CHAPTER 1

INTRODUCTION

This chapter includes objective of the project, the problems to be solved, purpose, scope and challenges to be solved by the project and organization of the report.

1.1 OVERVIEW

Link prediction in online social networks is used to determine new interactions among its members which are likely to occur in the future. Link prediction in the co-authorship network has been regarded as one of the main targets in link prediction researches so far. Researchers have focused on analyzing and proposing solutions to give efficient recommendation for authors who can work together in a science project. In order to give precise prediction of links between two ubiquitous authors in a co-authorship network, it is preferable to design a similarity metric between them and then utilizing it to determine the most possible co-author(s).

However, the relevant researches did not regard the integration of paper's content in the metric itself. This is important when considering the collaboration between scientists since it is possible that authors having same research interests are more likely to have a joint paper than those in different researches. In this paper, we propose a new metric for link prediction in the co-authorship network based on the content similarity as well as structural similarity. Mathematical notions of the link prediction in the co-authorship network and a link prediction algorithm are proposed. The new metric is experimentally validated on the public bibliographic collection.

1.2 OBJECTIVES

- The objective of the project "Co-Author Link prediction" is to determine an unlinked author in a co-author network using both structural and semantic similarity.

- Number of publications contain multiple authors, in order to predict future co-authors, we use three major approaches: Co-word analysis, co-citation analysis, and co-author analysis.

- Co-citation analysis involves searching for groups of references that are commonly cited together, co-word analysis seeks similarities in the frequency of words used in articles to define themes (cluster of words), and co-author analysis creates a network of interactions between authors and their co-authors.

1.3 PROBLEM STATEMENT

● Number of publications contain multiple authors, In order to predict future co-authors, we use three major approaches: Co-word analysis, co-citation analysis, and co-author analysis.

● Co-citation analysis involves searching for groups of references that are commonly cited together, co-word analysis seeks similarities in the frequency of words used in articles to define themes (cluster of words), and co-author analysis creates a network of interactions between authors and their co-authors.

1.4 ORGANIZATION OF THE REPORT

In this report, **Chapter 2** gives background on interactive discussion of some related works. **Chapter 3** describes the overall system design, algorithm and detailed description about each module. **Chapter 4** describes about dataset used for the implementation of the proposed framework, results of the experiments conducted, test cases, performance evaluation and comparison of the results of proposed system with existing system. **Chapter 5** discusses about the conclusion and scope for future work and finally the references.

CHAPTER 2

LITERATURE REVIEW

This chapter discusses the various ideas, methodologies that have incorporated in this project.

2.1 SIMILARITY METRICS FOR LINK PREDICTION

Pham Minh Chuan (2017) proposed the existing similarity metric for the link prediction problem. Most of the network topology-based link prediction methods make use of node similarities on neighborhood. The main assumption is that, the larger the numbers of common neighbors between two nodes are, more likely these nodes are able to connect in the future. Therefore, there are two main groups: the unweighted similarity scores and the weighted similarity scores. In what follows, we present some basic metrics in each group.

2.2 UNWEIGHTED SIMILARITY SCORES

Anderson Matos Medina (2018) had a good amount of studies on Link Prediction in social network. Most of the traditional link prediction algorithms use the common neighbors approach to measure the similarity.1) adamic-adar, 2) preferential attachment, 3) Jaccard coefficient, 4) common neighbor.

2.3 PREDICTORS OF INTERACTIONS AMONG AUTHORS

Dai T (2017) proposed the similarity of lines of thought between authors was measured using the similarity of words in the articles titles of author A is comparison with those of author B. This approach is based on the assumption that the words used in a text convey ideas; therefore, a set of words that co-occurs in a text should represent a line of thought. The same assumption underlies the use of co-word analysis. To measure word similarity, a list containing the titles of all publications was compiled and punctuations, numbers, and stop words were removed from the title using package tm. Furthermore, the 200 most frequent words were synonymized by removing plural suffixes manually (e.g.: The term interactions were synonymized to interaction). After cleaning the text for each time period, Sorensen similarity was calculated between the words used in all articles of author A (including articles not published with co-authors) and author B (including articles not published with co-authors).

2.4 PROPOSED LINK PREDICTION METHOD IN STOCHASTIC GRAPHS

Tien TN (2017) proposed a link prediction method based on learning automata (SLP) is proposed for stochastic social networks under the situation that the probability distribution functions of the weights associated with the links of the graph are unknown. The proposed algorithms take advantage of using learning automata to estimate the distribution of some chosen similarity metric in order to predict future links.

2.5 SEMANTIC SIMILARITY

S. Mahalakshmi (2018) proposed the generation of Deep Author Topic Models (DATMs) which completely represent a given author. DATMs are generated using Deep Hierarchical Topic Model (DSA-H ATM). A baseline comparison using traditional topic models is also presented.

2.6 COMMUNITY DETECTION

Zhu L (2017) proposed the Community detection has been considered as an important task in network analysis. Many methods have been proposed to explore community through exploring networks structure. During them, nonnegative matrix factorization method (NMF) is the most popular and widely used approaches. Wang et al. (2011) applied NMF to discovery community on undirected network, directed network and compound network. Lin et al. (2011) proposed a framework that extracts community structures from social media data. They modeled multi-relational social data as multiple conjunct data tensors and used meta-graph factorization method to extract latent communities. In order to take advantage of prior information, some research modified adjacency matrix of network by adding various constraints.

CHAPTER 3

SYSTEM DESIGN

This chapter discusses the overall system architecture and detailed description of all modules.

3.1 PROPOSED SYSTEM

The proposed system of the project is to predict link between co-authors using structural and semantic similarity of the co-author. In structural similarity you have to collect adamic-adar resource allocation index, Jaccard-coefficient and preferential attachment score, after that community need to be detected from the community common neighbor resource allocation and inter cluster common neighbor score are collected and these are combined scores are given to linear regression algorithm.

For semantic similarity abstract of the authors are collected and their abstracts are preprocessed and three features are related

- Number of Overlapping words used in abstract
- Temporal distance between the papers
- Number of co-authors

These features are given to classification algorithm SVM.

3.2 BLOCK DIAGRAM

Figure 3.1 describes the overall system of the project where the details of the module used for semantic and structural similarity in order to predict the link between the co-authors.

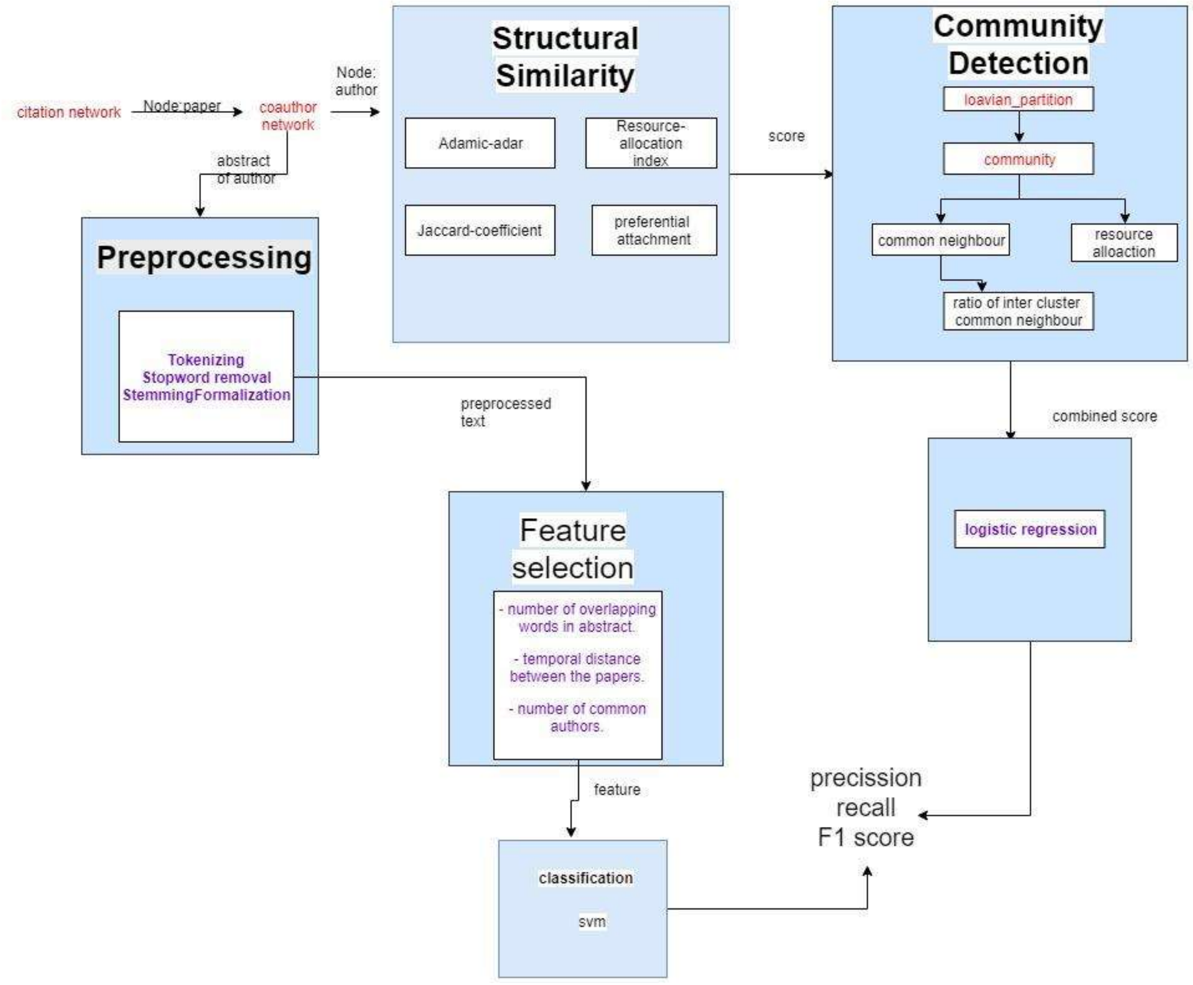


Fig 3.1 System Architecture

3.3 LIST OF MODULES

This section elaborately describes about the various modules of the proposed system. It also provides the algorithm details of each modules and depicts the snapshot of the results obtained. The modules of the proposed system are:

- Structural Similarity
- Community Detection
- Linear Regression
- Data Preprocessing
- Feature Extraction
- Classification using SVM

3.3.1 Structural Similarity

RESOURCE ALLOCATION INDEX:

Compute the resource allocation index of all node pairs in ebunch.

Parameters

G: graph

A NetworkX undirected graph.

ebunch: iterable of node pairs, optional (default = None) Resource allocation index will be computed for each pair of nodes given in the iterable. The pairs must be given as 2-tuples (u, v) where u and v are nodes in the graph. If ebunch is None then all non-existent edges in the graph will be used.

Default Value: None

Returns:

piter: iterator

An iterator of 3-tuples in the form (u, v, p) where (u, v) is a pair of nodes and p is their resource allocation index.

Examples

```
>>> import networkx as nx
>>> G = nx.complete_graph(5)
>>>preds = nx.resource_allocation_index(G, [(0, 1), (2, 3)])
>>> for u, v, p in preds:
  ...    '(%d, %d) -> %.8f' % (u, v, p)
  ...
  '(0, 1) -> 0.75000000'
  '(2, 3) -> 0.75000000'
```

Jaccard coefficient:

Compute the Jaccard coefficient of all node pairs in ebunch.

Parameters

G: graph

A NetworkX undirected graph.

ebunch: iterable of node pairs, optional (default = None) Jaccard coefficient will be computed for each pair of nodes given in the iterable. The pairs must be given as 2-tuple (u, v) where u and v are nodes in the graph. If ebunch is None then all non-existent edges in the graph will be used.

Default value: None.

Returns

piter: iterator An iterator of 3-tuples in the form (u, v, p) where (u, v) is a pair of nodes and p is their Jaccard coefficient.

Examples:

```
>>> import networkx as nx
>>> G = nx.complete_graph(5)
>>>preds = nx.jaccard_coefficient(G, [(0, 1), (2, 3)])
>>> for u, v, p in preds:
...    '(%d, %d) -> %.8f' % (u, v, p)
...
'(0, 1) -> 0.60000000'
'(2, 3) -> 0.60000000'
```

Adamic-Adar index:

Compute the Adamic-Adar index of all node pairs in ebunch.

Parameters

G: graph

NetworkX undirected graph.

ebunch: iterable of node pairs, optional (default = None) Adamic-Adar index will be computed for each pair of nodes given in the iterable. The pairs must be given as 2-tuples (u, v) where u and v are nodes in the graph. If ebunch is None then all non-existent edges in the graph will be used.

Default value: None.

Returns

piter: iterator

An iterator of 3-tuples in the form (u, v, p) where (u, v) is a pair of nodes and p is their Adamic-Adar index.

Examples:

```
>>> import networkx as nx
>>> G = nx.complete_graph(5)
>>>preds = nx.adamic_adar_index(G, [(0, 1), (2, 3)])
>>> for u, v, p in preds:
...    '(%d, %d) -> %.8f' % (u, v, p)
...
'(0, 1) -> 2.16404256'
```

```
'(2, 3) -> 2.16404256'
```

Preferential Attachment:

Compute the preferential attachment score of all node pairs in ebunch.

Parameters:

G: graph

NetworkX undirected graph.

ebunch: iterable of node pairs, optional (default = None) Preferential attachment score will be computed for each pair of nodes given in the iterable. The pairs must be given as 2-tuples (u, v) where u and v are nodes in the graph. If ebunch is None then all non-existent edges in the graph will be used.

Default value: None.

Returns

piter: iterator

An iterator of 3-tuples in the form (u, v, p) where (u, v) is a pair of nodes and p is their preferential attachment score.

Examples

```
>>> import networkx as nx
>>> G = nx.complete_graph (5)
>>>preds = nx.preferential_attachment(G, [(0, 1), (2, 3)])
>>> for u, v, p in preds:
```

```
...    '(%d, %d) -> %d' % (u, v, p)
...
'(0, 1) -> 16'
'(2, 3) -> 16'
```

ALGORITHM:

Input:

G: graph

A NetworkX undirected graph.

Output:

An iterator of 3-tuples in the form (u, v, p) where (u, v) is a pair of nodes and p is their resource allocation index, Adamic-Adar index, preferential attachment score, Jaccard coefficient.

Steps:

Step 1: Graph is read in the form of adjacent list graph using networkx package.

Step 2: Following scores are calculated for non-linked authors

- Jaccard index:

  $\text{Jaccard}(a, b) = |\Gamma(a) \cap \Gamma(b)| / |\Gamma(a) \cup \Gamma(b)|$

- common neighbour:

  $CN(a, b) = |\Gamma(a) \cap \Gamma(b)|$
- preferential attachment:

  $PA(a, b) = k(a) \times k(b)$.
- adamic adar index:

  $AAI(a, b) = \sum z \in \Gamma(a) \cap \Gamma(b)\ 1/\log kz$.

where z is a common neighbor to nodes both a and b and k is the degree of node z.

Step3: Combine all scores together.

3.3.2 Community Detection

Louvain algorithm:

- The method consists of two phases.
- It looks for "small" communities by optimizing modularity in a local way.
- It aggregates nodes of the same community and builds a new network whose nodes are the communities.
- These steps are repeated iteratively until a maximum of modularity is attained.
- The output of the program therefore gives several partitions. The partition found after the first step typically consists of many communities of small sizes.
- At subsequent steps, larger and larger communities are found due to the aggregation mechanism.
- This process naturally leads to hierarchical decomposition of the network.

Functions for computing and measuring community structure.

The functions in this class are not imported into the top-level networkx namespace.

- access these functions by importing the **networkx.algorithms.community** module, then accessing the functions as attributes of community

Partitions via centrality measures

Functions for computing communities based on centrality notions.

girvan_newman (G [, most_valuable_edge]) -> Finds communities in a graph using the Girvan–Newman method.

Validating partitions

Helper functions for community-finding algorithms.

is_partition (G, communities) --> return True if and only if communities is a partition of the nodes of **G**

Parameters

G: graph
NetworkX undirected graph.

Examples

```
>>> import networkx as nx

>>> from networkx.algorithms import community

>>> G = nx.barbell_graph(5, 1)
```

```
>>>communities_generator = community.girvan_newman(G)

>>>top_level_communities = next(communities_generator)

>>>next_level_communities = next(communities_generator)

>>>sorted(map(sorted, next_level_communities))

[[0, 1, 2, 3, 4], [5], [6, 7, 8, 9, 10]]
```

Common Neighbors:

- Count the number of common neighbors of all node pairs in ebunch using community information.
- For two nodes `u` and `v`, this function computes the number of common neighbors and bonus one for each common neighbor belonging to the same community as `u` and `v`.

Parameters

G : graph

A NetworkX undirected graph.

ebunch: iterable of node pairs, optional (default = None) The score will be computed for each pair of nodes given in the iterable.The pairs must be givCompute the resource allocation index of all node pairs in ebunch using community information.

For two nodes `u` and `v`, this function computes the resource allocation index considering only common neighbors belonging to the same community as `u` and `v`.en as 2-tuples (u, v) where u and v are nodes in the graph. If ebunch is None then all non-existent edges in the graph will be used. Default value: None.

Community: string, optional (default = 'community') Nodes attribute name containing the community information. G[u][community] identifies which community u belongs to. Each node belongs to at most one community.

Default Value: 'Community'

Returns

piter: iterator

An iterator of 3-tuples in the form (u, v, p) where (u, v) is a pair of nodes and p is their score.

Examples:

```
>>> import networkx as nx

>>> G = nx.path_graph(3)

>>>G.node[0]['community'] = 0

>>>G.node[1]['community'] = 0

>>>G.node[2]['community'] = 0

>>>preds = nx.cn_soundarajan_hopcroft(G, [(0, 2)])

>>> for u, v, p in preds:

...    '(%d, %d) -> %d' % (u, v, p)

...
```

'(0, 2) -> 2'

Resource Allocation Index:

G: graph

A NetworkX undirected graph.

ebunch: iterable of node pairs, optional (default = None) The score will be computed for each pair of nodes given in the iterable. The pairs must be given as 2-tuples (u, v) where u and v are nodes in the graph. If ebunch is None then all non-existent edges in the graph will be used.

Default value: None.

community: string, optional (default = 'community') Nodes attribute name containing the community information. G[u][community] identifies which community u belongs to. Each node belongs to at most one community.

Returns

piter: iterator

An iterator of 3-tuples in the form (u, v, p) where (u, v) is a pair of nodes and p is

Examples

```
>>> import networkx as nx

>>> G = nx.Graph()
>>> G.add_edges_from([(0, 1), (0, 2), (1, 3), (2, 3)])

>>>G.node[0]['community'] = 0

>>>G.node[1]['community'] = 0

>>>G.node[2]['community'] = 1

>>>G.node[3]['community'] = 0

>>>preds = nx.ra_index_soundarajan_hopcroft(G, [(0, 3)])

>>> for u, v, p in preds:

...    '(%d, %d) -> %.8f' % (u, v, p)

...

'(0, 3) -> 0.50000000'
```

Ratio of within- and inter-cluster:

- Compute the ratio of within- and inter-cluster common neighbors of all node pairs in ebunch.

- For two nodes `u` and `v`, if a common neighbor `w` belongs to the same community as them, `w` is considered as within-cluster common neighbor of `u` and `v`. Otherwise, it is considered as inter-cluster common neighbor of `u` and `v`. The ratio between the size of the set of within- and inter-cluster common neighbors is defined as the WIC measure.

Parameters

G: graph

A NetworkX undirected graph.

ebunch: iterable of node pairs, optional (default = None) The WIC measure will be computed for each pair of nodes given in the iterable. The pairs must be given as 2-tuples (u, v) where u and v are nodes in the graph. If ebunch is None then all non-existent edges in the graph will be used.

Default value: None.

delta: float, optional (default = 0.001) Value to prevent division by zero in case there is no inter-cluster common neighbor between two nodes. Default value: 0.001.

community: string, optional (default = 'community') Nodes attribute name containing the community information. G[u][community] identifies which community u belongs to. Each node belongs to at most one community.

Default value: 'community'.

Returns

piter: iterator

An iterator of 3-tuples in the form (u, v, p) where (u, v) is a pair of nodes and p is their WIC measure

Examples

```
>>> import networkx as nx

>>> G = nx.Graph()
>>> G.add_edges_from([(0, 1), (0, 2), (0, 3), (1, 4), (2, 4), (3, 4)])

>>>G.node[0]['community'] = 0

>>>G.node[1]['community'] = 1

>>>G.node[2]['community'] = 0

 >>>G.node[3]['community'] = 0

>>>G.node[4]['community'] = 0

>>>preds = nx.within_inter_cluster(G, [(0, 4)])

>>> for u, v, p in preds:

...    '(%d, %d) -> %.8f' % (u, v, p)

…

'(0, 4) -> 1.99800200'

>>>preds = nx.within_inter_cluster(G, [(0, 4)], delta=0.5)

>>> for u, v, p in preds:

...    '(%d, %d) -> %.8f' % (u, v, p)

...

'(0, 4) -> 1.33333333'
```

Input:

G: graph

A NetworkX undirected graph.

Output:

is_partition (G, communities) --> return True if and only if **communities** is a partition of the nodes of G.

Algorithm:

Step 1: louvian_partition package is imported using community package in networkx.

Step 2: partitions are found; it is formed into small clusters(community).

Step 3: From the clusters common neighbours within and between the clusters are found.

Step 4: Scores are attached to the already predicted scores.

3.3.3 Linear Regression

Linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

Input:

Combined scores.

Output:

Prediction for each pair of authors.

Algorithm:

Step1: Import stats models.

Step 2: Split dependent variable and independent variable.

Step 3: Y usually means our output/dependent variable.

Step 4: Add an intercept (beta_0) to our model.

Step 5: Calculate the prediction using imported package

```
    lm = linear_model.LinearRegression()

    model = lm.fit (X, Y)

    predictions = lm.predict (X)
```

3.3.4 Data preprocessing

The process of converting data to something a computer can understand is referred to as pre-processing. One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words.

Stop Words: A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. We would not want these words taking up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to be stop words. NLTK (Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. You can find them in the nltk_data directory.

Input:

Abstract of the authors.

Output:

Abstract after removal of stopword and stemmer formalization.

Algorithm:

```
Step 1: Import nltk
Step 2: nltk.download('punkt') # for tokenization nltk.download('stopwords')
Step 3:  For stopword and stemmer formalisation
stpwds=set(nltk.corpus.stopwords.words("english"))
            stemmer = nltk.stem.PorterStemmer ()
Step 4: Split the dataset into training set and test set.
Step 5: Convert to lowercase and tokenize.
Step 6: Remove stopwords.
```

3.3.5 Feature Extraction

In feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. Feature extraction is a dimensionality reduction process, where an initial set of raw variables is reduced to more manageable groups (features) for processing, while still accurately and completely describing the original data set.

Input:

Pre-processed data.

Output:

Training feature.

Algorithm:

Step1: Split into training set into source and target.

Step2: We will use three basic features:

- number of overlapping words in abstract.
- temporal distance between the papers.
- number of common authors.

Step3: Convert list into array.

Step4: Create training feature.

- documents as rows, unique words as columns (i.e., example as rows, features as columns)

```
training_features = np.array([overlap_abs, temp_diff, comm_auth]).T
```

- scale

```
training_features=preprocessing.scale(training_features)
```

3.3.6 Classification using SVM

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

Input:

Training feature after scaling.

Output:

f1 score.

Algorithm:

```
Step 1: Initialize basic SVM
            classifier = svm.LinearSVC()
step 2: train the dataset using fit function
          classifier.fit(X_train, y_train)
Step 3: Calculate the prediction after classification
            pred=classifier.predict(X_test)
Step 4:  f1 score is calculated
             sumf1+=f1_score(pred,y_test)
```

CHAPTER 4

IMPLEMENTATION AND RESULTS

This section discusses about the experiment adopted to efficiently evaluate the accuracy and performance of the proposed system against the existing approach.

4.1 DATASET DESCRIPTION

The dataset for citation network are downloaded from https://aminer.org/data from which co-author network were generated.

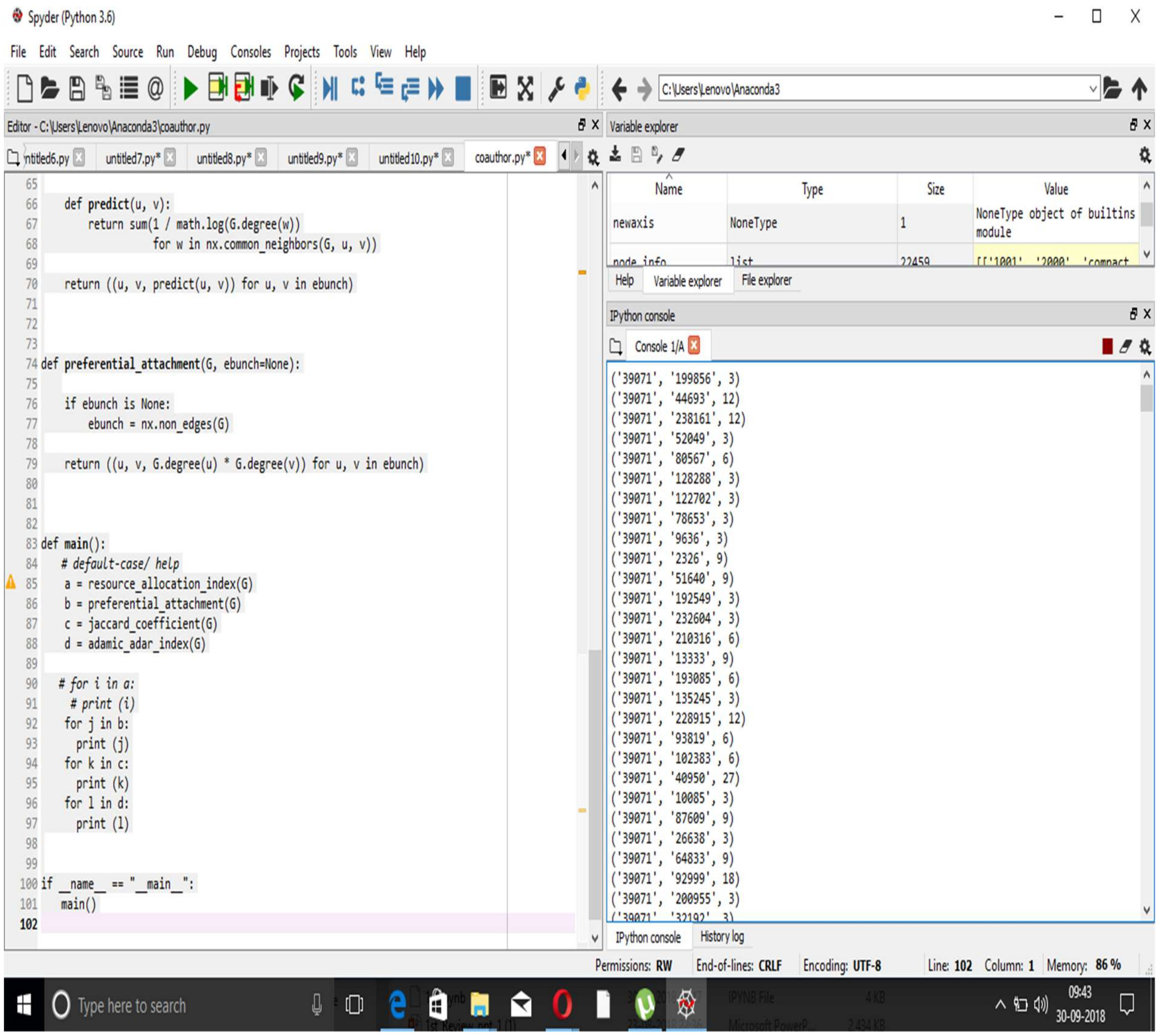
4.2 IMPLEMENTATION RESULTS

This section discusses the details about the modules of the proposed system and the output for each module.

4.2.1 Module description for Structural Similarity

In this module resource allocation index, Adamic-Adar index, preferential attachment score, Jaccard coefficient scores are collected which is shown in figure 4.1.

OUTPUT:



```
    def predict(u, v):
        return sum(1 / math.log(G.degree(w))
                   for w in nx.common_neighbors(G, u, v))

    return ((u, v, predict(u, v)) for u, v in ebunch)



def preferential_attachment(G, ebunch=None):

    if ebunch is None:
        ebunch = nx.non_edges(G)

    return ((u, v, G.degree(u) * G.degree(v)) for u, v in ebunch)



def main():
   # default-case/ help
    a = resource_allocation_index(G)
    b = preferential_attachment(G)
    c = jaccard_coefficient(G)
    d = adamic_adar_index(G)

   # for i in a:
     # print (i)
    for j in b:
      print (j)
    for k in c:
      print (k)
    for l in d:
      print (l)


if __name__ == "__main__":
   main()
```

Name	Type	Size	Value
newaxis	NoneType	1	NoneType object of builtins module
node_info	list	22459	[['1001', '2000', 'compact

```
('39071', '199856', 3)
('39071', '44693', 12)
('39071', '238161', 12)
('39071', '52049', 3)
('39071', '80567', 6)
('39071', '128288', 3)
('39071', '122702', 3)
('39071', '78653', 3)
('39071', '9636', 3)
('39071', '2326', 9)
('39071', '51640', 9)
('39071', '192549', 3)
('39071', '232604', 3)
('39071', '210316', 6)
('39071', '13333', 9)
('39071', '193085', 6)
('39071', '135245', 3)
('39071', '228915', 12)
('39071', '93819', 6)
('39071', '102383', 6)
('39071', '40950', 27)
('39071', '10085', 3)
('39071', '87609', 9)
('39071', '26638', 3)
('39071', '64833', 9)
('39071', '92999', 18)
('39071', '200955', 3)
('39071', '32192', 3)
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 102 Column: 1 Memory: 86 %

09:43
30-09-2018

Fig 4.1 Result of Structural Similarity

4.2.2 Module description for Community Detection

After collecting scores from structural similarity, community is detected from the adjacency graph displayed in figure 4.2 and again common neighbor within cluster is found results are shown in figure 4.3

OUTPUT:

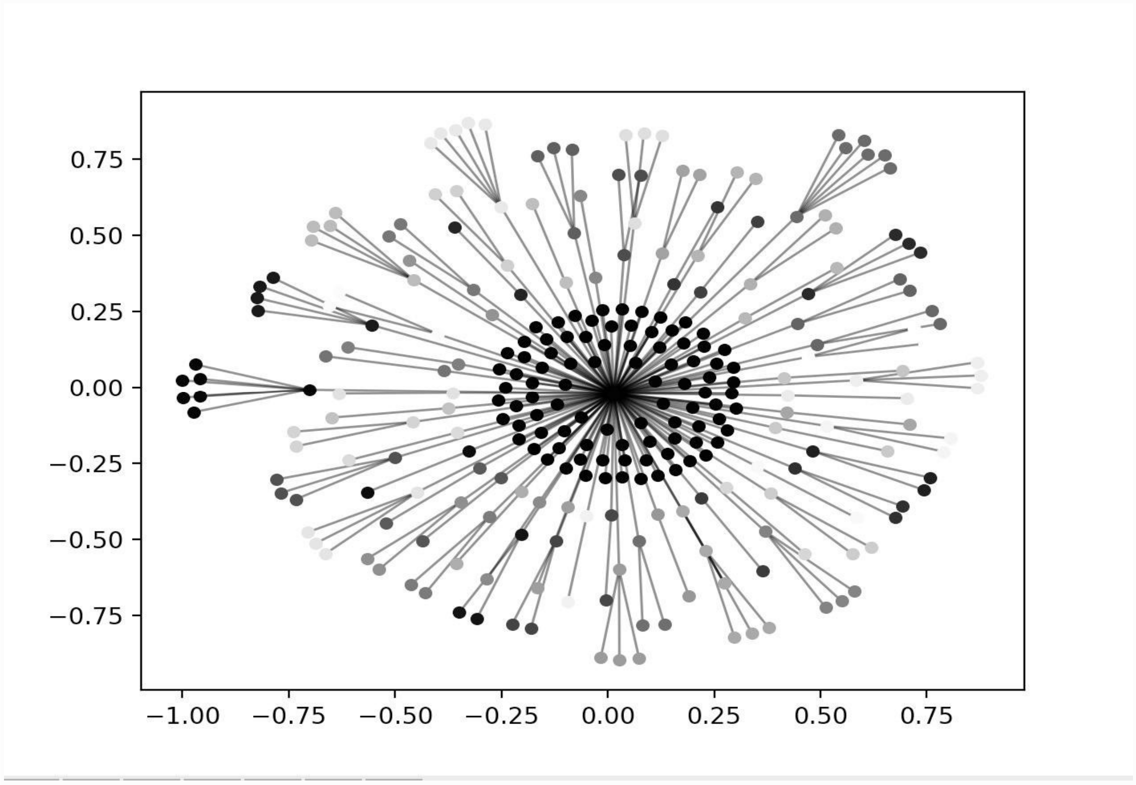
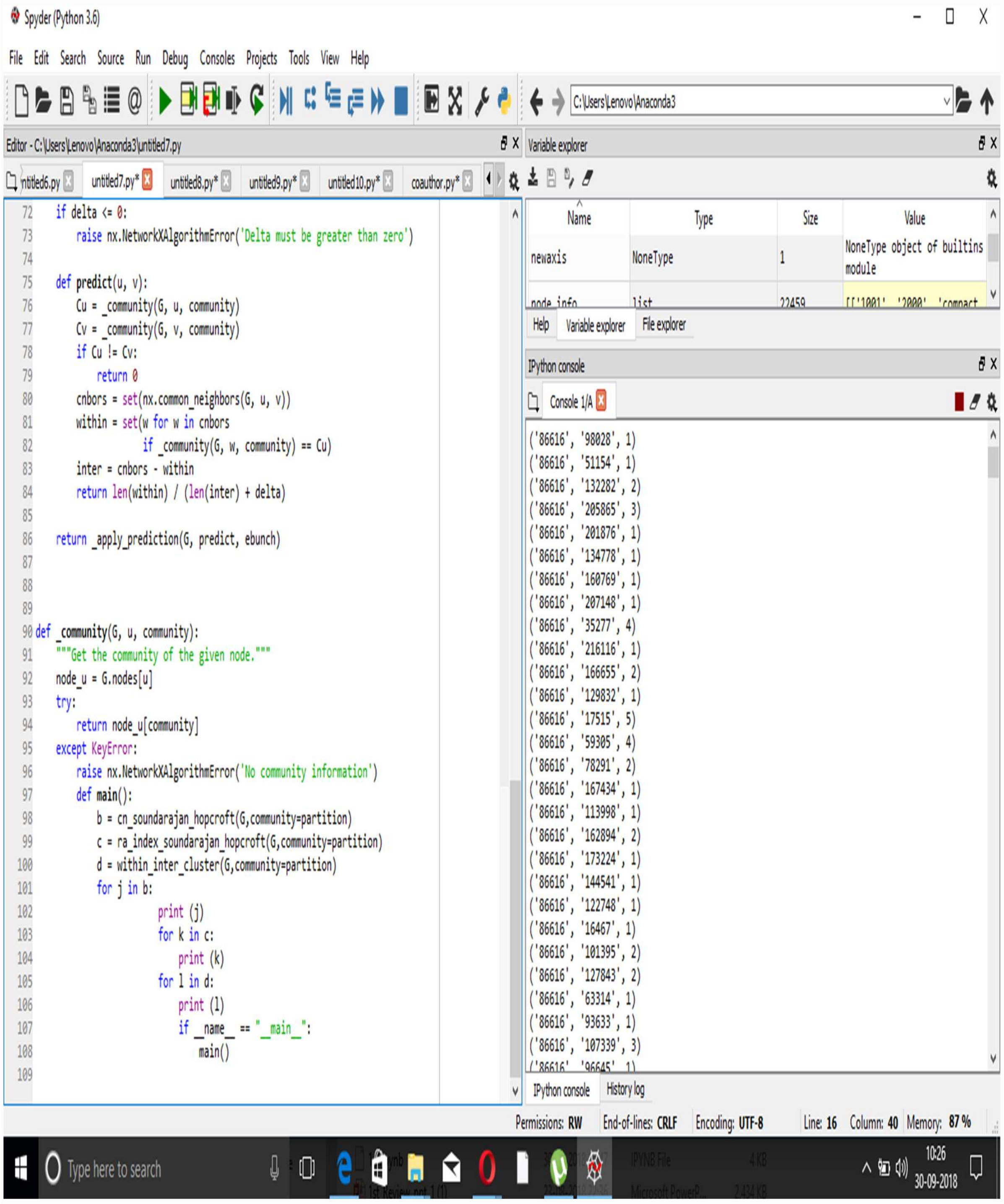


Fig 4.2 Output of Community Detection

OUTPUT:


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Lenovo\Anaconda3
Editor - C:\Users\Lenovo\Anaconda3\untitled7.py
ntitled6.py untitled7.py* untitled8.py* untitled9.py* untitled10.py* coauthor.py*
Variable explorer
Help Variable explorer File explorer
IPython console
Console 1/A
IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 16 Column: 40 Memory: 87 %
Type here to search
10:26
30-09-2018

```
    if delta <= 0:
        raise nx.NetworkXAlgorithmError('Delta must be greater than zero')

    def predict(u, v):
        Cu = _community(G, u, community)
        Cv = _community(G, v, community)
        if Cu != Cv:
            return 0
        cnbors = set(nx.common_neighbors(G, u, v))
        within = set(w for w in cnbors
                     if _community(G, w, community) == Cu)
        inter = cnbors - within
        return len(within) / (len(inter) + delta)

    return _apply_prediction(G, predict, ebunch)



def _community(G, u, community):
    """Get the community of the given node."""
    node_u = G.nodes[u]
    try:
        return node_u[community]
    except KeyError:
        raise nx.NetworkXAlgorithmError('No community information')
        def main():
            b = cn_soundarajan_hopcroft(G,community=partition)
            c = ra_index_soundarajan_hopcroft(G,community=partition)
            d = within_inter_cluster(G,community=partition)
            for j in b:
                    print (j)
                    for k in c:
                        print (k)
                    for l in d:
                        print (l)
                        if __name__ == "__main__":
                            main()
```

Name	Type	Size	Value
newaxis	NoneType	1	NoneType object of builtins module
node_info	list	22459	[['1001', '2000', 'compact

```
('86616', '98028', 1)
('86616', '51154', 1)
('86616', '132282', 2)
('86616', '205865', 3)
('86616', '201876', 1)
('86616', '134778', 1)
('86616', '160769', 1)
('86616', '207148', 1)
('86616', '35277', 4)
('86616', '216116', 1)
('86616', '166655', 2)
('86616', '129832', 1)
('86616', '17515', 5)
('86616', '59305', 4)
('86616', '78291', 2)
('86616', '167434', 1)
('86616', '113998', 1)
('86616', '162894', 2)
('86616', '173224', 1)
('86616', '144541', 1)
('86616', '122748', 1)
('86616', '16467', 1)
('86616', '101395', 2)
('86616', '127843', 2)
('86616', '63314', 1)
('86616', '93633', 1)
('86616', '107339', 3)
('86616', '96645', 1)
```

Fig 4.3 Score for Common Neighbor within the clusters

4.2.3 Module description for Linear regression

Once all the scores are collected from the above three modules, collected scores are given to linear regression algorithm to find probabilities for all pair of authors shown in figure 4.4, pair of authors having high score most likely to work together in future.

OUTPUT:

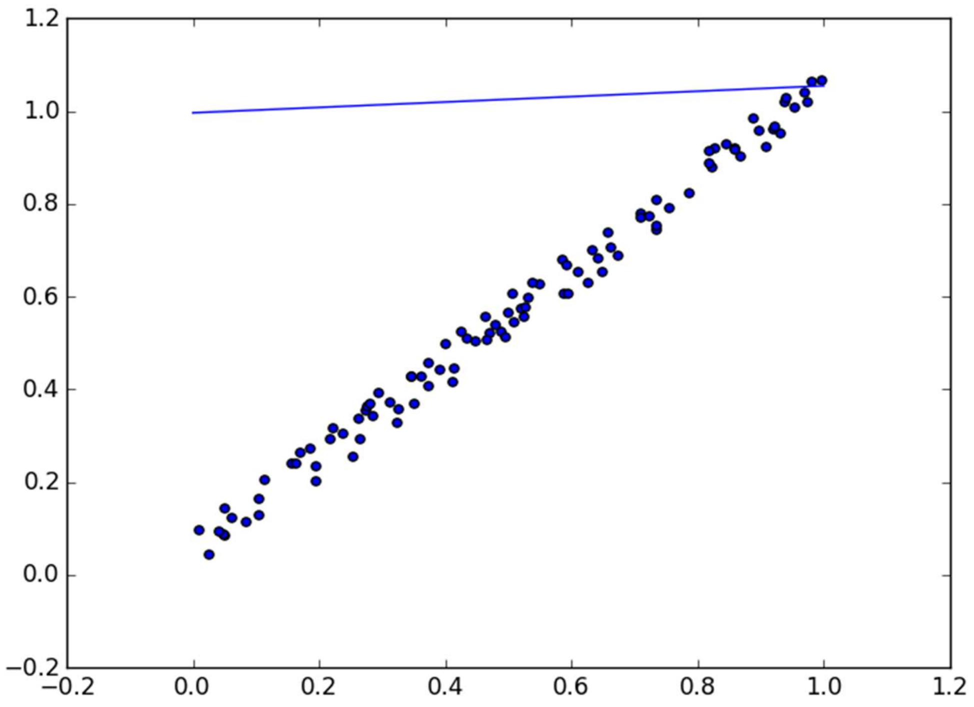
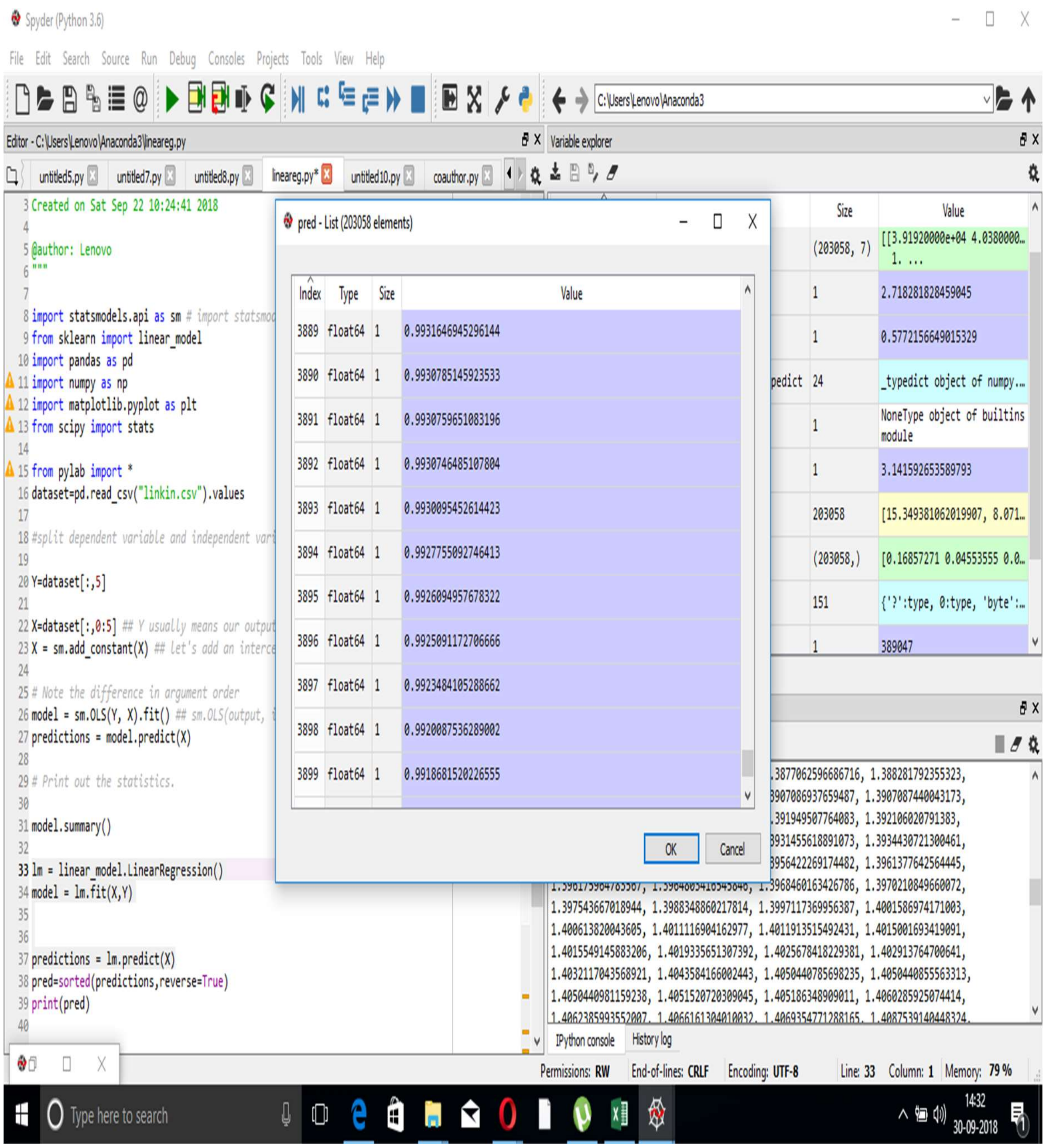


Fig 4.4 Result for Linear Regression plotted in graph



Editor - C:\Users\Lenovo\Anaconda3\lineareg.py

```python
Created on Sat Sep 22 10:24:41 2018

@author: Lenovo
"""

import statsmodels.api as sm # import statsmod
from sklearn import linear_model
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

from pylab import *
dataset=pd.read_csv("linkin.csv").values

#split dependent variable and independent vari

Y=dataset[:,5]

X=dataset[:,0:5] ## Y usually means our output
X = sm.add_constant(X) ## let's add an interce

# Note the difference in argument order
model = sm.OLS(Y, X).fit() ## sm.OLS(output, i
predictions = model.predict(X)

# Print out the statistics.

model.summary()

lm = linear_model.LinearRegression()
model = lm.fit(X,Y)


predictions = lm.predict(X)
pred=sorted(predictions,reverse=True)
print(pred)

```

pred - List (203058 elements)

Index	Type	Size	Value
3889	float64	1	0.9931646945296144
3890	float64	1	0.9930785145923533
3891	float64	1	0.9930759651083196
3892	float64	1	0.9930746485107804
3893	float64	1	0.9930095452614423
3894	float64	1	0.9927755092746413
3895	float64	1	0.9926094957678322
3896	float64	1	0.9925091172706666
3897	float64	1	0.9923484105288662
3898	float64	1	0.9920087536289002
3899	float64	1	0.9918681520226555

Variable explorer

	Size	Value
	(203058, 7)	[[3.91920000e+04 4.0380000… 1. ...
	1	2.718281828459045
	1	0.5772156649015329
pedict	24	_typedict object of numpy.…
	1	NoneType object of builtins module
	1	3.141592653589793
	203058	[15.349381062019907, 8.071…
	(203058,)	[0.16857271 0.04553555 0.0…
	151	{'?':type, 0:type, 'byte':…
	1	389047

```
.3877062596686716, 1.388281792355323,
3907086937659487, 1.3907087440043173,
.391949507764083, 1.392106020791383,
3931455618891073, 1.3934430721300461,
3956422269174482, 1.3961377642564445,
1.3961759647835567, 1.3964805416545846, 1.3968460163426786, 1.3970210849660072,
1.397543667018944, 1.3988348860217814, 1.3997117369956387, 1.4001586974171003,
1.400613820043605, 1.4011116904162977, 1.4011913515492431, 1.4015001693419091,
1.4015549145883206, 1.4019335651307392, 1.4025678418229381, 1.402913764700641,
1.4032117043568921, 1.4043584166002443, 1.4050440785698235, 1.4050440855563313,
1.4050440981159238, 1.4051520720309045, 1.405186348909011, 1.4060285925074414,
1.4062385993552007, 1.4066161304010032, 1.4069354771288165, 1.4087539140448324,
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 33 Column: 1 Memory: 79 %

Fig 4.5 Results for Linear Regression

4.2.4 Module description for Support Vector Machines

- clf= SVC(kernel="linear")

- Kernel is mainly used to find the similarity in the datasets, in this Support vector classifier it defines the linear hyperplane when training to the algorithm, it comes out as a classifier.

- clf.fit(features_train,labels_train)

- So, you’re passing training data to the fit function where the Linear kernel is used as the hyperplane.

- pred= clf.predict(features_test)

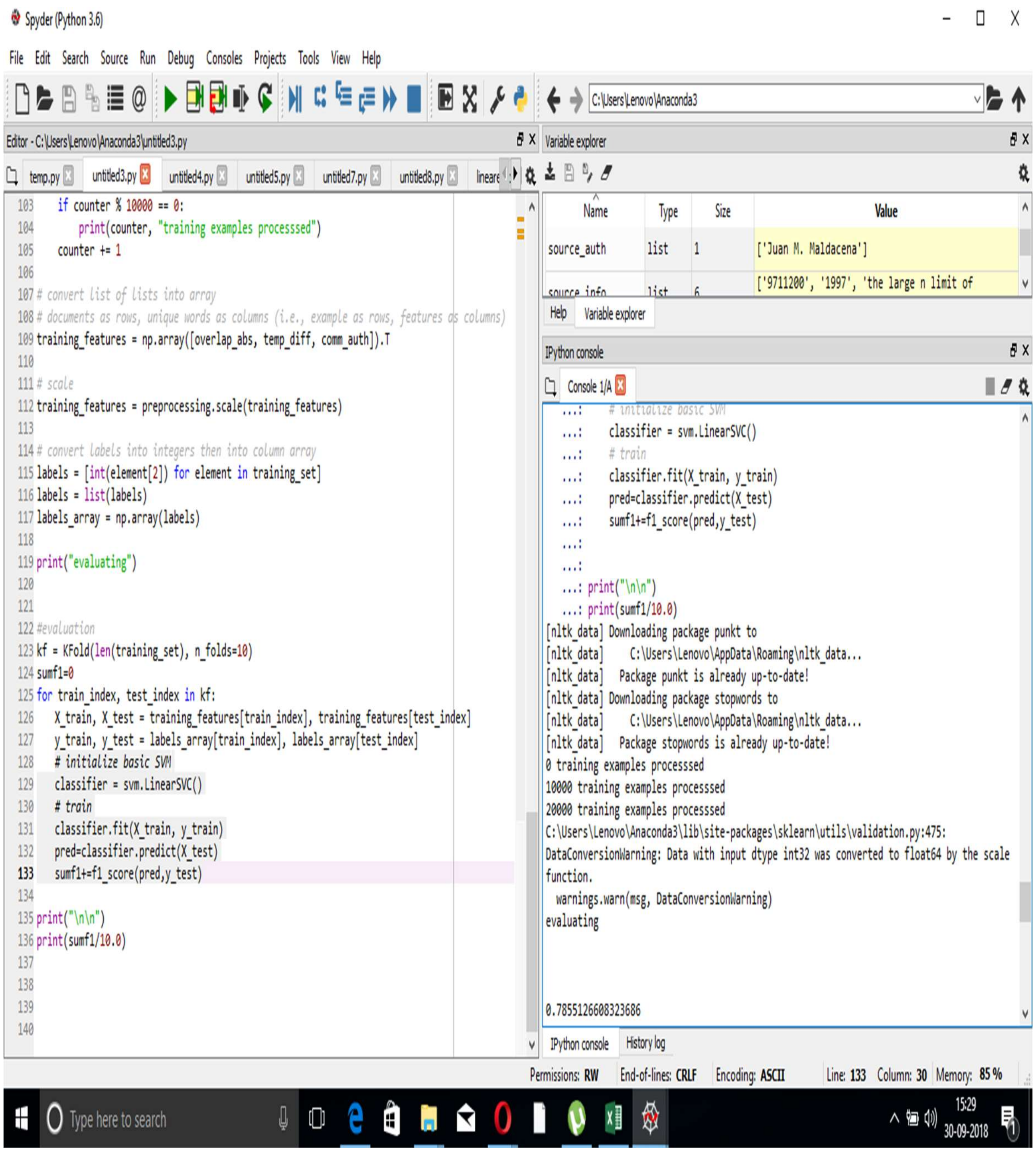
- Remember, we are going to predict the labels, so we have to pass one information to the predict function, so here test datasets are passed.

- After that, import accuracy for benchmark.

- acc= accuracy_score(pred,labels_test)

OUTPUT:

After Classification using SVM, the original dataset is shown in the figure 4.6


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Lenovo\Anaconda3
Editor - C:\Users\Lenovo\Anaconda3\untitled3.py
temp.py untitled3.py untitled4.py untitled5.py untitled7.py untitled8.py lineare
Variable explorer
Help Variable explorer
IPython console
Console 1/A
IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 133 Column: 30 Memory: 85 %
Type here to search
15:29
30-09-2018

```python
    if counter % 10000 == 0:
        print(counter, "training examples processsed")
    counter += 1

# convert list of lists into array
# documents as rows, unique words as columns (i.e., example as rows, features as columns)
training_features = np.array([overlap_abs, temp_diff, comm_auth]).T

# scale
training_features = preprocessing.scale(training_features)

# convert labels into integers then into column array
labels = [int(element[2]) for element in training_set]
labels = list(labels)
labels_array = np.array(labels)

print("evaluating")


#evaluation
kf = KFold(len(training_set), n_folds=10)
sumf1=0
for train_index, test_index in kf:
    X_train, X_test = training_features[train_index], training_features[test_index]
    y_train, y_test = labels_array[train_index], labels_array[test_index]
    # initialize basic SVM
    classifier = svm.LinearSVC()
    # train
    classifier.fit(X_train, y_train)
    pred=classifier.predict(X_test)
    sumf1+=f1_score(pred,y_test)

print("\n\n")
print(sumf1/10.0)
```

Name	Type	Size	Value
source_auth	list	1	['Juan M. Maldacena']
source_info	list	6	['9711200', '1997', 'the large n limit of

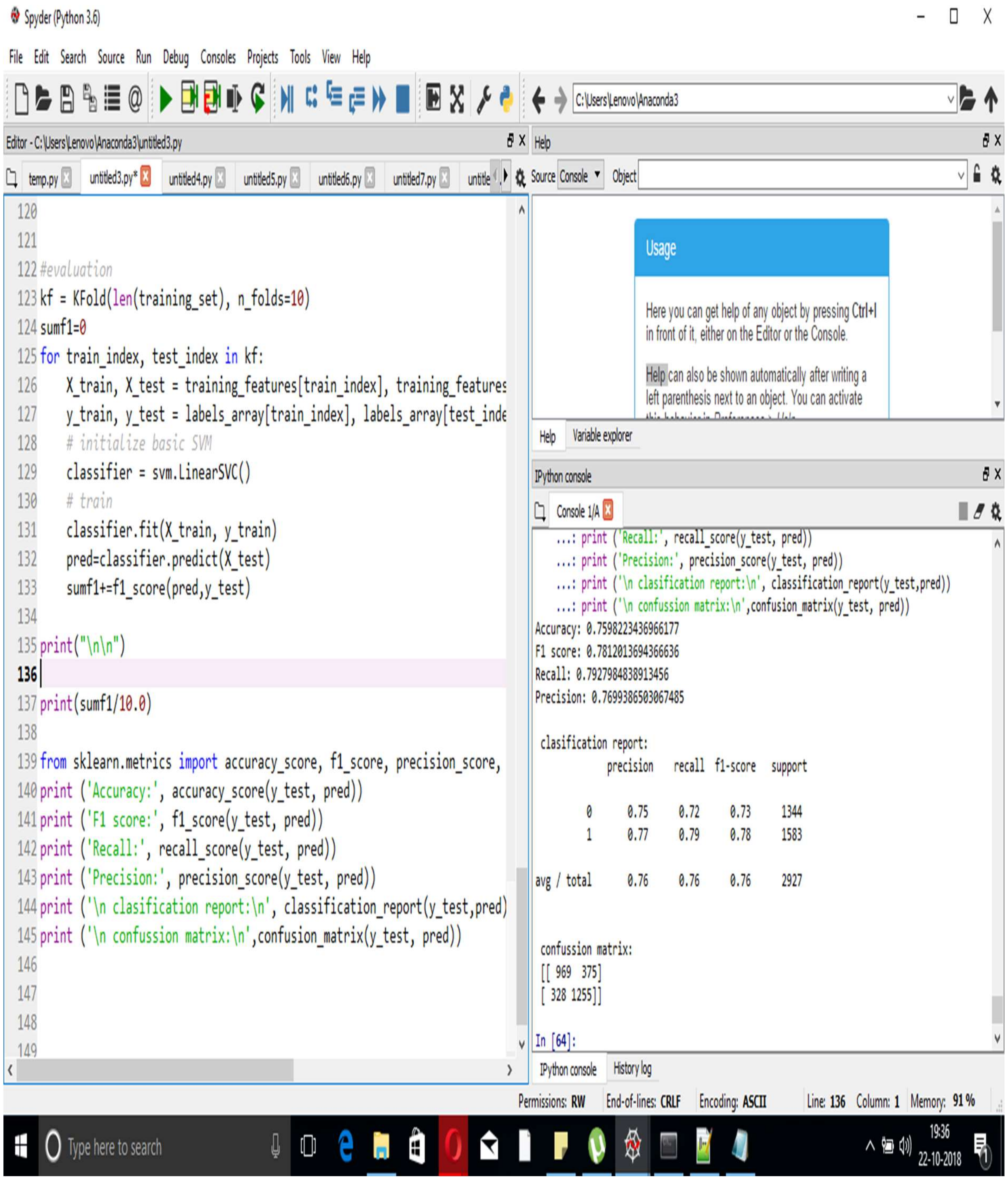
```
   ...:     # initialize basic SVM
   ...:     classifier = svm.LinearSVC()
   ...:     # train
   ...:     classifier.fit(X_train, y_train)
   ...:     pred=classifier.predict(X_test)
   ...:     sumf1+=f1_score(pred,y_test)
   ...:
   ...:
   ...: print("\n\n")
   ...: print(sumf1/10.0)
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
0 training examples processsed
10000 training examples processsed
20000 training examples processsed
C:\Users\Lenovo\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning: Data with input dtype int32 was converted to float64 by the scale
function.
  warnings.warn(msg, DataConversionWarning)
evaluating



0.7855126608323686
```

Fig 4.6 Results of Support Vector Machines

The classification report for SVM using the original dataset is shown in the figure 4.7


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Lenovo\Anaconda3
Editor - C:\Users\Lenovo\Anaconda3\untitled3.py
temp.py
untitled3.py*
untitled4.py
untitled5.py
untitled6.py
untitled7.py
Help
Source Console
Object
Help
Variable explorer
IPython console
Console 1/A
IPython console
History log
Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 136 Column: 1 Memory: 91 %
Type here to search
19:36
22-10-2018

```
#evaluation
kf = KFold(len(training_set), n_folds=10)
sumf1=0
for train_index, test_index in kf:
    X_train, X_test = training_features[train_index], training_features
    y_train, y_test = labels_array[train_index], labels_array[test_inde
    # initialize basic SVM
    classifier = svm.LinearSVC()
    # train
    classifier.fit(X_train, y_train)
    pred=classifier.predict(X_test)
    sumf1+=f1_score(pred,y_test)

print("\n\n")

print(sumf1/10.0)

from sklearn.metrics import accuracy_score, f1_score, precision_score,
print ('Accuracy:', accuracy_score(y_test, pred))
print ('F1 score:', f1_score(y_test, pred))
print ('Recall:', recall_score(y_test, pred))
print ('Precision:', precision_score(y_test, pred))
print ('\n clasification report:\n', classification_report(y_test,pred)
print ('\n confussion matrix:\n',confusion_matrix(y_test, pred))
```

Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate

```
   ...: print ('Recall:', recall_score(y_test, pred))
   ...: print ('Precision:', precision_score(y_test, pred))
   ...: print ('\n clasification report:\n', classification_report(y_test,pred))
   ...: print ('\n confussion matrix:\n',confusion_matrix(y_test, pred))
Accuracy: 0.7598223436966177
F1 score: 0.7812013694366636
Recall: 0.7927984838913456
Precision: 0.7699386503067485

 clasification report:
              precision    recall  f1-score   support

          0       0.75      0.72      0.73      1344
          1       0.77      0.79      0.78      1583

avg / total       0.76      0.76      0.76      2927


 confussion matrix:
 [[ 969  375]
 [ 328 1255]]

In [64]:
```

Fig 4.7 Classification Report for Support Vector Machines

4.3 TEST CASE

Table 4.1 Tabulation for Test Case

Test Case Name	**Test Case Description**	**Input**	**Expected Output**	**Actual Output**
Structural Similarity	Calculating resource allocation index, Adamic-Adar index, preferential attachment score, Jaccard coefficient	Only undirected graph need to be given.	Score for resource allocation index, Adamic-Adar index, preferential attachment score, Jaccard coefficient	As expected
Community Detection	Graph is partitioned into small communities.	Only undirected graph need to be given.	Graph is subdivided into small clusters.	As expected
Linear Regression	Predictions are done to each pair of authors.	Combined scores need to be given.	Prediction for each pair of authors.	As expected
Data Preprocessing	Abstract need to be preprocessed by removal of stop words and stemmer formalization	Abstract for authors	Removal of authors and stemmer formalization should be done successfully	As expected
Feature extraction	Three basic features need to be extracted - Number of overlapping words in abstract. - Temporal distance between the papers. - Number of common authors.	Preprocessed data should be given.	Three basic features will be extracted.	As expected

Test Case Name	**Test Case Description**	**Input**	**Expected Output**	**Actual Output**
Classification using SVM	f1 score will be calculated after prediction using SVM.	Training feature after scaling.	f1 score	As expected

4.4 PERFORMANCE EVALUATION

This section evaluates the performance of the proposed system based on precision, recall and accuracy. Implementation were performed using several parameter settings for the models for the same datasets.

The below mentioned are the parameters used to evaluate a binary classification:

True Positive (TP): Correct positive prediction

True Negative (TN): Correct negative prediction

False Positive (FP): Incorrect positive prediction

False Negative (FN): Incorrect negative prediction

Precision

It is calculated as the number of correct positive predictions divided by the total number of positive predictions.

Precision = TP / (TP + FP) (4.1)

Recall

It is calculated as the number of correct positive predictions divided by the total Number of positives.

Recall = TP / (TP + FN) (4.2)

Accuracy

It is calculated for overall accuracy rate for this proposal.

Accuracy = (TP + TN) / (TP + TN + FP + FN) (4.3)

4.4.1 Evaluation Metrics for the Project

Test set: 2006 and 2007

Accuracy: 0.6477622138708575
F1 score: 0.6620780072107506
Recall: 0.6340238543628374
Precision: 0.6927297668038409

Classification Report:

The f1-score gives you the harmonic mean of precision (0.65) and recall (0.65) refer table 4.2.

The scores corresponding to every class will tell you the accuracy of the classifier in classifying the data points in that particular class compared to all other classes.

The support (2927) is the number of samples of the true response that lie in that class from table 4.2.

Table 4.2 Tabulation for Classification Report1

	Precision	**Recall**	**F1- score**	**Support**
0	0.60	0.66	0.63	1334
1	0.69	0.63	0.66	1593
Avg/total	0.65	0.65	0.65	2927

Confusion matrix:

A confusion matrix is a table 4.3 that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

Table 4.3 Tabulation for Confusion matrix Report1

	0	1
0	886	448
1	583	1010

The dataset is classified into classes 0 and 1 precision, recall and support is calculated for both classes in table 4.2. above calculated values are plotted in bar chart figure 4.8

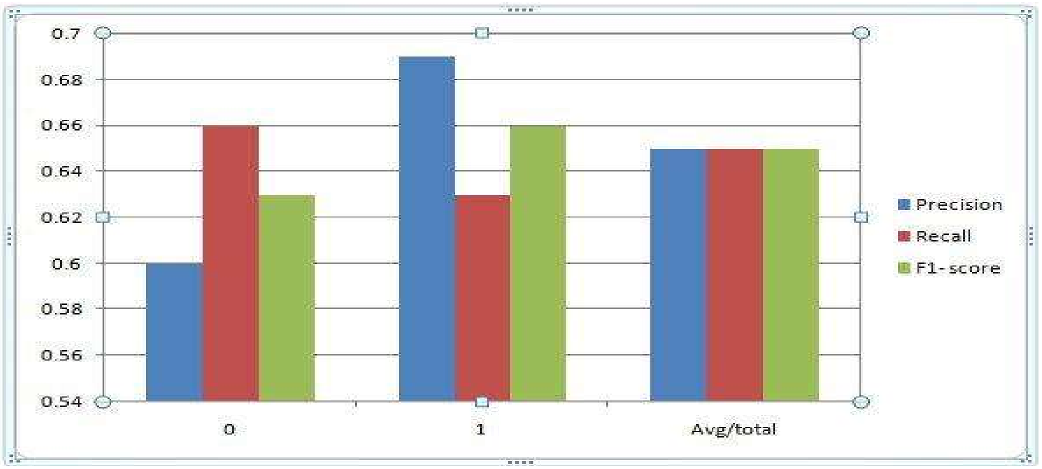


Fig 4.8 Precision, Recall and F1-score for Support Vector Machines (2006-2007)

Splitting actual dataset into training and testing

Accuracy: 0.7748548001366588

F1 score: 0.7901942056669851

Recall: 0.7939859245041587

Precision: 0.7864385297845374

Classification Report:

The f1-score gives you the harmonic mean of precision (0.77) and recall (0.77) refer table 4.4.

The scores corresponding to every class will tell you the accuracy of the classifier in classifying the data points in that particular class compared to all other classes.

The support (2927) is the number of samples of the true response that lie in that class from table 4.4.

Table 4.4 Tabulation for Classification Report2

	Precision	**Recall**	**F1- score**	**Support**
0	0.76	0.75	0.76	1364
1	0.79	0.79	0.79	1563
Avg/total	0.77	0.77	0.77	2927

Confusion matrix:

A confusion matrix is a table 4.5 that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

Table 4.5 Tabulation for Confusion matrix Report2

	0	1
0	1027	337
1	322	1241

The dataset is classified into classes 0 and 1 precision, recall and support is calculated for both classes in table 4.4 above calculated values are plotted in bar chart figure 4.9

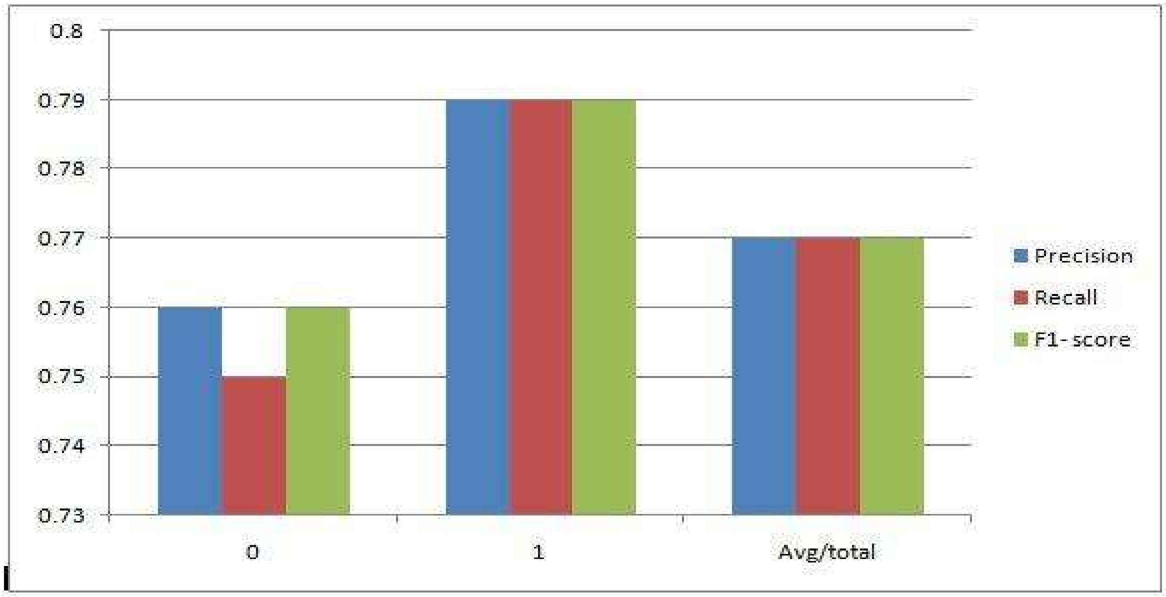


Fig 4.9 Precision, Recall and F1-score for Support Vector Machines (original dataset)

CHAPTER 5

CONCLUSION AND WORK SCHEDULE FOR PHASE 2

5.1 CONCLUSION

The conclusion of the project is that we have structurally predicted using linear regression, semantically abstract of the authors are collected classified using SVM and predicted the similarities.

5.2 WORK SCHEDULE FOR PHASE 2

- To predict when would the author become co-authors, estimation of time period when they will work together.

- Co-author network is split into data of some window size, prediction algorithm is applied to estimate after how many years they work together.

REFERENCES

[1] Pham Minh Chuan, Le Hoang Son, Mumtaz Ali, Tran DinhKhang, Le ThanhHuong, NilanjanDey, Springer (2017) Link prediction in co-authorship networks based on hybrid content similarity metric Applied Intelligence, vol 10209. Springer Published 2017 in Applied Intelligence.

[2] Anderson Matos Medina (2018) Why do ecologists search for co-authorships? Patterns of co authorship networks in ecology (1977–2016) vol 10245. Springer Received: 30 January 2018 Akade ´miaiKiado ´, Budapest, Hungary 2018.Springer

[3] G. S. Mahalakshmi, G. MuthuSelvi, S. Sendhilkumar, P.Vijayakumar, Yongxin Zhu, Victor Chang (2018), Sustainable Computing Based Deep Learning Framework for Writing Research Manuscripts, IEEE Transactions on Sustainable Computing frontiers:1:14, PP 175-185.

[4] Lakshmi JT, Bhavani DS (2017) Link Prediction in Temporal Heterogeneous Networks. In: Wang G, Chau M, Chen H (eds) Intelligence and Security Informatics. PAISI 2017. Cham

[5] Dai T, Zhu L, Cai X, Pan S, Yuan S (2017) Explore semantic topics and author communities for citation recommendation in bipartite bibliographic network. J Ambient Intell Humanized Comput:1–19.

[6] Moradabadi B, Meybodi MR (2017) Link prediction in stochastic social networks: learning automata approach. Journal of Computational Science. Department of Computer Engineering, Amirkabir University of Technology, Tehran, Iran, Journal of Computational Science 24 (2018), PP 313–328.

[7] Tien TN, Harper FM, Terveen L, Konstan JA (2017) User Per-sonality and User Satisfaction with Recommender Systems. Information Systems Frontiers:1–17. Computer Science and Engineering Published - Sep 3 2017.

[8] Akcora CG, Carminati B, Ferrari E (2011) Network and pro le based measures for user similarities on social networks. In: Proceedings of the 2011 IEEE International Conference on Information Reuse and Integration (IRI), pp 292–298

[9] Alghamdi .R and Alfalqi .K, "A Survey of Topic Modeling in Text Mining", International Journal of Advanced Computer Science and Applications, 6:1, pp.147-153, 2015

[10] Fan Lin, JianbingXiahou, and ZhuxiangXu. 2016. TCM clinic records data mining approaches based on weighted-LDA and multi-relationship LDA model. Multimedia Tools Appl. 75, 22 (November 2016), 14203- 14232.