Lehrstuhl für Data Science

Comparing Oblique Decision Tree with CART in Text classification

Masterarbeit von

Subash Ale Magar

1. Prüfer	2. Prüfer
Prof. Dr. Michael Granitzer	Prof. Dr. Harald Kosch

May 30, 2020

Contents

Abstract

Decision trees play a vital role in machine learning. Begin a simple and interpretable makes it outstanding over other approaches therefore popular in a wide range of application. Recursive partition of feature space into sub-region using axis parallel split makes the process simple. However, for some problem partition in axis parallel split can produce complicated decision structure. As an alternative, oblique splits are used to partition the feature space aiming to simplify those complicated decision boundaries. There exists a various way to compute oblique decision tree yet the performance has not been tested on text classification domain using document embedding. Therefore our thesis tries to find interesting results in several prospective by comparing oblique tree with axis parallel split. The goal of the thesis is to find a way to create oblique boundaries and apply in text-domain to compare with classification and regression tree (CART) as an axis-parallel decision tree. As a research, we performed a classifier combining approach and weak random split approach to find oblique split. Document to vector (Doc2Vec) is applied to create a feature for binary text classification problem. A random process carried out to select a linear combination feature. We observed our propose oblique decision trees has upper hand over CART in all test datasets. In most of the time, we find quite small depth with high accuracy.

Acknowledgements

In my master studies, there are countless people to thank who helped and encouraged me in numerous ways. However, some individuals have been at the forefront and had a vital contribution in bringing me thus far which I would like to pay tribute to them.

Since the last year I frequently in touch with Dr. Johannas Joveksey which I felt very lucky to have such advisor. All his kind advice, guidance, encouragement and importantly freedom to work on my imagination and research interest. He was always ready to help me every time I asked. Finally, I am deeply indebted to his tremendous supervision in limitless ways.

Once again, I was lucky to have Prof Dr. Michael Granitzer as my supervisor. All of his kind support to finished my experiment and clam words gave me more strength and motivation. During the experiment when it was taking more time than I expected he assured to finish properly and guided my focus. In a pandemic situation, his helpful response easies my mental pressure. I would also like to convey thank to Prof. Dr. Harald Kosch to begin my second supervisor.

During my study, I have made some precious friends they have not only supported in studies but established a strong bond. I am thankful to all my friends and family in Passau as well as those who wish and loved me.

Finally, I would like to thank my parents for their continuous support and enormous scarifies they made to ensure that I get the best possible education. As a result, I got an opportunity to pursue my master degree University of Passau focusing in the area of Data science and machine learning.

"For me, everything comes after you"

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

CART ..... Classification and regression tree

AI ..... Artificial intelligence

ML ..... Machine learning

DT ..... Decision tree

IG ..... Information gain

E ..... Entropy

G ..... Gini

GI ..... Gini index

UDT ..... Univariate decision tree

ODT ..... Oblique decision tree

NLP ..... Natural language processing

BOW ..... Bag of words

Doc2Vec/D2v Document to vector

Word2Vec .... Word to vector

CBOW ..... Continuous bag of words

PV-DM ..... Distributed memory paragraph to vector

PV-DBOW .. Distributed bag of words

LR ..... Logistic regression

RANSAC .... Random sample consensus

CARTLC .... Classification and regression tree with linear combination

SADT ..... Simulated annealing decision tree

OC1 ..... Oblique classifier 1

QUEST ..... Quick unbiased efficient statistical tree

GDT ..... Geometric decision tree

HHCART .... HouseHold classification and regression tree

RFS ..... Random feature selection

acc ..... Accuracy score

pre ..... Precision score

rec ..... Recall score

f1 ..... F1 score

max_depth ... Max depth

dep ..... depth

RM ..... rec.motercycles

CG ..... com.graphics

pos ..... Positive

neg ..... Negative

sncr ..... Sincere

insncr ..... Insincere

CSV ..... Comma separated value

algo ..... Algorithm

fet size ..... Feature size

min leaf ..... Min leaf point

1 Introduction

1.1 Introduction

The rise of data and computation power has drastic improvements in computer science. The biggest step up can be seen in the field of artificial intelligence (AI). It is also known as machine intelligence, an intelligence demonstrated by machines in contrast to the natural intelligence displayed by humans and animals[**artificial˙2020**]. Nowadays machines can do such human-level complex tasks that are almost impossible for a couple of decades before. All these are possible because of advance and intelligent algorithms and lots of data with computation power. Machine learning (ML) is a subfield of AI which uses data to learn for a specific purpose. Learning and improving from data is much like a conscious living creation routine which they learn from their daily activities then use that gained knowledge to other similar types of problem. Sets of examples from a particular task are called data. Holding data alone is insufficient to get knowledge therefore it needs a procedure to tell them how to learn and optimize. A learning procedure is known as ML algorithm.

The name ML was created by Arthur Samuel in 1959. Tom M. Mitchell provided a widely quoted, more formal definition of the algorithm studied in the machine learning field.

> "*A computer program is said to learn from experience E with respect to some class of task T and performance measure P if it's performance at tasks in T, as measure by P, improved experience E*" - Tom M. Mitchell, Machine Learning 1st Edition[**mitchell˙machine˙1997**]

Rather than teaching everything they need to know about the problem and how to carry out a solution, it might be possible to teach them to learn for themselves. Algorithms

are differ based on the type of task and data which are available. Several algorithms exist to solve a related problem unless we try it appears difficult to tell which work best.

Classification

Supervised ML is a branch of machine learning. The learning process happens with a set of input and output pairs for either classification or regression purpose. The ideal learned model c must approximate relation of X, Y is given as $f(c) = X \rightarrow Y$. The goal is to minimize the cost function by comparing predicted output h ($f(c)$) output on X after training) with true output Y in the training process. In supervised ML, we separate dataset in two different subsets, $\mathcal{D}(\text{dataset}) = (X, Y)_{train} + (X', Y')_{test}$. One is used for training an algorithm to create model or approximation function then the second one is used for evaluating the performance of that model. Classification and regression are two approaches to supervised problems in ML.

Classification is a process of predicting the class, category or label of a given data. Algorithms attempt to estimate the mapping function f from with input variables X to discrete output variables Y. In classification, the model is trained to classify classes to predict classes whose labels are unknown. A simple example of this type of problem is spam classification in E-mails or messages. This is a binary classification task since it has only two categories, either it can separate in the spam or is not spam. When classes have more than labels they called multi-class or multi-label classification problem. If we take the previous example of spam classification, first we train our model by using train dataset this includes both (X, Y) variables. In each iteration, the algorithm must self optimize to reduce the error rate until some condition satisfied. Then final evaluation takes place when model h output on (X') compare with the true output of Y'. This evaluation is known as testing, it is always good to test on unknown data because in train evaluation we cannot distinguish if the model is memorizing data. Two common problems are overfitting and underfitting can be detected via testing evaluation. Overfitting occurs when the model does good in training but worse in testing. Likewise under fitting happens if both evaluations have very poor performance. Most of the algorithm designed to tackle these two problems on top of that other solutions are exist out there.

1.1.1 Decision tree classifier

> "*The possible solutions to a given problem emerge as the leaves of a tree, each node representing a point of deliberation and decision.*" - Niklaus Wirth in 1934, Programming language designer[**niklaus**]

Decision classifier is a tree classifier, has flowchart similar structure. Decision tree (DT) is used for both classification and regression purpose. It is a non-parametric make no distributional assumptions on the data. The most important property of DT is the interpretability of the model. Since the structure is a tree-based question and answer flowchart, we can easily see and understand the model's approximation function. Unlike linear models, it can map non-linear relationships as well, therefore considering a go-to algorithm. Tree building is a process of asking a sufficient series of conditional questions. A decision node represents a "test" on the basis of an attribute (e.g. whether an E-mail is a spam or not spam), each branch represents the outcome of the test, and each leaf node represents a class label. The paths from the root to leaf represent classification rules[**decision˙2019**]. Applications of DT ubiquitous from data mining, decision support system and information retrieval. In table 1.1 we can find commonly used terms in DT literature.

Let consider basic notation and definition

1. $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{test} = \mathcal{D}_{tr} \cup \mathcal{D}_{te}$, where $\mathcal{D}_{tr} \cap \mathcal{D}_{te} = \emptyset$, $\mathcal{D}$ is dataset consist of train and test subsets.
2. $\mathcal{D}_{tr} = (X, Y)$ and $\mathcal{D}_{te} = (X', Y')$, those subsets must have input and output features $X = x, Y = y$ respectively.
3. $\mathcal{D} = (D_{1+\ldots+n}) \neq 0$, $\mathcal{D}$ is full dataset and D_1 to D_n are subset after splits. Each D consists of $X = (X_{1+\ldots+n}) \neq 0$ and $Y = (Y_{1+\ldots+n}) \neq 0$ which are subset X and Y.
4. $\mathcal{T} = (t_{1+..+n})$, $\mathcal{T}$ is tree and t_1 to t_n are inner nodes of the $\mathcal{T}$ each t contains D eg. a t_1 stores D_1 subset which have heterogeneous data, then next iteration will happen until all the points are from one category or meet any criteria.
5. $h_\theta(\mathcal{D}_{tr})$ = model, $\theta = \beta_{0+..+len(X)}$, where β_0 = bias & rest coefficient of X.
6. *Err(h, $\mathcal{D}$)*, is an error rate where output of h is compare to true value of $\mathcal{D}$.
7. lastly $\mathcal{C} = (\mathcal{C}_{1+..+n})$ is unique value of y_i and $x = x_{1+..+n}$ is number of features available in X or X' must equal.

Common terminologies in decision tree

Term	Description
Root node	A node where it creates the first split on the entire dataset.
Splitting	Process of dividing node into two or more subsets.
Depth/Height	Length of the longest path from a root to a leaf node.
Inner/Decision node	Split creates sub-node which is called decision or inner node.
Leaf/Terminal node	End node where split doesn't happen further.
Branch/Sub Tree	A subsection of the entire tree is called branch or sub-tree.
Parent node	A node, which is divided into sub-nodes is called a parent node.
Child node	sub-nodes are the children of a parent node.
Pruning	When we remove sub-nodes of a decision node.

Table 1.1: Common terms in Decision Tree

Basic tree classification process

The purpose of classifying data is to determine the class X' on $\mathcal{D}_{te}$ by using estimated function *(h)* which is trained on sets of $\mathcal{D}_{tr}$. Let's examine an example where data points are scattered in two dimensions. In figure 1.1 we can see the scattered email data. In this example we have two feature variables x_1 and x_2 represented in two axis. *spam and not spam* are two labels of Y. 'x' are used to identify spam emails and 'o' for not spam.

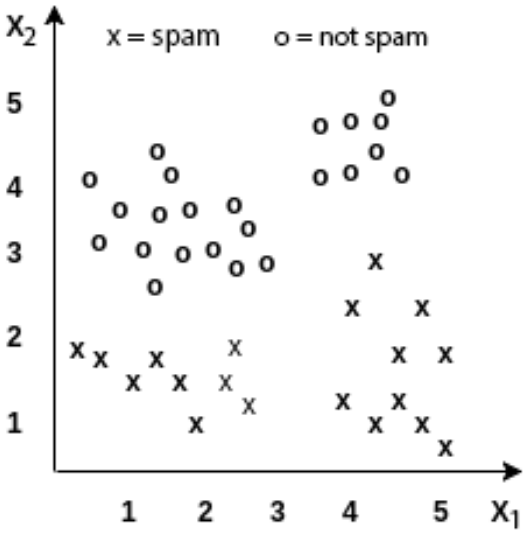


Figure 1.1: An example of email data in a scatter plot.

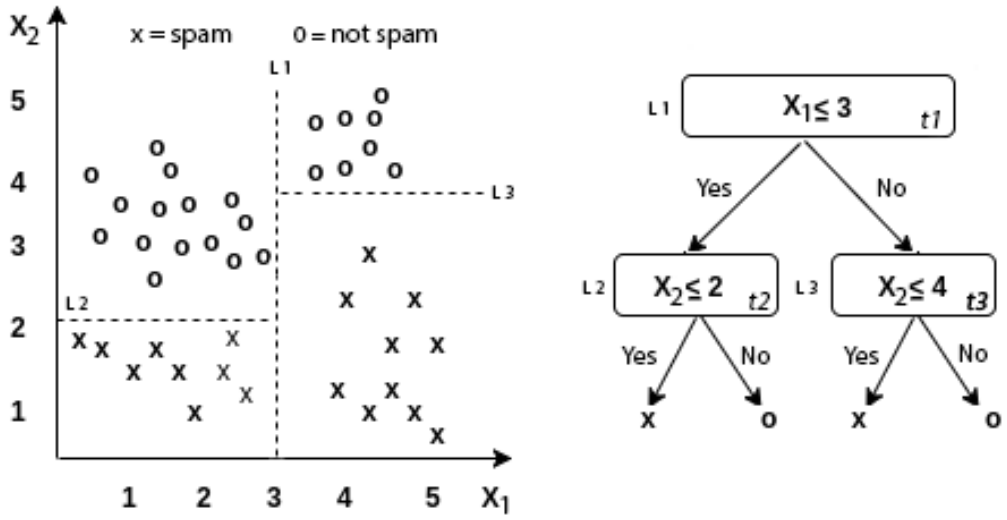


Figure 1.2: An example of decision boundary and tree, we can see how binary decision tree split above dataset and create decision boundary as well as how decision rule or tree is formed.

The above figure is a given solution to the problem. This is a simple example of binary classification. At a time it uses one feature point to find the best splitting line. The process is known as a univariate splitting process. Since it uses only feature point at a time, therefore, the decision boundary is always in axis-parallel. The line L1, L2 and L3 are decision boundary where it uses to ask questions. For example in L1 is a root node which has a depth of 0, we can ask a question if X_1 is less than 3 or not. Then question splits data into two subsets Yes and No branch. If the split data doesn't give pure child node then we ask a further question for each subset of data. For example, L2 and L3 are inner nodes now the tree has a depth of 1, here model asked questions to find a pure node. But if the split data has a pure node (consist only one category) then we stop building tree. The rule generated by DT looks:

Decision 1: IF $X_1 \leq 3 \wedge X_2 \leq 2$ = spam else not spam
Decision 2: IF $X_1 \leq 3 \wedge X_2 \leq 4$ = spam else not spam

Here the model has given clear comprehensible rules that one can understand. A full-grown tree, as well as a graphical illustration of the decision boundary, provide helps to understand deeper. With the help of both, We can make a precise reason why a particular question is asked on each level. In the next part, we will see how DT knows

which question is to ask first and keep asking more. Let's understand how DT grows and predict a new instance from the pseudo-process.

Steps of top-down DT induction

1. Start with all training sample at the root.
2. Choose the best attribute using impurity measurement, add as a decision rule.
3. Split data using the above measurements & create branches.
4. Repeat 1-3 step for each branch & their associated inner branches.
5. If All the data points in the branch have the same class label, mark that as the leaf node.

Steps of top-down DT deduction

1. Check unknown instance x_i from root node.
2. If the leaf node then return label.
3. Else match the condition by digging into branches.
4. Until leaf node appears.

Decision trees can handle high dimensional data with good accuracy. Furthermore, the rules simple and comprehensible it consider as a white box type of ML algorithm. It shares internal decision-making logic, which is unavailable in the black box type of algorithms such as Neural Network. It has a faster training time compared to the neural network algorithm. The time complexity of decision trees is a function of the number of records and the number of attributes in the given data, details are presented in the result chapter. The subsequent content will explain a process to measure the goodness of split.

Impurity measure

Different tree-based algorithms use a different strategy to measure impurity. These generally measure the homogeneity of the target variable within split subsets. Various strategies affect the accuracy of the model and also in model attributes like depth or number of nodes. The decision criteria are different for classification and regression trees. Split create sub-nodes and in sub-nodes must increase the homogeneity of the available category.

Asking the right question or decision rule is crucial. Identifying the best question can give an optimal tree. Where finding such optimal tree requires to create all possible tree which is practically impossible. Therefore, Decision tree measures the importance of feature value through impurity function. If the tree cannot give optimal rule we at least look for sub-optimal decisions. Those sub-optimal trees consist of two or more local optimal decision rules. An impurity criterion is used for binary and non-binary classification as well as a regression problem.

Let's see by an example of how impurity measurement is estimated. In the given explain we consider only categorical attributes. The same concept is applicable in any type of attribute plus can handle missing value for any attribute. Following two famous impurity measurements are used in our experiment.

1. Information Gain
   This function is based on information theory proposed by Shanon[**chiu˙mathematical**], less impure node requires less information to describe and more impure requires more. It uses entropy (E) function to measure impurity at a node. Information Gain (IG) is the E of the parent node minus the E of the child nodes. The use of information gain for DT was proposed by Quinlan in 1986 [**quinlan˙induction˙1986**]. It is calculated as:

   $$E(\mathcal{D}_{tr}) = \sum_{i=1}^{\mathcal{C}} -p_i log_2 p_i$$

   $E(\mathcal{D}_{tr})$ is total E of $\mathcal{D}_{tr}$ and is the average amount of information needed to identify the class label of a data $E(\mathcal{D}_{tr})$. p_i is the frequency of label i at a node and $\mathcal{C}$ is the number of unique labels. If the sample is completely homogeneous, then the entropy is zero and if the sample is equally divided (50% — 50%), it has an entropy of one.

   $$E(x_i, \mathcal{D}_{tr}) = \sum_{j=1}^{k} \frac{|D_j|}{|D|} E(D_j)$$

   $E(x_i, \mathcal{D}_{tr})$ is entropy on x_i attribute value. The expected information required to classify a tuple from $\mathcal{D}_{tr}$), based on the partitioning by attribute x_i is $E(x_i, \mathcal{D}_{tr})$. k is the total number of splits in our case we are it is 2 since we split in two branches.

   $$IG(x_i) = E(\mathcal{D}_{tr}) - E(x_i, \mathcal{D}_{tr})$$

   IG is defined as difference between beginning $E(\mathcal{D}_{tr})$ and $E(x_i, \mathcal{D}_{tr})$. After all attribute gain calculation we choose best gain which have less information.

2. Gini Index:
   It is a measurement of the likelihood of incorrect classification of a new instance of a random variable if that new instance were randomly classified according to the distribution of class labels from the data set. Lower the Gini impurity much homogeneous data we will have. The use of the Gini Index measure for DT was proposed by[**breiman˙classification˙1984**] in his book. The Gini index considers a binary split for each attribute. It is calculated as

$$G(\mathcal{D}_{tr}) = 1 - \sum_{i=1}^{\mathcal{C}}(p_i)^2$$

   p_i is the probability of an object being classified to a particular class $\mathcal{C}$. Perfectly classified, Gini Index would be zero and evenly distributed would be 1 – (1/number of classes)

$$G(xi, \mathcal{D}_{tr}) = \sum_{j=1}^{k} \frac{|Dj|}{|\mathcal{D}_{tr}|} G(D_j)$$

   If a binary split on xi then $\mathcal{D}_{tr}$ partition into two D_1 and D_2. The gini index (GI) of $\mathcal{D}_{\sqcup\nabla}$ given that partitioning is $G(xi, \mathcal{D}_{tr})$. We compute a weighted sum of the impurity of each resulting partition.

$$GI(xi) = G(\mathcal{D}_{tr}) - G(xi, \mathcal{D}_{tr})$$

   Concerning each attribute, each of the possible binary split is considered. For a discrete-valued attribute, the subset that gives the minimum GI index for that attribute is as its splitting attribute. In case of a discrete-valued attribute, the subset that gives the minimum Gini index for that chosen is selected as a splitting attribute. In the case of continuous-valued attributes, the strategy is to select each pair of adjacent values as a possible split-point and point with smaller Gini index chosen as the splitting point.

Threshold split on real inputs

Thus far we have seen splitting example on categorical inputs. In some case, like image or text classification we encounter a large number of continuous independent features which is also known as real features value. Handling real values are crucial, in one hand it saves computation time and secondly it can be counter-attack for overfitting. For example, if we have unique 100k floating-point value as a feature then we have to

compute impurity for each unique values therefore this process doesn't give any better result than learns strict rule. As a solution discretization of a continuous variable is a must. Many DT applies this transformation approach whereas CART also tackles real inputs by dividing into reasonable bins by different algorithm vendors. In the case of continuous-valued attributes, sklearn DT uses the strategy is to select each pair of adjacent values as a possible split-point[**datacamp˙decision˙2018**]. following approach for real value before implementing impurity measure.

1. Every time a tree is grown by splitting a dataset is sorted.
2. Then the mean of every adjacent pair x[i], x[j] of feature values is considered as a candidate split, except if the pair is less than 1e-7 [**sklearn˙DT˙2020**]. The best split, according to the G/E split criterion is used to split the dataset into those points with x < (x[i] + x[j]) / 2 and those with higher value for x.

Categorization of tree based on split orientation

Determining all possible tree from the dataset is impractical to accomplish, therefore greedy divide and conquer approach are used to approximate the best task. In the decision tree, the approach recursively creates multiple partitions based on the impurity function. The dividing process either perform by single feature value or multiple feature combination. Decision tree induction is the method of learning the decision trees from the training set. There exist distinct characteristics for building tree some are based on impurity measure, tree construction approach (top-down and bottom-up) and some are about feature selection strategy. Our thesis shows two different approaches of building tree which based on the orientation of decision boundary also known as a feature selection method, in another word we when to pick single data point from a feature then it creates an axis-parallel line to split data. This is the way how univariate decision trees work likewise if we pick a combination of features points to split data then it called a multivariate or oblique decision tree.

Univariate decision tree (UDT)

UDT tree are old and powerful trees has an axis parallel split. They can handle missing value and feature engineering problem. Impurity measure finds best homogeneous child nodes from a single data point of a feature therefore internally it does feature engineering

process. The main issue with these kinds of problem is unable to map appropriate function except in orthogonal space. Means if the data are linearly separable then it doesn't understand that possibility of separation. This leads to the problem of interpretability and overfitting. The most popular algorithms which use univariate split are CART and C.45 tree.

Oblique decision tree (ODT)

Unlike UDT, oblique decision tree select more than one feature hence creates an oblique split. It performs linear computation, for example in two-dimensional space with help of coefficient and bias it creates a linear line to separate data points wherein the above dimension it creates hyperplane. CART-Linear Combination (CART-LC), Oblique Classifier 1 (OC1), House Hold Matrix-CART (HHM-CART) are some algorithm of ODT[ODT reference]. The main benefit of this type of tree is to overcome the problem of UDTs. The most important difficulty of this technique is to choose a combination of features. Possibility of selecting features for a decision node can increase exponentially. For example, if the dataset has x_1 to x_4 features then we have possibility to take 2, 3 and all 4 features combination therefore we have total 11 different selection combination per t.

$$1 + \sum_{k=2}^{x-1}(xCk_i = \frac{x!}{k_i!(x-k_i)!})$$

$$(x = 4) = \frac{4!}{2!(4-2)!} + \frac{4!}{3!(4-3)!} + 1 = 11$$

The ODT build process follows the same approach as UDT except here we consider a combination of feature to create a linear rule. That rule provides almost the best partition to split the data further. The tree deduction procedure follows the same concept of Top-Down DT deduction.

Steps of top-down ODT induction

1. Start with all training sample at the root.
2. Select feature for linear combination.
3. Find θ of multivariate test such results in the best partition.
4. Build tree recursively for each partition.
5. Until All the data points in the branch have the same class label, mark that as a leaf node.

1.1.2 Text representation using document embeddings

Text representation is a crucial part of natural language processing (NLP). Text is a natural way of expressing information which easily understandable by a distinct human. In the case of computers, the text is unsuitable to process directly hence we represent them in the form of feature vectors. Feature engineering of natural text is very different from other domain like image and audio signals. Vector conversion is not the only thing to do but putting human-level sense is also crucial. Non-semantic representation process like word frequency count, labelling/one-hot encoding or bag of words (BOW) are some example. These processes are very simple to lose the contextual meaning of data. As an example one-hot encoded vector could produce sparse representation meaning, most indices are zero as word length gets bigger so the sparseness. At last, language is an art, they can be formed with simple to complex structure. Sometimes the meaning of words its self changes based on how they used and sometimes we have to conclude by reading the whole sentence or paragraph.

An example of BOW vectorizing
1.*John likes to watch movies. Mary likes movies too.*
2. *John also likes to watch football games. Mary hates football.*

List of vocabularies from all sentences:
["John", "likes", "to", "watch", "movies", "Mary", "too", "also", "football", "games", "hates"].

Final representation:
BOW1: *[1, 2, 1, 1, 2, 1, 1, 0, 0, 0, 0]*
BOW2: *[1, 1, 1, 1, 0, 1, 0, 1, 2, 1, 1]*

Creating a fixed length of feature is important while most of the BOW techniques fulfil this requirement but lacks ordering of words and semantics. Then based on the index of vocabularies we count words. In the below example we can see sentence 1 doesn't have words *["also", "football", "games", "hates"]* and become 0. The information is lost because of the word order for example "Joh likes Mary" and "Mary likes John" because of corresponding to identical vectors[**gensim˙D2V˙2020**]. There is a solution: bag of n-grams models consider word phrases of length n to represent documents as fixed-length vectors to capture local word order but suffer from data sparsity and high

dimensionality. In the other hand words which have antonym meaning between likes and hates or contextual similarity between football and games are not preserved.

To bridge the gap between representation and understanding we use distributed semantic language models. An unsupervised algorithm is used to create a variable based fixed-length feature. Word embedding is a semantic language model which are plays vital role to add a layer of similarity. Computational models that build contextual semantic representations from corpus data in the form of vector. The vector is a learned model via shallow neural network and can capture some semantic concept of given corpus. Such a concept later manipulate to get different relations between words, like synonyms, antonyms, or analogies. The unsupervised algorithms are word to vector (Word2Vec) and document to vector (Doc2Vec). The author states these methods achieve a new state of the art result on several classifications and sentiment analysis tasks[**d2v˙embedding**]. In our thesis experiment, we have used Doc2Vec embeddings to solve classification task.

Basic of word2Vec

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, three-layer (input, hidden and output) neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located close to one another in the space[**wiki˙word2vec˙2020**]. Vector space gives an intuition of knowing other similar words as human. We can extract information like common words, similarity distance and can do all kind of vector manipulation to retrieve more information. A higher dimensional embedding can capture fine-grained relationships between words but takes more data to learn.

Word2vec model is trained on two different architecture. Continuous bag of words (CBOW) and skip-gram two models. CBOW learns to predict a word from surrounding words whereas skip-gram is used one word to predict all surrounding words, just opposite of CBOW[**mikolov˙efficient˙2013**]. Word2Vec model calculates vectors for each word in a document but if we want to calculate vector for the entire document then requires explicit modification. To resolve this problem again the author provides a paragraph vectoring algorithm which is known as Doc2Vec.

Basic of Doc2Vec

Doc2vec is an extension to the word2vec approach towards documents or paragraphs. The intention is to encode all documents, consisting of lists of sentences, rather than lists of words in a sentence. While Word2Vec computes a feature vector for every word in the corpus, Doc2Vec computes a feature vector for every document in the corpus. Word2Vec works on the intuition that the word representation should be good enough to predict the surrounding words, the underlying intuition of Doc2Vec is that the document representation should be good enough to predict the words in the document. Doc2Vec explores the above observation by adding additional input nodes representing documents as additional context. Each additional node can be thought of just as an id for each input document. At the end of the training process, we will have word plus documents embeddings for documents in the training corpus[**d2v˙embedding**].

Doc2vec has two approaches for vectoring document. Distributed memory version of paragraph (PV-DM) is a small extension to the CBOW model. Instead of using just words to predict the next word, we also added another feature vector, which is a unique document. As in word2vec, another algorithm, which is similar to skip-gram is Distributed bag of words version of paragraph vector (PV-DBOW). Authors state that they recommend using a combination of both algorithms, though the PV-DM model is superior and usually will achieve a state of the art results by itself detail working mechanism can be found in [**le˙distributed˙2014**].

1.2 Motivation

We knew a greedy process cannot guarantee to return the globally optimal DT hence finding it is NP-complete problem. Another issue DT learners create biased trees if some classes dominate. We identified UDTs take single feature point where ODTs takes given number of features combination. In some case where data are linearly scattered, UDT can create over-complex trees that do not generalize the data well hence becomes overfit. In that case, an oblique split could be generalizations of an axis-parallel split. Therefore axis-parallel splits are suitable when the class boundaries are parallel to the feature axes. Oblique splits are useful when the class boundaries can be represented as linear combinations of feature variables. Both techniques have their advantage and limitation, they can work best within their best condition. But from prospective of data,

their properties and representation data plays an important role to choose the right algorithm. In the real world, the dataset contains many features which are impossible to visualize without using any dimension reduction techniques. Selecting a particular task with a hypothesis may reduce the problem.

Generally, UDT and ODT models are the choices of trade off between generalization and interpretation of the model. We have also seen that finding oblique splits can be more computationally expensive than searching for the axis-parallel split unless we provide optimized selection mechanism. Simply, We don't know which combination of features can provide the best split at what nodes. This also doesn't assure best is a global optimal tree. Meaning best separation in root node doesn't assure the best tree at the end, a case example is available in appendix A.2. A good part of ODTs is it guarantees small tree with competitive performance. Many studies have shown that trees which use oblique splits generally produce smaller trees with better accuracy compared with axis parallel trees[**wickramarachchi˙hhcart˙2015**]. When feature space is in higher dimension using linear combination would be a good choice to reduce the size and computation. We assume that Doc2Vec representation creates somewhat a linear feature space. Therefore as an assumption in this thesis, we experiment to explore all possible dimension of results.

Text classification domain is an example of a higher dimension problem. The amount of data required to solve a particular problem is enormous. Normally requires thousand of an example where an example can consist of more than hundreds of words or a couple of paragraphs. Hence, representing text as a feature can make ten to hundreds of dimensions. Our Doc2Vec approach takes an argument as vector size to learn model in and to provide a fixed length of given vector dimensions. Depending upon problem selecting the right size of a vector can make a difference in the result. This is also an evaluation part of our experiment. Let's take the previously given example of spam classification, the feature is placed in the linearly separable form in two-dimension and compare the possibility.

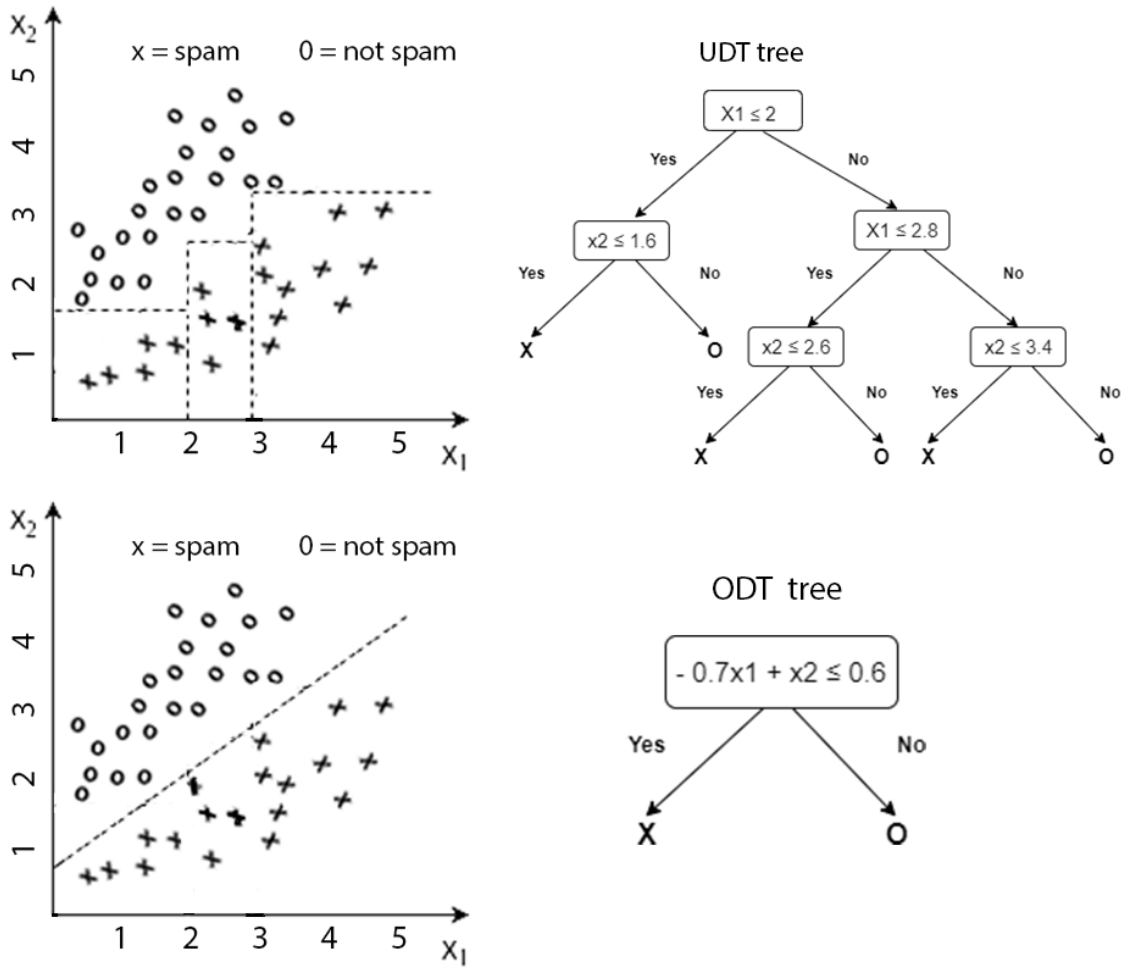


Figure 1.3: The general approach of UDT (top) has poor approximation and ODT (bottom) is the one we need, a linear function

Previously, In figure 1.2 we saw how well UDTs estimate function if feature points are separable in axis-parallel. Whereas in figure 1.3 they can not approximate well because this time regions can not separable in axis-parallel. The target concept that we need to learn is a linear function on $\mathcal{D}_{tr}$. A univariate decision tree (top) approximates the target concept very roughly. The model generates a complex tree and its overfitted. On the other hand, a multivariate decision tree (bottom) learns the linear decision boundary using only a single split hence more generalized and short depth. Therefore we are interested to see in the domain of text classification whether we get similar results or not. In the next section, we define a more focused research question for our experiment.

1.3 Research question

Our research question is classified into 3 scenarios. In which we are going to see if the oblique tree is better than a normal decision or not as well as comparing two ODTs. We will seek comparison in terms of:

- **General evaluation matrix:** We will compare four evaluation matrix accuracy, precision, recall and F1 for all algorithm.
- **Efficiency of tree attribute:** It will be an overall comparison of tree depth, number of nodes, branch size. In the case of ODTs, we also compare intermediate result focus on depth versus accuracy.
- **Evaluation based on embedding sizes of Doc2Vec model:** It will be interesting for us to see if the size of embedding makes any difference with the performance or not. To evaluate this we adapt the above two questions.

1.4 Proposed approach

Based on the research question we have three different algorithms to experiment. We select CART as our UDT algorithm likewise for ODT we have used the induced method on CART algorithm. Those two induced methods are from a linear model of logistic regression (LR) and RANdom SAmple Consensus (RANSAC).

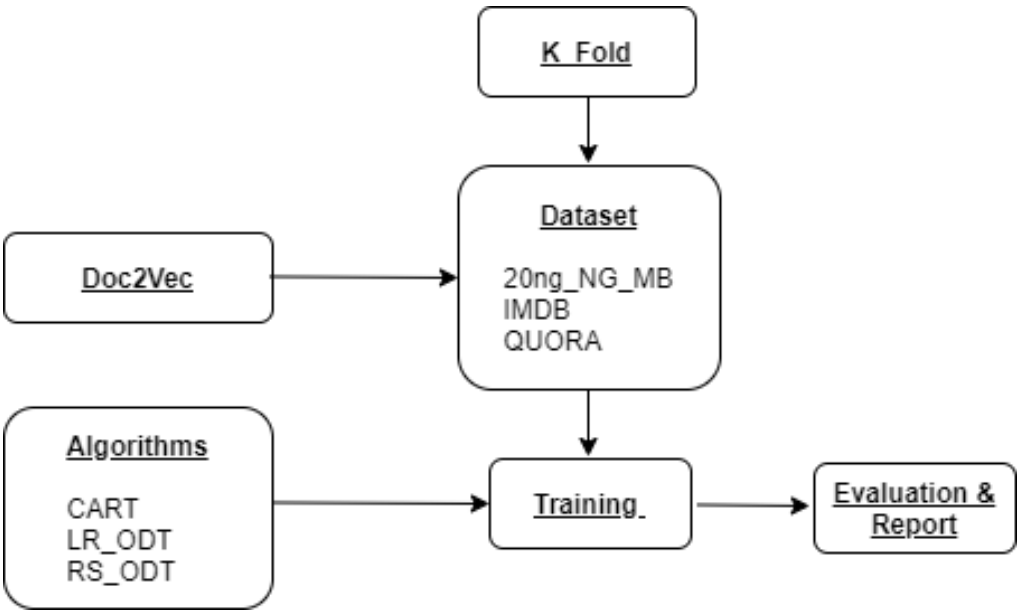


Figure 1.4: Basic work flow of proposed approach.

We have a CART and two induced ODT algorithms. We will test on three different datasets. These datasets have different properties and have two classes only. With the help of k_fold validation, we get an average result. We use Doc2Vec to train k_fold split datasets to train and building features. Then we fit on all available algorithm. In the end, we analyze and report our findings based on our research question. The detail process and setting of each are parts expressed in method chapter no. 3. In the next section, we discuss the structure of our thesis.

1.5 Structure of thesis

Heretofore, In chapter 1 we familiar with basic of decision tree and their types, continuously about NLP semantic model word embedding which leads to motivation of our thesis then we finish with research question and approach. Chapter 2 we will give more information about the history of the problem, what solutions exist, model merging technique and feature selection strategies. In-depth information of ODTs and CART is available in chapter 3. A systematic workflow, Configuration of experiment, data and preparation and result exposing process will take place in chapter 4 which gives the required result to answer our research question in. Finally, in Chapter 5 and 6 we discuss and conclude our thesis.

2 Background

2.1 Related work

In this chapter, a brief survey of oblique DT induction methods is presented as related work. We will see available algorithm merging concept, feature selection and engineering process. A detailed description of the partitioning strategy used in CART is given in Chapter 3. Though the other related methods are not directly relevant to the principal methodology proposed in this thesis, some of them are briefly illustrated in this chapter because those are the building block for construction and evaluation.

The common top-down DT induction has two stages; the first stage is responsible for building the full tree and second is for making it generalized by pruning the tree to avoid overfitting. However, some tree induction algorithms use pre-pruning techniques so that pruning is done while the tree is being built. The main job of the tree growing stage is to search for the best split. Choosing a lookup method for the best oblique split is crucial in DT induction. Time efficiency of the tree building process depends heavily on it. To great extend [**Heat˙et˙al˙SADT**] proves that the problem of finding an optimal oblique split is NP-Complete using the number of miss classified examples as the error measure. Many DT induction methods work in different ways. Our thesis is mainly influenced by the prior concept of merging linear classifier and deterministic search method from section 2.1.1. Therefore, in this section, we expose some oblique DT induction methods in brief based on creation. This section is divided into two part one which shows induction algorithms that use optimization, heuristic and statistic approach and next is combining algorithms.

2.1.1 Existing algorithms

CART Linear Combination (CARTLC)Uses a deterministic hill-climbing algorithm to search for the best split at an inner node. At each inner node, the algorithm perturbs each coefficient of the hyperplane until the algorithm finds a split that produces the maximum impurity reduction. To reduce the risk of a local minimum is found, each perturbation starts from three different pre-specified locations. A backward feature elimination process is also carried out to delete irrelevant features from the split[**breiman˙classification˙1984**].

A randomization approach is introduced called Simulated Annealing DT (SADT) which uses the simulated annealing optimization algorithm to search for the best split[**Heat˙et˙al˙SADT**]. At each non-terminal node, an initial hyperplane is set such that it is not parallel to any feature axis. Next, the algorithm picks one coefficient at a time randomly and adds a random quantity. The resulting hyperplane is then tested using an impurity measure and if the impurity reduction is negative, then the new hyperplane split is always accepted. If it is positive, then the new hyperplane is accepted with a probability function. Initially, T is set large so that when is small compared with T, the probability of accepting a worse hyperplane is approximately equal to 1. However, T is gradually decreased and hence, the probability of choosing a worse hyperplane tends to zero. This process is repeated until there is no further reduction in impurity. A distinct feature of this algorithm is that a series of locally optimal decisions are not necessarily made. Accepting a worse split from time to time can potentially lead to a globally optimal tree. The main disadvantage of the algorithm is the time taken to find the best split. In some cases, it may require the evaluation of the tens of thousands of hyperplane before finding an optimal split[**murthy˙system˙1994**].

[**murthy˙system˙1994**] combine the concept of CARTLC and SADT to introduce a new oblique DT methodology called OC1. First, it uses a deterministic hill-climbing algorithm to perturb the hyperplane until a local minimum of an impurity function is found. The hyperplane is then perturbed randomly to potentially leave the local minimum. These two steps are performed several times. Each time, the algorithm starts with a different initial guess. One of the initial guesses is the best axis-parallel split. A random hyperplane is also used as initial guesses. Since each initial guess potentially converges to a different hyperplane, the one that maximizes the impurity reduction is taken as the splitting hyperplane.[**murthy˙system˙1994**] show that the time complexity

at each non-terminal node for OC1 in the worst-case scenario is O(pn2 logn) provided that Max-Minority or Sum-Minority impurity measures are used. For other functions, obtaining a similar upper bound is an open question. Furthermore, the amount of work that OC1 does at a non-terminal node mostly depends on the number of times that OC1 evaluates the impurity function to find the best split at a non-terminal node. The number of impurity function evaluations made by OC1 can be derived as follows. The OC1 algorithm is used in the experiments of this thesis. One feature of both the SADT and OC1 algorithms is that they can construct different DTs on different runs using the same learning sample. Therefore, it is possible to run these algorithms multiple times and pick a tree which produces the minimum classification rate. However, realizing this advantage is tough when the learning sample contains a large number of examples and features.

[**Lim˙et˙al˙2000**] purpose a statistical method to generation unbiased tree called Quick Unbiased Efficient Statistical Tree (QUEST). Splits are determined by running quadratic discriminant analysis using the selected input on groups formed by the target categories. This method results in speed improvement over exhaustive search (CART) to determine the optimal split. It uses a sequence of rules, based on significance tests, to evaluate the input fields at a node. For selection purposes, as little as a single test may need to be performed on each input at a node. QUEST can find oblique splits which are a linear combination of qualitative and quantitative features.

Geometric DT (GDT) is another oblique DT that exploits the geometric structure of the data [**manwani˙geometric˙2012**]. For a two-class classification problem, the algorithm generates two clustering hyperplanes, one for each class. Loosely speaking, each clustering hyperplane tries to minimise the distance to examples in one class while maximising the distance to examples in the other class. The separating hyperplane is found by calculating the angular bisector of the two clustering hyperplanes. Since there are two angular bisectors, the one that minimizes an impurity measure is chosen as the splitting hyperplane. For a multi-class classification problem, the authors suggest forming two super-classes where one super-class contains the class which has the most examples and the remaining examples from the other classes are grouped into the other super-class. GDT does not require a search procedure to select splitting hyperplanes. At each non-terminal node, it only requires two evaluations of an impurity function. An algorithm proposed in this research is used to improve the performance of GDT.

The CARTopt algorithm introduced by [**robertson˙cartopt˙2013**], uses a two-class

oblique DT to find a minimiser of a non-smooth function. Initially, the examples are labelled into two classes: "high" and "low" depending on their value of f(x). One of the main tasks of the CARTopt algorithm is to identify a rectangular region which contains the most "low" points. The authors use axis-parallel partitions to identify the rectangular region. However, if the orientation of the "low" points is not aligned with the coordinate axes, the axis-parallel partitions will approximate the entire rectangular region by a series of small rectangular regions. To simplify the partition structure, they use a transformation, by which the orientation of the "low" points becomes parallel to one of the coordinate axes in the transformed space. The axis-parallel splits can then be searched in the transformed space to find the rectangular partition structure which contains the "low" points. The transformation is done using the Householder matrix and further details of the Householder matrix. In this study, the concept used in the CARTopt algorithm to create partitions is extended in several ways to develop a complete oblique DT for statistical data classification.

The method used in the CARTopt algorithm to construct feature space partitions is extended in many ways to develop a complete oblique DT called HouseHolder Classification and Regression Tree (HHCART). First, CARTopt is designed to classify two classes whereas HHCART can handle multi-class classification problems. Second, CARTopt reflects the training examples only at the root node whereas HHCART performs reflections at each non-terminal node during tree construction. This is an important part of the proposed algorithm, particularly for multi-class data classification. Finally, CARTopt deals only with quantitative features whereas HHCART is capable of finding oblique splits which can be linear combinations of both quantitative and qualitative features. This step enables HHCART to be applied in any feature space and hence, broadens the applicability of the algorithm. Two versions of HHCART are proposed: HHCART is based on all possible eigenvectors of all classes, and HHCART is based on only the dominant eigenvector of each class.

At node t, the first approach of HHCART finds all eigenvectors of the estimated covariance matrix for each class whereas in the second approach HHCART finds only the dominant eigenvector of each class. A Householder matrix is constructed for each eigenvector. Then Dt is reflected using each Householder matrix and axis-parallel splits are performed along each coordinate axis in the reflected space. The best axis-parallel split is chosen as the separating hyperplane at node t. However, if an eigenvector is already parallel to any of the feature axes, no reflect is done and hence, axis-parallel splits are

searched in the original space. The hyperplane found by the search divides node t into two child nodes. The algorithm is recursively run on all child nodes until each child node satisfied either: The miss classification rate at the child node is not greater than a user-specified threshold or the number of examples in the node is less than or equal to a user-specified threshold. The empirical results show that HHCART induces better trees, in terms of accuracy and the tree size than that of other DT algorithms for most of the problem domains. The algorithm is designed to convert qualitative features into quantitative features and thereby HHCART is capable of handling both qualitative and quantitative features in the same oblique split. In HHCART, the creation of a new feature (or artificial feature) spaces using the Householder reflection can be viewed as an attempt to expand the search space[**wickramarachchi˙hhcart˙2015**].

2.1.2 Merge algorithms

There has been some improvement found by combining classification algorithm over classic CART. Most of these merging is done to get best split over any direction. The split given is taken which creates oblique decision trees. [**boukis˙evolving˙2007**] purpose ODT classifiers using Support Vector Machines to compute the optimal separating hyper-plane for branching tests using subsets of the numerical attributes of the problem. The resulting decision trees maintain their diversity through the inherent instability of the decision tree induction process. We then describe an evolutionary process by which the population of base classifiers evolves during run-time to adapt to the newly seen instances. Another SVM application is used in [**nima˙corr**; **Vlado˙SVM˙Oblique˙tree**].

[**niel˙logistic˙2005**] has used linear regression functions at the leaves. They present an algorithm that adapts this idea for classification problems, using LR. Using a stage-wise fitting process to construct the LR models that can select relevant attributes in the data in a natural way, the models represent decision at the leaves by incrementally refining those constructed at higher levels in the tree}. Like, wise our one experiment is based on LR. Each non-terminal node gets its decision from LR fitting more the detail explanation is available in Chapter 3 method. DT induction methodology was also influenced by the development of artificial neural networks,evolutionary and genetic algorithm[**boukis˙evolving˙2007**; **siu˙automatic˙2019**; **hehn˙end–end˙2020**; **bandar˙genetic˙1999**].

2.2 Linear model

In this section, we discuss two linear models which are uses as a core part of our induction method. These models are used for classification purpose.

2.2.1 Logistic regression

LR is a statistical function used in ML. LR is linear regression model, instead of a linear function it uses Sigmoid function(σ) or logistic function as cost function. The σ hypothesis h is lies between 0 and 1. Our approach takes advantage of LR to find θ and split data on each decision node.

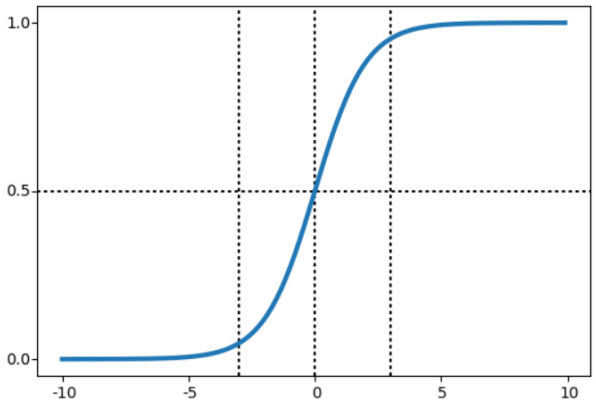


Figure 2.1: Sigmoid curve

$\sigma = f(x) = \frac{1}{1+e^{-(x)}}$, this a formula of sigmoid function, when we use linear formula h will be $h_\theta(x) = \beta_0 + \beta_1 X$ but slight change in of σ will use to get out put in the form of 0 and 1 therefore the formula for LR will be $h_\theta(x) = \sigma(\beta_0 + \beta_1 X)$. β represent parameter of model where βo is bias and $\beta 1$ is coefficient of X hence final function would be

$$h_\theta(x) = \frac{1}{1 + e^{-(\beta_0 + \sum \beta_i X i)}}$$

We expect the LR model output would be using probability when we pass the inputs through a prediction function and returns a probability score between 0 and 1. We basically decide with a threshold value which helps to predict classes for example if threshold value 0.5 then if probability ≥ 0.5 then $y = 1$ else $y = 0$. This means predicting class 1 whenever $\beta_0 + \sum \beta_i.X$ is non-negative and 0 otherwise[**stat˙cmu˙edu**].The requirement

of a problem can make a change in threshold which is known as threshold tuning. The cost function is in log base.

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Lets see an example how cost above function works, if $y = 1$ and $h(x) = 1$, then cost is 0 but when $h(x)$ is 0 then cost will be infinite hence the learning algorithm will penalized by large cost. Then fitting approach is applied when the prediction of the class goes wrong. The objective of the training is to set the parameter vector θ so that the model estimates high probabilities for positive instances (y = 1) and low probabilities for negative instances (y = 0). The cost will be close to 0 if the estimated probability is close to 0 for a negative instance or close to 1 for a positive instance. Gradient descent applies to minimize LR cost function by tweaking the β, in-depth explanation is available in [**ruder˙overview˙2017**].

[**fan˙liblinear**] said the solver liblinear is a good choice for document classification. It uses for performing linear-classifier learning both binary, multi-class, and regression. It supports various training methods and objective, such as SVM and Logistic Regression, with different regularization terms. For our experiment purpose we are using scikit-learn based logistic regression algorithm we have selected liblinear[**scikit-learn˙2020**]. In the next section, we learn about the RANSAC algorithm.

2.2.2 RANSAC based random sample splitting

Although our second ODT strategy is based on RANSAC, we only utilized a random sample selection process. Our approach does random sample selection from each class to find linear split. RANSAC we proposed by [**fischler˙random˙1981**], an iterative general parameter estimation approach designed to cope with a large proportion of outliers in the input data. It assumes that all of the data are comprised of both inliers and outliers. Inliers can be explained by a model with an approximation fit, while outliers do not fit that model in any circumstance. It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability, with this probability increasing as more iterations are allowed. A random sample of observation are used to learn the model. RANSAC has two main step hypothesis and test.

1. **Hypothesis**
   First, minimal sample sets are randomly selected from the input $\mathcal{D}$ and the model parameters θ are computed using only the elements of minimal sample sets. The cardinality of a random sample set is the smallest sufficient to determine the model parameters. As opposed to other approaches, such as least squares, where the parameters are estimated using all the data available, possibly with appropriate weights.

2. **Test**
   In the second step RANSAC check which elements of the entire dataset are consistent with the model, the model instantiated with the parameters estimated in the first step. The set of such elements is called a consensus set

These two steps are repeated until the given iteration. Whenever the best consensus set is found it updates the value. Iteration of random selection here, Let p be the probability sampling from $\mathcal{D}$. Assuming at least one outlier is picked then the probability choosing one point yields an inliner is 1-p. i is a number of iteration then it must be larger enough to reduce probability (1-p)^i is smaller or equal than a certain probability threshold ϵ, i.e (1-p)$^i \leq \epsilon$, hence we can write

$$i \leq \frac{log\epsilon}{log(1-p)}$$

Algorithm 1 Pseudo algorithm of RANSAC

Input: $\mathcal{D}_{tr}$, e: number of iteration to select random data, N: Number of sample to select
Output: θ: estimated fit

1: Selects N data items at random
2: Estimates parameter θ from cost function
3: Update θ which reduce the cost
4: Repeat 1 to 4 e times and return θ

This procedure is repeated a fixed number of times, each time producing either a model which is rejected because too few points are classified as inliers or a refined model together with a corresponding error measure. In the latter case, we keep the refined model if its error is lower than the last saved model. RANSAC is a non-deterministic algorithm producing only a reasonable result with a certain probability, which is dependent on the number of iterations (see max_trials parameter). Typically used for linear problems or find outliers and especially popular in the field of photogrammetric computer vision[**sklearn˙ransac˙2020**].

2.3 Other feature selection strategies

Feature selecting process has always been a crucial step in ML. As the amount of feature spaces increase or dimension of data several challenges occurs. Comprehensibility and efficiency of the model have highly depended on dimension. The following content explains some of the feature selection process, which aims to remove irrelevant features or reducing dimension.

[**yu˙feature**] purpose a novel concept of predominant correlation, introduce an efficient way of analyzing feature redundancy, and design a fast correlation-based filtering approach. By skipping the pairwise process of checking redundancy they have used a heuristic approach to skip some of the processes to get fast computation. A similar approach is taken by [**shu˙d-cca˙2020**] using a matrix to get 3 different properties; low-rank common matrix that captures the shared information across datasets, a low-rank distinctive matrix that characterizes the individual information within a single dataset, and an additive noise matrix. The latter guarantees that no more shared information is extracted able from the distinctive matrices and similar correlation-based approach are in[**steeg˙discovering˙nodate**; **doshi˙correlation˙2014**].

The various methods used for dimensionality reduction include principal component analysis, linear discriminant analysis, generalized discriminant analysis have been used in different ways to select features in lower dimension without losing much amount of information[**yata˙effective˙2012**; **blei˙latent˙nodate**; **singh˙generalized˙2009**; **baudat˙generalize**
[**goos˙feature˙1998**] has shown a feature transformation approach via use of neural network like wise clustering and evolutionary algorithms also been applicable as feature selection process[**hutchison˙genetic˙2014**; **khanbabaei˙use˙2013**; **zhang˙decision˙2008**].
A summarized survey on ODT and existing feature selection strategies is available in [**brodley˙multivariate˙1995**].

3 Methods

In this section, we are going to know about our 3 algorithms, how they are induced what strategy has been applied. An algorithmic form is used for each to understand more.

3.1 CART

A famous binary splitting decision tree for classification and regression. Gini impurity function uses to measure the goodness of split for each node. It learns to partition on the basis of the single feature value. A greedy recursive approach uses to build a full-grown tree. In the beginning whole training set $\mathcal{D}_{tr}$ is considered at the root. Feature values are preferred to be categorical, if values are continuous they are discretized prior to building the model. We have selected a scikit-learn library to use decision tree algorithm [**sklearn˙DT˙2020**]. This algorithm will be our base comparison for other two ODT. For our experiment we have selected two hyperparameters they are;

1. **min_samples_leaf:** A threshold point where algorithm stops growing tree and return probability each class by being as a leaf node, Then we choose dominated class from probability, In the case of equal instance we select a random approach, In section 4 we share detail explanation of each hyperparameters.

2. **criterion**: Since we want to take CART algorithm for UDT and sklearn offers two variations of impurity measurement we choose to stick with Gini impurity. We propose a comparison on the full-grown tree, therefore, rest of the hyperparameters are not taken.

Typically CART induction works in top-down order. Starting from root node till leaf nodes. Decision tree grows in depth-first order in each step followed by divide and conquer strategy [**breiman˙classification˙1984**]. First, select an attribute to place at

the root node and make two branches for this attribute based on criteria Gini index which is also known as feature selection measure. Afterwards, training instances are splits into two subsets, one for each branch extending from the root node. Normally left to right process is taken to create branches. Summation of these two branches or child node is equal to their root node or parent node. All instances at a node have the same class label (pure node) then splitting stops by putting information (typically label and number of instance) of the pure node. This splitting process continues until all nodes have pure instance from one class or some given condition is fulfilled as an example maximum depth of tree or minimum samples leaf. If a leaf node contains more than one class labels due to early stop then use majority voting is used to set its class. Tree deduction process follows the same approach as given in section 1.1.1 in steps of top-down DT deduction.

Algorithm 2 Pseudo algorithm of CART induction

Input: $\mathcal{D}_{tr}$, min_leaf_point: minimum number of sample in a node to split
Output: T, full grown tree as approximation function
Ensure: $\mathcal{D}_{tr} \neq 0$

1: Start with all training sample at the root.
2: Choose the best attribute using Gini impurity measurement, add as decision rule.
3: Split data using above measurement & create two branches.
4: Repeat 1-3 step for each branches their associated inner branches.
5: Mark as leaf node if all data point in branch have same class or min_leaf_point condition meet

3.2 Random Feature Selection (RFS)

We saw many examples of feature selection in section 2.1.1 and 2.3. In our experiment, we propose random techniques to select a feature. This process requires to define fix number combination size n and then use that size to select random feature index from x of *2 to len(x)* for every t node. This is a greedy approach and we assume that given n will give the full-grown tree. It is known that finding the best set of feature is an NP-Complete problem[**goos˙feature˙1998**]. Since the problem is very hard, without using optimization or feature engineering techniques we cannot guarantee global optimal tree. Hence for our experiment, we choose to stick with a set of n features and evaluate based on that result.

Algorithm 3 Pseudo algorithm of RFS

Input: $\mathcal{D}_{tr}$, n: number of features
Output: Di: subset of select features data only,
Ensure: n $\geq$ 1 and $\leq$ len(X)
1: Get n feature index (i) from random process
2: Use i to create new data ($rand\_D$) set containing only i features
3: **return** $rand\_D$

3.3 Induced logistic regression decision tree

Previously in chapter 2.2, we saw how LR work. For a recall, logistic regression is a linear function which is used for classification. With the help of coefficients, it creates hyperplane. The sigmoid function gives result based on a default of 0.5 probability threshold value which can be configurable based on the requirement. In our experiment sklearn based LR model is used [**sklearn˙LR˙2020**]. Here shows, the process of the growing tree happens in 3 stages.

1. **RFS**: We select n different type of features randomly.
2. **LR fit**: After training on the random feature set from RFS, we get a decision boundary for node t as θ
3. **Building Tree**: Using inbuilt predict method we split the entire data into two branches and follows the same process again.

At first, n combination of feature is generated then we train the LR model only using that n features. We use the model coefficient and intercept as the decision boundary. Afterwards, we separate based on the predict function of that model. We apply to predict method on the same feature and get output. Predict method is given function in sklearn based LR. Finally based on the output index we separate the whole dataset. Now the newly created subsets again go through the same process by fixing the same size of features. This process repeatedly happens until the subset contains data from the same label or meet some stopping criteria. In the case, where n are not enough to map linear combination then we stop growing tree. In the future update will discuss updating RFS index by keeping the same size of n to overcome a large number of the impure partition. This process lets our algorithm to not stick with the worse split but somewhat gives a split. If no improvement by measuring entropy=0 then we make that branch as a leaf node and continue rest.

Algorithm 4 Pseudo algorithm of LR_ODT induction

Input: $\mathcal{D}_{tr}$: training dataset, e: number of epochs, E: entropy, m_point: min_leaf_point, n_fet: feature size.

Output: T, all full grown tree as approximation function.

procedure BUILD_TREE($\mathcal{D}_{tr}$, e, m_point, n_fet)
    If $\mathcal{D}_{tr} = c_{i=1}$ return c_i as leaf node ▷ or E=0
    If len($\mathcal{D}_{tr} \leq min\_point$) return c_i
    θ, pred = LR_FIT(e, RFS(n_fet, $\mathcal{D}_{tr}$) ▷ pred index use for partition
    D1, D2 = Partition($\mathcal{D}_{tr}$, pred)
    Recursively train for each subsets, BUILD_TREE(D_i) ▷ D1, D2 are $\mathcal{D}_{tr}$ for next iteration
    return T(θ, n_fet)
end procedure

Find best split using sklearn LR method

procedure LR_FIT(e, X, y) ▷ epochs and subset from RFS
    LR(e).fit(X, y)
    return θ, predict(X)
end procedure

Algorithm 5 Pseudo algorithm of ODT deduction

Input: T: tree, x_i: unknown instance

Output: Class prediction of x_i

procedure PREDICTION(T, x_i)
    if instanceof(T, Leaf) **then** return c_i ▷ probability measure(if = get random)
    else pt = x_i[T.indexes, 1] ▷ getting indexes from T plus bias
        r = T.θ^T.dot(pt)
    end if
    if r $\geq$ 0 **then** PREDICTION(T.true_branch, x_i) ▷ 0: tunable threshold
    else PREDICTION(T.false_branch, x_i)
    end if
end procedure

A prediction approach follows the same approach for both explained ODTs. At each inner node, θ value and feature index are stored to predict new instance. Prediction process takes a tree and instance tuples then the process checks for the leaf node to return the class as predict output. We first find feature index and θ from the learned model next with feature index we select only that feature for the new instance to apply matrix multiplication as $\theta^T.X$. If the result is less than or equal to 0 or negative value we go for false branch else true branch. At some point, the branch finds a leaf node where prediction value comes. Algorithm 5 shows our pseudo approach for ODT deduction.

3.4 Induced random split decision Tree

In Chapter 2.2.2 we learned the basic mechanism of RANSAC algorithm. Our proposed implementation uses RANSAC's iterative random process to find the best hyperplane from the given number of iteration. Entropy has applied to measure the goodness of the hyperplane and splitting process follows. Since RANSAC is a linear model to apply in classification problem internal changes is required. We need to modify in such a way that θ values can be utilized to create hyperplane for splitting $\mathcal{D}$. We are using [**nghiaho12˙mvdt˙2012**] approach for creating line in DT. A linear regression model also available in [**sklearn˙ransac˙2020**]. The oblique line is created through two random sample points from each class. A line is created that goes directly between two points by finding the middle point which works as intercept and coefficient is the difference of two points. based on θ we split our data less than equal 0 conditions. Here shows, the process of growing tree in 3 stages.

1. RFS: Process of getting n different type of features randomly is same with LR_ODT
2. Random sample fitting: Once we get a subset of features from RFS, we first separate classes into two classes, hence each partition will homogeneous. Afterwards, we pick one data instance from each class then we create a line. Finally, we test the entropy of hyperplane for the whole dataset. This process will continue until the given iteration finishes. Then we take a hyperplane which gives high information gain.
3. Building tree: We take that hyperplane parameter as our decision boundary. We split the dataset into two branches and continue the same process from 1.

Algorithm 6 Pseudo algorithm of RS_ODT

Input: $\mathcal{D}_{tr}$: training dataset, e: number of epochs, m_point: min_leaf_point, n_fet: feature size.

Output: $\boldsymbol{T}$, all full grown tree as approximation function.

procedure BUILD_TREE($\mathcal{D}_{tr}$, e, m_point, n_fet)
    If $\mathcal{D}_{tr} = c_{i=1}$ return c_i
    If len($\mathcal{D}_{tr} \leq min\_point$) return c_i ▷ probability measure(if = get random)
    θ, pos_side, neg_side = GET_BEST(e, RFS(n_fet, $\mathcal{D}_{tr}$), PARTITION($\mathcal{D}_{tr}$)) ▷ classes data points(pos & neg)
    D1, D2 = PARTITION($\mathcal{D}_{tr}$, pred)
    Recursively train for each subset, BUILD_TREE(D_i) ▷ D1, D2 are $\mathcal{D}_{tr}$ for next iteration
    return T(θ, n_fet)
end procedure

Find best split after e iteration

procedure GET_BEST(e, X, y, pos, neg)
    for e **do**
        p, n = random(pos, neg) ▷ get random two points from each class
        θ, G, pos_pred & neg_pred = SPLIT_DATA(X, y, pos, neg, n, p)
        if G ¿ previous G **then** update G, pos_pred, neg_pred
        end if
    end for
    return θ, pos_pred, neg_pred
end procedure

Finding decision boundary θ

procedure split_data(X, y, pos, neg, p, n)
    Find coefficient (β_X) = p-n
    Find mind point (m) = (p+n)/2
    Find bias (β_0) = -β_X.dot(m), $\theta = [\beta_X, \beta_0]$
    if X.dot(θ^T) ≥ 0 **then** $c_{i=1}$, pos_side ▷ decision boundary
        Else $c_{i=0}$, neg_side
    end if
    Compute G ▷ find information gain after split
    return θ, G, pos_side and neg_side ▷ pos, neg are after prediction
end procedure

4 Results

In this section, we are going to see how the whole experiment is conducted with a flow chart, intermediate preparation steps, configuration and various angle of results observation to answer our research question.

4.1 Experimental setup

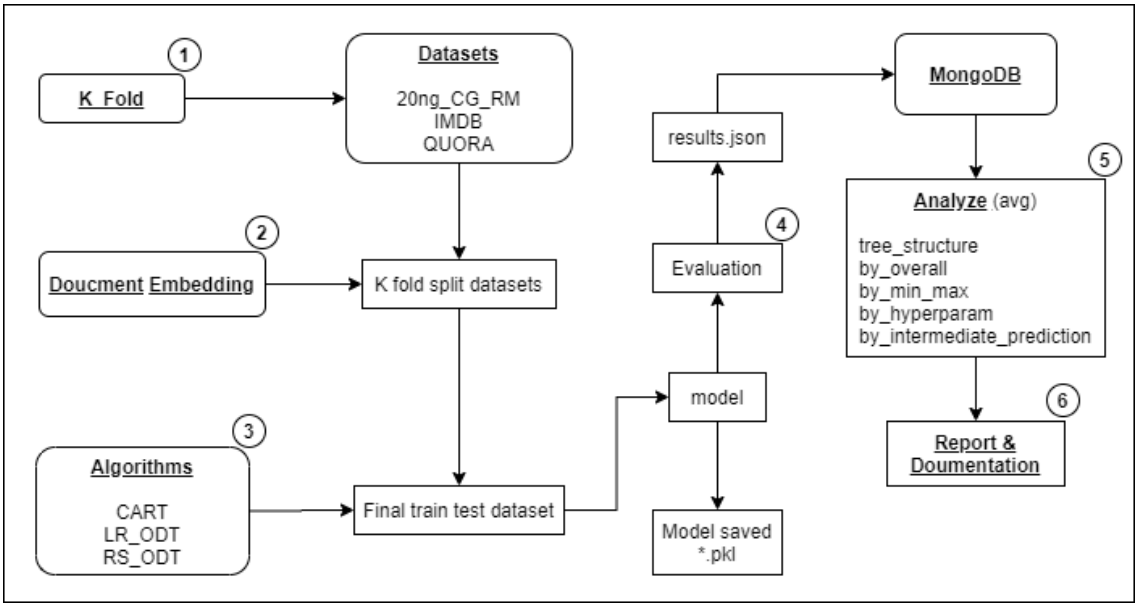


Figure 4.1: Flow char of our experiment process

1. **K_Fold cross-validation:**
   Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. Normally, we split our dataset in training and testing part. After the training process completes we test the performance of our model in the test dataset. Comparison both sets gives a clear evaluation of our model

but the case is not always true. However, splitting only a one set of train and test parts can mislead the evaluation. This situation is possible because even though the splitting process is random it can create differences in results if we take two different sets of train and test. One set can give excellent result than other hence we can not confidently rely on it. To overcome this situation a K fold cross-validation process is applied. K fold divides into a given number of folds ensuring each fold is used as a testing set at some point. The average performance of K will be the final result.

Figure 4.2: K fold cross-validation, when K = 5

Above figure show K_fold cross-validation process and the average formula. In this process, whole data are being utilized for training and evaluation. In our experiment, we have chosen K = 5 for each dataset. When K is 5 it generates 20% test data rest for training. In the next step, we use Doc2Vec for each K folds.

2. **Document embedding:**
   Once the k fold data are created we use Doc2Vec embeddings on train set of each. The model learns via Doc2Vec algorithm which is take form [**gensim˙2020**] library. The learn model maps the similarity of each document in a higher dimension, then we create feature embeddings for both train and test sets using the same model. Newly created features will be our final dataset. We have five different vector sizes for our experiment they are 10, 25, 50, 75 and 100. These ranges help us to study the impact on DT. Here are following the setting of this language model;

   a) Vector_size: [10, 25, 50, 75, 100]

   b) Implementation: PV-DM

c) Epochs: 40

d) Min_count: 2

3. **Algorithms:**
   We have three different algorithms, Once we have full feature representation ready we apply each algorithm for training and testing. Hyperparameters setting is available in chapter 4.3. After finishing the fitting process we saved our model and start the evaluation process next.

   a) Sklearn based DT (CART)

   b) LR induced DT (LR_ODT)

   c) RANSAC induced DT (RS_ODT)

4. **Evaluation:**
   In this step we extract performance of model this includes accuracy (acc), precision (pre), recall (rec), f1 score (f1) and confusion matrix likewise also extract the tree attributes such as maximum depth (max_depth), a number of nodes (inner and leaf), left and right branch node counts, prediction label, true label, training time and intermediate prediction till depth 19. The intermediate prediction does prediction with only using a certain depth of the tree. As an example we if give depth 1 then the prediction happens only root decision node where if give depth 19 then model is considered as till tree has 19th max depth. If in the case where some model less depth than given then we consider as none. The result of the evaluation information is saved in results.json file. After the process ends we analyze them using MongoDB technology.

5. **Analyze:**
   At first, a python script is used to store information in the database. We selected MongoDB because it provides easy manipulable query environment. We can access a various range of inbuilt queries such as averaging, sorting, grouping and parameterized selection are some. Again we use python script with MongoDB query to separate our results. For our experiment, we use four main information to extraction. All the information is based on test data of given vector sizes with each algorithm of each dataset.

a) Overall evaluation: It is the average evaluation of all hyperparameters separated by vector sizes with algorithms for three datasets, this evaluation includes a, p, r, f1, max_depth and standard deviation (sd) of each.

b) Min Max: This show maximum and minimum accuracy result, including p, r, f1, feature size, depth, inner nodes count, branch sizes and training time.

c) Hyper parameter wise evaluation: We are considering 4 hyperparameters epochs, k fold, min leaf points, combination of features. The Average result of each includes a, f1, depth and sd for all.

d) Intermediate Prediction: Here we store average depth and average accuracy of train and test sets. This intermediate approach is applied for LR_ODT and RS_ODT.

6. **Report & Documentation:**
   The final step would be proper reporting of findings and fact documentation based on the previous step. We explain our results in the form of text, tables and charts.

4.2 Datasets, properties and preparation

We are testing on three different datasets which all are English text. All datasets have two classes only. Below are the problem domain with a few examples.

1. 20_news_group: 20 different categories of news are combined in this dataset. We randomly selected comp.graphics and rec.motercycles we call this 20ng_CG_RM. The dataset contains news related to computer graphics and motercycles races, eg.

   rec.motorcycles (RM): *And they're rather tasty. Per gallon (bushel) perhaps. Unfortunately they eat the same amount every day no matter how much you ride them. And if you don't fuel them they die. On an annual basis, I spend much less on bike stuff than Amy the Wonder Wife does on horse stuff. She has two horses, I've got umm, lesseee, 11 bikes. I ride constantly, she rides four or five times a week. Even if you count insurance and the cost of the garage I built, I'm getting off cheaper than she is. And having more fun (IMHO). Go fast. Take chances.*
   com.graphics (CG): *Why would it have to be much faster (it probably is) ? Assuming an ARM is about as efficient as a MIPS R3000 for integer calculations,*

doing a Compact-Video-like digital video codec is an easy task. For Software Motion Pictures (which is a lot like Compact Video, though it predates it), we get 48 frames/sec. at 320x240 on a DECstation 5000/200. That machine has a 25 Mhz MIPS R3000. Burkhard Neidecker-Lutz"

2. IMDB: It is a sentiment classification problem of movie review from the imdb website. The dataset has two classes positive and negative sentiment, eg.

   Positive (pos): *If you like original gut wrenching laughter you will like this movie. If you are young or old then you will love this movie, hell even my mom liked it. Great Camp!!!*
   Negative (neg): *It's terrific when a funny movie doesn't make smile you. What a pity!! This film is very boring and so long. It's simply painfull. The story is staggering without goal and no fun. You feel better when it's finished*

3. QUORA: It is a Kaggle competition problem[Quora Insincere Questions Classification]. Will be predicting whether a question asked on Quora is sincere or not. The questions are categorized based on the tone of words, disparaging or inflammatory, false information and harassment words. Some portion of the dataset which contains 40k sincere question and 160k insincere.

   #1 Sincere (sncr): *Which is the best football country to never win a world cup? What year were they best?*
   #1 Insincere (insncr): *Do some football fans on Quora ever thought they are being low and unhelpful when they post "no shit Sherlock" answers with a random photo to football questions for upvotes?*

   #2 Sincere: *What are the advantages of having Donald Trump as a president?*
   #2 Insincere: *Why is it racist to refer to Obama as a monkey? Yes it's rude, but I don't see how it's racist.*

   #3 Sincere: *What are some reasons for the disappearances in the Bermuda Triangle?*
   #3 Insincere: *What is below the underworld? I found a tunnel that leads to space below the actual underworld*

Attributes		20ng_CG_RM		IMDB		QUORA	
		cm	***rm***	***pos***	***neg***	***sncr***	***isncr***
docs		973	996	25,000	25,000	20,000	80,000
Word len/doc	***min length***	3		4		2	
	max length	9,181		2,469		62	
All words count		267,548		11,513,119		1,346,847	
Unique words		22,883		49,582		56805	

Table 4.1: Basic properties of datasets

The table shows the basic properties of each dataset. The properties have calculated after basic text cleaning processes. It includes lowering word, removing punctuation and escape sequences like '\n'. In 20ng_CGn_RM datasets, we ignore the document which has less than 3 words. From the table, we can see 20ng_CG_RM and IMDB has an equal distribution of labels whereas QUORA has 20% sincere documents and 80% are insincere documents. Minimum and maximum word length per datasets also gives an insight into how the dataset is generated. Total words and unique words in the whole dataset gives a clear idea of how some words have high occurred and most of the words have a low frequency. Let's see an example of minimum word length documents per datasets with their label.

1. ***20ng_CG_RM:*** *Squeaky BMW riders*
2. ***IMDB:*** *Primary plotPrimary directionPoor interpretation*
3. ***QUORA:*** *Religious Tolerance*

As a further cleaning step, we applied Gensim inbuild function. Such functions are responsible for converting a document into a list of lowercase tokens, ignoring tokens that are too short or too long. In our experiment, we set the default value of those parameters min_len=2 and max_len=15. Since we have used K-fold cross-validation our Doc2Vec model goes through each document and their words at some fold.

Above table 4.2 below shows 10 least and most frequent words and their count for all datasets. As expected in English text top most commons word are always articles, preposition and verbs. In the other part least common words doesn't form any meaning. Therefore, exploiting least and most common word per label does not give any information. It could be an improvement part of our thesis.

20ng_CG_RM		IMDB		QUORA	
Least	***Most***	***Least***	***Most***	***Least***	***Most***
rubberized:1	it:2732	manyaryans:1	this:149359	baronies:1	do:20858
undercoating:1	for:3294	romanceoz:1	i:151966	udacity's, 1	and:21994
von:1	in:3383	viewingthats:1	it:152795	falsly:1	i:22292
welch:1	i:3779	oldtimebbc:1	in:184804	voilent:1	of:26142
vwelchncsauiucedu:1	is:3818	polari:1	is:210134	poltical:1	in:28343
group'93:1	of:5138	halliwell's: 1	to:266749	beleifs:1	what:31258
subaru:1	and:5647	petter:1	of:288360	savegely:1	a:31823
4wd:1	to:6540	habitatbr:1	a:320391	meetic:1	to:33417
aninnovator:1	a:6615	discerns:1	and:320652	redecorate:1	is:33668
microprocessing:1	the:11168	dressedup:1	the:662539	documenting:1	the:53280

Table 4.2: Ten least and most word frequency count of all three datasets

4.2.1 Process of document representation in vector space

Explain how we created doc2vec representation for each dataset as well as how we store final representation. This must include various techniques and hyperparameters setting if required. Explain two techniques and their setting for safe comparison. We can put a formula that shows how many iterations it requires to create all embeddings.

The datasets have different variety of text file and comma separated value files (CSV). First, we create transform various format in one CSV format by manipulating their original structure. Then we follow the given steps in Doc2Vec official site in order to make feature embeddings. Below steps contains the necessary process of creating Doc2Vec embeddings.

1. The feature simple_preprocess in Gensim library create tokens of words per document. At first, the process takes one documents and tokenized then with a unique id same document is tagged. The tagged document later use as separate neuron while training. Internally, tokenization comes with converting the document in lowercase, ignoring tokens that are too short or too long features. In our experiment, we kept the default parameter of too short and too long tokens which is 2 and 15 respectively. The tagged document uses only for train document.

2. Tokenized list of documents is used to build vocabulary to create a similarity matrix in a distributed manner. Three-layer NN maps the relationship in given

higher dimension as explained in chapter 1.1.2. Every iteration on training process minimizes the rate of error by reducing the distance between two documents.

3. Once the learning process completes we use inference method to generate Doc2Vec Features embeddings. Gensim provides an easy method as infer_vector. It uses the learned model and tokenized corpus of train and test data.

4. Above method gives representation for both subsets of data. We save representation in the form of CSV format combining with a true label of it. We get final dataset in tabular format then apply in our purposed algorithms

4.3 Algorithmic configuration

Explain how two algorithms are defined. Briefly describe two used algorithms since the detailed explanation is already up. Algorithms hyperparameters settings are available in the table below.

Algorithms	**Impurity**	**Min leaf point**	**Epochs**	**N_features**
CART	Gini	[2, 5, 10, 20, 30]	-	-
LR_ODT	Sigmoid & Entropy	[2, 5, 10, 20, 30]	[100, 300, 500, 800, 1000]	[2, 5, 10, 20, all]
RS_ODT	Entropy	[2, 5, 10, 20, 30]	[100, 300, 500, 800, 1000]	[2, 5, 10, 20, all]

Table 4.3: Algorithms hyperparameters setting list

4.4 Experimental Result

4.4.1 General time complexity

Runtime analysis of an algorithm depends on many factors. In reality, those factors differentiate by only some constant numbers. Therefore to know general algorithm performance we use BigO notation. Order of growth of algorithm performance is measure through asymptotic notation which is known as BigO notation. It gives a logical interpretation to measure time and space complexity in a large order of input. We keep the assumptions of

1. A full-grown tree with fix finite hyperparameters sizes
2. Considering only worst-case scenario using BigO but not BigΩ or Bigθ.
3. n is number of training example, m is number of feature and e is number of iteration.
4. Global optimal solution is NP-hard problem

Time complexity of CART

[**bramer˙computational˙2018**] has explained the complexity of the decision tree including CART. Even the conducted experiment support the logical explanation.

1. At first estimated complexity of computing the probability for each class labelled is bounded by the size of the sample, hence the cost is O(n).
2. The computation performed on one input attribute requires $O(n \log n)$ and since all attributes are considered then the total cost for this operation will be $O(mn \log n)$.
3. Similarly to analyze the recursive call of the algorithm to the subset of the training set, the estimated complexity for such operation is $O(n \log n)$ since at each partition the algorithm considers the instances and their respective target values.
4. Hence the total running time complexity can be estimated by combining the cost for each the basic operations in decision tree building as:

$$T(\mathcal{D}_{tr}) = O(n) + O(mn \log n) + O(n \log n) = O(mn \log n)$$

5. Based on a binary tree, searching for a node in each branch takes $O(\log n)$ because every time opposite half of the tree branch will not consider for each depth. While matching value takes constant time. Therefore estimated complexity for CART becomes $O(\log n)$.

Time complexity of LR_ODT

This approach mainly consists of two steps; one to fit a logistic model for a given random subset of data and tree construction.

1. The estimated time for creating subset of data based on random selection of features is $O(n)$.
2. LR does matrix multiplication, inversion to find θ, regardless of various solver the time complexity become $O(m^2n)$ [**iyer˙pennsylvania˙nodate**]. Our random feature selection affect computation by constant factor of time hence final complexity would becomes $O(m^2n)$.
3. Similarly to analyze recursive call of the algorithm for each subset it take $O(n \log n)$.
4. Hence sum of all operation and final complexity of written as

$$T((\mathcal{D}_{tr}) = O(n) + O(m^2n) + O(n \log n) = O(m^2n \log n)$$

5. The recursive search takes $O(\log n)$ while every decision node computes θ by $O(n)$ complexity on each level of depth, therefore final estimated prediction time will be $O(n \log n)$.

Time complexity of RS_ODT

Similarly, this approach again follows as steps. The only difference is while creating a linear decision boundary. RS_ODT takes only one observation of from given random samples and creates a line to fit given data.

1. Random subset data creating and sample selection takes $O(n) + O(n) = O(n)$.
2. Similarly, finding θ takes $O(m^2n)$ for matrix manipulation whereas entropy calculation takes constant time.
3. Construction of tree in recursive fashion takes $O(n \log n)$.
4. Finally we can compute time complexity as:

$$T(\mathcal{D}_{tr}) = O(n) + O(m^2n) + O(en \log n) = O(m^2n \log n)$$

5. Again the recursive half tree search take $O(\log n)$ while matrix multiplication on θ for each level becomes $O(n \log n)$ which is the final complexity.

4.4.2 Model structure comparison

Tree Structure		Zero	Perfect	Almost	Left	Right
Dataset	Algo	depth	balance	balanced	skewed	skewed
20ng_CG_RM	LR_ODT	2	250	326	928	1619
	RS_ODT	0	42	205	2151	727
IMDB	LR_ODT	1	85	950	648	1441
	RS_ODT	0	7	964	977	1177
QUORA	LR_ODT	379	16	12	146	2572
	RS_ODT	0	2	236	490	2397

Table 4.4: Tree structure analysis, measure in 5 different types.

A tree is a perfect balanced when both branches from the root node have equal inner nodes regardless of inner branch structures. Almost balanced is measure when both sides have a difference less than 10 percentage. Left and right-skewed are measures when branches more inner nodes in respective sides. We observed RS_ODT doesn't have any zero-depth which confirms approach performs weak learning procedure. In the opposite LR_ODT creates much perfect balance tree than others. Both trees have a similar number of the tree when branch difference is less than 10 per cent. We saw LR_ODT has generates many right-skewed trees where RS_ODT prefers left-skewed tree.

4.4.3 Overall comparison

Our experiment will have greatly affected by various factors such as hyperparameters and vector size from Doc2Vec (D2V). Figure 4.3 shows the average relationship between accuracy and depth concerning each dataset of all vector size. Since CART has only 25 total run per fold and rest have 1250 runs, we can see the effect as well. All the comparison will be based on CART vs ODT and LR_ODT vs RS_ODT. Though each algorithm (algo) has drawback Chapter 6.3 (algo limitation) we look for only interesting reasonable facts.

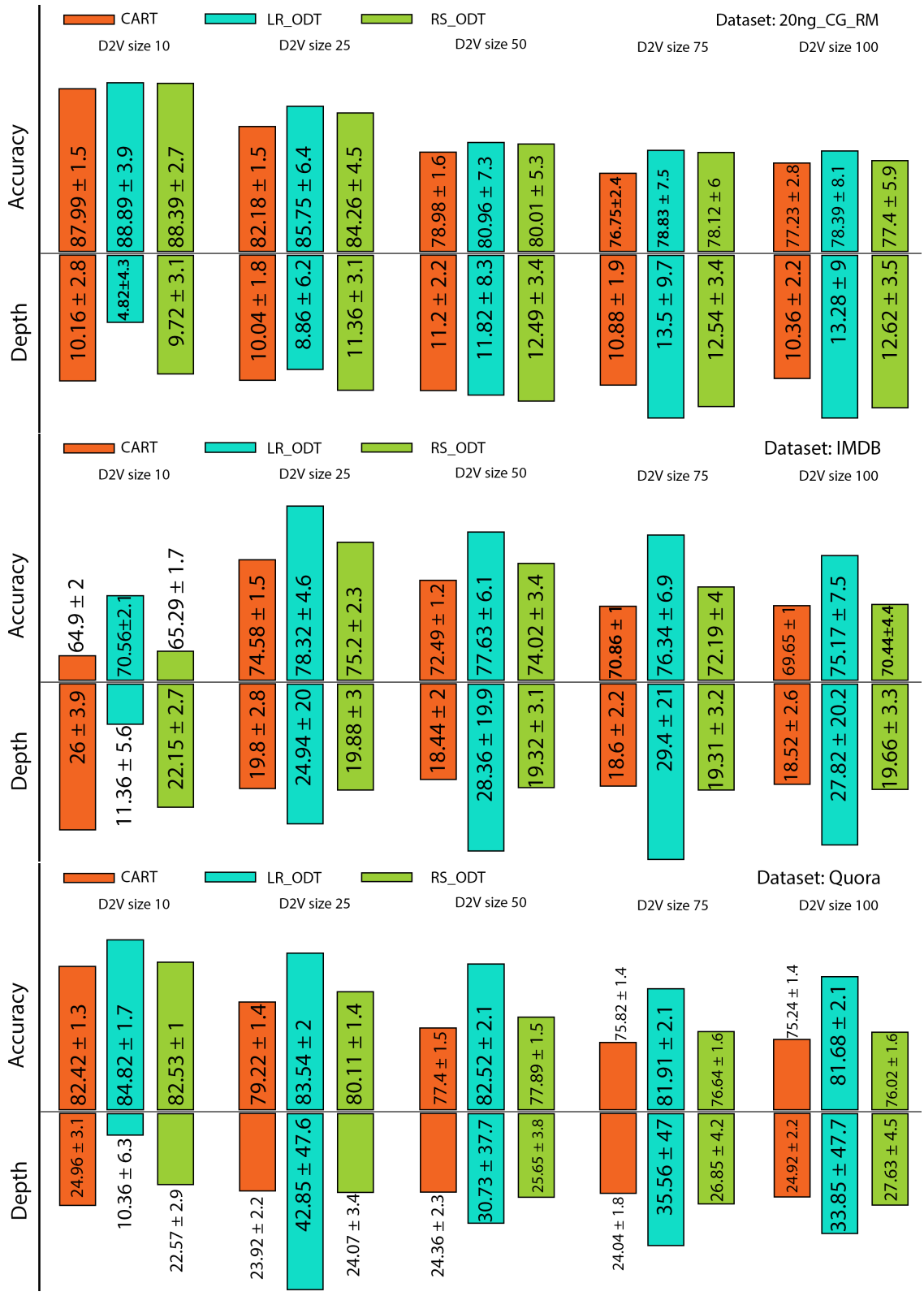


Figure 4.3: Overall average accuracy and average depth for each algorithms on each datasets' embeddings

data	d2v	CART					LR_ODT					RS_ODT				
		acc	pre	rec	f1	dep	acc	pre	rec	f1	dep	acc	pre	rec	f1	dep
20 ng cg rm	10	87.99 ± 1.5	87.59 ± 1.2	88.87 ± 2.6	88.2 ± 1.41	10.16 ± 2.8	88.89 ± 3.9	87.72 ± 3.7	90.87 ± 5.4	89.18 ± 3.91	4.82 ± 4.3	88.39 ± 2.7	87.56 ± 3	89.88 ± 3.8	88.65 ± 2.7	9.72 ± 3.1
	25	82.18 ± 1.5	82.22 ± 3.1	82.75 ± 3.5	82.39 ± .78	10.04 ±1.8	85.75 ± 6.4	84.37 ± 5.9	88.24 ± 8.2	86.13 ± 6.37	8.86 ± 6.2	84.26 ± 4.5	83.14 ± 4.6	86.47 ± 5.6	84.7 ± 4.5	11.36 ± 3.1
	50	78.98 ± 1.6	79.1 ± 3.4	79.46 ± 3.9	79.17 ± 2.19	11.2 ± 2.2	80.96 ± 7.3	79.93 ± 6.8	83.38 ± 9.3	81.45 ± 7.39	11.82 ± 8.3	80.01 ± 5.3	79.14 ± 5.3	82.15 ± 6.7	80.52 ± 5.41	12.49 ± 3.4
	75	76.75 ± 2.4	76.19 ± 3.7	79.09 ± 3.6	77.48 ± 1.76	10.88 ± 1.9	78.83 ± 7.5	77.88 ± 7	81.33 ± 9.6	79.39 ± 7.56	13.5 ± 9.7	78.12 ± 6	77.36 ± 6	80.32 ± 7.3	78.7 ± 5.99	12.54 ± 3.4
	100	77.23 ± 2.8	76.9 ± 3.6	78.81 ± 3.9	77.75 ± 2.6	10.36 ± 2.2	78.39 ± 8.1	77.41 ± 7.4	80.93 ± 10.7	78.9 ± 8.3	13.28 ± 9	77.4 ± 5.9	76.73 ± 5.8	79.4 ± 7.8	77.92 ± 6.15	12.62 ± 3.5
IMDB	10	64.9 ± 2	65.35 ± 2.2	63.41 ± 2.1	64.36 ± 2.08	26 ± 3.9	70.56 ± 2.1	71.24 ± 2.6	69.15 ± 3.2	70.12 ± 2.25	11.36 ± 5.6	65.29 ± 1.7	65.37 ± 1.8	65.03 ± 2	65.2 ± 1.81	22.15 ± 2.7
	25	74.58 ± 1.5	75.22 ± 1.7	73.32 ± 1.5	74.25 ± 1.51	19.8 ± 2.8	78.32 ± 4.6	78.76 ± 5	77.79 ± 5.3	78.19 ± 4.72	24.94 ± 20	75.2 ± 2.3	75.54 ± 2.5	74.55 ± 2.2	75.04 ± 2.27	19.88 ± 3
	50	72.49 ± 1.2	72.92 ± 1.4	71.54 ± 1.6	72.21 ± 1.35	18.44 ± 2	77.63 ± 6.1	78.03 ± 6.6	77.17 ± 6.2	77.53 ± 6.11	28.36 ± 19.9	74.02 ± 3.4	74.28 ± 3.6	73.52 ± 3.4	73.89 ± 3.44	19.32 ± 3.1
	75	70.86 ± 1	71.43 ± 1.2	69.52 ± 1.5	70.45 ± 1.13	18.6 ± 2.2	76.34 ± 6.9	76.68 ± 7.4	75.98 ± 7	76.26 ± 6.98	29.4 ± 21	72.19 ± 4	72.36 ± 4.1	71.82 ± 4	72.09 ± 3.97	19.31 ± 3.2
	100	69.65 ± 1	70.04 ± 1.1	68.68 ± 1.9	69.34 ± 1.16	18.52 ± 2.6	75.17 ± 7.5	75.6 ± 7.9	74.63 ± 7.8	75.03 ± 7.58	27.82 ± 20.2	70.44 ± 4.4	70.55 ± 4.5	70.2 ± 4.3	70.37 ± 4.36	19.66 ± 3.3
QUORA	10	82.42 ± 1.3	57.51 ± 4.3	48.12 ± 0.8	52.32 ± 1.95	24.96 ± 3.1	84.82 ± 1.7	68.04 ± 12	41.81 ± 13.8	50.6 ± 15.13	10.36 ± 6.3	82.53 ± 1	57.85 ± 3.6	47.85 ± 1.6	52.31 ± 1.7	22.57 ± 2.9
	25	79.22 ± 1.4	48.12 ± 3.8	40.9 ± 1.8	44.08 ± 1.27	23.92 ± 2.2	83.54 ± 2	62.72 ± 15.5	35.03 ± 17.8	42.66 ± 19.63	42.85 ± 47.6	80.11 ± 1.4	50.6 ± 4	41.85 ± 3.6	45.68 ± 3.05	24.07 ± 3.4
	50	77.4 ± 1.5	43.37 ± 3.3	40.03 ± 1.8	41.5 ± 1.17	24.36 ± 2.3	82.52 ± 2.1	57.43 ± 19.1	29.13 ± 19.2	35.73 ± 21.53	30.73 ± 37.7	77.89 ± 1.5	44.14 ± 3.9	37.78 ± 4.2	40.54 ± 3.33	25.65 ± 3.8
	75	75.82 ± 1.4	39.39 ± 2.4	37.71 ± 2.5	38.41 ± 1.1	24.04 ± 1.8	81.91 ± 2.1	52.03 ± 22	25.97 ± 19.7	31.74 ± 22.13	35.56 ± 47	76.64 ± 1.6	40.82 ± 3.6	36.19 ± 4.3	38.19 ± 3.18	26.85 ± 4.2
	100	75.24 ± 1.4	38.61 ± 2.4	39.37 ± 1.9	38.89 ± 1.03	24.92 ± 2.2	81.68 ± 2.1	50.67 ± 23	24.72 ± 20.2	29.98 ± 22.63	33.85 ± 47.7	76.02 ± 1.6	39.41 ± 3.5	36.06 ± 4.3	37.49 ± 3.18	27.63 ± 4.5

Table 4.5: Overall average test performance

Overall average test performance, findings from Figure 4.3 and Table 4.4

1. LR_ODT has high performance on all dataset yet there is drastic different difference in depth. For smallest vector size 10 LR_ODT proven best choice where as for other vector sizes we compromise with depth.

2. Performance of RS_ODT has slightly better than CART for both accuracy and depth while still accuracy lack behind LR_ODT.

3. If we look to standard deviation we can find that CART has pretty low compare to LR_ODT and slight high to RS_ODT for both accuracy and depth. It means both ODT algorithms can perform better.

4. We can see that depth of LR_ODT has scatter in vast range some standard deviate of depth are greater than mean depth.

5. We can see that the f1 of RS_ODT is grate to other algorithm on QUORA dataset while for other datasets LR_ODT has high f1.

6. Except QUORA dataset pre, rec and f1 are almost equal or slightly above to accuracy. Either all algorithms have predict almost equal number of false positive and false negative or algorithm are biased to some one class. It can also be possible since both dataset has around 50% of equal class distribution.

4.4.4 Min max comparison

Previously we analyze average overall performance, in this section we see what are the minimum and maximum performance on each vector size of datasets. This comparison is based on the minimum and maximum test accuracy. In this evaluation, we will also include max depth, branch size (left and right) and training time plus hyperparameters settings. Finding the best hyperparameters setting complex task often requires heavy amount task but still we can see if any improvement we can get or not. Finally, in node sizes, we can assume tree structure like a balanced tree or left-right skewed tree. At last training, time may not give true information since CART and LR are optimized code from sklearn yet we show how long it took for each training process.

d2v	algo	max min	feat size	epo	min leaf	acc	pre	trec	f1	dep	node size	time (s)
10	CART	max	0	0	5	89.85	87.82	91.53	89.64	13	0,0	0.04
		min	0	0	30	83.76	87.32	82.49	84.83	7	0,0	0.02
	LR_ ODT	max	5	1k	10	93.15	91.75	94.18	92.95	5	5,8	0.35
		min	2	500	15	70.05	63.67	93.4	75.72	3	0,6	0.16
	RS_ ODT	max	20	100	15	93.65	91.84	95.24	93.51	16	19,1	2.14
		min	2	500	30	76.9	80.11	71.57	75.6	6	12,6	7.88
25	CART	max	0	0	5	84.52	83.01	86.8	84.86	13	0,0	0.07
		min	0	0	15	79.19	77.72	79.37	78.53	10	0,0	0.07
	LR_ ODT	max	20	100	5	94.16	92.35	95.77	94.03	6	7,5	3.25
		min	2	1k	15	65.48	63.2	74.11	68.22	12	48,0	0.5
	RS_ ODT	max	20	800	30	92.64	92.11	92.59	92.35	8	2,8	9.96
		min	2	100	20	67.26	69.1	62.44	65.6	17	0,50	1.89
50	CART	max	0	0	30	81.93	83.17	82.78	82.97	8	0,0	0.12
		min	0	0	10	75.89	71.92	79.35	75.45	11	0,0	0.13
	LR_ ODT	max	50	100	5	93.91	91.88	95.77	93.78	2	2,1	2.64
		min	2	800	10	60.81	60	78.95	68.18	6	0,11	0.18
	RS_ ODT	max	50	1k	15	90.36	89.12	91.01	90.05	15	5,18	18.37
		min	2	300	15	58.88	58.54	60.91	59.7	11	47,11	11.7
75	CART	max	0	0	20	80.96	78.5	83.07	80.72	11	0,0	0.16
		min	0	0	10	72.59	71.5	75.13	73.27	11	0,0	0.13
	LR_ ODT	max	75	100	5	91.37	87.08	96.3	91.46	2	2,2	5.64
		min	2	1k	30	60.05	64.77	54.55	59.22	10	17,14	0.4
	RS_ ODT	max	10	100	30	89.34	87.31	91.01	89.12	8	4,9	2.09
		min	2	800	15	59.64	57.23	53.8	55.46	14	23,40	31.28
100	CART	max	0	0	15	81.22	84.02	75.13	79.33	11	0,0	0.16
		min	0	0	5	70.81	70.5	71.57	71.03	13	0,0	0.2
	LR_ ODT	max	100	100	5	91.62	87.94	95.11	91.38	3	3,1	8.27
		min	2	100	15	58.38	55.81	63.49	59.41	9	13,11	0.37
	RS_ ODT	max	100	500	5	89.59	90	91.24	90.62	16	32,28	23.31
		min	2	800	5	60.15	60.1	60.41	60.25	18	21,92	26.49

Table 4.6: 20ng_CG_RM maximun and minimum test accuracy and its parameters setting by all algorithm for each vector sizes

d2v	algo	max min	feat size	epo	min leaf	acc	pre	rec	f1	dep	node size	time (s)
10	CART	max	0	0	30	69.17	69.68	68.48	69.07	25	0,0	1.02
		min	0	0	5	61.22	60.29	59.92	60.1	31	0,0	1.29
	LR_ ODT	max	10	100	5	72.86	74.3	70.37	72.28	7	2,36	22.3
		min	2	500	10	61.92	68.61	43.33	53.11	11	41,0	1.86
	RS_ ODT	max	10	100	30	69.54	69.54	70.15	69.84	18	279, 391	128.39
		min	20	500	5	61.03	59.87	60.86	60.36	28	1879, 1264	761.03
25	CART	max	0	0	30	77.33	77.64	75.14	76.37	16	0,0	2.14
		min	0	0	5	72.22	72.56	71.96	72.26	23	0,0	2.6
	LR_ ODT	max	25	100	5	83.9	84.32	82.28	83.28	13	14,29	51.66
		min	2	300	15	64.1	73.82	43.21	54.51	13	62,0	2.16
	RS_ ODT	max	20	500	30	79.43	78.38	79.82	79.09	16	194, 284	404.19
		min	2	100	5	68.76	68.67	68.5	68.58	23	803, 2027	50.86
50	CART	max	0	0	30	74.4	74.63	74.36	74.5	16	0,0	4.32
		min	0	0	5	70.27	72.1	68.25	70.12	21	0,0	3.66
	LR_ ODT	max	50	100	30	85.51	86.88	83.7	85.26	9	11,13	45.23
		min	2	800	15	61.58	60.83	69.78	65	15	0,129	5.06
	RS_ ODT	max	50	500	30	79.45	79.63	79.46	79.54	18	205, 301	536.42
		min	2	1000	10	65.49	65.43	65.03	65.23	21	580, 1253	413.99
75	CART	max	0	0	30	73	73.26	72.56	72.91	17	0,0	4.7
		min	0	0	10	69.29	69.79	67.54	68.65	21	0,0	5.32
	LR_ ODT	max	75	100	5	86.21	87.81	84.14	85.94	8	13,18	49.34
		min	2	300	30	61.38	58.83	77.23	66.79	16	0,98	6.43
	RS_ ODT	max	75	500	30	78.6	79.15	77.98	78.56	16	280, 244	686.25
		min	2	500	5	62.85	64	62.46	63.22	23	2109, 1057	693.4
100	CART	max	0	0	30	71.87	73.04	71.28	72.15	16	0,0	6.19
		min	0	0	5	68.28	68.49	67.88	68.18	20	0,0	7.57
	LR_ ODT	max	100	100	30	86.28	88.12	84.05	86.03	5	6,9	241.52
		min	2	100	15	59.26	57.36	70.73	63.35	14	12,59	1.44
	RS_ ODT	max	100	800	30	78.27	78.51	78.18	78.35	19	137, 390	1035.99
		min	2	500	10	61.49	62.78	60.58	61.66	21	940, 1127	503.25

Table 4.7: IMDB maximun and minimum test accuracy performance of each algorithms for all d2v sizes

d2v	algo	max min	feat size	epo	min leaf	acc	pre	rec	f1	dep	node size	time (s)
10	CART	max	0	0	30	84.29	63.32	50.58	56.24	19	0,0	2.34
		min	0	0	5	80.07	49.92	47.75	48.81	27	0,0	1.89
	LR_ ODT	max	10	100	10	86	71.79	48.81	58.11	6	3,25	13.26
		min	2	100	10	80.76	56.83	11.25	18.78	5	1,10	2.781
	RS_ ODT	max	5	300	30	84.59	65.47	47.75	55.22	19	473, 257	228.17
		min	2	1k	5	80.13	50.92	46.52	48.62	28	1117, 2508	1161.85
25	CART	max	0	0	30	81.21	53.46	38.29	44.62	21	0,0	6.13
		min	0	0	5	76.17	41.06	41.13	41.1	26	0,0	7.47
	LR_ ODT	max	25	100	30	85.92	71.54	48.58	57.87	7	6,19	41.13
		min	2	100	10	79.81	51.11	2.86	5.43	4	3,4	1.86
	RS_ ODT	max	10	800	30	82.5	58.93	43.75	50.22	20	318, 473	864.26
		min	2	500	5	76.15	40.32	37.47	38.85	31	3363, 985	632.06
50	CART	max	0	0	30	79.86	50.1	38.72	43.68	20	0,0	11.9
		min	0	0	5	74.36	36.97	40.45	38.63	27	0,0	13.76
	LR_ ODT	max	50	100	30	86.13	71.01	51.17	59.48	8	7,24	76.28
		min	2	100	20	79.82	42.85	0	0.01	0	0,0	0.93
	RS_ ODT	max	10	100	30	80.77	52.47	35.51	42.36	19	108, 746	123.42
		min	2	500	5	73.38	34.5	35.25	34.87	31	3447, 1094	657.41
75	CART	max	0	0	30	78.23	44.95	35.32	39.56	22	0,0	18.52
		min	0	0	5	73.27	35.41	39.51	37.35	26	0,0	20.62
	LR_ ODT	max	75	100	15	86.07	70.16	52.17	59.84	12	7,36	187.84
		min	2	300	10	79.75	41.18	0.35	0.69	1	1,0	1.35
	RS_ ODT	max	20	500	30	79.64	49.42	38.82	43.48	28	245, 737	551.33
		min	2	1k	5	71.62	31.63	35.05	33.25	31	813, 3948	1400.53
100	CART	max	0	0	30	78.08	44.24	39.18	41.56	22	0,0	17.58
		min	0	0	5	73.02	34.86	40.58	37.5	26	0,0	27.38
	LR_ ODT	max	100	100	5	86.23	70.64	52.73	60.38	9	12,36	157.2
		min	5	500	20	79.77	48.39	1.11	2.18	2	0,2	4.09
	RS_ ODT	max	20	300	30	79.07	46.87	36.62	41.11	19	92,867	489.86
		min	2	1k	5	71.05	29.87	33.78	31.7	36	3805, 906	1393.23

Table 4.8: QUORA maximum and minimum test accuracy performance of each algorithms for all d2v sizes

Min Max evaluation, findings from Table 4.5, 4.6 and 4.7

1. For all dataset ODTs approach has higher accuracy than CART where LR_ODT obtained best for accuracy. In opposite both ODTs can also lower than CART depending on hyperparameters setting.

2. Considering feature size (feat size) shows interesting light on our experiment. We assumed that representation of D2V model is in a somewhat linear form of higher dimension as proof most of the maximum performance is achieved when we took all the features. Most of the benefit has been taken by LR_ODT Where RS_ODT comes second. Likewise, the minimum accuracy obtained by taking the lowest feature combination.

3. Epochs alone doesn't show any useful information for both ODT. It seems like epochs tightly bound by other parameters which are equal or more important.

4. For small dataset 20ng_CG_RM CART has high performance on least min leaf point where other larger datasets have high performance on maximum min leaf point, therefore, it looks like it is behaving as pruning. In the same way, LR_ODT is also performing whereas RS_ODT has exactly opposite symptoms except on 20ng_CG_RM.

5. CART and RS_ODT have high accuracy when depth is high for 20ng_CG_RM while in other two datasets high accuracy has gained when depth is low. In contrast, LR_ODT got maximum accuracy in low depth and vice versa on both IMDB and 20ng_CG_RM whereas in QUORA accuracy and depth has equal relation for increment and decrement.

6. Structure of tree doesn't provide much information alone yet most of the time high accuracy has gained when the tree is almost balance in ODTs. The same effect also applied for training time. As explain above RS_ODT took the longest time.

4.4.5 Hyperparameters based comparison

In this section, we will evaluate algorithmic performance based on their hyperparameters. We analyze our result by accuracy, F1 score and max depth as well as the standard deviation for each.

Epochs	d2v	LR_ODT			RS_ODT		
		accuracy	f1	depth	accuracy	f1	depth
100	10	88.74 ± 4.1	88.97 ± 4.46	4.98 ± 4.1	88.17 ± 2.7	88.44 ± 2.68	10.14 ± 2.7
	25	85.62 ± 6.1	85.98 ± 6.42	8.3 ± 5.6	83.56 ± 4.5	83.97 ± 4.61	11.8 ± 3.1
	50	80.87 ± 7.2	81.45 ± 7.32	11.9 ± 7.8	79.36 ± 5.2	79.94 ± 5.54	13.1 ± 3.1
	75	78.8 ± 7.5	79.33 ± 7.49	13.31 ± 10.5	76.95 ± 5.8	77.53 ± 5.91	13.5 ± 3.6
	100	78.51 ± 8.4	79.12 ± 8.06	12.95 ± 10.5	76.88 ± 5.9	77.46 ± 6.48	13.13 ± 3.9
300	10	88.92 ± 3.6	89.19 ± 3.69	5.15 ± 4.3	88.24 ± 2.8	88.49 ± 2.77	9.91 ± 3.2
	25	85.84 ± 6.3	86.22 ± 6.68	9.08 ± 6.5	84.09 ± 4.2	84.54 ± 4.2	11.12 ± 3
	50	81.17 ± 7.5	81.65 ± 7.83	11.58 ± 9	79.72 ± 5.3	80.29 ± 5.21	12.69 ± 3.5
	75	78.62 ± 7.7	79.26 ± 7.83	13.28 ± 8.4	78.35 ± 6.1	79.04 ± 6.19	12.75 ± 3.4
	100	78.55 ± 8.1	79.01 ± 8.28	13.6 ± 8.4	76.92 ± 5.7	77.62 ± 6.01	12.78 ± 3.4
500	10	88.79 ± 4.1	89.17 ± 3.73	4.68 ± 4.3	88.33 ± 2.7	88.59 ± 2.72	9.62 ± 2.9
	25	85.95 ± 6.5	86.27 ± 6.16	8.89 ± 6.9	84.48 ± 4.5	84.98 ± 4.37	11.26 ± 3.3
	50	80.98 ± 7.3	81.48 ± 7.29	11.3 ± 8.9	80.07 ± 5.4	80.57 ± 5.23	12.1 ± 3.3
	75	78.98 ± 7.5	79.53 ± 7.5	13.01 ± 10.1	78.34 ± 6	78.92 ± 6	12.22 ± 3.1
	100	78.4 ± 8	79.05 ± 8.25	14.04 ± 8.9	77.55 ± 5.7	78.07 ± 6.13	12.39 ± 3.4
800	10	88.87 ± 3.9	89.13 ± 3.94	4.45 ± 4.8	88.48 ± 2.8	88.73 ± 2.41	9.38 ± 3.2
	25	85.55 ± 6.4	85.98 ± 6.33	9.22 ± 6.1	84.5 ± 4.9	84.94 ± 4.84	11.11 ± 3
	50	80.8 ± 7.2	81.14 ± 7.2	12.47 ± 6.9	80.41 ± 5.1	80.87 ± 5.65	12.3 ± 3.3
	75	78.82 ± 7.4	79.31 ± 7.57	14.51 ± 10.1	78.13 ± 6.1	78.7 ± 6.18	12.3 ± 3.5
	100	78.16 ± 8.2	78.55 ± 7.98	12.18 ± 9.1	77.6 ± 6.2	78.01 ± 5.88	12.58 ± 3.3
1000	10	89.12 ± 3.6	89.42 ± 3.65	4.82 ± 4	88.73 ± 2.5	89.03 ± 2.88	9.56 ± 3.2
	25	85.76 ± 6.6	86.2 ± 6.27	8.8 ± 5.8	84.66 ± 4.2	85.09 ± 4.34	11.53 ± 3.2
	50	80.95 ± 7.4	81.51 ± 7.29	11.86 ± 8.6	80.47 ± 5.5	80.95 ± 5.35	12.24 ± 3.6
	75	78.96 ± 7.4	79.49 ± 7.43	13.39 ± 9.2	78.81 ± 5.6	79.32 ± 5.48	11.92 ± 3.1
	100	78.33 ± 7.7	78.77 ± 8.87	13.6 ± 7.8	78.05 ± 6.2	78.45 ± 6.17	12.25 ± 3.4

Table 4.9: 20ng epochs wise performance

k fold	d2v	CART			LR_ODT			RS_ODT		
		accuracy	f1	max depth	accuracy	f1	max depth	accuracy	f1	max depth
1	***10***	88.32 ± 0.4	87.7 ± 0.33	11.6 ± 3.4	89.34 ± 4	88.69 ± 4.72	4.83 ± 2.1	88.73 ± 2.5	88.15 ± 2.78	10.01 ± 2.9
	25	82.49 ± 1	81.36 ± 1.01	9.8 ± 1.7	86.36 ± 6.7	85.99 ± 6.3	9.6 ± 4.3	84.78 ± 4.3	84.13 ± 4.52	11.61 ± 3.1
	50	77.97 ± 1.1	77.16 ± 1.01	11.2 ± 2	81.38 ± 7.6	80.8 ± 7.78	12.13 ± 8.1	80.19 ± 5.3	79.31 ± 5.78	12.02 ± 3.2
	75	76.6 ± 1.3	76.85 ± 0.82	12 ± 2.4	79.07 ± 7.2	78.38 ± 7.28	12.05 ± 10.8	78.28 ± 6	77.53 ± 6.59	12.7 ± 3.5
	100	79.34 ± 0.8	78.74 ± 1.3	10.6 ± 2.4	78.51 ± 8	77.92 ± 8.29	13.82 ± 9.3	77.45 ± 5.8	76.76 ± 5.67	12.57 ± 2.8
2	***10***	85.94 ± 0.5	87 ± 1.41	10.8 ± 2.9	87.44 ± 3.7	88.59 ± 3.46	6.88 ± 6.9	87.86 ± 2.7	88.99 ± 2.54	10.23 ± 3.1
	25	81.68 ± 0.8	82.97 ± 0.81	9.6 ± 1.9	84.93 ± 5.9	86.25 ± 6.55	8.49 ± 6	83.67 ± 3.9	85.28 ± 4.27	10.7 ± 2.9
	50	78.38 ± 1	79.64 ± 0.33	12 ± 1.8	80.48 ± 7.3	82.33 ± 7.08	11.09 ± 7.2	79.45 ± 4.7	81.44 ± 4.43	12.75 ± 2.9
	75	74.62 ± 0.6	76.94 ± 1	10.4 ± 1.9	78.63 ± 7.4	80.45 ± 7.12	12.05 ± 7.7	77.41 ± 6.2	79.59 ± 5.04	11.9 ± 3.1
	100	76.95 ± 1.9	78.97 ± 2.04	9.6 ± 2.2	78.32 ± 8.2	80.1 ± 8.31	12.98 ± 9	77.14 ± 5.5	79.26 ± 6.32	12.17 ± 4
3	***10***	87.21 ± 0.3	87.25 ± 0.53	8.2 ± 2.4	88.19 ± 3.9	88.64 ± 3.85	2.91 ± 3	87.48 ± 2.6	87.82 ± 2.61	9.43 ± 3
	25	83.96 ± 0.5	84.6 ± 1.47	9.8 ± 1.3	84.79 ± 6.3	85.24 ± 6.42	6.72 ± 6.1	82.86 ± 4.9	83.33 ± 3.7	11.26 ± 3.2
	50	78.68 ± 0.7	78.17 ± 1.38	12.4 ± 2.4	79.46 ± 7.2	79.8 ± 6.9	11.09 ± 8	78.54 ± 5.3	78.98 ± 5.72	13.06 ± 3.6
	75	74.97 ± 1.9	75.93 ± 1.84	10.2 ± 1.7	77.08 ± 7.1	77.75 ± 7.22	14.23 ± 8	76.51 ± 6	77 ± 5.83	12.45 ± 3.2
	100	72.84 ± 1.4	73.79 ± 1.92	10.4 ± 1.9	76.03 ± 8.3	76.51 ± 7.71	13.42 ± 8.7	75.43 ± 5.7	75.83 ± 5.95	12.82 ± 3.4
4	***10***	89.39 ± 1.5	89.11 ± 0.36	9.8 ± 1.9	90.29 ± 3.6	90.03 ± 3.58	4.81 ± 4.7	89.78 ± 2.5	89.49 ± 2.62	9.38 ± 3.1
	25	80.61 ± 1	80.38 ± 1.24	10.2 ± 2.1	86.55 ± 6.3	86.29 ± 5.93	8.46 ± 6.2	85.55 ± 4.3	85.3 ± 4.52	11.43 ± 2.9
	50	78.78 ± 1.7	78.11 ± 0.92	10 ± 1.9	82.84 ± 7.4	82.54 ± 7.57	10.48 ± 6.5	81.67 ± 5.2	81.42 ± 5.35	11.97 ± 3.4
	75	80.41 ± 0.4	79.85 ± 0.61	10.8 ± 1.5	80.56 ± 7.6	80.42 ± 8.13	12.39 ± 7.2	80.28 ± 5.7	79.96 ± 5.94	12.17 ± 3.5
	100	79.04 ± 1.5	78.12 ± 1.3	10.6 ± 2.1	80.36 ± 7.6	80.1 ± 7.84	11.74 ± 8	79.47 ± 6	79.16 ± 6.36	11.9 ± 3.4
5	***10***	89.11 ± 0.9	89.95 ± 0.98	10.4 ± 1.7	89.19 ± 3.6	89.92 ± 3.52	4.66 ± 1.6	88.11 ± 2.7	88.81 ± 2.63	9.55 ± 3.1
	25	82.19 ± 1.4	82.64 ± 0.31	10.8 ± 1.4	86.1 ± 6.4	86.87 ± 6.54	11.01 ± 7.2	84.45 ± 4.5	85.47 ± 4.99	11.82 ± 3.4
	50	81.07 ± 0.8	82.79 ± 0.78	10.4 ± 2.1	80.61 ± 6.8	81.76 ± 7.24	14.34 ± 10.5	80.18 ± 5.6	81.47 ± 5.06	12.63 ± 3.6
	75	77.15 ± 1.2	77.82 ± 1.13	11 ± 1.4	78.82 ± 7.7	79.93 ± 7.62	16.78 ± 12.7	78.1 ± 5.2	79.44 ± 5.84	13.47 ± 3.5
	100	77.96 ± 1.5	79.14 ± 1.54	10.6 ± 2.2	78.73 ± 7.7	79.86 ± 8.65	14.42 ± 9.7	77.53 ± 6	78.6 ± 5.6	13.66 ± 3.6

Table 4.10: 20ng_CG_RM K_fold wise average test performance

min leaf	d2v	CART			LR_ODT			RS_ODT		
		accuracy	f1	max depth	accuracy	f1	max depth	accuracy	f1	max depth
5	***10***	88.27 ± 0.7	88.48 ± 0.86	14 ± 2	88.92 ± 4	89.19 ± 4.19	5.1 ± 4.2	88.55 ± 2.5	88.73 ± 2.19	12.89 ± 2.3
	25	82.53 ± 1.4	82.64 ± 1.56	12.8 ± 0.4	86.12 ± 6.1	86.55 ± 6.23	8.58 ± 5.5	84.48 ± 4	84.9 ± 3.96	14.38 ± 2.6
	50	79.38 ± 1.1	79.73 ± 2.11	14.6 ± 1.2	81.75 ± 7.1	82.3 ± 7.21	12.02 ± 7.9	80.28 ± 5.2	80.72 ± 5.44	15.53 ± 2.6
	75	76.43 ± 2.1	77.02 ± 1.26	13.4 ± 1.4	79.4 ± 7.2	80.17 ± 7.49	13.65 ± 9.6	78.15 ± 5.7	78.66 ± 5.7	15.67 ± 2.8
	100	76.13 ± 3	76.55 ± 2.88	13.6 ± 0.6	78.95 ± 8.1	79.63 ± 7.92	12.92 ± 9	77.15 ± 5.9	77.61 ± 6.05	16.12 ± 3.1
10	***10***	88.42 ± 1.7	88.65 ± 0.96	11.6 ± 1.9	89.02 ± 4	89.34 ± 3.96	4.97 ± 4.3	88.54 ± 2.8	88.76 ± 2.82	10.91 ± 1.8
	25	83.04 ± 1.2	83.24 ± 1.43	10.8 ± 0.4	86.05 ± 6.3	86.54 ± 5.82	8.54 ± 5.3	84.25 ± 4.2	84.67 ± 4.25	12.38 ± 2.4
	50	78.01 ± 1.6	78.52 ± 1.5	12 ± 1	81.21 ± 7.2	81.86 ± 6.99	12.09 ± 7.6	79.76 ± 5.3	80.33 ± 5.39	13.84 ± 2.9
	75	76.13 ± 2.1	76.77 ± 1.6	11.8 ± 0.7	79.14 ± 7.3	79.68 ± 7.5	13.02 ± 11	78.27 ± 5.6	78.9 ± 5.44	13.91 ± 2.5
	100	77.4 ± 2.9	77.59 ± 2.81	11.6 ± 0.8	78.97 ± 7.6	79.64 ± 8.19	13.38 ± 9.5	77.27 ± 6	77.84 ± 6.5	13.9 ± 2.9
15	***10***	87.76 ± 1.4	87.9 ± 1.48	9.6 ± 0.8	89 ± 3.8	89.36 ± 3.79	4.56 ± 4.5	88.54 ± 2.2	88.85 ± 3.22	9.75 ± 2.2
	25	81.82 ± 1.8	81.87 ± 2.09	9.8 ± 0.7	85.8 ± 6.2	86.2 ± 6.16	8.97 ± 6.4	84.39 ± 4.3	84.83 ± 4.39	10.9 ± 2.5
	50	79.28 ± 1.3	79.53 ± 2.12	10.8 ± 0.7	80.76 ± 7.4	81.23 ± 7.35	12.5 ± 8.9	79.94 ± 5.6	80.42 ± 5.62	12.54 ± 2.6
	75	77.2 ± 2.7	78.36 ± 1.74	11 ± 0.9	78.88 ± 7.5	79.46 ± 7.07	14.01 ± 9.1	77.74 ± 6.4	78.2 ± 6.74	12.38 ± 2.8
	100	78.36 ± 2.4	78.71 ± 1.89	10 ± 0.6	78.72 ± 8.2	79.31 ± 7.56	13.42 ± 8.7	77.86 ± 5.8	78.3 ± 6.08	12.35 ± 2.4
20	***10***	87.61 ± 1	87.9 ± 1.41	8.6 ± 0.8	88.82 ± 3.6	89.04 ± 3.46	4.62 ± 4.1	88.29 ± 2.7	88.57 ± 2.61	8.49 ± 2.2
	25	81.67 ± 1.4	81.79 ± 2.16	8.8 ± 0.7	85.65 ± 6.2	85.93 ± 6.44	9.04 ± 7.3	84.28 ± 4.8	84.74 ± 4.79	10.52 ± 2.2
	50	79.38 ± 1.1	79.37 ± 2.62	9.8 ± 1.3	80.63 ± 7.4	81.07 ± 7.49	11.88 ± 9	80.2 ± 4.9	80.7 ± 5.12	11.07 ± 2.6
	75	76.99 ± 2.5	77.55 ± 1.95	9.8 ± 0.7	78.98 ± 7.2	79.33 ± 7.4	12.99 ± 9.6	78.4 ± 5.9	79.06 ± 5.87	11.13 ± 2.5
	100	77.5 ± 2.9	78.2 ± 3.18	9 ± 0.5	77.96 ± 8.5	78.36 ± 9.18	12.98 ± 8.9	77.24 ± 6.2	77.73 ± 6.09	11.13 ± 2.7
30	***10***	87.91 ± 2.2	88.08 ± 1.91	7 ± 0.6	88.68 ± 4	88.95 ± 4.08	4.83 ± 4.5	88.04 ± 3.3	88.35 ± 2.54	6.57 ± 2.3
	25	81.87 ± 0.9	82.42 ± 0.97	8 ± 0.6	85.11 ± 7	85.41 ± 7.09	9.15 ± 6.3	83.9 ± 5.1	84.38 ± 4.99	8.64 ± 2.5
	50	78.82 ± 2	78.71 ± 2.19	8.8 ± 0.7	80.42 ± 7.6	80.78 ± 7.8	10.63 ± 7.7	79.85 ± 5.4	80.45 ± 5.46	9.46 ± 2.4
	75	76.99 ± 2.3	77.7 ± 1.74	8.4 ± 0.5	77.77 ± 8.1	78.3 ± 8.2	13.84 ± 9	78.03 ± 6.2	78.69 ± 6.07	9.61 ± 2.7
	100	76.74 ± 2	77.7 ± 1.16	7.6 ± 0.5	77.36 ± 7.8	77.57 ± 8.36	13.67 ± 8.8	77.5 ± 5.8	78.12 ± 5.97	9.62 ± 2.3

Table 4.11: 20ng min leaf point average test performance

n_feat	d2v	LR_ODT			RS_ODT		
		accuracy	f1	depth	accuracy	f1	depth
2	*10*	82.65 ± 2.1	82.8 ± 1.89	7.9 ± 5.3	84.6 ± 1.5	84.83 ± 1.43	10.86 ± 2.9
	25	75.78 ± 3.4	76.25 ± 2.5	8.86 ± 6.3	77.24 ± 2.5	77.69 ± 3.36	12.06 ± 3
	50	71.03 ± 3.8	71.53 ± 1.29	8.94 ± 0.4	72.27 ± 2.4	72.57 ± 2.08	13.1 ± 3.5
	75	69.05 ± 1.1	69.57 ± 2.85	9.82 ± 0.5	69.3 ± 3.5	69.86 ± 4.04	13.07 ± 3
	100	68.35 ± 3.7	68.64 ± 4.41	9.58 ± 3.6	68.97 ± 3	69.13 ± 3.21	12.91 ± 2.9
5	*10*	88.92 ± 0.9	89.21 ± 4.21	8.38 ± 4.9	88.65 ± 2.9	88.91 ± 2.89	9.22 ± 3.3
	25	82.72 ± 2.3	82.98 ± 1.76	13.83 ± 5.1	83.15 ± 2	83.6 ± 2.5	10.74 ± 2.5
	50	77.4 ± 3.2	77.74 ± 3.89	16.44 ± 5.1	77.8 ± 2.9	78.36 ± 2.91	12.21 ± 3.9
	75	74.99 ± 3.6	75.41 ± 1.05	17.61 ± 3.8	75.94 ± 3.1	76.48 ± 2.98	12.38 ± 3.2
	100	74.09 ± 3.7	74.48 ± 0.78	17.06 ± 7.5	74.69 ± 2.3	75.18 ± 2.14	12.06 ± 4.4
10	*10*	90.96 ± 0.9	91.29 ± 0.61	2.6 ± 1.2	89.51 ± 1.8	89.77 ± 1.78	9.48 ± 2.7
	25	87.01 ± 1.9	87.44 ± 0.79	13.17 ± 0.8	86.03 ± 2	86. ± 1.87	10.97 ± 3.2
	50	80.74 ± 2.9	81.26 ± 3.08	18.23 ± 8.8	81.24 ± 2.4	81.83 ± 2.5	11.87 ± 3.2
	75	78.59 ± 2.9	79.2 ± 3.1	22.19 ± 10.2	79.42 ± 2.8	80 ± 2.8	12.36 ± 3.7
	100	77.83 ± 3.2	78.36 ± 3.07	21.52 ± 8.8	78.37 ± 2.8	79.01 ± 2.8	12.7 ± 3.4
20	*10*	90.96 ± 4	91.29 ± 0.61	2.6 ± 1.2	89.68 ± 1.6	89.97 ± 1.43	9.57 ± 3.1
	25	90.77 ± 1.4	91.19 ± 1.28	7.02 ± 4.2	87.39 ± 2	87.78 ± 1.82	11.37 ± 3.2
	50	84.34 ± 2.2	84.97 ± 2.14	13.3 ± 3.7	83.48 ± 3.3	84.1 ± 3.72	12.37 ± 2.9
	75	81.6 ± 2.7	82.27 ± 4.2	15.28 ± 5.7	81.54 ± 2.7	82.19 ± 2.4	12 ± 3.2
	100	80.96 ± 3	81.76 ± 3.44	15.86 ± 6	80.74 ± 2.7	81.4 ± 2.51	12.46 ± 3.4
all	*10*	82.65 ± 2.1	82.8 ± 1.89	7.9 ± 5.3	84.6 ± 1.5	84.83 ± 1.43	10.86 ± 2.9
	25	92.43 ± 0.9	92.79 ± 3.5	1.4 ± 2.7	87.5 ± 3	87.94 ± 1.76	11.67 ± 3.4
	50	91.26 ± 1.7	91.75 ± 3.42	2.2 ± 7.8	85.24 ± 2.1	85.77 ± 1.91	12.89 ± 3.1
	75	89.94 ± 3	90.47 ± 2.49	2.6 ± 9.6	84.38 ± 2.3	84.99 ± 2.06	12.89 ± 3.7
	100	90.72 ± 1	91.27 ± 4.08	2.36 ± 0.6	84.24 ± 3.3	84.87 ± 3.68	12.99 ± 3.2

Table 4.12: 20ng n feature wise performance

Hyperparameters wise evaluation of 20ng_CG_RM dataset, Findings from Table (4.8, 4.9, 4.10, 4.11)

1. In Table 4.8 evaluation on epochs, for all epochs LR_ODT performs slightly better in terms of accuracy, f1 and depth by keeping low scatter around each mean.

2. In Table 4.9 evaluation based on 5 folds, Most of the times performance of RS_ODT has slightly higher than CART even though the number of training are 5 times higher whereas LR_ODT outperforms RS_ODT by keeping surprisingly low depth.

3. Similarly for minimum leaf point (min leaf), the performance of all model has the same effect in Table 4.10. There has been small increment when choosing least min point for ODT where selecting the largest value reduces the depth.

4. In Table 4.11 evaluation by the number of feature combination, RS_ODT has performed almost equal or slightly better performance except for depth in partial feature selection. Whereas when all features are selected LR_ODT outperforms in all given matrix.

Epochs	d2v	LR_ODT			RS_ODT		
		accuracy	f1	depth	accuracy	f1	depth
100	*10*	70.55 ± 2.1	70.07 ± 2.14	10.99 ± 5.3	65.66 ± 1.7	65.61 ± 1.83	21.14 ± 2.7
	25	78.27 ± 4.7	78.15 ± 4.79	26.09 ± 20.5	75.31 ± 2.3	75.17 ± 2.31	19.3 ± 2.9
	50	77.61 ± 6.1	77.57 ± 5.9	27.99 ± 19.5	73.74 ± 3.3	73.62 ± 3.38	19.42 ± 2.9
	75	76.3 ± 7	76.26 ± 6.96	30.25 ± 19.7	71.91 ± 4	71.81 ± 3.68	19.53 ± 3
	100	75.05 ± 7.6	74.99 ± 7.45	27.97 ± 21.7	69.94 ± 4.3	69.87 ± 4.53	19.99 ± 3.3
300	*10*	70.54 ± 2.1	70.17 ± 2.44	11.23 ± 5.6	65.36 ± 1.7	65.27 ± 1.8	21.78 ± 2.5
	25	78.22 ± 4.6	78.04 ± 5.13	23.6 ± 20.6	75.18 ± 2.3	75.01 ± 2.17	19.86 ± 3
	50	77.69 ± 6.4	77.6 ± 6.06	28.09 ± 19.6	74.05 ± 3.4	73.9 ± 3.42	19.38 ± 2.9
	75	76.25 ± 6.9	76.15 ± 7.09	29.13 ± 22.1	72.17 ± 3.7	72.07 ± 3.96	19.3 ± 2.9
	100	75.31 ± 7.3	75.26 ± 7.59	27.62 ± 19.7	70.42 ± 4.3	70.37 ± 4.14	19.74 ± 3.3
500	*10*	70.57 ± 2	70.02 ± 2.68	11.94 ± 4.9	65.22 ± 1.7	65.08 ± 1.81	22.06 ± 2.6
	25	78.24 ± 4.2	78.15 ± 4.61	24.39 ± 19.5	75.19 ± 2.4	75.06 ± 2.29	20.03 ± 2.8
	50	77.59 ± 5.9	77.59 ± 6	28.87 ± 20.5	74.13 ± 3.5	73.99 ± 3.59	19.02 ± 3.4
	75	76.42 ± 7	76.3 ± 7.04	29.49 ± 20.7	72.15 ± 3.9	72.07 ± 4.06	19.11 ± 3.2
	100	75.16 ± 7.4	75.02 ± 7.75	28.41 ± 18.6	70.53 ± 4.5	70.45 ± 4.24	19.42 ± 3.5
800	*10*	70.58 ± 2	70.28 ± 2.12	11.38 ± 6.5	65.16 ± 1.7	65.05 ± 1.82	22.74 ± 2.9
	25	78.53 ± 4.9	78.54 ± 3.97	24.74 ± 21.1	75.1 ± 2.2	74.93 ± 2.19	20.02 ± 3.1
	50	77.46 ± 6.1	77.29 ± 5.99	27.76 ± 18.4	74.11 ± 3.4	73.98 ± 3.3	19.22 ± 3.3
	75	76.35 ± 7	76.23 ± 7.03	28.93 ± 20.8	72.4 ± 4.1	72.3 ± 3.97	19.38 ± 3.2
	100	75.12 ± 7.5	74.9 ± 7.78	27.91 ± 20.9	70.6 ± 4.1	70.52 ± 4.51	19.7 ± 3.3
1000	*10*	70.56 ± 2.1	70.05 ± 1.76	11.23 ± 5.3	65.08 ± 1.6	64.97 ± 1.72	23.03 ± 2.5
	25	78.32 ± 4.6	78.06 ± 5	25.89 ± 18.2	75.2 ± 2.2	75.01 ± 2.36	20.22 ± 3
	50	77.78 ± 5.9	77.62 ± 6.55	29.08 ± 21.6	74.09 ± 3.6	73.97 ± 3.51	19.56 ± 3.1
	75	76.39 ± 6.8	76.37 ± 6.75	29.23 ± 21.4	72.31 ± 4.1	72.18 ± 4.14	19.22 ± 3.3
	100	75.22 ± 7.5	74.97 ± 7.29	27.22 ± 20.3	70.71 ± 4.5	70.62 ± 4.32	19.46 ± 3.3

Table 4.13: imdb ephocs wise performance

k fold	algo.	CART			LR_ODT			RS_ODT		
	d2v	accuracy	f1	max depth	accuracy	f1	max depth	accuracy	f1	max depth
1	***10***	66.37 ± 1.5	66 ± 1.97	29.2 ± 3.6	71.73 ± 1.8	71.38 ± 1.7	9.74 ± 7.2	66.6 ± 1.4	66.65 ± 1.43	21.73 ± 2.7
	25	74.2 ± 1.4	74.17 ± 1.34	19.2 ± 2.5	78.19 ± 4.1	78.12 ± 4.38	25.5 ± 22.1	74.88 ± 2.3	74.78 ± 2.14	19.75 ± 2.9
	50	73.25 ± 1.1	73.15 ± 0.77	18.2 ± 2.1	77.94 ± 6.3	77.96 ± 5.68	27.36 ± 22.5	74.22 ± 3.5	74.17 ± 3.5	19.13 ± 3.1
	75	71.46 ± 0.7	71.51 ± 0.84	17.8 ± 2.3	76.52 ± 6.9	76.72 ± 6.77	30.19 ± 25.3	72.42 ± 4	72.44 ± 3.97	19.5 ± 3.4
	100	68.85 ± 0.7	68 ± 0.95	19.8 ± 1.7	75.36 ± 7.5	75.25 ± 7.45	32.29 ± 24.2	70.72 ± 4.3	70.74 ± 4.47	19.66 ± 3.1
2	***10***	64.01 ± 1.8	62.88 ± 2.06	24 ± 3.6	70.35 ± 2.1	69.46 ± 2.8	14.16 ± 5.2	64.59 ± 1.5	63.82 ± 1.57	22.03 ± 2.8
	25	75.46 ± 1.4	74.46 ± 1.42	18.8 ± 1.9	79.65 ± 4.7	79.31 ± 5.08	23.94 ± 18.7	76.29 ± 2.1	75.66 ± 2.29	19.5 ± 2.8
	50	71.45 ± 1	70.66 ± 0.61	18.2 ± 1.9	77.68 ± 6.2	77.45 ± 6.56	30.81 ± 21.7	74.1 ± 3.5	73.51 ± 3.41	19.34 ± 3.1
	75	70.68 ± 0.7	69.5 ± 0.56	18 ± 1.9	76.26 ± 7.1	75.8 ± 7.12	26.55 ± 21.5	72.15 ± 4	71.59 ± 3.98	19.46 ± 3.3
	100	70.25 ± 0.9	69.72 ± 0.87	19 ± 3.1	75.47 ± 7.6	74.96 ± 7.31	25.05 ± 15.9	70.24 ± 4.5	69.75 ± 4.36	19.76 ± 3.4
3	***10***	63.97 ± 1.6	64.01 ± 1.75	26 ± 2.9	70.08 ± 2	70.12 ± 1.91	11.73 ± 5	64.57 ± 1.4	64.88 ± 1.45	22.46 ± 2.9
	25	75.28 ± 0.6	75.29 ± 1.57	19.6 ± 2.7	78.34 ± 4.9	78.38 ± 4.3	21.66 ± 20.3	75.65 ± 2.3	75.88 ± 2.08	19.9 ± 3
	50	72.1 ± 0.7	72.03 ± 1.28	18.6 ± 1.7	77.12 ± 5.9	77.12 ± 5.99	29.68 ± 19.6	73.77 ± 3.4	73.98 ± 3.45	19.42 ± 3.1
	75	70.3 ± 0.6	70.3 ± 0.44	18.2 ± 1.5	76.06 ± 6.7	76.27 ± 6.86	30.7 ± 14.9	71.9 ± 4	72.17 ± 3.75	19.31 ± 3.2
	100	69.61 ± 0.4	69.83 ± 0.81	17.8 ± 2.4	74.98 ± 7.3	75.17 ± 7.6	26.7 ± 18.6	70.33 ± 4.3	70.61 ± 4.24	19.5 ± 3.4
4	***10***	65.4 ± 1.9	64.99 ± 1.58	25.4 ± 3	70.83 ± 2	70.18 ± 1.97	10.88 ± 4.4	66.17 ± 1.4	66.12 ± 1.47	22.1 ± 2.7
	25	73.43 ± 1.5	73.11 ± 1.47	19.8 ± 3.3	77.27 ± 4.3	77.12 ± 5.35	27.08 ± 17.7	74.21 ± 2.1	74.05 ± 2.24	20.06 ± 2.9
	50	73.29 ± 1	73.13 ± 1.08	18 ± 1.4	77.39 ± 6	77.22 ± 5.81	27.75 ± 17.7	73.81 ± 3.3	73.7 ± 3.32	19.26 ± 3.2
	75	69.93 ± 0.5	69.5 ± 0.81	20.4 ± 2	76.22 ± 7.1	76.07 ± 6.79	35.56 ± 22.3	72.14 ± 3.8	72 ± 4.09	18.99 ± 2.8
	100	70.28 ± 1.2	69.95 ± 0.35	18.8 ± 3.2	74.96 ± 7.2	74.83 ± 7.82	25.38 ± 17.8	70.31 ± 4.3	70.18 ± 4.33	19.71 ± 3.3
5	***10***	64.75 ± 2	63.92 ± 1.56	25.4 ± 4.1	69.81 ± 1.9	69.46 ± 2.15	10.26 ± 4.3	64.54 ± 1.4	64.51 ± 1.46	22.42 ± 2.5
	25	74.53 ± 1.5	74.23 ± 0.76	21.6 ± 2.6	78.13 ± 4.7	78.02 ± 4.1	26.52 ± 20.7	74.94 ± 2.1	74.8 ± 2.09	20.21 ± 3.2
	50	72.34 ± 0.6	72.09 ± 1.07	19.2 ± 2.5	78.01 ± 6.1	77.92 ± 6.39	26.19 ± 17.2	74.21 ± 3.5	74.1 ± 3.5	19.47 ± 3.1
	75	71.91 ± 0.6	71.46 ± 0.77	18.6 ± 2.1	76.65 ± 7	76.46 ± 7.28	24.02 ± 17.1	72.32 ± 4	72.22 ± 3.99	19.26 ± 3
	100	69.24 ± 0.8	69.18 ± 1.32	17.2 ± 1.3	75.09 ± 7.6	74.94 ± 7.68	29.7 ± 22.6	70.59 ± 4.3	70.55 ± 4.32	19.7 ± 3.4

Table 4.14: imdb average test performance by k_folds

min leaf	d2v	CART			LR_ODT			RS_ODT		
		accuracy	f1	max depth	accuracy	f1	max depth	accuracy	f1	max depth
5	***10***	62.34 ± 0.7	61.92 ± 1.13	31.6 ± 1.4	70.67 ± 2.1	70.18 ± 2.03	12.46 ± 5.7	63.27 ± 1.1	63.13 ± 1.09	25.3 ± 2
	25	72.79 ± 1	72.47 ± 1.01	23.6 ± 1.4	78.3 ± 4.5	78.11 ± 5	26.72 ± 20.1	73.49 ± 1.8	73.33 ± 2.2	23.21 ± 2
	50	71.18 ± 0.8	70.88 ± 1.01	21.4 ± 0.5	77.45 ± 6.3	77.35 ± 6.33	30.74 ± 21	72.6 ± 3.1	72.45 ± 3.08	22.75 ± 2
	75	70.1 ± 0.7	69.86 ± 0.89	21.8 ± 1.5	76.2 ± 7	76.01 ± 7.34	30.5 ± 21.2	70.91 ± 3.8	70.77 ± 3.75	22.55 ± 2.3
	100	68.73 ± 0.4	68.48 ± 1.17	22.4 ± 0.5	75.26 ± 7.6	75.01 ± 7.76	28.38 ± 20.9	69.14 ± 4.4	69.06 ± 4.31	22.97 ± 2.8
10	***10***	63.69 ± 0.8	62.95 ± 1.07	27.4 ± 1.9	70.56 ± 2.1	70.09 ± 2.53	11.18 ± 5.3	64.51 ± 1	64.38 ± 1.23	23.44 ± 1.7
	25	73.57 ± 1.1	73.12 ± 0.83	21.6 ± 1.5	78.37 ± 4.8	78.18 ± 4.68	25.26 ± 18.7	74.71 ± 1.9	74.54 ± 2.05	21.32 ± 1.8
	50	71.95 ± 0.8	71.51 ± 1.13	19.4 ± 1	77.66 ± 6.1	77.51 ± 6.25	28.88 ± 20	73.65 ± 3.2	73.49 ± 3.24	20.49 ± 2.1
	75	70.56 ± 0.8	70 ± 0.92	19.4 ± 0.5	76.31 ± 6.9	76.21 ± 6.77	29.32 ± 20.4	71.75 ± 4	71.63 ± 3.79	20.66 ± 2.2
	100	68.99 ± 0.6	68.51 ± 0.95	19.6 ± 0.6	75.01 ± 7.3	74.91 ± 7.47	28.53 ± 18.9	70.07 ± 4.4	69.99 ± 4.43	21.01 ± 2.8
15	***10***	65.03 ± 1.1	64.53 ± 1.15	25.4 ± 1.9	70.54 ± 2.1	70.17 ± 2.18	11.06 ± 5.6	65.46 ± 1.1	65.37 ± 1.27	21.88 ± 1.8
	25	74.84 ± 0.8	74.55 ± 0.87	19.4 ± 1.1	78.31 ± 4.5	78.14 ± 4.88	24.58 ± 20.8	75.4 ± 2.2	75.24 ± 2.16	19.76 ± 2
	50	72.63 ± 0.6	72.36 ± 0.82	18 ± 0.6	77.62 ± 5.9	77.67 ± 6.08	27.05 ± 19.8	74.29 ± 3.5	74.16 ± 3.33	19.11 ± 2.5
	75	70.79 ± 1	70.29 ± 1.11	18.2 ± 1	76.46 ± 6.8	76.43 ± 7.11	30.72 ± 22.6	72.41 ± 4.1	72.31 ± 3.96	19.1 ± 2.1
	100	69.53 ± 0.9	69.23 ± 0.42	18 ± 2.1	75.16 ± 7.7	75.13 ± 7.78	26.63 ± 22.2	70.54 ± 4.2	70.46 ± 4.23	19.55 ± 2.4
20	***10***	66.19 ± 0.9	65.54 ± 1.27	23.8 ± 2.5	70.55 ± 2.2	70.13 ± 2.09	10.98 ± 5.5	66.11 ± 1.1	66.03 ± 1.21	20.82 ± 1.6
	25	75.49 ± 0.8	75.14 ± 0.35	18 ± 1	78.37 ± 4.7	78.44 ± 4.82	24.68 ± 21	75.94 ± 2.1	75.79 ± 1.76	18.54 ± 1.7
	50	72.99 ± 0.7	72.79 ± 0.93	17.4 ± 1	77.74 ± 6.2	77.54 ± 5.87	26.97 ± 18	74.55 ± 3.3	74.47 ± 3.45	17.98 ± 2.3
	75	71.2 ± 0.7	70.72 ± 1.08	17.2 ± 1	76.23 ± 7.2	76.19 ± 6.73	28.7 ± 21.1	72.71 ± 3.7	72.62 ± 3.8	17.82 ± 2.4
	100	70.08 ± 0.8	69.8 ± 0.42	17 ± 1.5	75.12 ± 7.5	74.93 ± 7.36	29.54 ± 19.8	71.01 ± 4.3	70.93 ± 4.37	18.3 ± 2.1
30	***10***	67.25 ± 1.1	66.87 ± 0.83	21.8 ± 1.9	70.49 ± 1.9	70.03 ± 2.39	11.09 ± 5.6	67.11 ± 1	67.07 ± 1.21	19.31 ± 1.8
	25	76.21 ± 0.5	75.98 ± 0.73	16.4 ± 0.8	78.22 ± 4.6	78.08 ± 4.17	23.47 ± 19.4	76.43 ± 2.2	76.26 ± 1.91	16.58 ± 2.1
	50	73.68 ± 0.9	73.52 ± 0.96	16 ± 0.6	77.66 ± 6	77.59 ± 5.99	28.15 ± 20.5	75.02 ± 3.6	74.9 ± 3.55	16.27 ± 2.2
	75	71.64 ± 0.8	71.4 ± 0.92	16.4 ± 1.3	76.51 ± 6.8	76.47 ± 6.9	27.78 ± 19.1	73.16 ± 3.8	73.09 ± 4.11	16.41 ± 2.5
	100	70.9 ± 0.5	70.68 ± 0.84	15.6 ± 0.6	75.31 ± 7.2	75.16 ± 7.48	26.04 ± 19	71.43 ± 4.1	71.4 ± 4.08	16.5 ± 2.2

Table 4.15: imdb average test performance by min leaf point

n_feat	d2v	LR_ODT			RS_ODT		
		accuracy	f1	depth	accuracy	f1	depth
2	*10*	67.08 ± 0.6	66.88 ± 0.72	11.84 ± 2.2	65.13 ± 1.7	65.05 ± 1.66	23.35 ± 2.9
	25	70.23 ± 0.9	70.14 ± 0.87	12.98 ± 3.8	71.65 ± 1.4	71.53 ± 1.33	20.34 ± 2.9
	50	67.49 ± 0.4	67.5 ± 0.46	15.41 ± 17.8	68.16 ± 1	68.05 ± 1.18	20.06 ± 2.5
	75	65.34 ± 0.7	65.28 ± 0.73	16.74 ± 12.2	65.64 ± 1.1	65.55 ± 1.17	20.28 ± 2.4
	100	63.88 ± 1.7	63.56 ± 3.21	16.67 ± 5.4	63.55 ± 1	63.51 ± 1.28	20.77 ± 2.5
5	*10*	70.48 ± 0.6	70.63 ± 0.72	19.62 ± 2.2	65.59 ± 1.7	65.43 ± 1.79	21.69 ± 2.6
	25	77.24 ± 2.3	77.28 ± 0.75	19.25 ± 3.2	75.21 ± 1.5	75.06 ± 1.33	18.18 ± 2.5
	50	75.04 ± 0.4	75.05 ± 2.89	23.57 ± 2.2	72.96 ± 1.1	72.82 ± 1.08	17.59 ± 2.3
	75	72.94 ± 1.6	73 ± 2.92	25.02 ± 3.8	70.6 ± 1.1	70.52 ± 1.11	17.9 ± 2.7
	100	71.27 ± 0.4	71.31 ± 0.82	24.9 ± 19	68.37 ± 1.1	68.31 ± 1.24	18.18 ± 2.5
10	*10*	71.75 ± 0.9	71.03 ± 0.94	8.44 ± 5.9	65.29 ± 1.5	65.21 ± 1.87	21.82 ± 2.6
	25	80.19 ± 1.2	80.13 ± 3.65	25.86 ± 5.5	76.26 ± 1.3	76.06 ± 1.34	18.96 ± 2.5
	50	78.94 ± 0.6	78.86 ± 0.68	34.58 ± 10.6	75.3 ± 1.1	75.18 ± 1.14	17.9 ± 2.6
	75	77.31 ± 0.5	77.31 ± 0.32	37.51 ± 21.4	73.24 ± 1	73.13 ± 1.09	17.8 ± 3.3
	100	75.69 ± 0.6	75.62 ± 0.39	34.97 ± 11.5	71.36 ± 1.2	71.26 ± 1.17	18.08 ± 3.9
20	*10*	71.75 ± 1.7	71.03 ± 3.11	8.44 ± 3.4	65.17 ± 1.7	65.08 ± 1.85	22.07 ± 2.6
	25	81.59 ± 0.8	81.39 ± 1.41	57.85 ± 18.9	76.46 ± 1.4	76.28 ± 1.46	20.48 ± 2.7
	50	81.72 ± 1.8	81.59 ± 0.35	58.75 ± 3.3	76.54 ± 1.1	76.39 ± 1.1	18.85 ± 2.6
	75	80.5 ± 1	80.38 ± 1.33	58.82 ± 1.9	74.91 ± 1.1	74.79 ± 1.04	18.51 ± 2.2
	100	79.16 ± 0.7	79.07 ± 0.53	56.78 ± 1.6	73.15 ± 1.2	73.06 ± 1.28	18.53 ± 2.4
all	*10*	67.08 ± 0.6	66.88 ± 0.72	11.84 ± 2.2	65.13 ± 1.7	65.05 ± 1.66	23.35 ± 2.9
	25	82.32 ± 0.9	82 ± 0.81	8.76 ± 8.7	76.4 ± 1.4	76.25 ± 1.37	21.46 ± 3.2
	50	84.94 ± 1	84.67 ± 1.09	9.48 ± 5.7	77.15 ± 1.1	77.02 ± 1.16	22.2 ± 3.2
	75	85.61 ± 0.4	85.36 ± 0.56	8.92 ± 6.2	76.55 ± 1.1	76.44 ± 1.17	22.04 ± 2.7
	100	85.86 ± 1.1	85.59 ± 1.36	5.8 ± 4.2	75.76 ± 1.2	75.69 ± 1.07	22.77 ± 2.4

Table 4.16: imdb average test performance by n features

Hyperparameters wise evaluation of IMDB dataset, Findings from Table (4.12, 4.13, 4.14, 4.15)

1. Table 4.12 comparison based on epochs, for all epochs LR_ODT performs better in terms of accuracy and f1 whereas most of the time RS_ODT has lower tree size.

2. In Table 4.13 shows the similar result as Table 4.9, both ODT performs better than CART where LR_ODT has a high performance by keeping tree size high.

3. Similarly, based on minimum leaf points has also the same performance of all model showing in Table 4.14. The low point has high depth and vice versa whereas low min point doesn't show high accuracy.

4. Feature selection results in Table 4.15 shows, RS_ODT has performed slightly better performance when selecting only 2 features but the size of the tree also high. For the rest of the feature selection, LR_ODT performs better keeping most low depth most of the time.

Epochs	d2v	LR_ODT			RS_ODT		
		accuracy	f1	depth	accuracy	f1	depth
100	*10*	84.87 ± 1.8	51.43 ± 15.74	10.26 ± 7.3	82.71 ± 1	52.24 ± 1.65	21.94 ± 3.1
	25	83.53 ± 2	42.43 ± 19.84	41.48 ± 48.8	80.21 ± 1.3	44.84 ± 2.9	23.78 ± 3.5
	50	82.52 ± 2	35.86 ± 21.56	30.16 ± 36.7	78 ± 1.5	39.4 ± 2.97	25.86 ± 3.7
	75	81.92 ± 2.1	31.82 ± 22.37	37.26 ± 51.5	76.79 ± 1.5	36.92 ± 3.01	26.9 ± 4
	100	81.69 ± 2.1	30.03 ± 22.78	35.5 ± 51.3	76.17 ± 1.6	36.22 ± 3.01	27.83 ± 5
300	*10*	84.94 ± 1.7	52 ± 15.44	10.99 ± 6	82.55 ± 1	52.26 ± 1.65	22.59 ± 2.7
	25	83.52 ± 2	42.41 ± 19.71	43.01 ± 49.6	80.16 ± 1.4	45.67 ± 2.93	23.92 ± 3.3
	50	82.53 ± 2.1	35.86 ± 21.2	31.05 ± 37.9	77.92 ± 1.5	40.38 ± 3.48	25.78 ± 4.2
	75	81.89 ± 2.1	31.51 ± 22.12	36.74 ± 48.3	76.65 ± 1.5	38.03 ± 2.98	26.65 ± 4.5
	100	81.67 ± 2.1	29.98 ± 22.66	32.51 ± 42	76.05 ± 1.6	37.39 ± 3.05	27.46 ± 4.4
500	*10*	84.8 ± 1.6	50.28 ± 13.05	10.22 ± 7.1	82.5 ± 1	52.29 ± 1.7	22.54 ± 3
	25	83.57 ± 2	43.07 ± 19.03	41.94 ± 47.5	80.11 ± 1.4	45.9 ± 3.02	23.94 ± 3.3
	50	82.5 ± 2.1	34.82 ± 21.03	28.73 ± 35.4	77.87 ± 1.6	40.75 ± 3.46	25.48 ± 3.7
	75	81.92 ± 2	32.32 ± 21.64	36.42 ± 46	76.65 ± 1.6	38.51 ± 3.2	26.81 ± 4.1
	100	81.71 ± 2.1	29.94 ± 22.55	32.25 ± 47.8	76.04 ± 1.6	37.74 ± 3.13	27.67 ± 4.4
800	*10*	84.73 ± 1.5	49.24 ± 17.45	9.86 ± 5.8	82.46 ± 1	52.41 ± 1.72	22.77 ± 2.7
	25	83.52 ± 2	42.41 ± 19.26	43.78 ± 48.6	80.03 ± 1.4	45.92 ± 3.08	24.5 ± 3.3
	50	82.53 ± 2	36.38 ± 21.38	32.1 ± 39.6	77.86 ± 1.5	41.05 ± 3.2	25.36 ± 3.7
	75	81.93 ± 2.1	31.44 ± 22.24	31.53 ± 41.6	76.6 ± 1.5	38.67 ± 3.18	26.82 ± 4.2
	100	81.66 ± 2.1	29.57 ± 22.84	33.78 ± 45.8	75.96 ± 1.6	37.94 ± 3.11	27.58 ± 4.5
1000	*10*	84.77 ± 1.8	50.06 ± 13.39	10.46 ± 5.3	82.43 ± 1.1	52.34 ± 1.78	23.02 ± 2.8
	25	83.56 ± 2	43 ± 20.26	44.06 ± 42.9	80.06 ± 1.4	46.08 ± 3.14	24.23 ± 3.4
	50	82.52 ± 2.1	35.74 ± 22.44	31.61 ± 38.4	77.82 ± 1.5	41.12 ± 3.22	25.78 ± 3.9
	75	81.89 ± 2.1	31.63 ± 22.24	35.86 ± 47	76.51 ± 1.7	38.8 ± 3.17	27.1 ± 4
	100	81.67 ± 2.1	30.35 ± 22.28	35.23 ± 50.9	75.89 ± 1.7	38.14 ± 3.2	27.62 ± 4.3

Table 4.17: quora average test performance by epochs

k	algo.	CART			LR_ODT			RS_ODT		
fold	d2v	accuracy	f1	max depth	accuracy	f1	max depth	accuracy	f1	max depth
1	***10***	82.66 ± 1.3	53.17 ± 2.16	24 ± 3.5	84.92 ± 1.7	51.22 ± 15.61	8.82 ± 6	82.53 ± 1.1	52.51 ± 1.72	22.58 ± 3
	25	79.23 ± 1.5	44.16 ± 0.55	23.2 ± 2.4	83.6 ± 1.9	43.24 ± 18.89	48.02 ± 53.5	80.18 ± 1.3	45.96 ± 2.99	24.21 ± 3.3
	50	77.23 ± 1.7	40.63 ± 1.02	24.6 ± 2.1	82.55 ± 2	36.35 ± 20.88	35.3 ± 39.6	77.96 ± 1.5	40.81 ± 3.22	25.33 ± 3.8
	75	75.66 ± 1.3	38.17 ± 0.36	24.6 ± 2.1	81.94 ± 2	31.79 ± 22.22	34.03 ± 40.3	76.68 ± 1.6	38.17 ± 3.31	26.44 ± 4.3
	100	74.76 ± 1.3	37.57 ± 0.84	23.6 ± 1.7	81.7 ± 2.1	29.71 ± 22.18	39.29 ± 39.9	75.98 ± 1.5	37.52 ± 3.18	27.77 ± 4.6
2	***10***	82.39 ± 1.4	51.88 ± 1.87	24 ± 3.4	85.04 ± 1.5	52.06 ± 13.1	11.74 ± 8.2	82.54 ± 1	52.16 ± 1.76	22.6 ± 2.9
	25	79.35 ± 1.4	43.82 ± 0.85	24.2 ± 2.6	83.64 ± 1.9	42.78 ± 19.71	38.24 ± 37.5	80.14 ± 1.3	45.54 ± 3.08	23.98 ± 3.2
	50	77.26 ± 1.3	41.47 ± 0.67	23.6 ± 2.1	82.56 ± 2	35.83 ± 21.37	29.19 ± 34.6	77.82 ± 1.5	40.41 ± 3.44	25.52 ± 3.7
	75	76.43 ± 1.3	40.14 ± 0.46	24.4 ± 1.3	82.02 ± 2	31.65 ± 21.35	30 ± 48.7	76.59 ± 1.5	38.17 ± 3.01	26.46 ± 3.7
	100	75.31 ± 1.3	38.55 ± 0.39	25.8 ± 1.9	81.72 ± 2.1	30.56 ± 23.07	38.46 ± 57.7	76.07 ± 1.6	37.54 ± 3.21	27.49 ± 4.4
3	***10***	82.29 ± 1.3	51.71 ± 1.7	24.4 ± 2.3	84.96 ± 1.7	50.56 ± 15.41	9.29 ± 6.3	82.7 ± 1	52.35 ± 1.67	22.58 ± 2.8
	25	79.36 ± 1.5	44.4 ± 1.12	24 ± 2.3	83.7 ± 2	42.9 ± 19.51	51.94 ± 59.2	80.34 ± 1.4	45.75 ± 3.14	23.98 ± 3.6
	50	77.52 ± 1.4	42.02 ± 0.78	24 ± 1.7	82.78 ± 2.1	36.63 ± 21.55	31.54 ± 38.3	78.04 ± 1.6	40.69 ± 3.3	25.58 ± 4
	75	75.69 ± 0.9	37.49 ± 0.88	24.4 ± 1	82.13 ± 2.1	32.75 ± 22.27	36.24 ± 43.5	76.76 ± 1.6	38.19 ± 3.24	27.31 ± 4.2
	100	75.77 ± 1.5	39.92 ± 0.84	24.4 ± 1.6	81.97 ± 2.2	30.36 ± 22.98	30.91 ± 45.2	76.15 ± 1.6	37.52 ± 3.19	27.84 ± 4.7
4	***10***	82.23 ± 1.3	52.36 ± 1.6	25.4 ± 2.1	84.65 ± 1.8	49.27 ± 16.7	10.45 ± 4.5	82.51 ± 1	52.66 ± 1.64	22.44 ± 2.8
	25	79.64 ± 1.2	45.51 ± 0.76	23.2 ± 1.5	83.49 ± 2.1	41.63 ± 21.19	32.44 ± 38.2	80.04 ± 1.4	45.96 ± 2.9	24.07 ± 3.5
	50	78.07 ± 1.4	42.84 ± 0.92	24.8 ± 3.4	82.64 ± 2.2	35.85 ± 22.33	21.67 ± 22.1	77.96 ± 1.5	40.91 ± 3.34	26 ± 3.8
	75	75.87 ± 1.7	38.29 ± 0.74	24 ± 1.7	81.89 ± 2.2	31.41 ± 22.76	40.85 ± 56.9	76.72 ± 1.6	38.56 ± 3.34	27.24 ± 4.5
	100	74.95 ± 1.3	39.33 ± 0.35	25.6 ± 2.7	81.62 ± 2.1	29.85 ± 22.76	29.34 ± 39.7	76.04 ± 1.6	37.8 ± 3.11	27.83 ± 4.3
5	***10***	82.52 ± 1.4	52.49 ± 2.02	27 ± 2.5	84.54 ± 1.5	49.9 ± 14.44	11.5 ± 5.4	82.38 ± 1	51.85 ± 1.59	22.66 ± 3.1
	25	78.49 ± 1.2	42.52 ± 0.72	25 ± 1.3	83.27 ± 2	42.77 ± 18.69	43.62 ± 42.8	79.87 ± 1.3	45.2 ± 3.05	24.12 ± 3.2
	50	76.93 ± 1.3	40.56 ± 0.37	24.8 ± 1.4	82.09 ± 2	33.99 ± 21.41	35.94 ± 47.3	77.69 ± 1.5	39.87 ± 3.24	25.83 ± 3.8
	75	75.45 ± 1.3	37.94 ± 0.5	22.8 ± 2	81.56 ± 2	31.12 ± 21.98	36.69 ± 43.1	76.45 ± 1.5	37.84 ± 2.97	26.82 ± 4.1
	100	75.43 ± 1.3	39.1 ± 0.7	25.2 ± 2.2	81.39 ± 2	29.4 ± 22.11	31.26 ± 52.4	75.86 ± 1.6	37.05 ± 3.13	27.23 ± 4.6

Table 4.18: quora average test performance by k fold

min leaf	algo	CART			LR_ODT			RS_ODT		
	d2v	accuracy	f1	max depth	accuracy	f1	max depth	accuracy	f1	max depth
5	***10***	80.22 ± 0.3	49.48 ± 1.08	29 ± 1.5	84.85 ± 1.6	50.99 ± 13.89	11.21 ± 6.9	80.9 ± 0.3	50.31 ± 0.87	26.24 ± 1.6
	25	76.97 ± 0.5	42.74 ± 1.08	27.2 ± 1.2	83.52 ± 2	42.26 ± 20.52	43.5 ± 48.9	78.29 ± 0.8	44.65 ± 2.43	28.08 ± 2.1
	50	75.23 ± 0.5	40.28 ± 0.93	27.6 ± 1.4	82.48 ± 2.1	34.95 ± 22.03	31.51 ± 39.4	75.86 ± 0.9	40.15 ± 2.73	29.86 ± 2.9
	75	73.88 ± 0.3	38.39 ± 0.83	26.6 ± 1.2	81.87 ± 2.1	31.53 ± 22.33	38.64 ± 52.2	74.5 ± 1	38.18 ± 2.55	30.82 ± 3.1
	100	73.35 ± 0.2	38.25 ± 0.69	28.2 ± 1.6	81.63 ± 2.1	30.18 ± 22.62	37.11 ± 53.2	73.87 ± 1	37.48 ± 2.58	31.68 ± 3.8
10	***10***	81.8 ± 0.2	51.07 ± 0.38	26.2 ± 0.4	84.72 ± 1.9	49.16 ± 17.86	10.42 ± 6.7	82.1 ± 0.4	51.81 ± 1.1	23.77 ± 1.8
	25	78.61 ± 0.4	43.81 ± 1.07	24.6 ± 0.7	83.53 ± 2	42.29 ± 18.72	42.2 ± 44.2	79.62 ± 0.8	45.59 ± 3.46	25.68 ± 1.8
	50	76.53 ± 0.3	41.35 ± 1.12	25.4 ± 1.2	82.52 ± 2.1	35.75 ± 21.6	30.26 ± 37.1	77.23 ± 0.9	40.62 ± 2.9	27.01 ± 2.6
	75	75.08 ± 0.5	38.43 ± 1.18	24.8 ± 1	81.9 ± 2.1	32.32 ± 21.64	35.43 ± 47.6	75.93 ± 0.7	38.47 ± 2.77	28.63 ± 3
	100	74.41 ± 0.5	38.49 ± 0.86	25.6 ± 0.8	81.65 ± 2.1	30.15 ± 22.63	35.33 ± 48.5	75.26 ± 0.7	37.88 ± 2.84	29.55 ± 3.7
15	***10***	82.74 ± 0.1	52.66 ± 0.39	25.2 ± 1.9	84.85 ± 1.6	51.08 ± 13.73	10.47 ± 5.8	82.8 ± 0.4	52.67 ± 1.19	22.34 ± 1.9
	25	79.45 ± 0.5	44.58 ± 1.04	23.6 ± 1.4	83.58 ± 1.9	43.51 ± 18.46	41.78 ± 43.8	80.37 ± 0.8	45.91 ± 3.03	23.84 ± 2.1
	50	77.51 ± 0.4	41.63 ± 1.03	24.2 ± 0.4	82.52 ± 2.1	35.49 ± 21.92	32.07 ± 40.1	78.16 ± 0.8	40.79 ± 3.06	25.51 ± 3
	75	75.88 ± 0.5	38.72 ± 0.88	23.6 ± 0.5	81.91 ± 2.1	31.45 ± 22.31	34.31 ± 45.3	76.88 ± 0.8	38.46 ± 3.07	26.96 ± 3
	100	75.2 ± 0.6	39.23 ± 0.53	24.4 ± 0.5	81.7 ± 2.1	30.09 ± 22.49	33.15 ± 50	76.24 ± 0.8	37.71 ± 3.08	27.5 ± 3.6
20	***10***	83.45 ± 0.1	53.77 ± 0.45	22.8 ± 1.9	84.84 ± 1.7	50.9 ± 14.94	10.23 ± 6.8	83.17 ± 0.4	53.06 ± 1.31	21 ± 1.7
	25	80.21 ± 0.5	44.67 ± 1.17	22.4 ± 1	83.52 ± 2	42.31 ± 20.08	45.39 ± 51.7	80.86 ± 0.9	46.12 ± 3.32	22.29 ± 2.1
	50	78.44 ± 0.5	42.14 ± 1.05	23.2 ± 0.4	82.53 ± 2	36.42 ± 20.77	29.9 ± 32.9	78.76 ± 0.8	40.86 ± 3.59	24 ± 2.7
	75	76.61 ± 0.4	38.19 ± 1.3	23.2 ± 0.7	81.92 ± 2.1	31.71 ± 22.1	34.06 ± 43.3	77.56 ± 0.8	38.24 ± 3.41	24.98 ± 2.9
	100	76.12 ± 0.4	39.29 ± 0.98	23.6 ± 0.8	81.71 ± 2.1	30.01 ± 22.37	29.09 ± 37.5	76.93 ± 0.8	37.42 ± 3.38	25.74 ± 3.3
30	***10***	83.88 ± 0.1	54.62 ± 0.51	21.6 ± 1.7	84.86 ± 1.6	50.87 ± 14.8	9.46 ± 5.2	83.68 ± 0.5	53.69 ± 1.58	19.51 ± 1.8
	25	80.84 ± 0.3	44.61 ± 0.73	21.8 ± 1	83.55 ± 2	42.95 ± 20.23	41.41 ± 48.6	81.42 ± 0.8	46.13 ± 2.6	20.48 ± 2.3
	50	79.3 ± 0.4	42.11 ± 0.47	21.4 ± 1	82.57 ± 2.1	36.04 ± 21.3	29.9 ± 38.4	79.46 ± 0.8	40.28 ± 4.13	21.88 ± 2.3
	75	77.65 ± 0.6	38.3 ± 1.16	22 ± 0.6	81.95 ± 2.1	31.7 ± 22.22	35.36 ± 46.1	78.32 ± 0.9	37.59 ± 3.87	22.88 ± 3.5
	100	77.15 ± 0.3	39.21 ± 1.39	22.8 ± 1.7	81.71 ± 2.1	29.45 ± 23.01	34.58 ± 47.5	77.81 ± 0.9	36.93 ± 3.78	23.68 ± 3.4

Table 4.19: quora average test performance by min leaf point

n_feat	d2v	LR_ODT			RS_ODT		
		accuracy	f1	depth	accuracy	f1	depth
2	*10*	81.77 ± 1.3	24.93 ± 3.59	6.19 ± 4.6	82.05 ± 0.9	50.55 ± 1.16	23.23 ± 2.9
	25	80.37 ± 0.5	7.92 ± 9.02	4.7 ± 10.2	78.67 ± 1.2	40.21 ± 1.15	24.88 ± 3.3
	50	80.09 ± 0.5	2.6 ± 10.64	1.32 ± 7.2	76.47 ± 1.3	34.8 ± 1.28	27.04 ± 2.8
	75	80.03 ± 0.4	1.06 ± 2.29	0.81 ± 1.8	75.29 ± 1.4	32.87 ± 1.22	28.03 ± 3.3
	100	80.03 ± 0.2	0.98 ± 2.39	0.29 ± 0.7	74.77 ± 1.6	32.43 ± 1.26	28.52 ± 3.5
5	*10*	85.07 ± 0.4	54.73 ± 17.3	19.56 ± 7.6	82.8 ± 1.1	53.19 ± 1.58	21.85 ± 2.5
	25	82.44 ± 0.7	37.86 ± 8.32	23.84 ± 4.8	80.19 ± 1.1	45.86 ± 1.13	22.41 ± 2.7
	50	80.94 ± 0.2	21.57 ± 4.56	14.84 ± 2	77.71 ± 1.4	40.02 ± 1.61	23.76 ± 3.3
	75	80.45 ± 0.2	12.91 ± 8.91	10.93 ± 7.7	76.38 ± 1.5	37.37 ± 1.56	24.47 ± 3.1
	100	80.27 ± 0.3	8.04 ± 6.84	7.14 ± 6.3	75.63 ± 1.4	36.28 ± 1.59	25.14 ± 3.2
10	*10*	85.76 ± 0.2	57.78 ± 0.74	8.68 ± 1.9	82.59 ± 1	52.58 ± 1.47	22.68 ± 3.1
	25	84.01 ± 0.3	51.97 ± 1.06	56.84 ± 27.4	80.61 ± 1.1	47.38 ± 1.23	22.94 ± 3.2
	50	82.32 ± 0.4	43.1 ± 3.24	36.25 ± 12.7	78.35 ± 1.3	42.18 ± 1.26	23.82 ± 3.3
	75	81.36 ± 0.3	37.6 ± 4.13	40.75 ± 15.6	76.97 ± 1.3	39.42 ± 1.14	24.91 ± 3.3
	100	81.07 ± 0.3	35.33 ± 4.93	38.05 ± 13.7	76.26 ± 1.4	38.47 ± 1.15	25.34 ± 3.1
20	*10*	85.76 ± 0.2	57.78 ± 0.74	8.68 ± 1.9	82.63 ± 1	52.63 ± 1.47	22.42 ± 2.7
	25	85.17 ± 0.2	57.92 ± 0.48	117.9 ± 4.7	80.62 ± 1.1	47.63 ± 1.19	24.82 ± 3.5
	50	83.39 ± 0.4	52.32 ± 1.09	90.24 ± 42.1	78.58 ± 1.3	43.05 ± 1.31	25.17 ± 3.3
	75	81.96 ± 0.4	48.06 ± 1.13	112.52 ± 49.6	77.27 ± 1.3	40.56 ± 1.26	26 ± 3.3
	100	81.26 ± 0.4	45.99 ± 1.33	114.18 ± 47.5	76.59 ± 1.4	39.76 ± 1.26	26.62 ± 3.7
all	*10*	85.76 ± 0.2	57.78 ± 0.74	8.68 ± 1.9	82.59 ± 1	52.58 ± 1.47	22.68 ± 3.1
	25	85.72 ± 0.1	57.64 ± 0.58	11 ± 41.8	80.48 ± 1.2	47.32 ± 1.22	25.31 ± 3
	50	85.87 ± 0.2	59.06 ± 0.9	11 ± 5.1	78.36 ± 1.3	42.65 ± 1.14	28.47 ± 4
	75	85.75 ± 0.4	59.08 ± 0.8	12.8 ± 3.3	77.27 ± 1.4	40.71 ± 1.23	30.86 ± 4.1
	100	85.77 ± 0.3	59.54 ± 0.69	9.6 ± 1.2	76.85 ± 1.3	40.49 ± 1.11	32.55 ± 4.4

Table 4.20: quora average test performance by n feature

Hyperparameters wise evaluation of QUORA dataset, Findings from Table (4.16, 4.17, 4.18, 4.19)

1. In Table 4.16 shows the comparison based on epochs, for all epochs, LR_ODT performs high accuracy and depth are almost same where RS_ODT has high f1 score than LR_ODT.

2. Table 4.17 shows the results by each fold. As in our previous evaluation for other datasets, the performance of LR_ODT has high accuracy by 2-5 percentage and depth has also high where RS_ODT has around 1 percentage high accuracy and f1 score while keeping equal depth size as CART. CART and RS has high f1 score

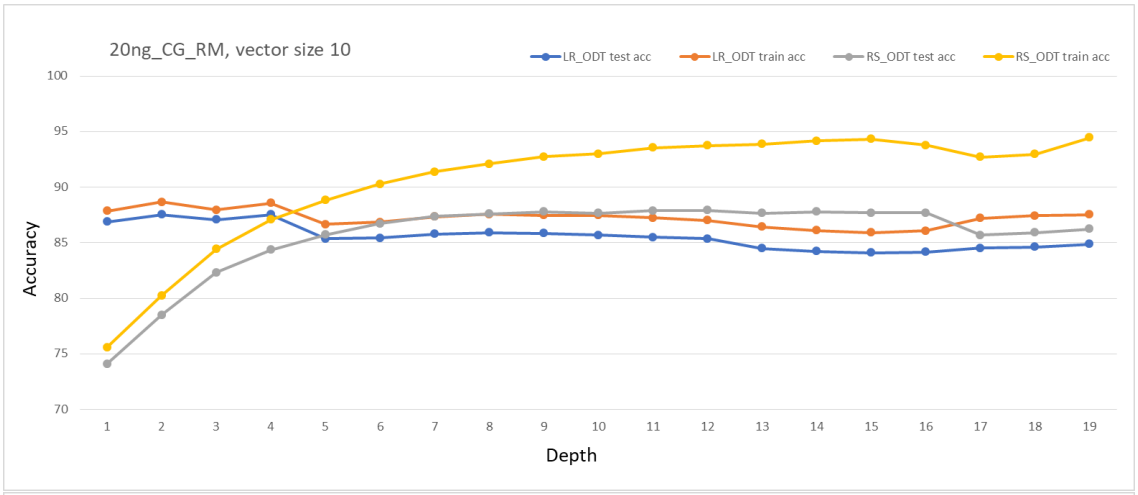
3. Again, Table 4.18 shows similar results as in Table 4.17 when analyzing with several min leaf points. CART and RS_ODT show the same range of f1 score while LR_ODT lacks in this evaluation.

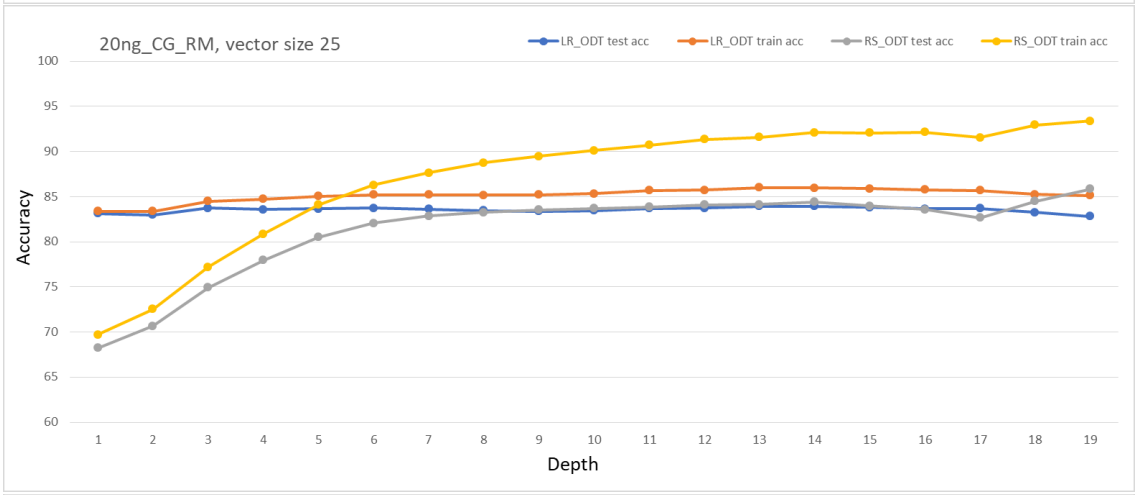
4. While comparing by various combination of features in Table 4.19 shows the same effect as epochs wise comparison in Table 4.16. LR_ODT has upper hand on accuracy and RS_ODT has higher f1 score. Interestingly LR_ODT has short depths.

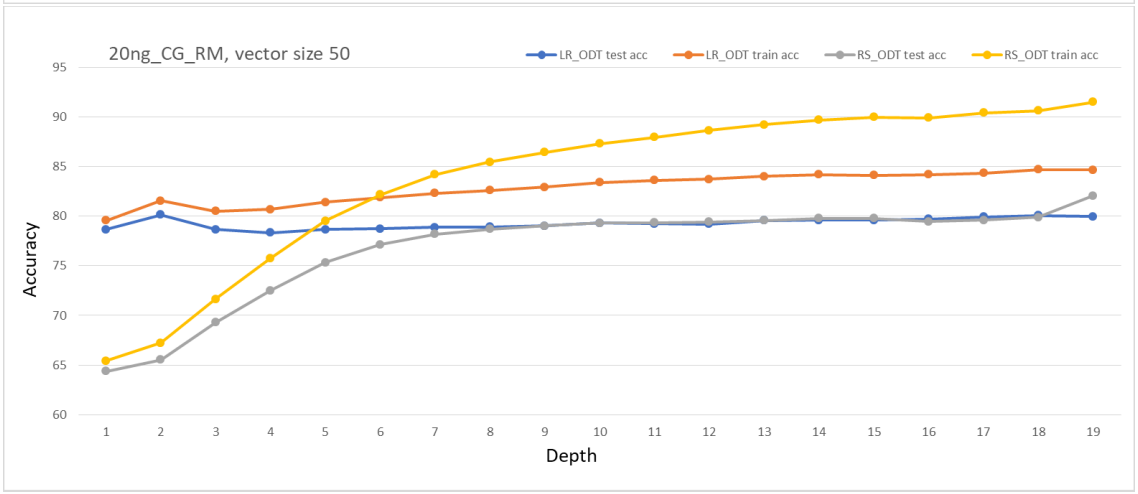
4.4.6 Intermediate accuracy vs depth comparison

In this section, we analyze the intermediate performance of our algorithms. We plot training and testing result of both accuracy and depth to analyze the intermediate behaviour of algorithms. Since CART has taken as granted from sklearn we didn't change its behaviour whereas the other two ODT has implemented this feature.

So far we acknowledged only the final result where we did not consider the growth of the tree with respect to performance matrix. In this evaluation, we can say that LR_ODT may not have many trends because LR tries finding the best split from the root node. In the other hand, it's interesting to observe the RS_ODT modelling process because the only single instance is considered to create decision boundary. The following line chart shows depth and accuracy score base on each D2V size for all datasets.


20ng_CG_RM, vector size 10
LR_ODT test acc
LR_ODT train acc
RS_ODT test acc
RS_ODT train acc
Accuracy
Depth


20ng_CG_RM, vector size 25
LR_ODT test acc
LR_ODT train acc
RS_ODT test acc
RS_ODT train acc
Accuracy
Depth


20ng_CG_RM, vector size 50
LR_ODT test acc
LR_ODT train acc
RS_ODT test acc
RS_ODT train acc
Accuracy
Depth

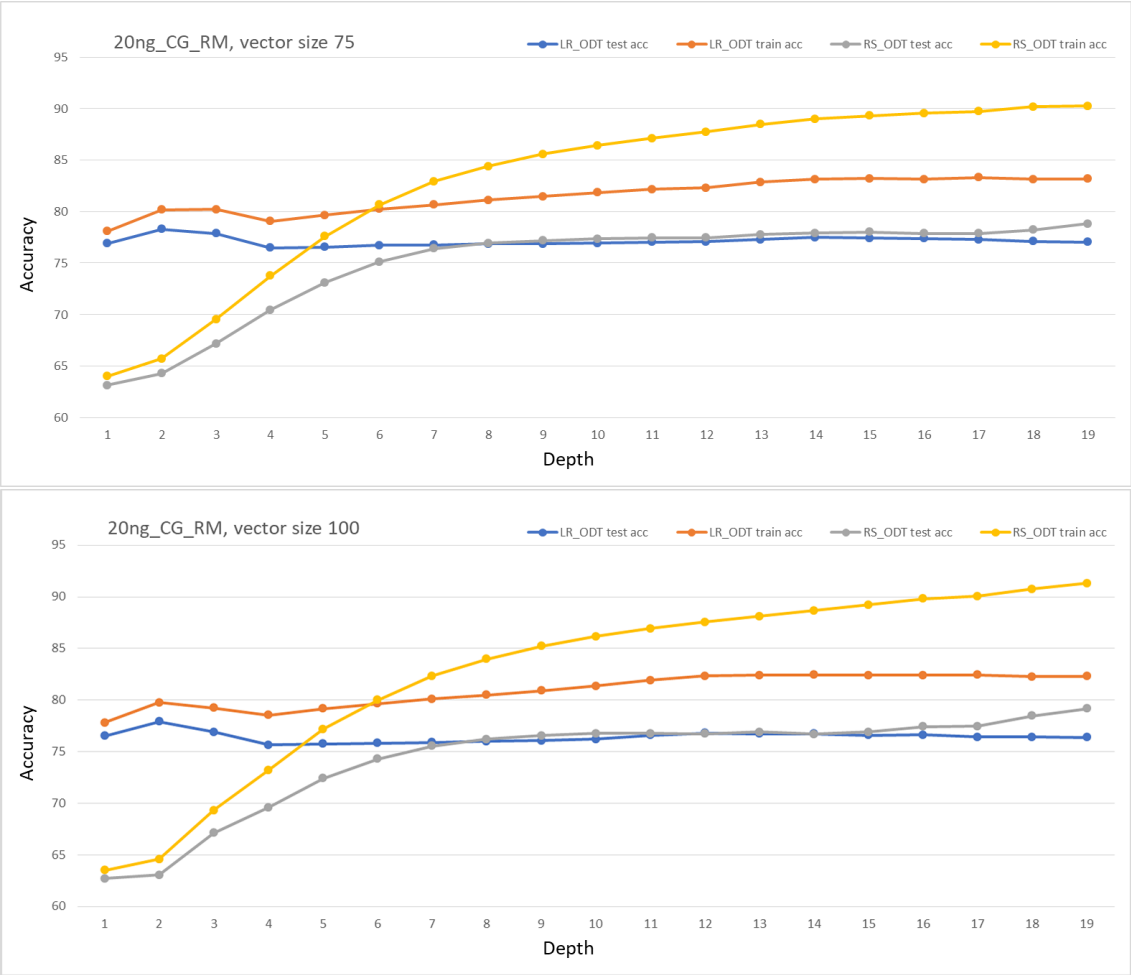
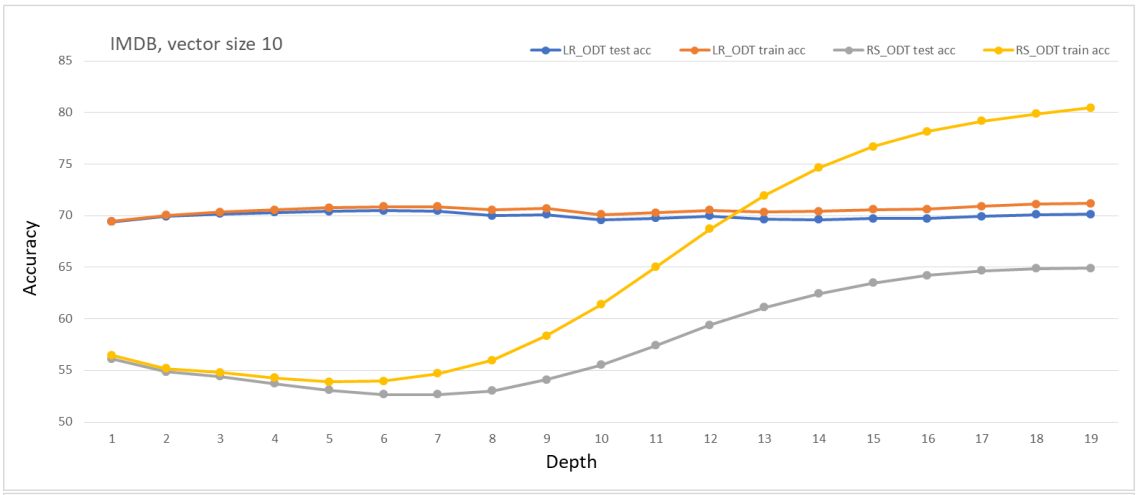
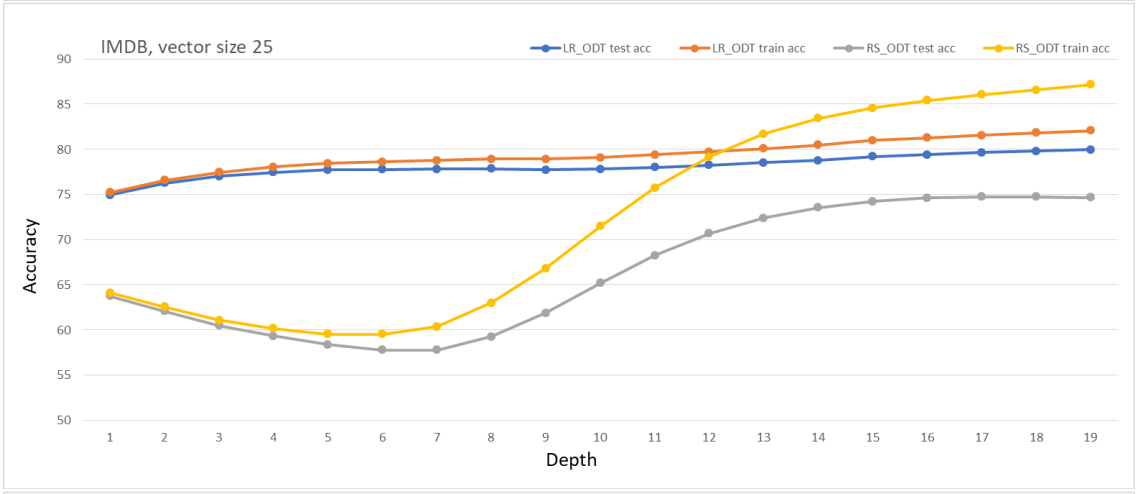


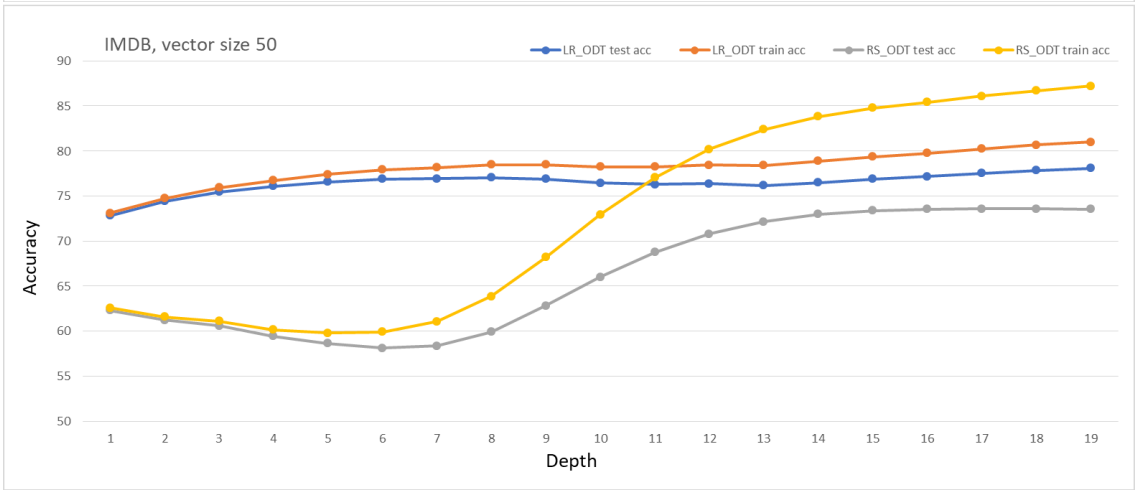
Figure 4.4: 20ng intermediate model evaluation; comparison of depth and corresponding accuracy for each D2V sizes

LR_ODT shows a small percentage of improvement in accuracy when depth increases while in testing it is almost stable. Noticeable changes have happened between 1 to 4th depth while the rest of change happens at the end. Since the result is an accumulated average result, consequently, we can see slight drops in accuracy even if depth increase for both train and test. It shows around 5 percentage of difference in LR_ODT.

The results show for all vector sizes RS_ODT has performed well in training phase than LR_ODT but in the testing phase, it drops by more than 10 percent which can be an overfitting problem. It also shows the possibility of improvement in future work. The growth of RS_ODT model shows an interesting pattern by growing gradually.


IMDB, vector size 10
LR_ODT test acc
LR_ODT train acc
RS_ODT test acc
RS_ODT train acc
Accuracy
Depth


IMDB, vector size 25
LR_ODT test acc
LR_ODT train acc
RS_ODT test acc
RS_ODT train acc
Accuracy
Depth


IMDB, vector size 50
LR_ODT test acc
LR_ODT train acc
RS_ODT test acc
RS_ODT train acc
Accuracy
Depth

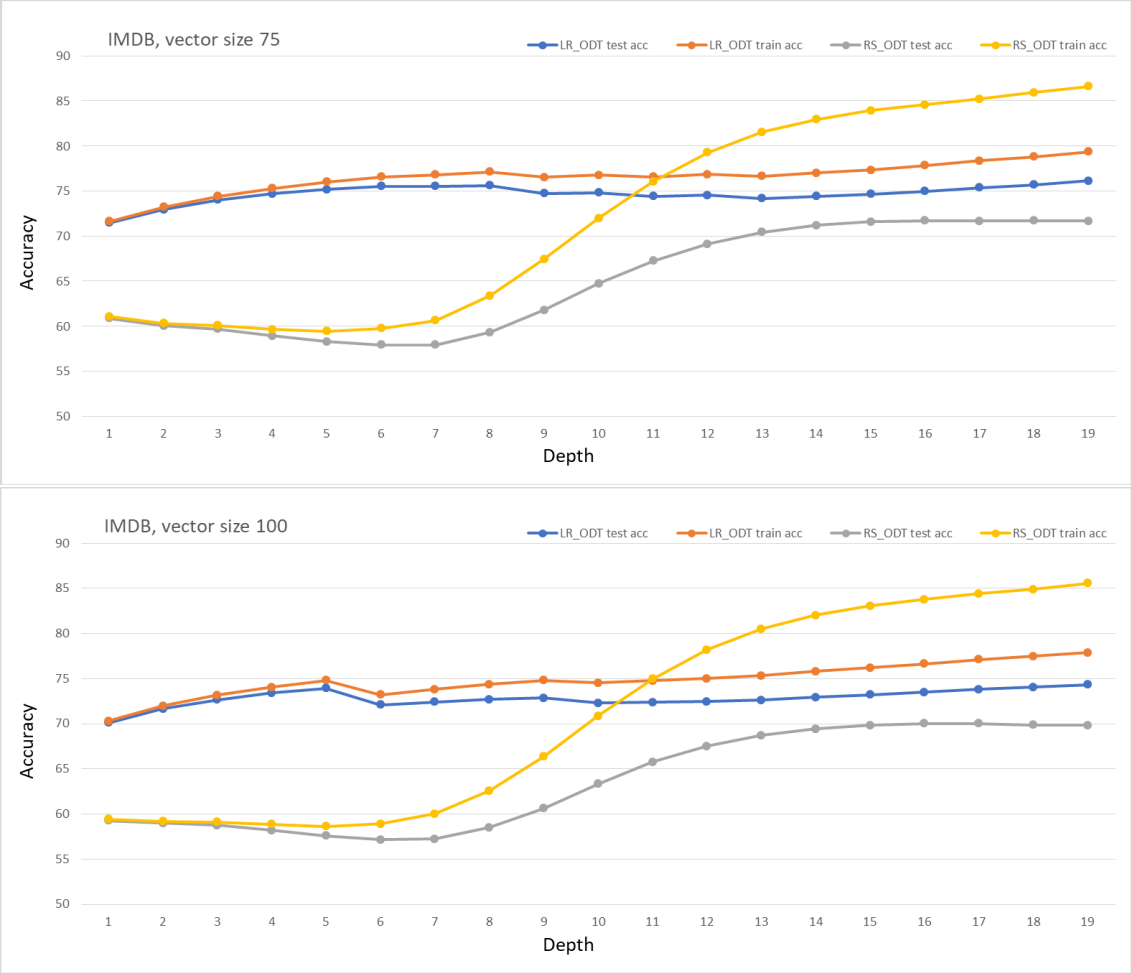
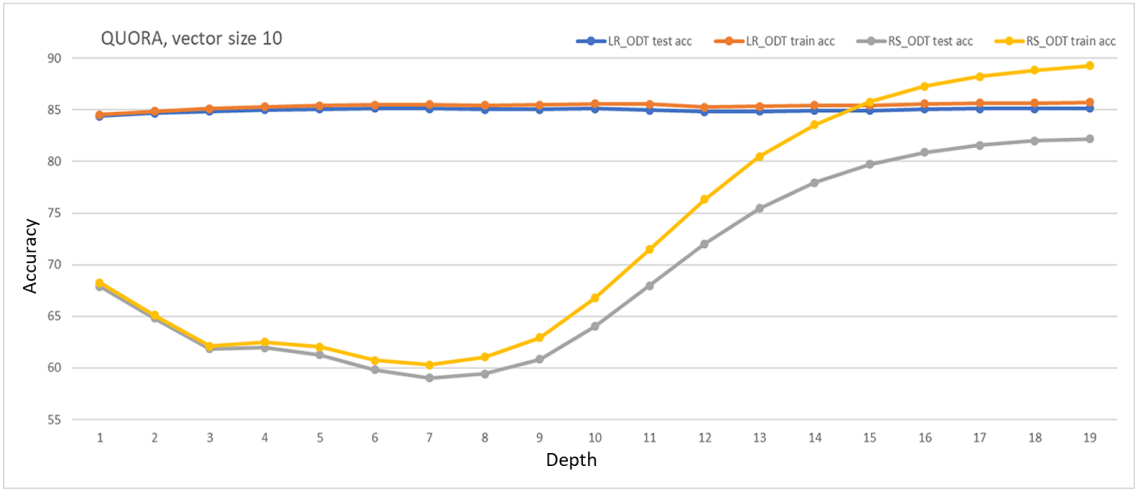
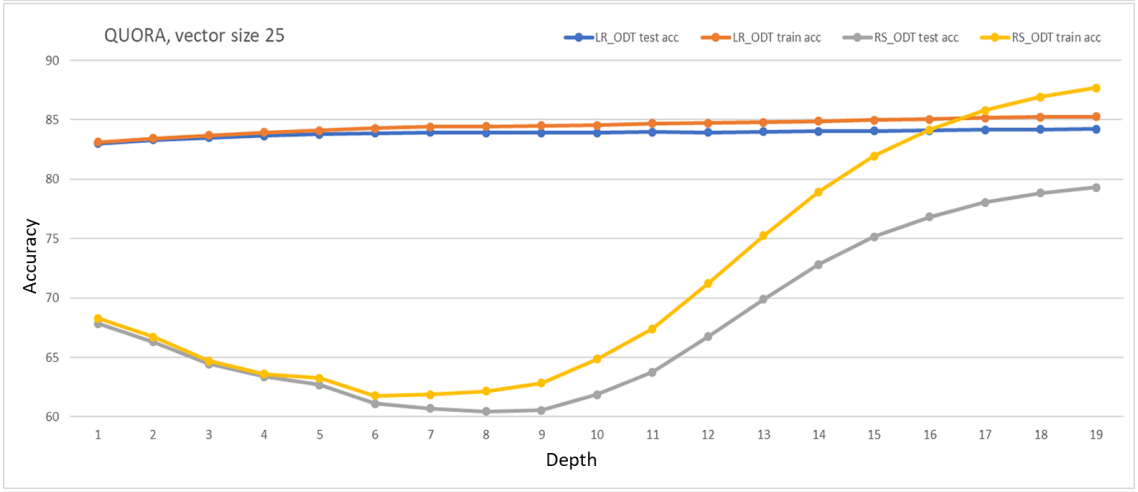


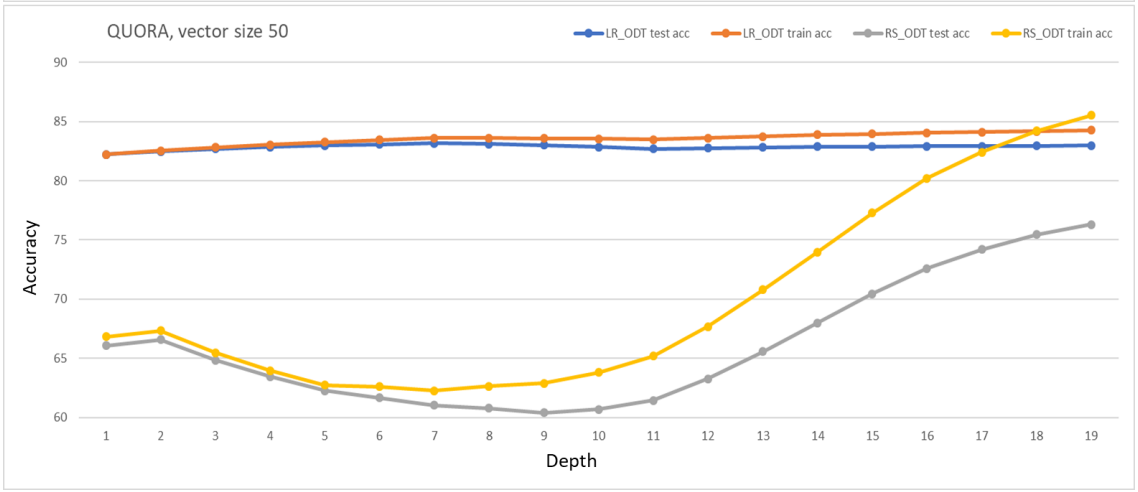
Figure 4.5: imdb intermediate model evaluation; comparison of depth and corresponding accuracy for each D2V sizes.

For IMDB LR_ODT show a slight upward trend for both training and testing. The increment on accuracy is around 5 percent and the difference between training and testing are 4 percent. This fact also confirms LR can be via improved combining approach.

By observing RS_ODT we find no improvement till depth 6 where eventually it improves at a decent rate. Repeatedly we find similar behaviour of RS_ODT where the distance between train and test is very different.


QUORA, vector size 10
LR_ODT test acc
LR_ODT train acc
RS_ODT test acc
RS_ODT train acc
Accuracy
Depth


QUORA, vector size 25
LR_ODT test acc
LR_ODT train acc
RS_ODT test acc
RS_ODT train acc
Accuracy
Depth


QUORA, vector size 50
LR_ODT test acc
LR_ODT train acc
RS_ODT test acc
RS_ODT train acc
Accuracy
Depth

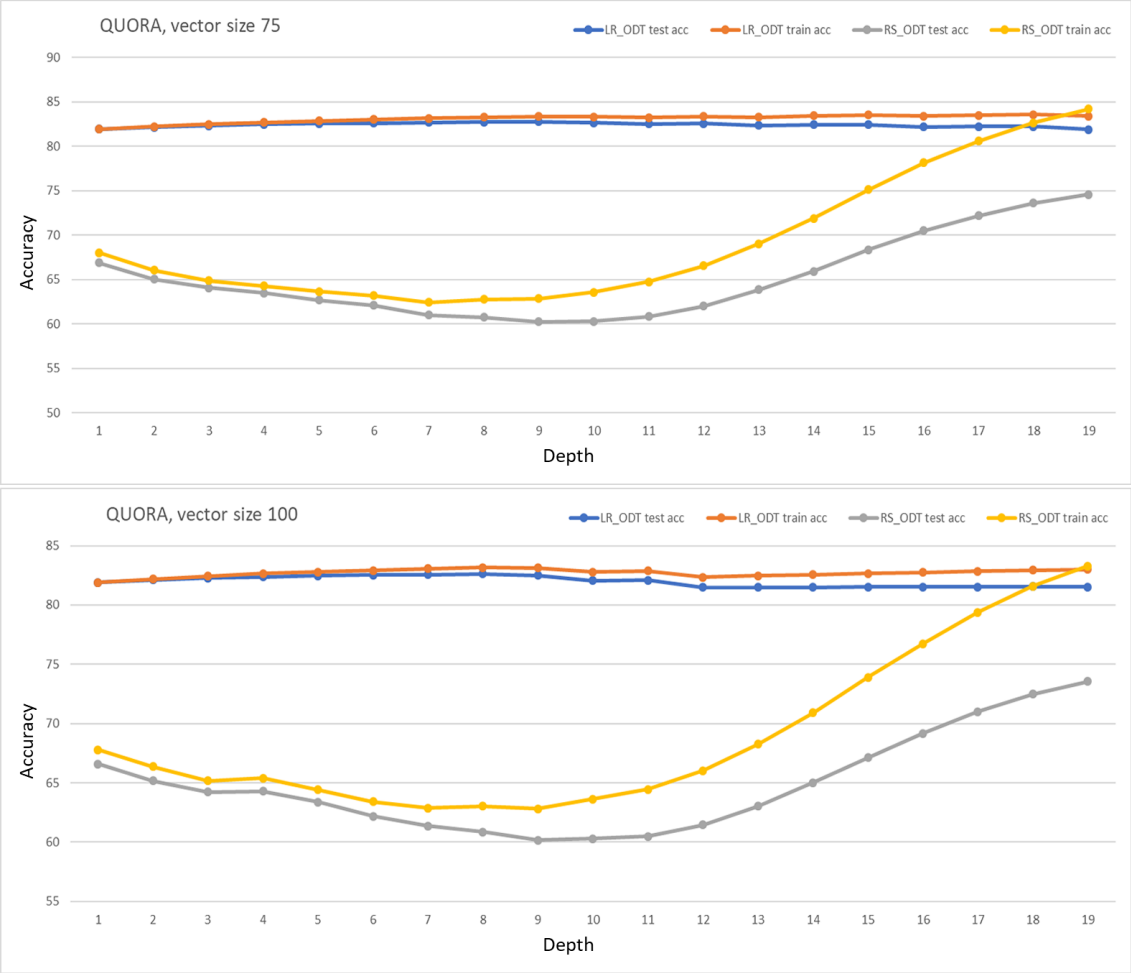


Figure 4.6: quora intermediate model evaluation; comparison of depth and corresponding accuracy for each D2V sizes.

In QUORA dataset LR_ODT shows tiny improvement on both training and testing phase. The difference between them is also less.

In RS_ODT shows unpredictable behaviour by going underneath until the depth of 9 and taking pace to uphill. This behaviour could from those model which low accuracy in short depths.

5 Discussion

In this section, we will discuss our findings afterwards point our experiment limitation and finally provide some undiscovered area as a future improvement.

5.1 Exchange of view

Our experiment covered six aspects of evaluation to compare the pros and cons based CART and provided ODTs in the area of the text classification problem. Most of the time pattern of each algorithm was alike. Combination of LR was most promising than random picking. Here we discussed experimental thoughts on topic comparison.

1. **Time complexity**
   We measured the run time complexity of train and test for each algorithm. Our ODTs takes much time to learn the model and predict than CART. At every inner node, a linear computation has to performed by ODTs at both learnings and predict process the length of time automatically increased by a linear factor $O(n)$. There is no optimized way of fitting θ value in RS_ODT, hence, the process becomes faster by a constant factor.

2. **Tree structure**
   .....

3. **Overall results**
   Here, we saw LR based approach outperforms others inaccuracy but also increase the depth. When classifiers find very small data point to separate it creates more depth. Post pruning or early stopping criteria could the problem. In the other hand, RS_ODT has performed better than CART in many cases by keeping its depth equal. Surprising we saw precision, recall and f1 score are similar to accuracy.

After reviewing the number of miss classification in both classes we confirm the issue arise when an equal amount of prediction happens for both classes. Precision and recall are the difference by false positive and false negative where f1 is a harmonic mean of them.

4. **Minimum and maximum accuracy**
   This demonstration provides different prospective to review experiment results based on only the highest and lowest accuracy and their respective hyperparameters value on vector sizes of datasets. We find on every iteration LR_ODT has the highest performance by more than 3 percentage on both low and high accuracy where RS_ODT comes second. We observed major benefit received because of the LR model. The reason could be the D2V representation of text. The given result doesn't explain the important ratio of hyperparameters therefore we analyze based on a small set of list.

5. **Hyperparameters findings**
   In the process of hyperparameters wise comparison, we saw epochs more epochs gives better accuracy in less vector size and hence opposite relation. Similarly model also generates a small tree when vector size less and epochs are high. Although K fold validation is not hyperparameters, we thought it could be interesting to see how much changes we will have on each fold. As a constant performer CART shows a similar pattern over ever folds whereas ODTs took a random subset of data hence some fluctuation has occurred. On each fold, the smallest vector size shows the smallest tree size while the rest has unpredictable behaviour. F1 seems to be around the accuracy range. Minimum leaf point help to reduce overfitting problem in train phase. Our results show minimum value create high depth which doesn't lead to high accuracy whereas most of the less depth which has high min leaf value show good accuracy. In the other hand higher the vector size show increment in depth.

   Selection of features was a crucial process of our experiment. While our experiment doesn't provide any self-optimization or feature pre-engineering process for taking the best set of features we choose a random method. In random process fixed number of feature random choice is taken. We mostly want to explore effect when we select of least number of features to all features. In our observation, we encounter less number of feature combination gives more accuracy than more or

all combination. By saying this trend also affected by the size of the vector in the opposite direction.

6. **Intermediate observation**
   To examine model performance during full growth of tree we produce each depth versus accuracy prediction on train and test datasets. We found an only slight improvement in LR_ODT whereas RS_ODT can be observed clearly. Since logistic regression itself powerful classification model it tries to find best split in from the initial. At the first depth, most of the data are already in good condition therefore very less data point is left for separation. The left instance from first depth might also suffer from overfitting or those data point could be hard to separate via a linear line.

   In the other hand, RS_ODT gradually improves its purity. Since this process takes no assumption about other features or neighbour data points it generates many local minima there we find a comparably large tree. Unlike logistic regression, it can be used for any shape of data with implicit changes. Due to the same reason in training performance is better than testing performance. From the above observation, we find some aspect to improve and some limitation.

5.2 Limitation

The main drawback of our approach is the time taken to find the best split. Therefore we provided a certain number of fixed features selection approach in a random way. Numbers of hyperparameters makes this process more complicated to find the best hyperplane. In this section, we discussed some important limitation of our experiment.

- **Limitation on feature importance selection**
  Finding an optimal split requires a large number of iteration using brute force step. the feature is either selected randomly or all consider which doesn't make any scoring technique. The selection process becomes the most resource-consuming part.

- **Limitation of LR_ODT approach**
  Classifier merging technique automatically increases time and storage. In the other hand, the merging classifier has its own hyperparameters to tune. Some of the

hyperparameters of sklearn based LR are max_iter, solver and class weight. Since its a linear model it performs best when feature space is linearly separable. After n split subset of impure data could form in any shape in those case model performs worst and keeps increasing its depth with any accuracy gain.

- **Limitation of RS_ODT** The major limitation of this approach is to consider only two points while creating the split. The problem not only generates multiple local optima but also suffers from overfitting because of the same reason. Whereas another problem is finding a proper number of iteration it needs to find the best hyperplane. When a number of instances are extremely high than small times of the iteration couldn't best split.

5.3 Further Improvement

This section we spark some of the areas which can be improved as future work. Regardless of other problem domain, embedding techniques, tree induction approach, improvement in interpretability and regression problem. Our thesis raises the following searching questions. These question should be conducted to investigate possible outcomes.

- **Improvement in feature engineering**
  Performing the right kind of feature engineering always assists to boost model performance. Although our main goal was to perform an oblique split and to compare with CART in text domain yet we could perform the known technique to explore the benefit. Our experiment doesn't talk about feature engineering process but creating valuable new feature and transformation concerning algorithmic suitable form always supports to learn fast and accurate. Creating numeric attributes like a specific word based count, sentiment probability, merging or experimenting with other embedding techniques could be interesting to observer.

- **Improvement in feature selection**
  Previously, we saw how complex it can be to find the best combination of features. If we brute force all the possible combination respect to its value then it could take an exponential amount of time and space. Existing techniques like correlation base selection, clustering or dimensionality reduction could show different results.

- **Algorithmic improvement and parameters tuning**
  We found LR based approach was quite better in terms of accuracy whereas depth was also increased without giving much improvement. Therefore adding some functionality which could stop growing tree base on threshold could resolve this problem. A condition if no improvement then changes the feature index but keep a number of features the same can be applied. In the other hand for RS_ODT if we could increase data points to find the best split or adding the concept of K means algorithms or support vector machine to final hyperplane from the given data may increase the performance and also reduces the depth. Since logistic regression is robust linear classifier but performs poor in non-linear feature space to solve that problem applying random point split can give a better result. Therefore a combined approach in a decision tree with logistic regression and random point split could give better approximate in training phase but algorithm complexity also increases.

6 Conclusion

The main objective of the thesis is presented in Chapter 1. A thorough explanation of methodologies and results are carried out in chapter 3 and 4. Here, we summaries the outcome of this thesis briefly.

In this thesis, we performed two methods for finding oblique splits in the decision tree. One method was known linear classifier and second was weak approximation approach. The experiment conducted in text classification focusing on the binary problem. The text data represented via document to vector technique. The results are explored in a possible dimension to extract the pros and cons.

Oblique split requires a pair or more sets of feature where identifying best sets are computationally expensive than searching based on single feature value. Our second RS_ODT mostly suffers from the same problem where LR_ODT utilizes all feature values yet still the problem remains. Tuning model parameters increase run time. Finally, we prove its better to choose ODT for text domain than UDT when D2V embeddings are used. The possible improvements are exits in both representation and learning process.

We conclude that in text classification using D2V embedding oblique decision tree can perform better than a classic decision tree. We assume better in the sense of evaluation matrix and tree properties. Combining classifier gives comparably good results by compromising interpretability and runtime.

A Appendix

A.1 OC1 and CARTLC evaluation

dataset	algo	train_acc	test_acc	train_f1	test_f1
20ng_CG_RM	***oc1***	99.35 ± 0.001	81.58 ± 0.050	99.36 ± 0.001	81.87 ± 0.052
	cartlc	99.11 ± 0.007	86.75 ± 0.026	99.13 ± 0.007	87.03 ± 0.025
IMDB	***oc1***	99.11 ± 0.002	72.09 ± 0.05	99.11 ± 0.002	71.67 ± 0.051
	cartlc	92.26 ± 0.039	74.04 ± 0.049	92.25 ± 0.039	73.90 ± 0.049
QUORA	***oc1***	99.27 ± 0.001	74.89 ± 0.026	98.18 ± 0.003	43.26 ± 0.031
	cartlc	98.25 ± 0.02	77.86 ± 0.01	95.55 ± 0.02	46.95 ± 0.02

Table A.1: Average results of OC1 and CART LC, max depth is 20 and single runs for each folds therefore 25 times training on each data for one algorithm

A.2 LR_ODT and RS_ODT visualization

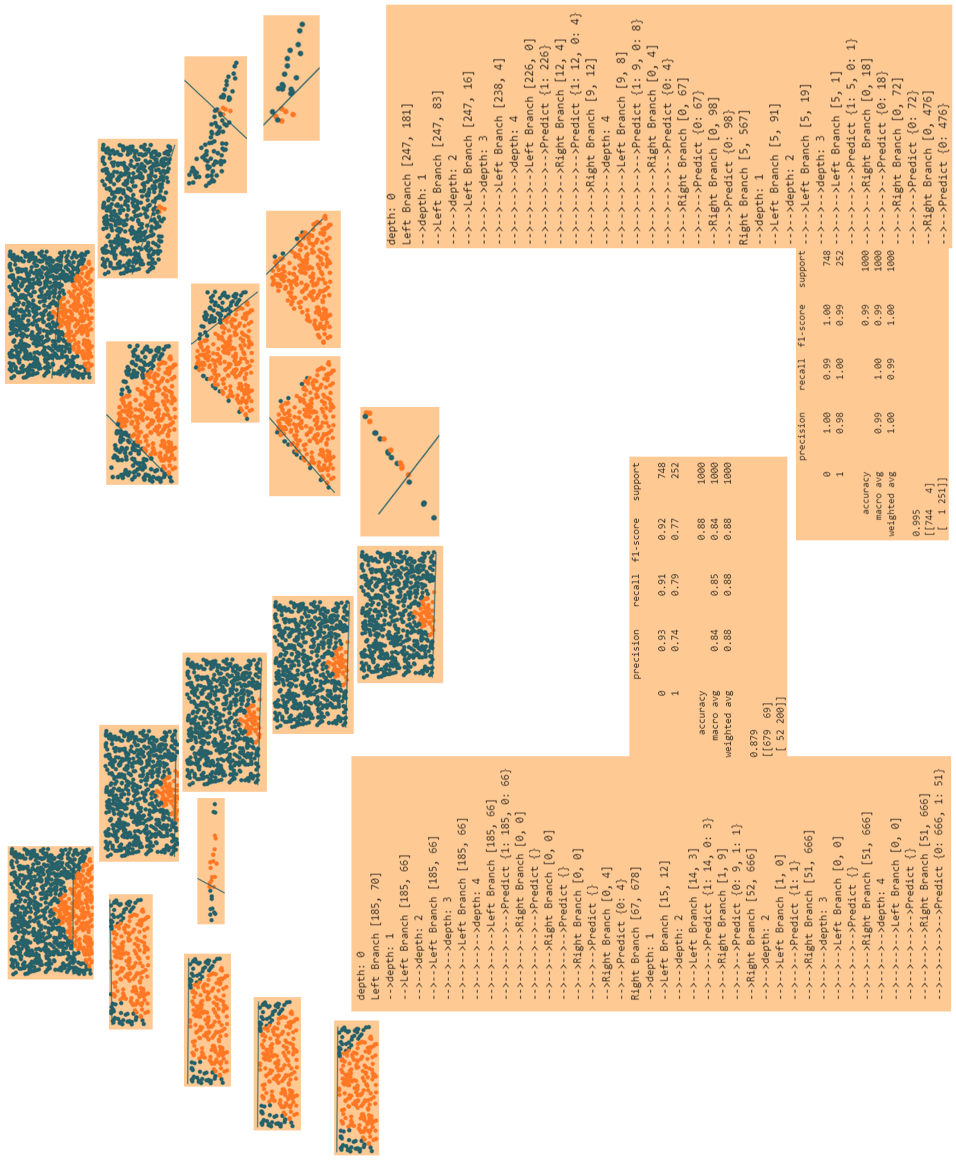


Figure A.1: Linearly separable 2D dataset consist of 1000 instances applied LR_ODT(before entropy condition) & RS_ODT shows learning process and evaluation when depth:4 & epoch:800

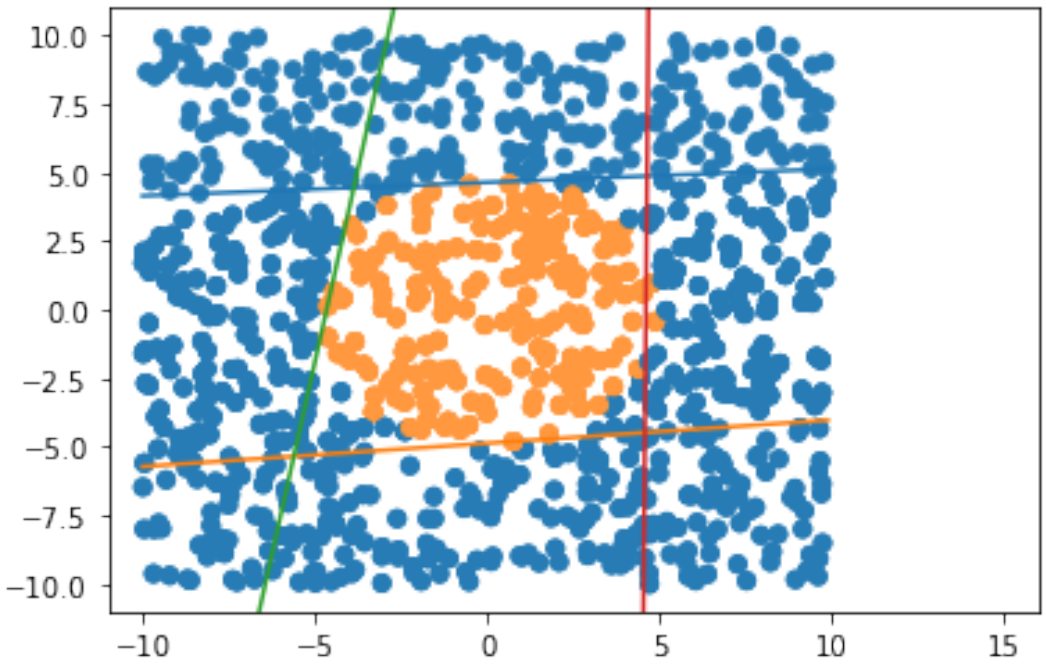


Figure A.2: An example where RS_ODT training on non linear dataset. A non linear 2D dataset consist of 1000 instances trained on RS_ODT shows decision boundaries when depth:4 & epoch:800. A linear model cannot approximate when set of feature are in non linear form.

A.3 PCA transformation of data

data & algo → accuracy ↓	Type	20ng_CG_RM		IMDB		QUORA	
		LR_ODT	RS_ODT	LR_ODT	RS_ODT	LR_ODT	RS_ODT
50-59	zero	2	0	1	0	379	0
	perfect						
	almost						
	left						
	right						
60-69	zero	250	42	85	7	16	2
	perfect						
	almost						
	left						
	right						
70-79	zero	326	205	950	964	12	236
	perfect						
	almost						
	left						
	right						
80-89	zero	928	2151	648	977	146	490
	perfect						
	almost						
	left						
	right						
90+	zero	1619	727	1441	1177	2572	2397
	perfect						
	almost						
	left						
	right						

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie, dass ich die Masterarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, May 30, 2020

AUTHOR