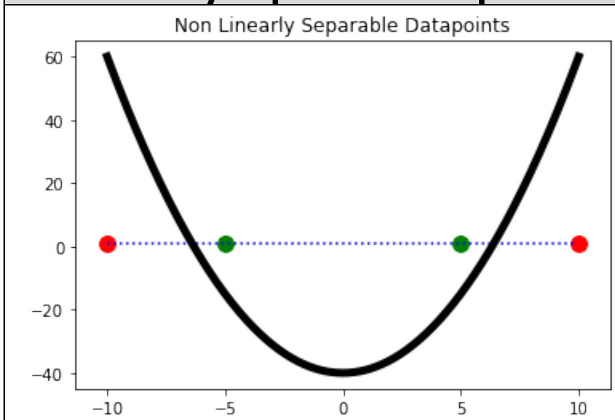
Simon Fraser University
Assignment 3 - CMPT 741 — Data Mining, Fall 2019

Date**: October 29th, 2019**

Name: **Anurag Bejju**
Student ID: **301369375**

1.
1.1.
The dataset of four 2-dimensional points with numerical features and binary class labels (+, -) that are not linearly separable are:

Non Linearly Separable Datapoints



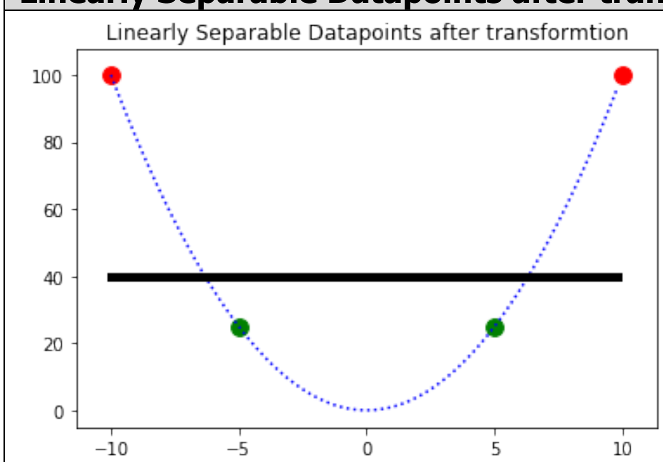
S.No	X	Y	Class	Color
Point 1	-10	1	-1	Red
Point 2	-5	1	1	Green
Point 3	5	1	1	Green
Point 4	10	1	-1	Red

Inference

As you can see in the attached graph for the given dataset. It is not possible to linear function (hyperplane) to separate the given points w.r.t their assigned class

Inorder to make these given data points linearly separable w.r.t their assigned class, we can apply a transformation function $\boldsymbol{\varphi} \rightarrow \boldsymbol{y} = \boldsymbol{x}^2$ on these points.

Linearly Separable Datapoints after transformation



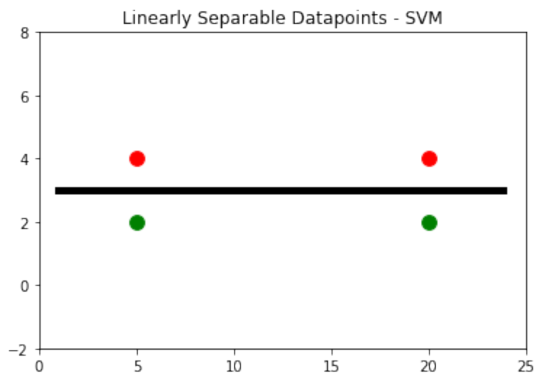
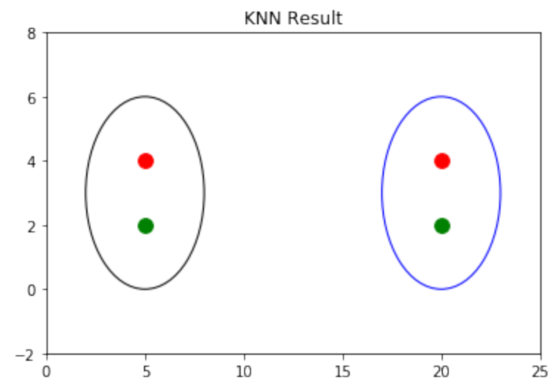
S.No	X	Y = X^2	Class	Color
Point 1	-10	100	-1	Red
Point 2	-5	25	1	Green
Point 3	5	25	1	Green
Point 4	10	100	-1	Red

Inference

As you can see in the attached graph for the given transformed dataset. It is now possible to have a linear function or hyperplane (y= 40 example) to linearly separate the given points w.r.t their assigned class.

1.2.

Dataset for 100% training accuracy with linear SVM and 0% with Nearest Neighbour classifier				
S.No	**X**	**Y**	**Class**	**Color**
Point 1	5	2	1	Green
Point 2	5	4	-1	Red
Point 3	20	2	1	Green
Point 4	20	4	-1	Red

Linear SVM	Nearest Neighbour classifier
	
As you can see in the plot above, the given data point are perfectly linear separable w.r.t to their classes. Therefore, our SVM would always return a 100% training accuracy when we test it on the same points.	In this case, when we run KNN with cluster size 2, you can see each cluster has two points belonging to 2 different classes. Since KNN uses majority (> 50%) to determine the class for points with in a cluster, there will be a 50% probability for any new points to be in each of the 2 classes. In worst case scenario, each point in training set can be misclassified due to this, leading to a 0% training accuracy.

2.

2.1. ***Relationship:*** The set of frequent itemsets FDB in the global database $\bigcup_{i=1}^{k} DB_i$ must be locally frequent in atleast one of the local sets of frequent itemsets FDB_i on the client side.

Assumption:

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n items present in the entire database

Let $W = \{t_1, t_2, \ldots, t_m\}$ be a set of m transactions, where each transaction t_i is a subset of I.

Proof.

Let x be an item set (subset of I). If $x's$ support $(x.supp_i)$ is smaller than the min supp for s_W $(min\ support\ threshold\ for\ all\ transactions\ W) \times DB_i$ for $i = 1, \ldots, K$ (different locations) then its support $x.supp$ will be smaller than $s_W \times DB$ (since $x.supp = \sum_1^k x.supp_i$ and $B = \bigcup_{i=1}^{k} DB_i$) and x cannot be globally frequent. In simpler terms, if x itemset support is lower than the min sup threshold in all the $DB_i's$ then it is not possible for it to be frequent in DB (with DB_i being disjoint in nature).

Therefore by proof of contradiction, if x is globally frequent, it must be locally frequent in at least one DB_i

2.2.

Communication Direction	Information Exchanged
Client (DB_i) to server (DB)	In the first phase of the algorithm, for every client (DB_i), it returns a set of locally large frequent itemsets of various lengths l ($\mathrm{FDB_i^1} \ldots \mathrm{FDB_i^l}$) and sends it to server (DB)
Server (DB) to Client (DB_i)	Once the server received all the FDB_i's, it merges them and request all clients ($\mathrm{DB_i}$) to get count for each global candidate itemset as well as their support.
Client (DB_i) to server (DB)	In the second phase of the algorithm, , for every DB_i, it returns the count, support for each global candidate itemset.
Server (DB) to Client (DB_i)	Finally, Once the server received all count and support for each global candidate itemset, it will filter out all the global candidate itemsets that donot meet the minSup criteria and return a final global frequent itemsets in DB (FDB). This FDB is broadcasted back to all Clients (DB_i's) and FDB_i's are updated.

2.3. Proposed Algorithm [1]

Legend for variables:

DB : *Global Database* $\left(i.e\ \bigcup_1^k DB_i\right)$ *or server*

DB_i : *Local Database at* i^{th} *location or the* i^{th} *client*

k : $1..k$ *clients or* $1..k$ *local databases in total*

FDB_i : *Local sets of frequent itemsets in* DB_i

FDB' : *Global set of* ***all*** *candiadte itemsets in* DB

Output: FDB : *Final global set of frequent itemsets in* DB

Algorithm:

 # Phase I *: creating large frequent itemsets (FDB_i) with different lengths l in each* DB_i

For $i = 1$ to k begin:

$read\_in\_local\_DB(DB_i)$

 # returns a set of locally large frequent item of various lengths l ($FDB_i^1 \ldots FDB_i^l$) in DB_i

$FDB_i = gen\_large\_itemsets(DB_i)$

end

 # Merge Phase : merging created large frequent itemsets (FDB_i) to create FDB'

$FDB' = \bigcup_1^k FDB_i$

 # Phase II *: finding and support for each global candidate itemset for all* $DB_i's$

For $i = 1$ to k begin:

$read\_in\_local\_DB(DB_i)$

 # count each global candidate itemset as find their support for all $DB_i's$

$\forall\ c \in FDB' \rightarrow gen\_count(c, DB_i)$

end

 # filter out all the global candidate itemsets that donot meet the minSup criteria and return

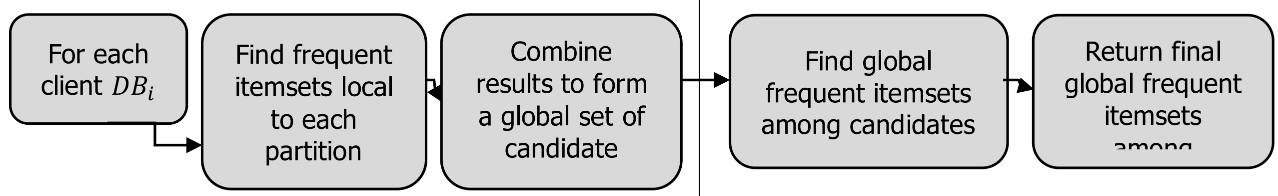
 # final global set of frequent itemsets in DB

$FDB = \{c \in FDB' \mid c.count \geq minSup\}$

return FDB

Simple Explanation for the proposed algorithm:

Phase 1	Phase 2



This algorithm does two scans in total. In scan 1, for every client database we find local frequent patterns. Since the client-database is relatively smaller than original, all frequent item sets are tested in one scan. Then in Scan 2 we consolidate global frequent patterns.

References:

[1] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association in large databases. In VLDB'95