DEPARTMENT OF ELECTRICAL ENGINEERING

IIT DELHI

SOCIAL NETWORK ANALYSIS

PROJECT REPORT

on

Link Prediction

Under The Guidance of

Dr. Amit Nanavati

Dr. Sougata Mukherjea

Submitted By:

Ashutosh Pandey(2021EET2114)

Nishant Chauhan(2021EET2377)

Pratik Wadhwa(2021EET2378)

Table Of Contents

Abstract

Let say we have an instance of a network, can we infer which new interactions among its members are likely to occur in the near future? We formalize this question as the link prediction problem, and develop approaches to link prediction based on measures for analyzing the "proximity" of nodes in a network. Experiments on large networks suggest that information about future interactions can be extracted from networks. Link Prediction methods anticipate the likelihood of a future connection between two nodes in a given network.

Introduction

Given a snapshot of a social network at time t, we seek to accurately predict the edges that will be added to the network during the interval from time t to a given future time t0. The link prediction problem is also related to the problem of inferring missing links from an observed network: in a number of domains, one constructs a network of interactions based on observable data and then tries to infer additional links that, while not directly visible, are likely to exist

As part of the recent surge of research on large, complex networks and their properties, a considerable amount of attention has been devoted to the computational analysis of networks—structures whose nodes represent entities, whose edges represent interaction, collaboration, or influence between entities and asking whether they reproduce certain global structural features observed in real networks. A network model is useful to the extent that it can support meaningful inferences from observed network data.

In this project, we seek to predict the future connectivity of flights between any pair of airports of the country. As a part of the survey there are a lot of problems faced by passengers to switch the flights from one airport to another inorder to reach the destination. This mainly occurs since there are no direct flights between their source and destination. So we analysed this problem and tried to extract features from the network and predict which airports can be connected in future which would make it easier for the passengers and in fact they would be motivated to travel by air and a lot of their time would be saved. Also we have seen that as the development of a country increases, the standard of living of people also increases as a result more people will prefer to commute through flights. Hence we will need strong connectivity or direct flights between the demanding airports.

Data and Analysis

The United States Bureau of Transportation Statistics (BTS) publishes a wealth of dataset on all things transportation related. One of those datasets under the unassuming name of "T-100 Domestic Segment (U.S. Carriers)" is described as:

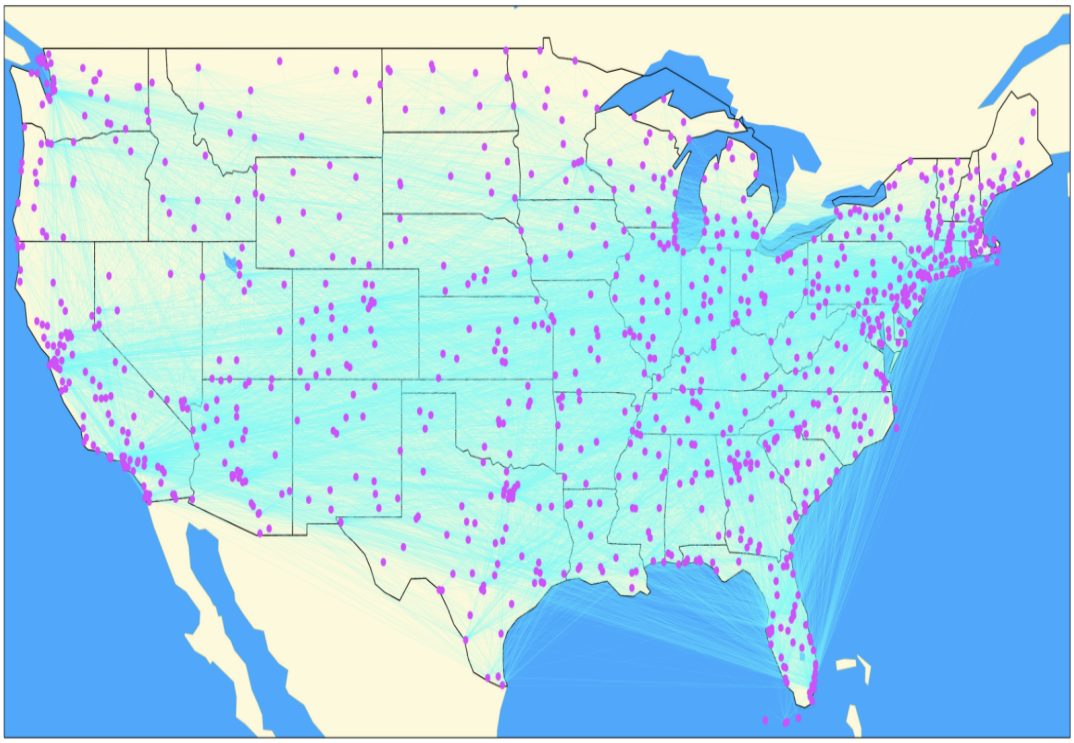
This table contains domestic non-stop segment data reported by U.S. air carriers, including carrier, origin, destination, aircraft type and service class for transported passengers, freight and mail, available capacity, scheduled departures, departures performed, aircraft hours, and load factor when both origin and destination airports are located within the boundaries of the United States and its territories.

This is essentially a large csv file where each row corresponds to an individual flight, including airline, origin and destination airports, number of passengers embarked, date and time, etc. If we take airports to be our nodes and flights to be edges then we have our very own network!

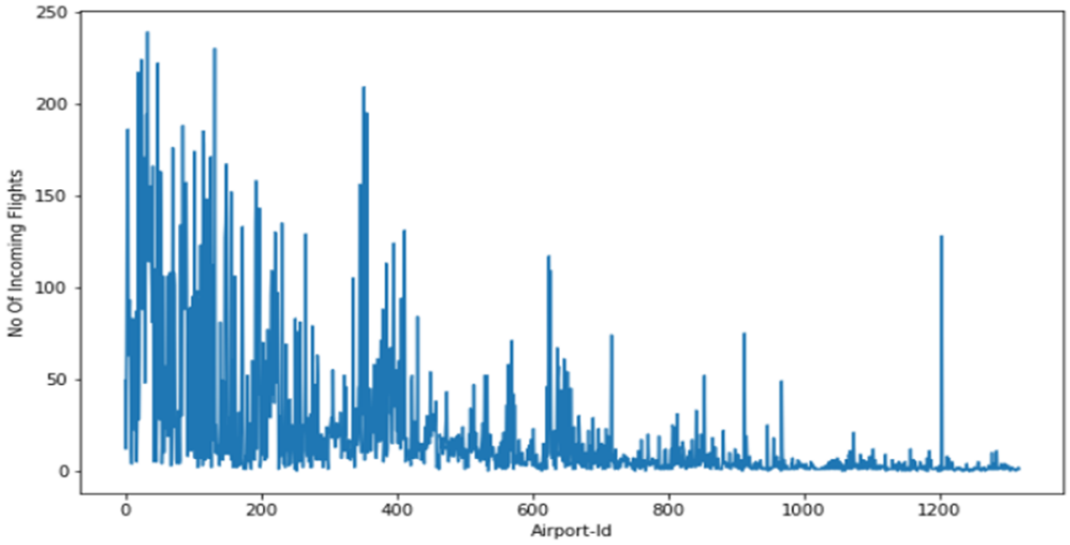
When downloading the file containing all the flights for 2019, we find 391,114 unique flights covering the 365 days of the year. When we sum up all the passengers traveling between the same Origin and Destination we find that we are down to 27,302 rows.

Each row of this reduced dataset now corresponds to an individual edge. This format is known as "**Edge List**" and is one of the most common forms to represent a Graph, just a list of edges, one per line.

In our project we are using the origin and destination airports data for each flight. Our final Edge List will have E=27,072 edges and we have N=1,318 unique nodes.

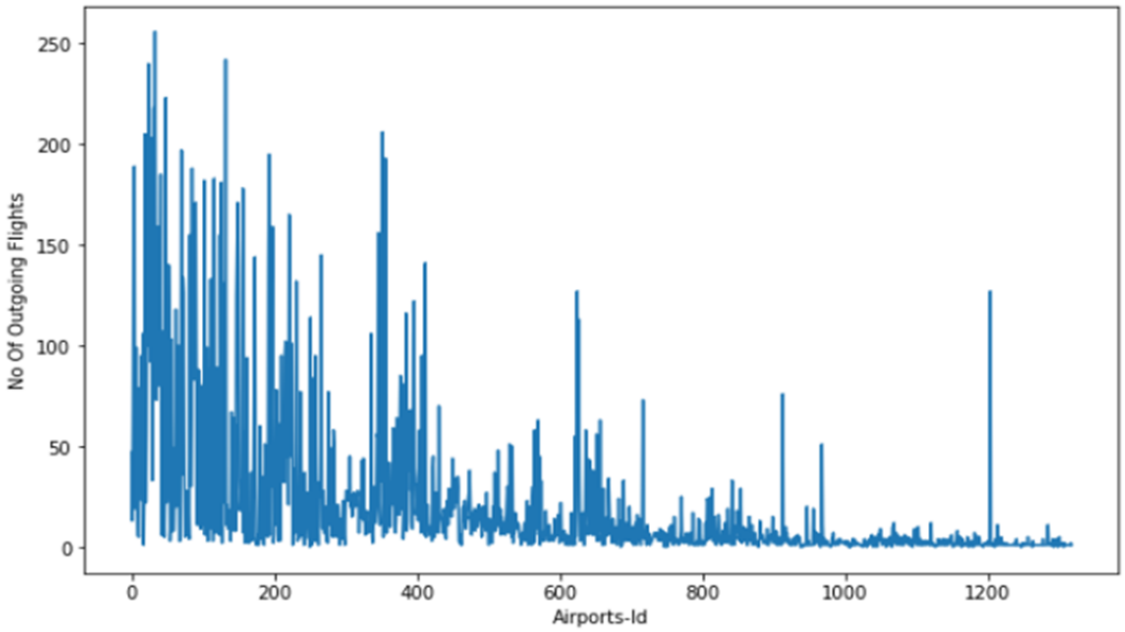


Here we have shown below an analysis of our dataset by plotting the incoming flights of each airport. Along with it we have also calculated the percentile values of the flights of airports. For instance, 90 percentile value is 60; this means only 10% of the total airports have 60 flights.



```
90 percentile value is 60.0
91 percentile value is 69.0
92 percentile value is 78.55999999999995
93 percentile value is 87.0
94 percentile value is 93.91999999999985
95 percentile value is 106.0
96 percentile value is 115.55999999999995
97 percentile value is 131.92000000000007
98 percentile value is 148.63999999999987
99 percentile value is 173.4599999999998
100 percentile value is 239.0
```

Here is the plot of outgoing flights of each airport and their percentile values.




```
90 percentile value is 59.0
91 percentile value is 67.0
92 percentile value is 76.0
93 percentile value is 84.74000000000001
94 percentile value is 95.0
95 percentile value is 102.09999999999991
96 percentile value is 122.55999999999995
97 percentile value is 135.84000000000015
98 percentile value is 157.91999999999962
99 percentile value is 187.4599999999998
100 percentile value is 256.0
```

We have also found during our analysis that the number of airports with no outgoing flights is about 2% and also the number of airports with no incoming flights is about 2%. And there is not any airport with no incoming flights and no outgoing flights.

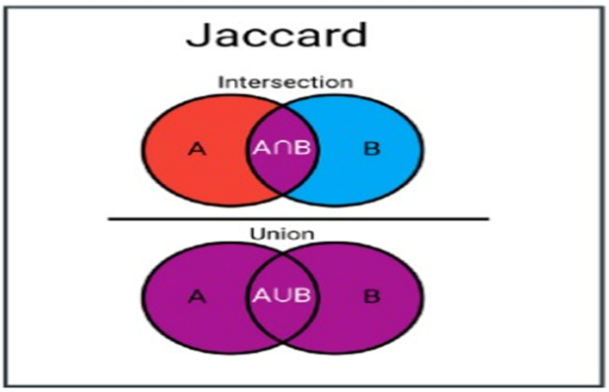
Inorder to perform classification we need to add missing edges. Since we have airports with edges between them hence we would have 1 between them and we should also have airports which do not have edges between them that is 0 between them. So we are adding missing edges between the airports which have shortest paths greater than 2. Now we have 27302 edges and 27302 missing edges.

Here we have splitted the data into train and test with 60:40 ratio. We are training positive edges and then adding negative edges to our data in order to do classification. We are giving one's to positive edges and zero's to negative edges.

Feature Vectors

1) Jaccard Coefficient: $|X \cap Y| / |X \cup Y|$

The Jaccard index, also known as the Jaccard similarity coefficient, is a statistic used for gauging the similarity and diversity of sample sets.If the ratio of common flights and total flights between two airports is more, then it is more likely to have flight between them.



2) Adamic Adar index:

Let's say we have two intersections of 2 airports a and b i.e. u1 and u2. If u1 has a large neighborhood then it might happen that it is the main airport and a,b doesn't need to have flight between them. But if it has a small neighborhood then it is not the main airport so a,b should be connected. Hence a and b should have a flight between them.

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log |N(u)|}$$

3) Preferential attachment:

Preferential attachment means that the more connected a node is, the more likely it is to receive new links.

$$PA(x, y) = |N(x)| * |N(y)|$$

N(u) is the set of nodes adjacent to u. A value of 0 indicates that two nodes are not close, while higher values indicate that nodes are closer.

4) Page Rank:

PageRank is a very very popular featurization technique for directed graphs as it was used by Google. PageRank was named after Larry Page, one of the founders of Google. It is an algorithm used by google search to rank web pages in their search engine results.

Page rank finds the nodes of high importance i.e airports with large no of incoming flights so there is a high chance that new airports will be connected to this important airport.

5) Katz centrality:

A measurement that takes all paths between two nodes in consideration while rating short paths more heavily. The measurement exponentially reduces the contribution of a path to the measure in order to give less weightage to longer paths. Therefore it uses a factor of l where l is the path length.

If Katz value between two nodes have a large value then this means there should be an edge(flight) between them.

$$\sum_{l=1}^{\infty} \beta^l \cdot \left|\text{paths}_{xy}^{\langle l \rangle}\right| .$$

6) Weakly Connected:

If there is a direct flight between airport a and b then we will first remove that edge from the graph and then we will find the shortest path between a and b if it exists. If there exists a shortest path between them then we will return 1 it means they are in the same wcc and have something in common else we will return 0 which means they are in different wcc and same for flight b to a.

7) Intersection:

If there is flight from A to {C,D...} and B to {C,D...} then there are high chances of flight to be present between A to B as they have a high intersection between their neighbours.

Algorithms

Naive Bayes Classifier

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification tasks. The crux of the classifier is based on the Bayes theorem.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Using Bayes theorem, we can find the probability of **A** happening, given that **B** has occurred. Here, **B** is the evidence and **A** is the hypothesis. The assumption made here is that the predictors/features are independent. That is, the presence of one particular feature does not affect the other. Hence it is called naive.

EXAMPLE: Let us assume we want to make a predictor to predict which airports will be connected in future.Now here our input features are Adamic Adar, Jaccard coefficient, Page Rank, Katz Centrality, Intersection, Weakly connected. Using these features we will predict whether there would be a link between the airports or not. So it's a binary classification problem, where '1' corresponds to the link that is present and '0' indicates the absence of the link.

As we know Naives Bayes makes an assumption that given a particular class all the features are independent. So given that link is present between the airport, then we can say that the features such as jaccard coefficient and Adamic Adar index are independent of each other.

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

The variable y is the class variable(link present/absent), which represents if link is present or absent given the features. Variable X represents the parameters/features.
X is given as:

$$X = (x_1, x_2, x_3, ..., x_n)$$

Here x_1,x_2….x_n represent the features, i.e they can be mapped toAdamic Adar, Jaccard coefficient,Page Rank,Katz Centrality,Intersection, Weakly connected. By substituting for **X** and expanding using the chain rule we get

$$P(y|x_1,...,x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

Now, you can obtain the values for each by looking at the dataset and substitute them into the equation. For all entries in the dataset, the denominator does not change, it remains static. Therefore, the denominator can be removed and a proportionality can be introduced.

$$P(y|x_1,...,x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

In our case, the class variable(**y**) has only two outcomes, yes or no. There could be cases where the classification could be multivariate. Therefore, we need to find the class **y** with maximum probability.

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y)$$

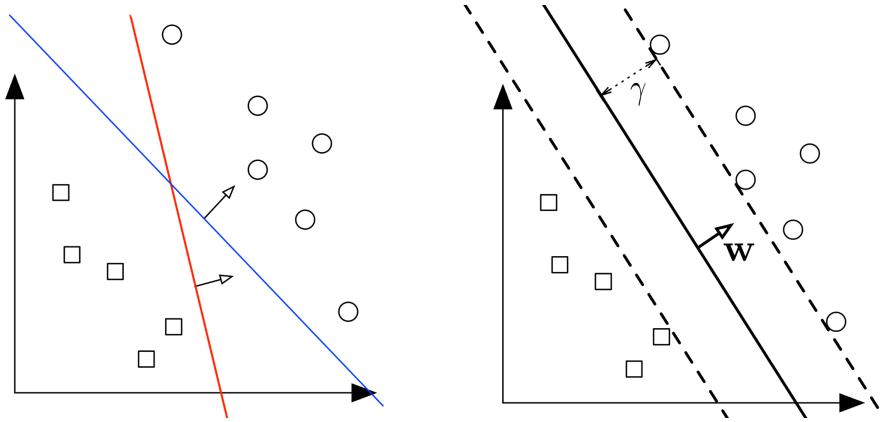
Using the above function, we can obtain the class, given the predictors.

The biggest disadvantage is the requirement of features to be independent. In most of the real life cases, the features are dependent, this hinders the performance of the classifier.

SVM

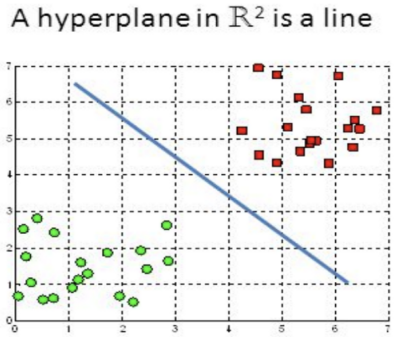
The Support Vector Machine (SVM) is a linear classifier that can be viewed as an extension of the Perceptron developed by Rosenblatt in 1958. The Perceptron guarantees that you find a hyperplane if it exists. The SVM finds the maximum margin separating hyperplane.

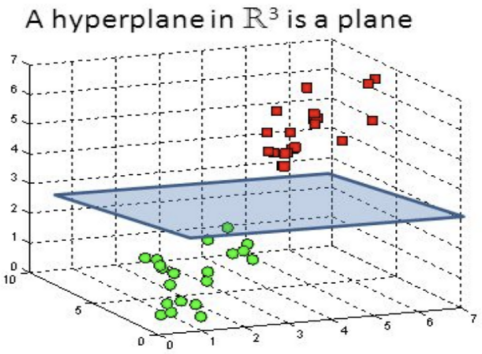
We define a linear classifier: h(x)=sign(wTx+b) and we assume a binary classification setting with labels {+1,−1}.



One question to ask is : What is the best separating hyperplane?
Answer to it is the one that maximizes the distance to the closest data points from both classes. We say it is the hyperplane with maximum margin.

In the above figure, two different separating hyperplanes for the same data set is shown. The right side is the one with maximum margin. The margin, γ, is the distance from the hyperplane (solid line) to the closest points in either class (which touch the parallel dotted lines). In the below figure we have a 3-d view of the hyperplane that divides the points with the maximum margin.





Max Margin Classifier

We can formulate our search for the maximum margin separating hyperplane as a constrained optimization problem. The objective is to maximize the margin under the constraints that all data points must lie on the correct side of the hyperplane:

$$\underbrace{\max_{\mathbf{w},b} \gamma(\mathbf{w},b)}_{maximize\ margin} \text{ such that } \underbrace{\forall i\ y_i(\mathbf{w}^T x_i + b) \geq 0}_{separating\ hyperplane}$$

Now margin of the point x from the hyperplane is defined as:

$$\gamma(\mathbf{w},b) = \min_{\mathbf{x}\in D} \frac{|\mathbf{w}^T\mathbf{x}+b|}{\|\mathbf{w}\|_2}$$

If we plug in this γ in our equation we obtain:

$$\underbrace{\max_{\mathbf{w},b} \underbrace{\frac{1}{\|\mathbf{w}\|_2} \min_{\mathbf{x}_i\in D} \left|\mathbf{w}^T\mathbf{x}_i + b\right|}_{\gamma(\mathbf{w},b)}}_{maximize\ margin} \quad s.t. \quad \underbrace{\forall i\ y_i(\mathbf{w}^T x_i + b) \geq 0}_{separating\ hyperplane}$$

Because the hyperplane is scale invariant, we can fix the scale of w,b anyway we want. Let's be clever about it, and choose it such that

$$\min_{\mathbf{x}\in D} \left|\mathbf{w}^T\mathbf{x} + b\right| = 1.$$

We can add this re-scaling as an equality constraint. Then our objective becomes:

$$\max_{\mathbf{w},b} \frac{1}{\|\mathbf{w}\|_2} \cdot 1 = \min_{\mathbf{w},b} \|\mathbf{w}\|_2 = \min_{\mathbf{w},b} \mathbf{w}^\top \mathbf{w}$$

Hence now our optimization problem becomes:

$$\min_{\mathbf{w},b} \mathbf{w}^\top \mathbf{w}$$
$$\text{s.t.} \quad \begin{aligned} \forall i,\ y_i(\mathbf{w}^T \mathbf{x}_i + b) &\geq 0 \\ \min_i \left|\mathbf{w}^T \mathbf{x}_i + b\right| &= 1 \end{aligned}$$

These constraints are still hard to deal with, however luckily we can show that (for the optimal solution) they are equivalent to a much simpler formulation.

$$\min_{\mathbf{w},b} \mathbf{w}^T \mathbf{w}$$
$$\text{s.t.} \quad \forall i\ y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

This new formulation is a quadratic optimization problem. The objective is quadratic and the constraints are all linear. We can solve it efficiently with any QCQP (Quadratically Constrained Quadratic Program) solver. It has a unique solution whenever a separating hyperplane exists. It also has a nice interpretation: Find the simplest hyperplane (where simpler means smaller $w^\top w$) such that all inputs lie at least 1 unit away from the hyperplane on the correct side.

SVM with soft constraints

If the data is low dimensional it is often the case that there is no separating hyperplane between the two classes. In this case, there is no solution to the optimization problems stated above. We can fix this by allowing the constraints to be violated ever so slight with the introduction of slack variables:

$$\min_{\mathbf{w},b} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^{n} \xi_i$$
$$s.t.\ \forall i\ y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i$$
$$\forall i\ \xi_i \geq 0$$

The slack variable ξi allows the input xi to be closer to the hyperplane (or even be on the wrong side), but there is a penalty in the objective function for such "slack". If C is very large, the SVM becomes very strict and tries to get all points to be on the right side of the hyperplane. If C is very small, the SVM becomes very loose and may "sacrifice" some points to obtain a simpler (i.e. lower the square of ∥ w ∥ 2) solution.

Let us consider the value of ξi for the case of C≠0. Because the objective will always try to minimize ξi as much as possible, the equation must hold as an equality and we have:

$$\xi_i = \begin{cases} 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) & \text{if } y_i(\mathbf{w}^T\mathbf{x}_i + b) < 1 \\ 0 & \text{if } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \end{cases}$$

This is equivalent to the following closed form:

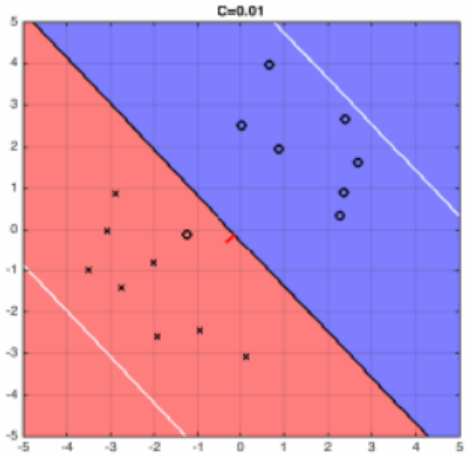
$$\xi_i = \max(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b), 0)$$

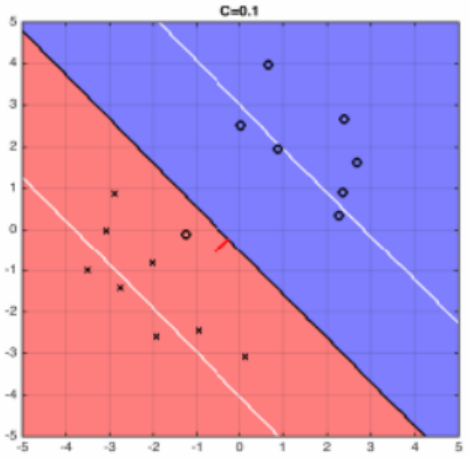
If we plug this closed form into the objective of our SVM optimization problem, we obtain the following unconstrained version as loss function and regularizer:

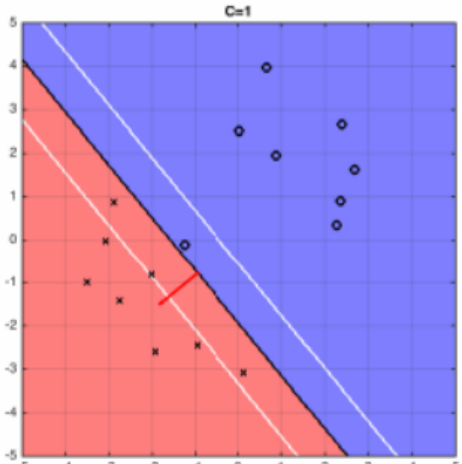
$$\min_{\mathbf{w},b} \underbrace{\mathbf{w}^T\mathbf{w}}_{l_2-regularizer} + C \sum_{i=1}^{n} \underbrace{\max\left[1 - y_i(\mathbf{w}^T\mathbf{x} + b), 0\right]}_{hinge-loss}$$

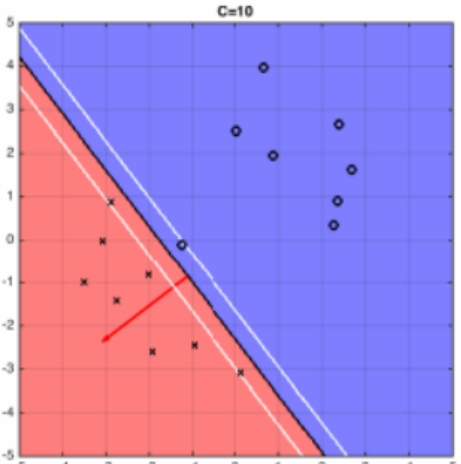
This formulation allows us to optimize the SVM parameters (w,b) just like logistic regression (e.g. through gradient descent). The only difference is that we have the hinge-loss instead of the logistic loss.

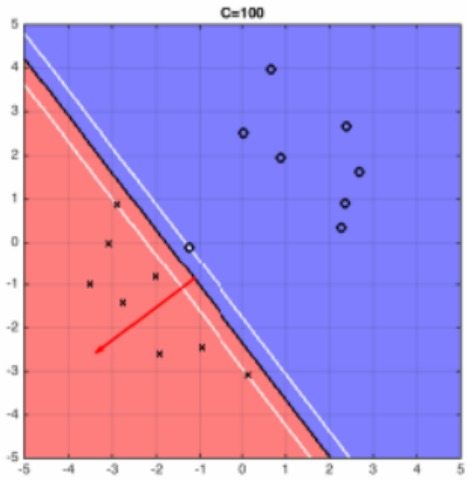
The five plots below show different boundaries of the hyperplane and the optimal hyperplane separating example data, when C=0.01, 0.1, 1, 10, 100.


C=0.01


C=0.1


C=1


C=10


C=100

KNN

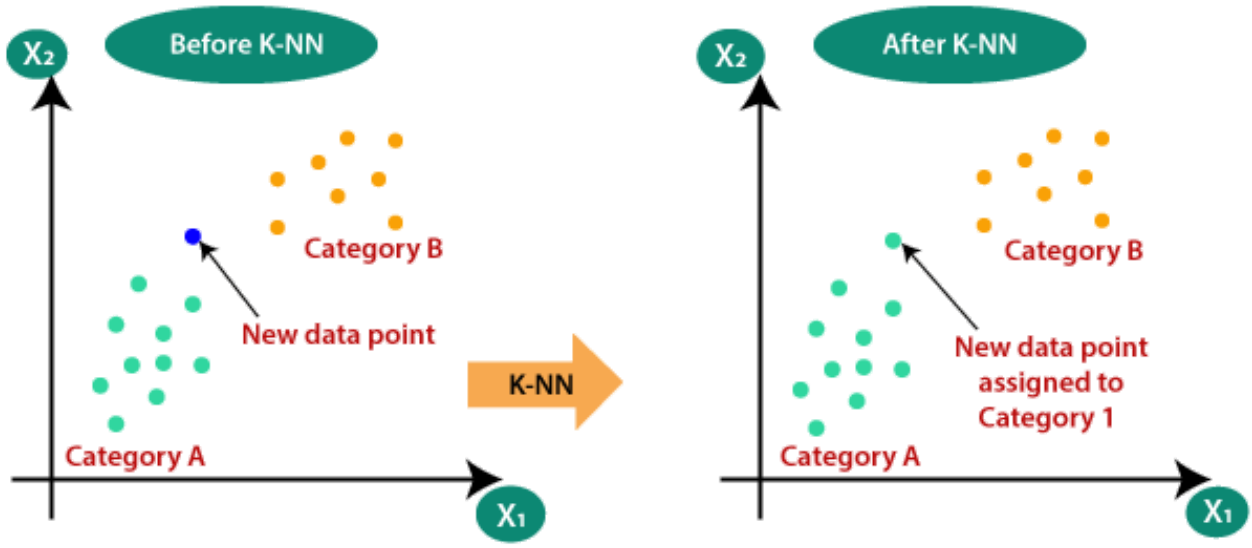
K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

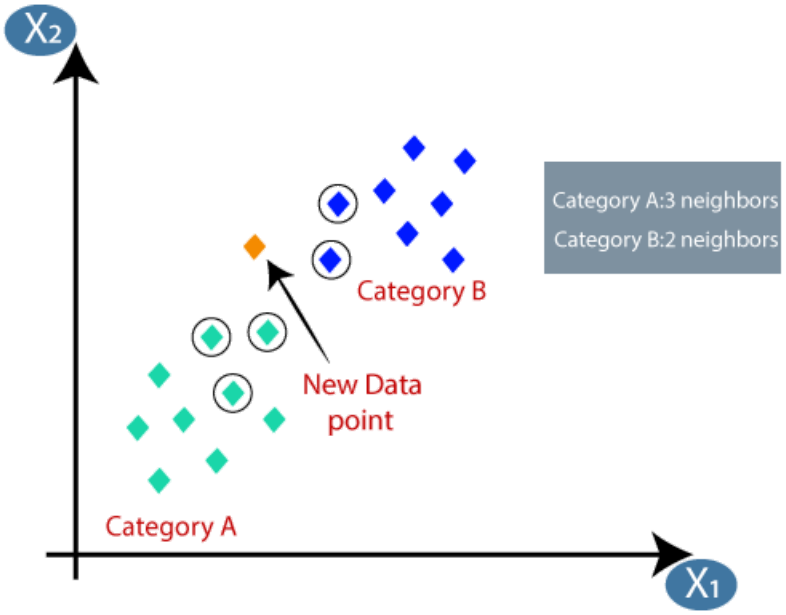
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suited category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- K-NN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Working of KNN

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.


Before K-NN
X2
Category B
New data point
Category A
X1
K-NN
After K-NN
X2
Category B
New data point
assigned to
Category 1
Category A
X1


X2
Category A:3 neighbors
Category B:2 neighbors
Category B
New Data
point
Category A
X1

As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

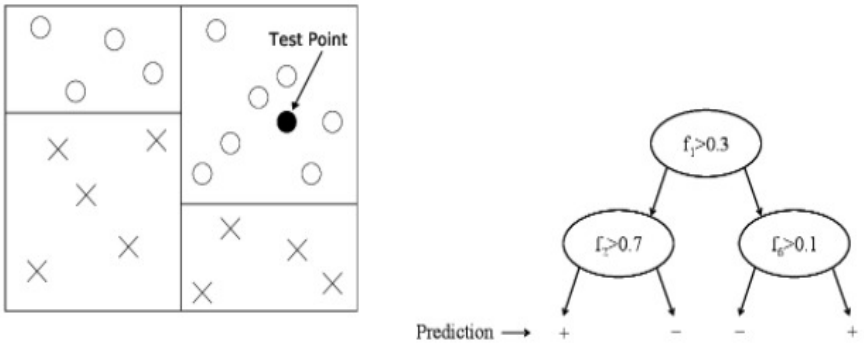
Our input features are Adamic Adar, Jaccard coefficient, Page Rank, Katz Centrality, Intersection, Weakly connected. Using these features we will predict whether there would be a link between the airports or not. So it's a binary classification problem, where ‘1’ corresponds to the link that is present and ‘0’ indicates the absence of the link.

Also there is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. In our project we got optimal results by choosing K=20.

DECISION TREES

Decision Trees are powerful algorithms. In Decision Trees we do not store the training data, instead we use the training data to build a tree structure that recursively divides the space into regions with similar labels. The root node of the tree represents the entire data set. This set is then split roughly in half along one dimension by a simple threshold t. All points that have a feature value ≥t fall into the right child node, all the others into the left child node. The threshold t and the dimension are chosen so that the resulting child nodes are purer in terms of class membership. Ideally all positive points fall into one child node and all negative points in the other. If this is the case, the tree is done. If not, the leaf nodes are again split until eventually all leaves are pure (i.e. all its data points contain the same label) or cannot be split any further (in the rare case with two identical points of different labels).

Decision trees have several nice advantages over nearest neighbor algorithms: 1. Once the tree is constructed, the training data does not need to be stored. Instead, we can simply store how many points of each label ended up in each leaf - typically these are pure so we just have to store the label of all points; 2. decision trees are very fast during test time, as test inputs simply need to traverse down the tree to a leaf - the prediction is the majority label of the leaf; 3. decision trees require no metric because the splits are based on feature thresholds and not distances.



We want to Build a tree that is Maximally compact and Only has pure leaves

It is always possible to find a consistent tree if and only if no two input vectors have identical features but different labels. The problem is, Finding a minimum size tree is NP-Hard. But we can approximate it very effectively with a greedy strategy. We keep splitting the data to minimize an impurity function that measures label purity amongst the children.

Gini Impurity

Let Sk⊆S where Sk={(x,y)∈S:y=k} (all inputs with labels k)

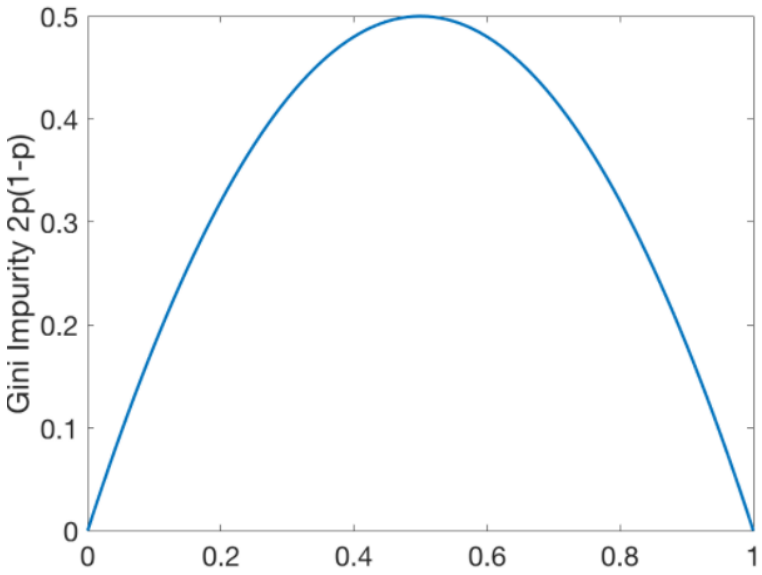
S=S1∪⋯∪Sc

Define:

pk=|Sk|/|S|←fraction of inputs in S with label k

Note: This is different from the Gini coefficient.

$$G(S) = \sum_{k=1}^{c} p_k (1 - p_k)$$



Gini impurity of a tree:

$$G^T(S) = \frac{|S_L|}{|S|} G^T(S_L) + \frac{|S_R|}{|S|} G^T(S_R)$$

where:

- $(S = S_L \cup S_R)$
- $S_L \cap S_R = \varnothing$
- $\frac{|S_L|}{|S|} \leftarrow$ fraction of inputs in left substree
- $\frac{|S_R|}{|S|} \leftarrow$ fraction of inputs in right substree

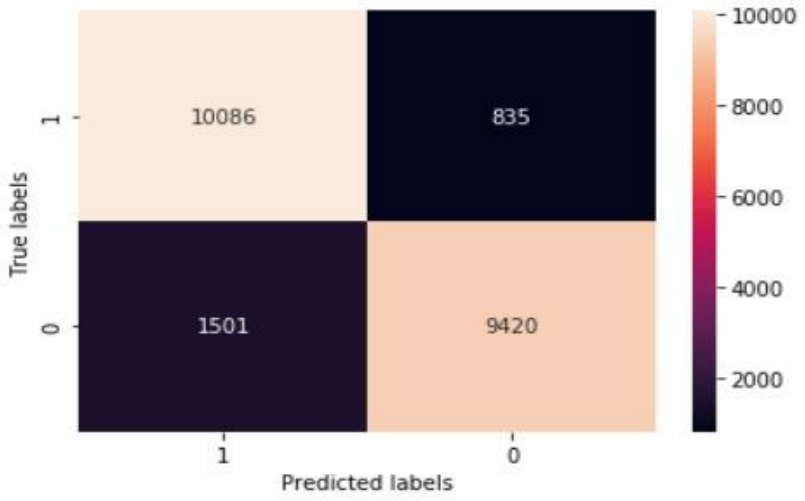
Results And Findings

We have used different algorithms for link prediction and on comparing the results of these algorithms we have found out that Naive Bayes and Decision tree were performing quite well . On the other hand SVM and KNN did not perform well i.e. giving lower accuracy when compared with the accuracy of Naive Bayes and Decision tree. Below table shows us the accuracy score obtained for different algorithms.

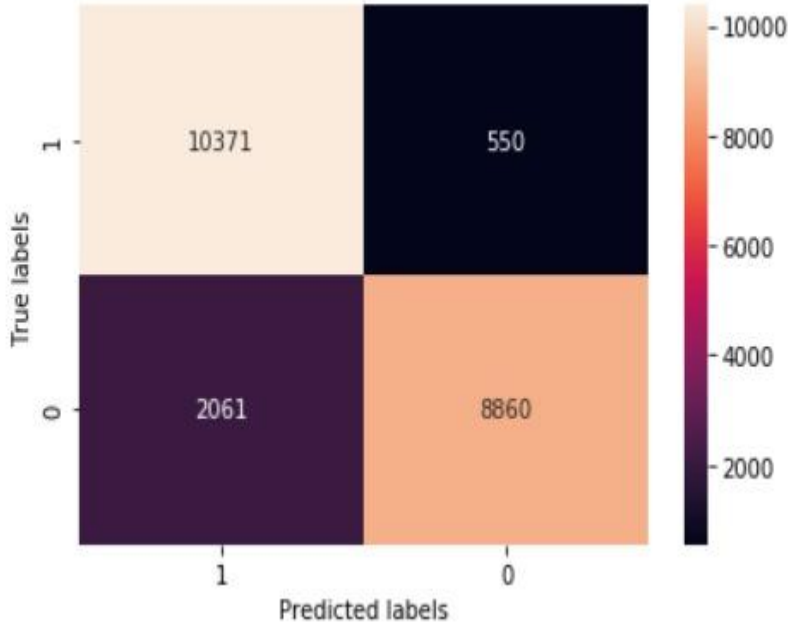
Algorithm	Accuracy
SVM	87.80%
KNN	89.61%
Naive Bayes	95.90%
Decision tree	95.36%

Feature vectors like Jaccard distance and Adar index were giving quite good results on the other hand feature vector like preferential attachment was not giving the good result.

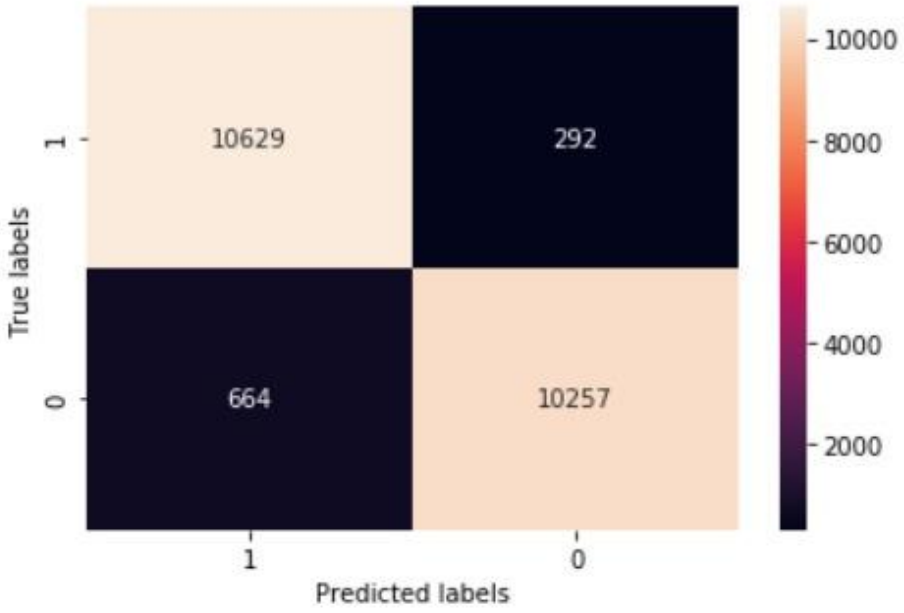
Confusion Matrix for KNN:



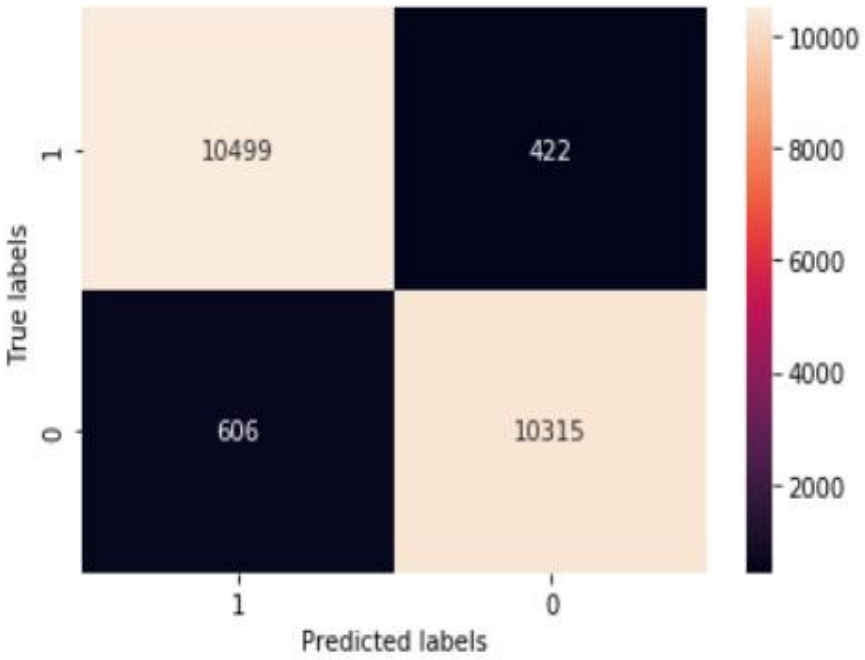
Confusion matrix for SVM:



Confusion Matrix for Decision Tree:



Confusion Matrix for Naive Bayes:



Conclusion & Future Work

We have implemented this on around 10 features and used 4 different algorithms to perform link prediction. More relevant features can be added to the link prediction model which would improve the link prediction accuracy, furthermore different algorithms can also be used which would perform better on this model. So in future we can predict which airports can be connected.

References

1)https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-019-1561-7#sec2

2)https://www.cs.cornell.edu/home/kleinber/link-pred.pdf

3)https://www.transtats.bts.gov/

4)M. E. J. Newman. Clustering and preferential attachment in growing networks. Physical Review Letters E, 64(025102), 2001

5)http://image.diku.dk/imagecanon/material/cortes_vapnik95.pdf

6)https://ecommons.cornell.edu/bitstream/handle/1813/31637/BU-1065-MA.pdf;jsessionid=B819F2B1B8036A42D9551925F716E95C?sequence=1

7)http://www.cs.cornell.edu/courses/cs4780/2015fa/web/lecturenotes/lecturenote05.html

8)https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote17.html

.