# Node Embedding Algorithms

(From S-VGAE paper)

Conjoint Triad https://github.com/MahaAmin/Conjoint-Triad

(From SkipGNN paper) https://github.com/shenweichen/GraphEmbedding

Node2vec (change walk length to 20 from default 80)

DeepWalk https://github.com/phanein/deepwalk

Struc2Vec

LINE: Large-Scale Information Network Embedding

SDNE: Structural Deep Network Embedding

(From DeepPPI paper)

iFeature https://github.com/Superzchen/iFeature

Amino acid composition (AAC)

Dipeptide composition (DPC)

Composition, Transition and Distribution (C/T/D)

Quasi-sequence-order descriptors (QSOrder)

Amphiphilic Pseudoamino Acid Composition (APAAC)

## Find best performing among graph embedding then combine with protein embedding

## Graph Embedding

(Results from 10-fold cross validation) Average scores

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Method | Accuracy | Sensitivity  (Recall) | Specificity | Precision | F1-Score | ROC-AUC | PR-AUC |
| One-hot | 0.7020 | 0.609 | 0.7963 | 0.7525 | 0.6561 | 0.8082 | 0.6966 |
| Node2Vec | 0.6809 | 0.5222 | 0.8402 | 0.7633 | 0.6151 | 0.8245 | 0.7089 |
| Deepwalk | 0.6878 | 0.5056 | 0.8708 | 0.8038 | 0.6129 | **0.8258** | **0.7281** |
| Struc2Vec |  |  |  |  |  |  |  |
| **LINE** | 0.7182 | 0.6341 | 0.8019 | 0.7597 | 0.6848 | 0.8144 | 0.6939 |

## Protein Embedding

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Method | Accuracy | Sensitivity | Specificity | Precision | F1-Score | ROC-AUC | PR-AUC |
| Conjoint Triad (CT) |  |  |  |  |  |  |  |
| AAC |  |  |  |  |  |  |  |
| DPC |  |  |  |  |  |  |  |
| C/T/D |  |  |  |  |  |  |  |
| APAAC |  |  |  |  |  |  |  |

4 (Graph embedding) \* 5 (Protein Embedding) = 20 combinations

10-fold average

Try on both Skip-GNN and S-VGAE

Compare results

## Protein-Protein Interactions Prediction Based on Graph Energy and Protein Sequence Information

Multi-information fusion

Graph energy: Physicochemical graph energy based on the ionization equilibrium constant and isoelectric point + contact graph energy based on the contact information of amino acids

Dipeptide composition method (DPC) –used for order information of amino acids

Principal component analysis (PCA) – implemented for eliminating noise and a robust weighted sparse representation-based classification (WSRC) classifier

## Deep Neural Network and Extreme Gradient Boosting Based Hybrid Classifier for Improved Prediction of Protein-Protein Interaction (DNN-XGB)

Fusion of three **sequence-based** (AAC – CT – Local Descriptor (LD))

Local Descriptor (LD)

## Combining sequence and network information to enhance protein-protein interaction prediction

**Sequence**: one-hot encoding for both amino acids

Each amino acid can be represented by a 20-dimensional vector consisting of 0 and 1

**Structure**: proteins’ position information

adjacency matrix and degree matrix of the graph

*first-order neighbor’s feature information*, which contains the position information of the protein in the PPIs networks graph

# iFeature

# Feature selection (Information gain) 🡪 Select protein embedding based on dataset

Dimensionality reduction – Principal component analysis (PCA)

### Grouped amino acid composition

* 20 amino acid types are first categorized according to their physicochemical properties, and then the composition of each category is calculated

### Composition, Transition and Distribution (C/T/D)

*References:*

*Dubchak,I. et al. (1995) Prediction of protein folding class using global description of amino acid sequence*

*Dubchak,I. et al. (1999) Recognition of a protein fold in the context of the Structural Classification of Proteins (SCOP) classification*

* Describe the global composition of a given amino acid property in a protein,
* The frequencies with which the property changes along the entire length of the protein,
* And the distribution pattern of the property along the sequence respectively

Amino acids are divided into three classes according to **attribute** (e.g. hydrophobicity, normalized van der Waals volume, polarity and polarizability), and each amino acid is encoded by one of the indices 1, 2, 3 according to which class it belongs.

Composition

* Global percent for each encoded class in the sequence

(Dimension = 3 classes x 11 attributes)

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Class 1** | **Class 2** | **Class 3** |
| hydrophobicity\_PRAM900101 | RKEDQN | GASTPHY | CLVIMFW |
| hydrophobicity\_ARGP820101 | QSTNGDE | RAHCKMV | LYPFIW |
| hydrophobicity\_ZIMJ680101 | QNGSWTDERA | HMCKV | LPFYI |
| hydrophobicity\_PONP930101 | KPDESNQT | GRHA | YMFWLCVI |
| hydrophobicity\_CASG920101 | KDEQPSRNTG | AHYMLV | FIWC |
| hydrophobicity\_ENGD860101 | RDKENQHYP | SGTAW | CVLIMF |
| hydrophobicity\_FASG890101 | KERSQD | NTPG | AYHWVMFLIC |
| Normalized Van Der Waals Volumne | GASTPDC | NVEQIL | MHKFRYW |
| Polarity | LIFWCMVY | PATGS | HQRKNED |
| Polarizability | GASDT | CPNVEQIL | KMHFRYW |
| Charge | KR | ANCQGHILMFPSTWYV | DE |
| Secondary structure | EALMQKRH | VIYCWFT | GNPSD |
| Solvent Accessibility | ALFCGIVW | RKQEND | MSPTHY |

Transition

* Transition from

class 1 to 2 (percent frequency with which 1->2 or 2->1)

class 1 to 3 (1->3 or 3->1)

class 2 to 3 (2->3 or 3->2)

Distribution (Dimension: 39 x 5 = 195)

* Distribution of each attribute in the sequence (class 1, 2 and 3 for each attribute)

5 “distribution” descriptors for each attribute

Position percent in the whole sequence for the first residue

(first, 25%, 50%, 75%, 100% residues respectively, for a specified encoded class)

### Quasi-Sequence-Order Descriptors

Sequence-order-coupling number (SOCNumber) Dimension: 60

### Grouped Amino Acid Composition (GAAC)

Physicochemical properties -> Composition of each category

|  |  |
| --- | --- |
| **Physicochemical properties** | **Amino Acids** |
| Aliphatic (non-polar and hydrophobic) | A, G, I, L, P, V |
| Aromatic | F, H, W, Y |
| Pos Charge | K, R |
| Neg Charge | D, E |
| Uncharge (Neutral) | ANCQGHILMFPSTWYV |

### Grouped Dipeptide Composition (GDPC)

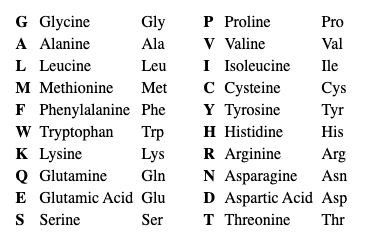
* Sliding window size = 2

E.g. (Aliphatic, Aliphatic) = AG, (Aliphatic, Aromatic) = AF, (Aliphatic, Pos Charge) = AK, (Aliphatic, Neg Charge) = AD, (Aliphatic, Uncharge) = AA

### Grouped Tripeptide Composition (GTPC)

* Sliding window size = 3

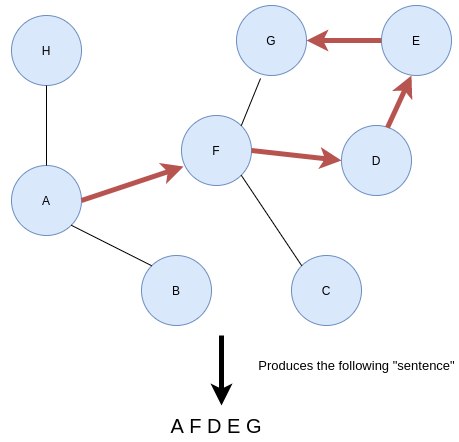
E.g. (Aliphatic, Aliphatic, Aliphatic) = AGI, (Aliphatic, Aliphatic, Aromatic) = AGF, (Aliphatic, Aliphatic, Pos Charge) = AGK, (Aliphatic, Aliphatic, Neg Charge) = AGD, (Aliphatic, Aliphatic, Uncharge) = AGA

Node2Vec (Node embedding algorithm)

*(Ref:* [*https://towardsdatascience.com/complete-guide-to-understanding-node2vec-algorithm-4e9a35e5d147*](https://towardsdatascience.com/complete-guide-to-understanding-node2vec-algorithm-4e9a35e5d147)*)*

*Source code: https://github.com/aditya-grover/node2vec*

* Resulting node embeddings are only affected by the network topology and not the node properties or attributes
* **Need the whole graph to be available** to learn the node embeddings
  + Cannot run separately on train and test data
  + When a new node is added to the graph, will need to re-run the node2vec algorithm on the whole graph to generate embedding for the new node
* Inspired by the skip-gram model
* Node2Vec = Skip-gram with Negative sampling (SGNS)
* Using random walks to generate a corpus of “sentences” from a given network



1. Start traversing the graph from a ‘start’ node (Node ‘A’ in this case)
2. Pick a neighboring node at **Random** and hop onto it
3. Repeat the process until a pre-defined **walk length** (defines how long the “sentences” will be)
4. For every node in the graph, node2vec algorithm generates a series of random walks with the particular node as the starting node

**Walks per node**: defines how many random walks should start from a particular node (i.e., how many “sentences” to generate starting from each node in the graph)

1. After “sentences” are generated using random walks, the algorithm inputs them into the skip-gram with negative sampling (SGNS) model and retrieve the hidden layer weights as node embeddings
2. Node2vec implements **second-order biased random walks (takes into account both the current and previous state)**

🡪 1st-order random walks: probability of returning to a previous node or any other node is equal

2nd-order: likelihood of backtracking the walk and immediately revisiting a node in the walk is controlled by the *return parameter p.*

(*higher p-value* = lower chance of revisiting a node. 🡪 Encourages moderate graph exploration

*Low p-value* = higher chances of backtracking in the walk 🡪 Keeps the random walk closer to the ‘start’ node)

Skip-Gram: Tries to predict the source context words (surrounding words) given a target word (centre word)

Word Vector (row of real-valued numbers): Represent words as multidimensional continuous floating point numbers where semantically similar words are mapped to proximate points in geometric space

Semantics of the word are embedded across the dimensions of the vector

(semantically similar words have similar vectors, words that are used in a similar context will be mapped to a proximate vector space)

Using *mathematical operators* on vectors (e.g. addition and subtraction) to create new word vectors:

E.g. *(king – man) + woman = queen*, where new word vector [*(king – man) + woman*] maps most closely to the word vector for [*queen*]

Input: One-hot encoded vector representing the input word

Output: One-hot encoded vector representing the context word

Goal: Learn weights of the hidden layer (i.e., word embedding)

Number of neurons in the hidden layer determines the embedding dimension (i.e, size of the vector representing each word in the vocabulary)

Influence of context ‘window size’ parameter:

*(Ref: Dependency-Based Word Embeddings by Levy & Goldberg)*

Larger 🡪 tends to capture more topic/domain information

Smaller 🡪 tends to capture more information about the word itself (e.g., what other words are functionally similar)

Negative sampling

1. Each training sample will update all of the weights in the neural network
2. Updating thousands of weights for every input context training pair is very expensive
   1. Negative sampling solves this performance issue by having each training sample modify only a small subset of the weights rather than all of them

Node2vec 🡪 builds on Deepwalk

(Is there difference in implementation between 2 papers)

Uses biased random walk based on depth/breath first search to consider both local and global network structure

20 walk lengths 🡪 longer walk lengths improve the embedding quality

# Deepwalk

*(https://towardsdatascience.com/deepwalk-its-behavior-and-how-to-implement-it-b5aac0290a15)*

Co-interactions within graphs can be captured and encoded by simple neural networks into embeddings

* Relies heavily on the Word2Vec language model (Skip-Gram algorithm)

Performs uniform distributed random walk and applies skip-gram model to learn a node-embedding (provide insights into localized structures within networks)

Random path-making through graphs to reveal latent patterns in the network

* These patterns are then learned and encoded by neural networks to yield the final embeddings

20 walk lengths

Different parameter configurations