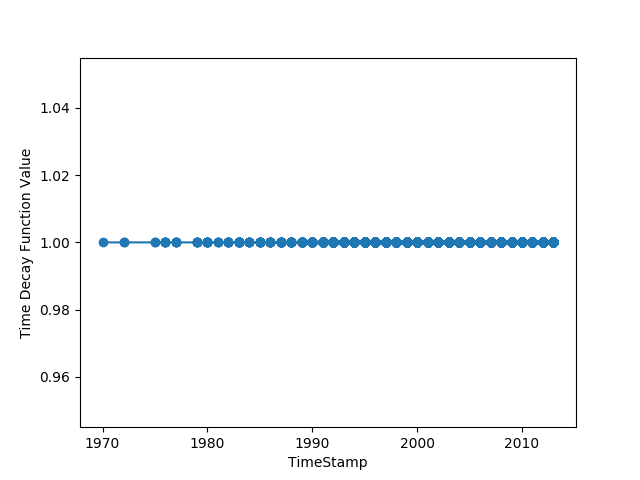
****

Fig: 3.2.2. 6 Timestamp Vs Time Decay function values using kernel 6.

**3.2.3 Embeddings for Nodes :**

* After generating corpus, we will generate the embeddings for each and every node having dimensions of **1 \* 128** (length = 128).
* This embeddings are generated by the most recent unsupervised algorithm , Word2Vec algorithm ***SkipGram model***, Using **Gensim’s** library.

Hyper parameters given for this Word2Vec model are:

1. Data set
2. Model = Skip gram(sg)
3. Sampling Methods: either a or b
4. Hierarchical Softmax = 1
5. Negative Sampling =5

3) Dimension of output embedding = 128

4) Window size = chosen from [1, 2, 5, 7, 10 ]

5) Min count = 1

**3.2.4 Edge Feature Generation**  :-

We got the embeddings for each node,

In order to model it as a **binary classification problem** , we are required to generate the corresponding **features of edges[edge-features]** between any two certain nodes.

Edge-features are generated for **four different Binary Operators**:--

a)- **Hadamard :--** If a and b are the two the two operands then this operator produces **a\*b** as a result.

b)- **Average :--** If a and b are the two the two operands then this operator produces (**a+b) / 2** as a result.

c)- **Weighted-L1 :--** If a and b are the two the two operands then this operator produces **abs(a - b)** as a result.

d)- **Weighted-L2 :--** If a and b are the two the two operands then this operator produces **pow(abs(a - b),2)** as a result.

[Reference :- <https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf>**(section 3.2.2)**]

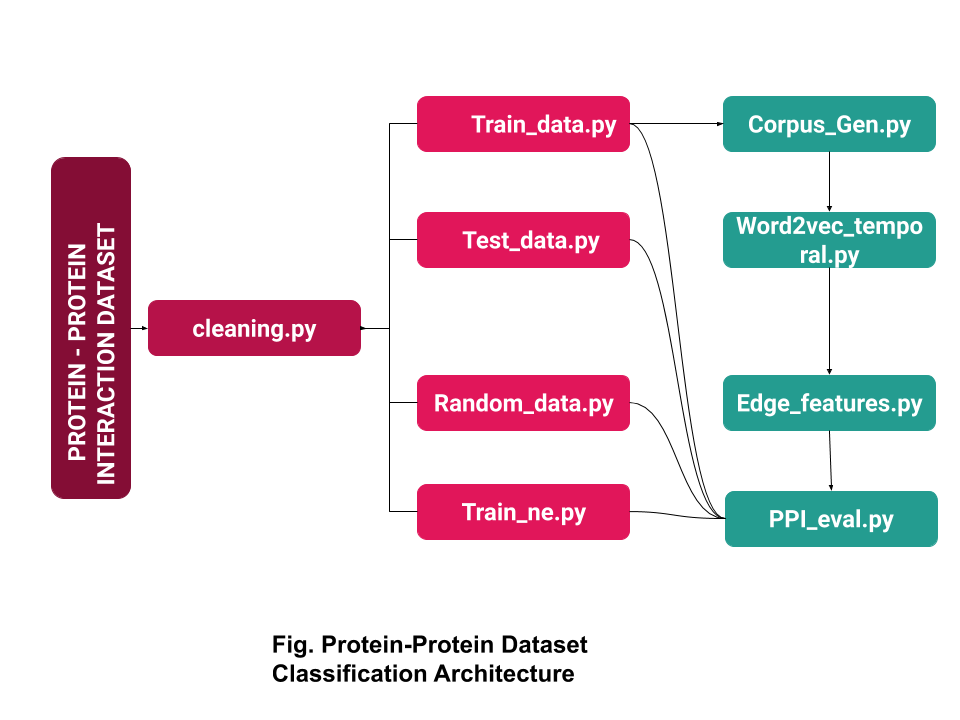
**3.3.5 Evaluation**

* To evaluate the quality of network embedding on predicting Protein-Protein interaction task, We used four state-of-art classifiers namely, Decision Tree (DT), Naive Bayes (NB), Random Forest (RF), and Logistic Regression (LR) on a random 80:20 train test split over combined test links (actual test links with randomly generated negative test links).
* We train those classifiers using the edge features generated using various binary operators, such as Hadamard, Average, L1 and L2 (by Paper ) for training data
* After evaluating with the test data the above mentioned classifiers generates the scores for F-1 Macro, F-1 Micro, AUC, Average precision and Accuracy.
* Finally the AUC score is considered to assess the performance of

Protein-Protein interaction prediction.

**Table of contents4. Model Architecture**

The below figure shows the architecture and data flow manner for PPI dataset.

****

**5. Results**

The below results obtained for Protein - Protein dataset in corresponding to these hyperparameters are as follows:-

**5.1 Hyperparameters:-**

1. Samples\_length = 100

2 Sigma = 16100

3. Dimension of vectors generated by Word2Vec = 128

And the Hyperparameters which we tuned are:

1. **Types of Kernel : Total 7**

|  |  |
| --- | --- |
| Kernel Name | Value |
| Kernel 0 | 0 |
| Gaussian | 1 |
| Triangle | 2 |
| Cosine | 3 |
| Circle | 4 |
| Passage | 5 |
| Baseline | 6 |

**2)** **No. Samples: Total 6 values [1, 2, 5, 10, 20, 30] .**

**3) window Size : Total 5 values [1 , 2, 5, 7, 10] .**

**4) Softmax in Word2Vec : Total 2 , hs = 1 and ns = 5.**

**5) Methods to generate feature embeddings for Edges: Total 4 values**

1. **Hadamard**
2. **Average**
3. **L1**
4. **L2**

**5.2: Total Number of Results:**

The total Number of results generated by the combinations of above hyperparameters are : 7 \* 6 \* 5 \* 2 \* 4 = 1,680

Among them we got the best results which are tabulated below:

**Table : 5.2.1** : **Best Kernel Results**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Method** | Softmax | Classifier | Kernel | Sample size | Window size | **F1-Macro** | **F1-Micro** | **AUC** | **Avg.**  **Precision** | **Accuracy** |
| Hadamard | HS | nb | 0 | 1 | 1 | 0.607134 | 0.609421 | 0.609421 | 0.697844 | 0.609421 |
| Average | NS | nb | 5 | 1 | 1 | **0.639162** | **0.641355** | **0.641355** | **0.594351** | **0.641355** |
| L1 | NS | nb | 5 | 1 | 1 | **0.639162** | **0.641355** | **0.641355** | **0.594351** | **0.641355** |
| L2 | NS | lr | 0 | 2 | 1 | **0.611047** | **0.611106** | **0.611106** | 0.706652 | **0.611106** |

**Table 5.2.2 : Best Baseline Results**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Method** | Softmax | Classifier | Kernel | Sample size | Window size | **F1-Macro** | **F1-Micro** | **AUC** | **Avg.**  **Precision** | **Accuracy** |
| Hadamard | NS | nb | 6 | 1 | 1 | **0.618448** | **0.625787** | **0.625787** | **0.708810** | **0.625787** |
| Average | NS | nb | 6 | 1 | 1 | **0.632779** | **0.635368** | **0.635368** | **0.589706** | **0.635368** |
| L1 | NS | lr | 6 | 10 | 1 | **0.615478** | **0.618868** | **0.618868** | **0.576831** | **0.618868** |
| L2 | NS | lr | 6 | 10 | 2 | **0.617950** | **0.618203** | **0.618203** | **0.573831** | **0.618203** |

**6. Conclusion:**

**6. 1**  **Comparison of Performance between** Kernels **and** Baseline**:**

Performance (P) is calculated using % , Kernel AUC values with respect to Baseline AUC value from the above tables 5.2.1 and 5.2.2 values

**P = (Kernel - Baseline) \* 100 / Baseline**

**Table 6.1.1 Performance:**

|  |  |  |  |
| --- | --- | --- | --- |
| **METHOD** | **KERNEL** | **BASELINE** | **P** |
| Hadamard | **0.609421** (HS) K=1 | **0.625787** | -2.61492 |
| Average | **0.641355** | **0.635368** | 0.9422886 |
| L1 | **0.641355** | **0.618868** | 3.63357 |
| L2 | **0.611106** | **0.618203** | -1.148005 |

By analysing the above table we can conclude that

* For Methods Average and L1 Kernel 5 is performing better than Baseline kernel by **0.9422886** and **3.633357** and are giving good performance.
* Where as for Methods Hadamard and L2 Baseline kernel is better than other kernels by 2.61492 and 1.148005.
* In Node2Vec generation SkipGram model using Softmax,Negative Sampling = 5 is worked better than Hierarchical Softmax.
* Naive Bayes (NB) and Logistic regression (LR) classifiers working good better than Decision tree(DT) and Random Forest (RF)

**References:**

1. Wikipedia <https://en.wikipedia.org/wiki/Machine_learning>.
2. Akash Anil, Uppinder Chugh, and Sanasam Ranbir Singh in On Applying Meta-path for Network Embedding in Mining Heterogeneous DBLP Network <https://arxiv.org/abs/1808.04799>
3. For dataset <https://github.com/urielsinger/PPI>

4. For kernels <https://dl.acm.org/citation.cfm?id=3133060>{Go to Pdf and

Open section 3.2}

5. For Node2vec <https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf>)

6. For Alias sampling

[https://hips.seas.harvard.edu/blog/2013/03/03/the-alias-method-efficient-](https://hips.seas.harvard.edu/blog/2013/03/03/the-alias-method-efficient-sampling-with-many-discrete-outcomes/)

[sampling-with-many-discrete-outcomes/ )](https://hips.seas.harvard.edu/blog/2013/03/03/the-alias-method-efficient-sampling-with-many-discrete-outcomes/)

7. For edge feature generation.

[ <https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf>**(section 3.2.2)**]