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

June 6, 2020

Contents

Abstract

Decision tree algorithms play a vital role in machine learning. The algorithm is famous for its simple nature and comprehensible decision rule makes the favourable choice for supervised problems. The algorithm does recursive partition of feature space into sub-region using axis-parallel splits. However, for some problems partitioning in axis parallel can produce complicated decision structure to understand having overfitting. As an alternative, the linear split could reduce the complexity while comprehensibility could decrease. There are various ways to compute oblique decision trees, yet they have not tested the performance on text classification domain using document embedding. Therefore, our thesis tries to find interesting results in several aspects by comparing oblique trees with axis-parallel splits. The goal of the thesis is to create oblique boundaries and apply in text-domain to compare with classification and regression tree (CART). Therefore, in this paper, we present the comparison of CART with linear rule induction for text classification domain. We induced logistic regression(LR) and a random sample split approach for creating a linear decision boundary or as a mini classifier for each decision node. It applies document to vector to create feature vectors for binary classes text dataset. A random process is carried out to select a linear combination feature. We observed our induced linear decision trees have higher accuracy than CART in all test datasets of all feature vectors. All linear induction doesn't resolve the overfitting problem rather depends on the classifier. We find D2V representations are suitable in linear induction than CART which creates a comparably small tree with better accuracy in small vectors whereas larger are better at accuracy. Finally, our induction tree model shows a better alternative approach for both CART and induced mini classifiers by improving performance with its interpretable decision rules.

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 last year I have been frequently in touch with Dr Johannes Jurgovsky which I felt very lucky to have such an 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 finish my experiment and calm words gave me more strength and motivation. During the experiment, when it was taking more time than I expected, he assured me to finish properly and guided my focus. In a pandemic situation, his helpful response easies my mental pressure. I would also like to convey gratitude 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 powerful bond. I am thankful to all my friends and family in Passau as well as those who wish and love me.

Finally, I would like to thank my parents for their continuous support and enormous sacrifices they made to ensure that I get the best possible education. As a result, I got an opportunity to pursue my master's degree at the 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

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

ln ..... log

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[20b]. Nowadays machines can do such human-level complex tasks that were almost impossible for a couple of decades before. All these are possible because of advanced 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 solve other similar types of problems. 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[Mit97]

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

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 the relation of X, Y is given as $f(c) = X \to Y$. 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 the dataset in two different subsets, $\mathcal{D}(\text{dataset}) = (X, Y)_{train} + (X', Y')_{test}$. One is used for training an algorithm to create a 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 or label of a particular type of data. Algorithms try to estimate the mapping function f from input variables X to discrete output variables Y. The classification model is trained to predict the unknown label data. A simple example of this type of problem is spam classification in email or messages. This is a binary classification task because it has only two categories, either it can separate into spam or it cannot separate into spam. If classes have more labels, they are called multi-class or multi-label classification problems. If we take the previous example of spam classification, we will first train our model using a training record containing both (X, Y) variables. In each iteration, the algorithm must be optimised to reduce the error or misclassification rate. The final evaluation is done when the output of the model h on (X') is compared to the true output of Y'. The basic analysis of model performance measures whether the model stores data or generalises. Two common problems are overfitting and underfitting, which can be identified by evaluating the train and test performance. Overfitting occurs when the model performs well in training but poorly in testing, and underfitting occurs when both evaluations show poor performance. Most algorithms are designed to solve these two problems, although there are other solutions.

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[20e]

Decision tree (DT) is a well-accepted supervised algorithm. The tree algorithm comprises tree algorithms by satisfying the assumption of the tree rules. A tree has flowchart similar structure made of questions and answers, therefore can easily interpret. The question represents a decision rule or inner branch that either carries another question or leaf node as output. To find solutions, a decision tree takes stepwise decisions from the flowchart and finds the result at the end. It can approximate both linear and non-linear feature relationships, which makes it more applicable. It uses greedy and recursive partition techniques to build full tree models. At a step, the greedy process makes optimal decisions and recursive splits divide the larger data into subsets. The recursive nature splits the dataset based on the impurity function to generate estimated decision boundaries. A decision node represents a "test" based on an attribute value; 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 or approximation function[19]. Applications of DT ubiquitous from data mining, decision support system and information retrieval. Below we can find commonly used terms in DT literature.

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.

Table 1.1: Common terms in Decision Tree

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. $\mathcal{D} = (D_{1+..+n}) \neq 0$, $\mathcal{D}$ is full dataset and D_1 to D_n are subset after splits. Each D consists of $X = (X_{1+..+n}) \neq 0$ and $Y = (Y_{1+..+n}) \neq 0$ 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 rule to split D.
5. $h_\theta(\mathcal{D}_{tr})$ = model, $\theta = \beta_{0+..+len(X)}$, where β_0 = intercept & rest coefficient of X.
6. *Err(h, $\mathcal{D}$)*, is an error rate where output of h is compare to true value of Y.
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.

Basic tree classification process

The purpose of classifying data is to determine the class X' by using estimated function *(h)*. Let's examine an example where data points are scattered in two dimensions. Figure 1.1 shows an example of email data. In this example we have two feature variables x_1 and x_2 represented in two axes.

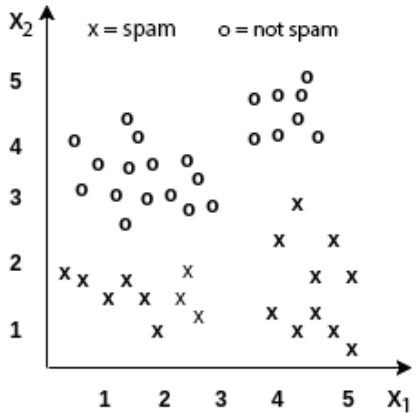


Figure 1.1: An example of email data in a scatter plot.

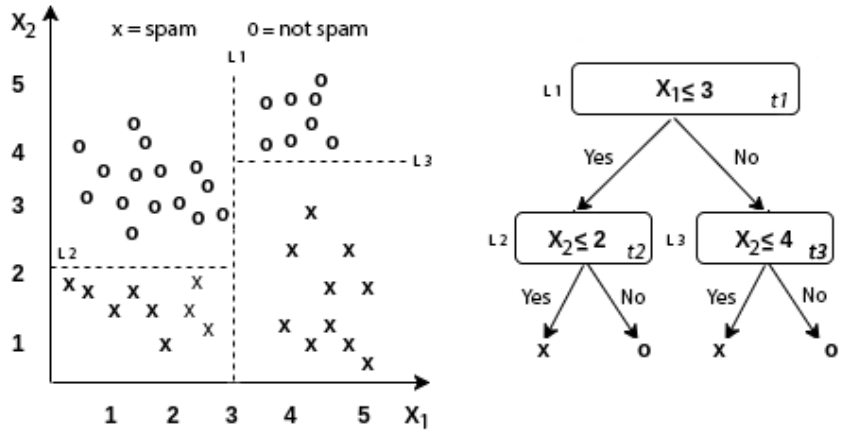


Figure 1.2: An example of decision boundary and tree rule, we can see how binary decision trees split above the dataset and create decision boundaries including tree rule.

Above figure shows a simple example of binary tree classification. In each decision rule or decision node, a feature value finds the best splitting line. The process is known as a univariate splitting process. Since it uses only one feature point at a time, therefore, the decision boundary is always in the axis-parallel. The line L1, L2 and L3 are decision boundaries where it is used 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. Afterwards, the question splits data into two subsets of Yes and No branches. If the split data doesn't give a pure child node, 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, where the 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 a tree. The rule generated by the above decision tree be like:

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 and a graphical illustration of the decision boundary helps understand deeper. With the help of both, we can make a precise reason why a particular question is asked on each level. The below steps show how a tree grows and predict a new instance from the.

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.

An efficient decision tree must try to split the branch in an equal number of instances. Only in search of pure nodes can lead to over generalise and complex rules, therefore splitting criteria play an important role. The subsequent content shows how the decision tree knows which question to ask first and keep asking more to grow a full tree.

Impurity measure

Different tree-based algorithms use their strategy to measure impurity. These measure the homogeneity of the target variable within split subsets. Split creates sub-nodes and in sub-nodes must increase the homogeneity of the category. Asking the right question or decision rule is crucial and identifying the best question can give an optimal tree. Where finding such an optimal tree requires creating all possible trees, which is practically impossible. Therefore, Decision tree measures the importance of feature value through impurity function. If the tree cannot give optimal rules, we at least look for suboptimal decisions. Those sub-optimal trees comprise two or more local optimal decision rules. An impurity criterion applies in all kinds of classification and regression problems.

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

   Information theory proposed by Shanon[Chi+] that states less impure data 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, subtracting the E of the child nodes. Quinlan proposed the application of information gain in a decision tree in 1986 [Qui86]. We calculate it 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 homogeneous, then the entropy is zero and when different categories equal (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, 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 the difference between beginning $E(\mathcal{D}_{tr})$ and $E(x_i, \mathcal{D}_{tr})$. After all attribute gain calculation we choose the best gain which has 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. Using the Gini Index to measure for DT was proposed by[Bre+84] 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 $\mathcal{D}_{\sqcup\triangledown}$ given that partitioning is $G(xi, \mathcal{D}_{tr})$. We compute a weighted sum of the impurity of each resulting partition.

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

Gini considers each of the possible binary splits. In case of a discrete-valued attribute, we select the subset that gives the minimum Gini index. Next section show how CART handles continuous-valued attributes.

Threshold split on real inputs

We have seen splitting examples on categorical inputs. Sometimes, like image or text classification, we encounter many continuous independent features which are also known as real features value. Handling real values is crucial in one hand it saves computation time and secondly it can be a counter-attack for overfitting. For example, if we have a unique 10k floating-point value as a feature then we have to compute impurity for each unique value therefore this process doesn't give any better result than learning strict rules. As a solution, the discretization of a continuous variable is a must. Many decision trees apply this transformation approach whereas our used CART algorithm also tackles real inputs by dividing into reasonable bins by different algorithm vendors. With continuous-valued attributes, DT uses such strategy to select each pair of adjacent values as a split-point[18]. The following steps show how CART measures continuous attributes.

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 [skl20c]. 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

We know determining all possible trees from the dataset is impractical to accomplish, therefore greedy divide-and-conquer approaches 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 performs by a single feature value or multiple feature combination. Decision tree induction is the method of learning the decision trees from the training set. There are distinct characteristics for building trees, some are based on impurity measures, tree construction approach (top-down and bottom-up) and some are about feature selection strategy. Our thesis shows two different approaches to building trees which are based on the orientation of the decision boundary also known as a feature selection method. The below content explains two rules of feature selection in the decision tree.

Univariate decision tree (UDT)
UDT trees are old and powerful trees having axes parallel split orientations. They can handle missing value and feature engineering problems. Impurity measure finds the best homogeneous child nodes from a single data point of a feature, therefore it doesn't require many features engineering process. The main issue with approach is it cannot generalise when the true decision boundaries are not aligned with the feature axes. 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 is the CART.

Oblique decision tree (ODT)
Unlike UDT, oblique decision trees 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 higher dimension it creates hyperplane. CART-Linear Combination (CART-LC), Oblique Classifier 1 (OC1), HouseHold Matrix-CART (HHM-CART) are some algorithms of ODT[ODT reference]. The major benefit of this tree is to overcome the problem of UDTs by not only strictly aligned to axis-parallel split. The most major 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 features 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 samples at the root.
2. Select feature for linear combination.
3. Find θ of a multivariate test such results in the best partition.
4. Build a 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

In today's world, text documents occupy a large value of space by growing rapidly. The booming web platform is giving the power to create any kind of document to all accessible people. So, understanding those data to add value in the business becomes a challenging task whereas machine learning is trying to give automation in several aspects. Text is a natural way of expressing information which is easily understandable by a distinct human. With computers, the text is unsuitable to process directly, hence we represent them in a suitable form. Feature engineering of the natural text differs from other domains like image or audio signals. Natural language processing is a subset of AI that helps computers to understand and manipulate human language. It comprises several tools and techniques to convert text into a set of features vectors. But vector conversion is not the only thing to do, but putting human-level sense is also crucial. Non-semantic representation processes like word frequency count, labelling/one-hot encoding or bag of words (BOW) are some examples. 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 is self changes based on how they are used and sometimes we have to conclude by reading the entire sentence or paragraph. Let's analyze two different methods of feature engineering.

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 fulfill this requirement but lacks ordering of words and semantics. In the below example we can see sentence 1 doesn't have words *["also", "football", "games", "hates"]* and become 0. They lose all information about word order: "John likes Mary" and "Mary likes John" correspond to identical vectors[20c]. 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. On the other hand, words which have antonym meaning of likes and hate, or it does not preserve the contextual similarity between football and games.

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 plays a vital role by adding a layer of similarity semantics. Computational models that build contextual semantic representations from corpus data in the form of vectors. The vector is a learned model via a shallow neural network and can capture some semantic concept of a corpus. Such a concept was later manipulated 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[DOL15]. In our thesis experiment, we have used Doc2Vec embeddings to measure performance of decision trees.

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 input of 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. It positions word vectors in the vector space they locate such that words that share common contexts in the corpus close to one another in the space[20g]. Vector space gives an intuition of knowing other similar words as humans. We can extract information like common words, similarity distance and can do all kinds of vector manipulation to retrieve more information. A higher dimensional embedding can capture fine-grained relationships between words but takes more data to learn.

There are two different architecture of Word2vec models. They are 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[Mik+13]. The Word2Vec model calculates vectors for each word in a document, but if we want to calculate vectors 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 comprising 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[DOL15].

Doc2vec has two approaches for vectoring documents. 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 like 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 state-of-the-art results by itself. We can find a detailed working mechanism in [LM14].

1.2 Motivation

In summary, we knew a greedy process cannot guarantee to return the globally optimal DT, hence finding it is an NP-complete problem. Another issue DT learners create biased trees if some classes dominate. We identified UDT takes a single feature point where ODT takes several features combinations. Sometimes where data are linearly scattered, UDT can create over-complex trees that do not generalise the data well hence becomes overfit. In that case, an oblique split could be generalisations 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 the perspective of data, their properties and representation data play an important role in choosing the right algorithm. In the real world, the dataset contains many features that are impossible to visualise using no dimension reduction techniques. Selecting a particular task with a hypothesis may reduce the problem.

Generally, UDT and ODT models are the choices of trade between generalisation 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 an optimised selection mechanism. Simply, We don't know which combination of features

can provide the best split in 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 suitable part of ODTs is it guarantees small trees with competitive performance. Many studies have shown that trees which use oblique splits produce smaller trees with better accuracy compared with axis parallel trees[Wic+15]. When feature space is in a higher dimension using a 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 dimensions 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 thousands of examples 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 the model in and to provide a fixed length of given vector dimensions. Depending upon the 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.

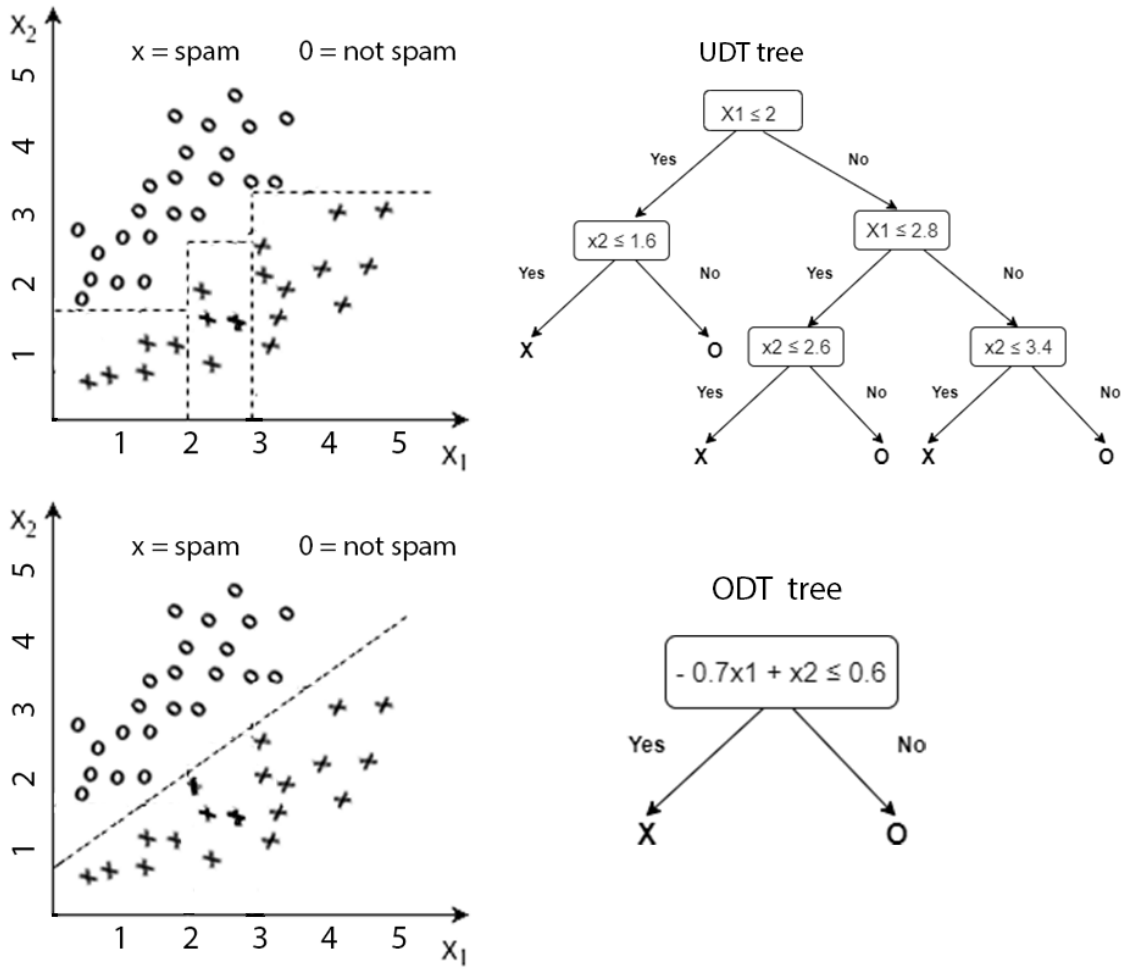


Figure 1.3: The general approach of UDT (top) has poor approximation and ODT (bottom) is the one we need, a linear function

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 be 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 roughly. The model generates a complex tree and is overfitted. On the other hand, a multivariate decision tree (bottom) learns the linear decision boundary using only a single split, hence more generalised and short depth. Therefore, we are interested to see in the domain of text classification whether or not we get similar results. In the next section, we define a more focused research question for our experiment.

1.3 Research question

We classify our research question into 3 scenarios. In which we will see if the oblique tree is better than a CART or not and compare two ODTs approaches. We will seek comparison in terms of:

- **General evaluation matrix:** We will compare four evaluation matrix accuracy, precision, recall and F1 for all algorithms.
- **Efficiency of tree attribute:** It will be an overall comparison of tree depth, number of nodes, branch size. With ODTs, we also compare intermediate result focus on depth versus accuracy.
- **Evaluation based on embedding sizes of Doc2Vec model:** It will interest for us to see if the size of embedding makes any difference with the performance or not. To test this, we adopt 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 split based on RANSAC process.

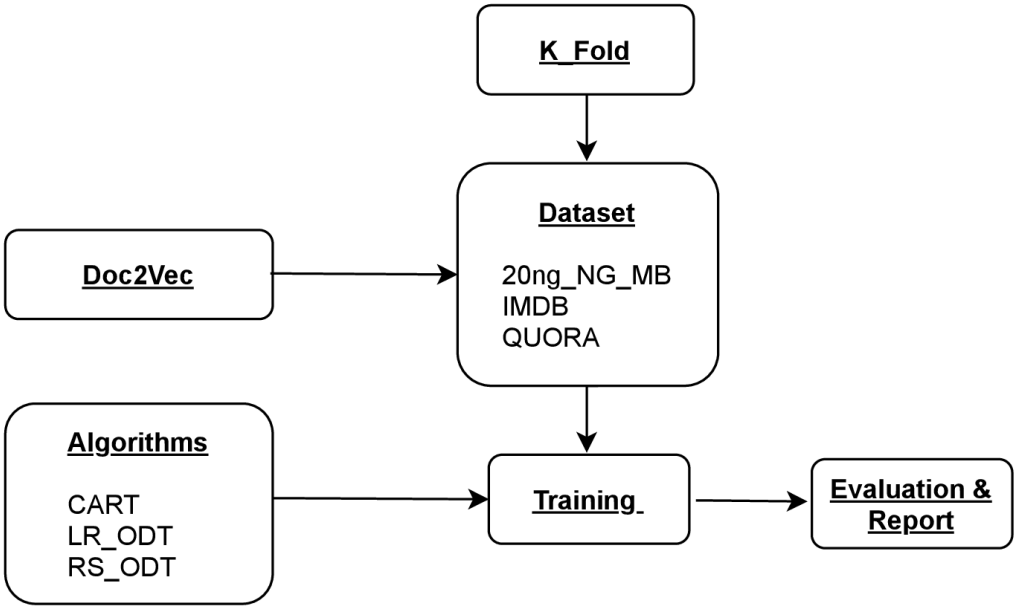


Figure 1.4: Basic work flow of proposed approach.

We have a CART and two induced ODT algorithms. We will test on three different datasets. These datasets have unique properties and have two classes only. The detailed information is available in chapter 4 results. With the help of k_fold validation, we will calculate an average result at the end of evaluation. We use Doc2Vec to train k_fold split datasets to train and build features. Then we fit all available algorithms. In the end, we analyse 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 are familiar with the basics of decision trees and their types, continuously about NLP semantic model of word embedding which leads to motivation of our thesis then we finish with research question and approach. In the next 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 about our ODTs approach and CART is available in chapter 3. A systematic workflow, Configuration of the 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, we present a brief survey of oblique DT induction methods as related work. We will see available algorithms, merging concepts, feature selection and engineering processes. A detailed description of the partitioning strategy used in CART is available in Chapter 3. Though the other related methods are not directly relevant to the principal method 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 generalised 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 learning. The primary job of the tree growing stage is to search for the best split. Choosing a lookup method for the best oblique split is crucial. Runtime and performance also depend on it. To a great extent [DS] proves that the problem of finding an optimal oblique split is NP-Complete. Many DT induction methods work in distinct ways. Our thesis is mainly influenced by the prior concept of merging linear classifier and the random search method from section 2.1.1. Therefore, in this section, we expose some oblique DT induction methods in brief. It divides this section into two parts one which shows induction algorithms that use optimisation, heuristic and statistic approach and next is combining algorithm.

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 different pre-specified locations. It also carries a backward feature elimination process and delete irrelevant features to find optimal the split[Bre+84].

It introduces a randomisation approach called simulated annealing DT (SADT) which uses the simulated annealing optimisation algorithm to search for the best split[DS]. At each non-terminal node, it sets an initial hyperplane 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. It then tests the resulting hyperplane is then using an impurity measure and if the impurity reduction is negative, then the new hyperplane split is always accepted. If it is positive, then it accepts the new hyperplane with a probability function. Initially, T is set large, so that when it 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. It repeats this process 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 might lead to a globally optimal tree. The major disadvantage of the algorithm is the time taken to find the best split. Sometimes, it may require the evaluation of the tens of thousands of hyperplanes before finding an optimal split[MKS94].

[MKS94] combine the concept of CARTLC and SADT to introduce a new oblique DT method 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. It performs these two steps several times. Each time, the algorithm starts with a different initial guess. One of the initial guesses is the best axis-parallel split. It also uses a random hyperplane as initial guesses. Since each initial guess potentially converges to a different hyperplane, the one that maximises it take the impurity reduction as the splitting hyperplane.[MKS94] 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, getting a similar upper bound is an open question. 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. 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, realising this advantage is tough when the learning sample contains many examples and features. In our appendix section we have presented overall all test results to conduct a high level analysis.

[00] proposed a statistical method of generating unbiased trees called Quick Unbiased Efficient Statistical Tree (QUEST). It determined the splits 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 test the input fields at a node. For selection, 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 [MS12]. For a binary classification problem, the algorithm generates two clustering hyperplanes, one for each class. Each clustering hyperplane tries to minimise the distance between examples in the same class while maximising the distance between other class examples. The separating hyperplane is found by calculating the angular bisector of the two clustering hyperplanes. Since there are two angular bisectors, the one that minimises an impurity measure is chosen as the splitting hyperplane. On a non-binary classification problem, the authors suggest forming two super-classes where one superclass contains the class which has the most examples and group the remaining examples from the other classes into the other superclass. 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 [RPR13], 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 key 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. Following algorithm extends the concept used in the CARTopt algorithm to create partitions in several ways to develop a complete oblique DT.

It extends the method used in the CARTopt algorithm to construct feature space partitions 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. 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. It constructs a Householder matrix 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 reflection is done and hence, axis-parallel splits are searched in the original space. The empirical results show that HHCART induces better trees, in terms of accuracy and the tree size than other DT algorithms for most of the problem domains. They design the algorithm to convert qualitative features into quantitative features, and HHCART can handle both qualitative and quantitative features in the same oblique split[Wic+15].

2.1.2 Merge algorithms

There has been some improvement found by combining classification algorithms 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. [CE07] 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. It describes an evolutionary process by which the population of base classifiers grows during run-time to adapt to the newly seen instances. Another, SVM implementation is available in [Nim11; Vla05].

[Fur05] has used linear regression functions at the decision nodes. They present an algorithm that adapts this idea for classification problems using LR. With a stage-wise fitting process to construct the LR models that can select relevant attributes in the data naturally, the models represent decision at the leaves by incrementally refining those constructed at higher levels in the tree. Similarly, our one approach is also based on LR. Each non-terminal node gets its decision from LR fitting more the detailed explanation is available in Chapter 3 method. DT induction method was also influenced by the development of artificial neural networks, evolutionary and genetic algorithm[CE07; Siu19; Heh20; BAC99].

2.2 Linear model

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

2.2.1 Logistic regression

Our approach takes advantage of LR to find θ and split data on each decision node. LR is a statistical function used in ML. It helps to estimate how the probability of categorical variables is influenced. Therefore, given the cause variable, we try to predict the probability of the effect variable. LR requires Y variable to be discrete where X variable can support both. Unlike linear regression which fit a line between X and Y variables via line function $y = mx + b$, a LR fit S curve between X and the probability of Y using a sigmoid function($\sigma = \frac{1}{1+e^{-(\beta_0+\beta X_i)}}$). The following figure shows the S curve with upper and lower bound limits.

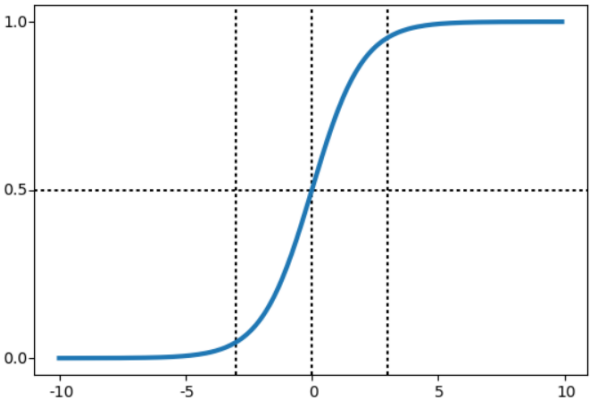


Figure 2.1: Sigmoid curve

A logistic regression is transformed in to a linear problem by using log (ln) transformation which is also known as logit of the probability (logit(p) = $\ln(\frac{p}{1-p}) = \ln(\frac{p(yi)}{1-p(yi)})$) or the log of odds of an event p. The odds of p = $\frac{p}{1-p}$ and the log of odds is ln odds (p) = ln(p) - ln(1-p). Once we apply logit transformation, it becomes identical to linear regression processing, therefore we get logit(p) = ln odds(p) = $\beta_0 + \beta X_i$. LR process involves finding the value of θ which best fits our data and plugins those values into S curve equation then given any value of X we can predict the probability of Y hence boundary always lies between 0 and 1. In other words 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 output in the form of probability therefore the formula for LR will be $h_\theta(x) = \sigma(\beta_0 + \beta_1 X)$. β represent a parameter of model where βo is intercept 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, probability ≥ 0.5 gives $y = 1$ else $y = 0$. This means predicting class 1 whenever $\beta_0 + \sum \beta_i . X$ is non-negative and otherwise 0[20a].The requirement of a problem can make a change in threshold which is known as threshold tuning.

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

Let's see an example from above how log-based cost function works. If $y = 1$ and $h(x)$

= 1, then cost is 0 but in the same case when *h(x)* is 0 then cost will be high hence the learning algorithm will penalised by large cost. Then θ tuning process happens, meaning when the prediction of the class goes wrong the optimisation method tunes the coefficients. The training aims 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). Again, 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. An optimiser like gradient descent applies to minimise cost function by tweaking the β, in-depth explanation is available in [Rud17].

[Fan+] said the solver liblinear is a good choice for document classification. It is used for performing linear-classifier learning both binary, multi-class, and regression. It supports various training methods and objectives, such as SVM and Logistic Regression, with different regularisation terms. For our experiment purpose, we are using scikit-learn based logistic regression algorithms. We have selected liblinear[20f] as a solver. In the next section, we explain how random splitting processes take place.

2.2.2 Random sample splitting

Although our second ODT strategy is based on RANSAC, we only utilised a random sample selection process. Our approach does random sample selection from each class to find linear split. RANSAC we proposed by [FB81], an iterative general parameter estimation approach designed to cope with a large proportion of outliers in the input data. It assumes that all the data comprise both inliers and outliers. A model with an approximation fit can explain inliers, 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 observations is used to learn the model. RANSAC has two fundamental steps: 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

Steps of RANSAC
1. Selects N sample data items at random.
2. Estimates parameter θ from linear function.
3. Update θ which reduce the error (best consensus).
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. The goodness of model highly depends 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[skl20b]. The actual uses of this method is discussed in chapter 3 Methods.

2.3 Other feature selection strategies

The feature selection process has always been a crucial step in ML. As the amount of data increases or dimension of data, several challenges occur. In some algorithms, comprehensibility and efficiency of the model have highly depended on dimension. The following content explains some feature selection process, which aims to remove irrelevant features or reduce dimension.

[YL] purpose a novel concept of predominant correlation, introduce an efficient way of analysing feature redundancy, and design a fast correlation-based filtering approach. By skipping the pair-wise process of checking redundancy they have used a heuristic approach to skip some processes to get fast computation. A similar approach is taken by [SWZ20] 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 characterises 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 approaches are in[SG; DC14].

The various methods used for dimensionality reduction include principal component analysis, linear discriminant analysis, generalised discriminant analysis have used in distinct ways to select features in a lower dimension without losing much amount of information[YA12; Ble; SS09; BA00]. [LS98] has shown a feature transformation approach via use of neural network likewise clustering and evolutionary algorithms also been applicable as feature selection process[Yan+14; KA13; ZLY08]. A summarised survey on ODT and existing feature selection strategies are available in [BU95].

3 Methods

In this section, we are getting about our 3 algorithms, how they are induced and what strategy we have applied. We use an algorithmic form for each to understand more.

3.1 CART

A famous binary splitting decision tree for classification and regression. It tries to separate two branches in equal purity and balance. The balance is the left branch and right branch having the same or similar number of examples. It measures purity with the variation of Gini coefficient. Gini impurity function is used to measure the goodness of split for each node. It learns to partition based on the single feature value. A greedy recursive approach is used to build a full-grown tree. In the beginning, the whole training set $\mathcal{D}_{tr}$ is considered at the root. The feature values are preferred to be categorical, if values are continuous they are discretized before building the model. We have selected a scikit-learn library to use the decision tree algorithm [skl20c]. This algorithm will be our base comparison for the other two ODT. For our experiment we have selected two hyperparameters they are;

1. **min_samples_leaf:** A threshold point where the algorithm stops growing trees and returns probability for each class by being as a leaf node. Afterwards, we choose the dominated class of probability, with an equal instance we select a random approach.

2. **criterion**: Since we want to take the CART algorithm for UDT and scikit-learn offers two variations of impurity measurement, we stick with Gini impurity. We proposed a comparison on the full-grown tree.

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 [Bre+84]. 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 split into two subsets, one for each branch extending from the root node. Normally a 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 instances) of the pure node. This splitting process continues until all nodes have a pure instance from one class or some given condition is fulfilled as an example maximum depth of tree or minimum sample leaf. If a leaf node contains more than one class label due to early stop, then majority voting is used to set class. Tree deduction process follows the same approach as given in Section 1.1.1 in steps of top-down DT deduction.

Algorithm 1 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 branch.
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 a fixed 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 features is a NP-Complete problem[LS98]. Since the problem is very hard, without using optimization or feature engineering techniques we cannot guarantee a global optimal tree. Hence for

our experiment, we choose to stick with a set of n features and evaluate based on that result.

Algorithm 2 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 works. For a recall, logistic regression is a linear function which is used for classification. With the help of coefficients, it creates a hyperplane. The sigmoid function gives results based on a default of 0.5 probability threshold value which can be configurable based on the requirement. In our experiment, scikit learn based LR model is used [skl20a]. 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 follow the same process again.

At first, n combination of features is generated then we train the LR model only using that n subset of features. We use the model coefficient and intercept as the decision boundary. Afterwards, we separate based on the prediction function of that model. We apply a predict method on the same feature and get output. Predict method is an inbuilt function.. 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 category or meets some stopping criteria. In the case where n is not enough to map linear combinations, then we stop growing trees. In the future update will discuss

updating the RFS index by keeping the same size of n to overcome a large number of the impure partitions. This process lets our algorithm not stick with the worse split but somewhat gives a split. If there is no improvement by measuring entropy=0, then we make that branch as a leaf node and continue to rest.

Algorithm 3 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*) ▷ true & false branches, ≥ 0
    BUILD_TREE(D_i) ▷ 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, $\mathcal{D}_{tr}$) ▷ epochs and subset from RFS
    LR(e).fit($\mathcal{D}_{tr}$)
    return θ, predict($\mathcal{D}_{tr}$.X)
end procedure

Algorithm 4 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 instances. Prediction process takes a tree and instance tuples, then the process checks for the leaf node to return the class as predicted output. We check if the model is Leaf, then we show its label. If the Leaf contains over one data point of classes, then based on probability we

select a label. If all classes data points are equal, we selected randomly. When the case model is not Leaf, then we do matrix calculation by selecting the same feature index from model and new predicting instance. If the result is less than 0 or negative value we go for false branch else true branch. We continue recursion with each selected branch, at some point we find a leaf node where prediction value is stored. 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 the RANSAC algorithm. Our proposed implementation uses RANSAC's iterative random process to find the best hyperplane from the given number of iterations. Entropy has been applied to measure the goodness of the hyperplane and splitting process follows. Since RANSAC is a linear model to apply in classification problems internal changes is required. We need to modify in such a way that θ values can be utilised to create hyperplane for splitting $\mathcal{D}$. We are using [Ngh12] approach for creating a line in DT. A linear regression model also available in [skl20b]. It creates the oblique line through two random sample points from each class. It creates a line that goes directly between two points by finding the middle point which works as an 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 trees in 3 stages.

1. RFS: Process of getting n different type of features randomly is the 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 be homogeneous. Afterwards, we pick one random data from each class then we create a line using the basic equation of a line. Finally, we test the goodness split of the hyperplane for the whole dataset using entropy. 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 5 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: 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
    θ, *pos_side, neg_side* = GET_BEST(e, RFS(n_fet, $\mathcal{D}_{tr}$), PARTITION($\mathcal{D}_{tr}$))
    D1, D2 = PARTITION($\mathcal{D}_{tr}$, *pos_side, neg_side*)
    BUILD_TREE(D_i) ▷ 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, $\mathcal{D}_{tr}$ *pos, neg*) ▷ pos, neg : data points respect to class
    for e **do**
        p, n = random(*pos, neg*) ▷ get two random points from each class
        θ, G, *pos_pred, neg_pred* = SPLIT_DATA($\mathcal{D}_{tr}$, *pos, neg, n, p*)
        if G > previous G **then** update θ, G, *pos_pred, neg_pred*
        end if
    end for
    return θ, *pos_pred, neg_pred*
end procedure

Finding decision boundary θ

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

4 Results

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

4.1 Experimental setup

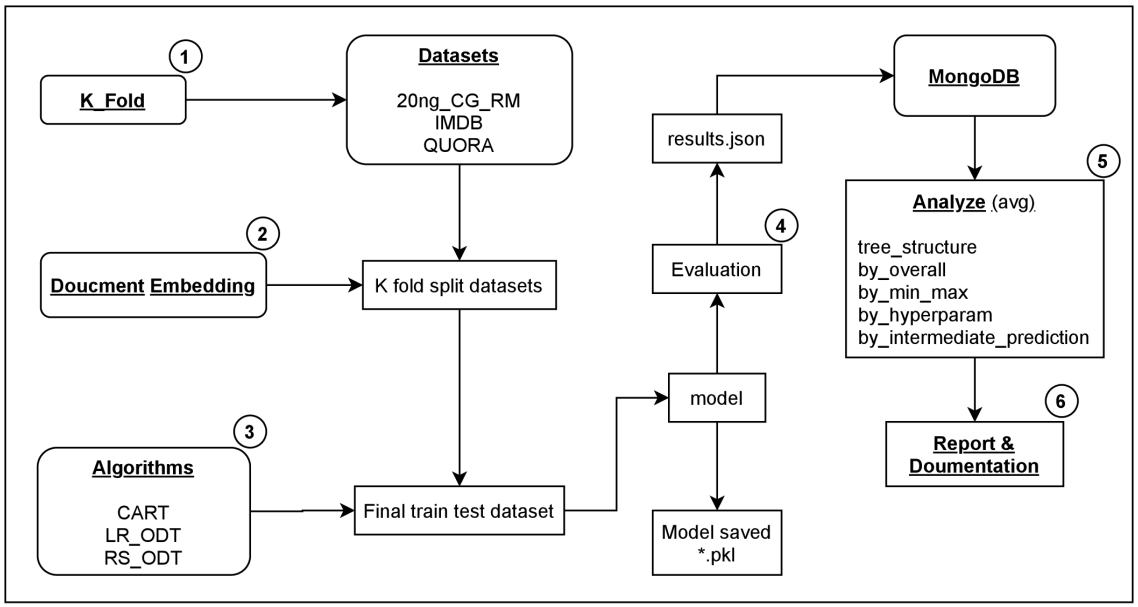


Figure 4.1: Flow char of our experiment process

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 the training and testing part. After the training process completes we test the performance in the test dataset. With comparison, both sets give a clear evaluation of our model.

However, splitting only one set of train and test 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 results than another, 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.

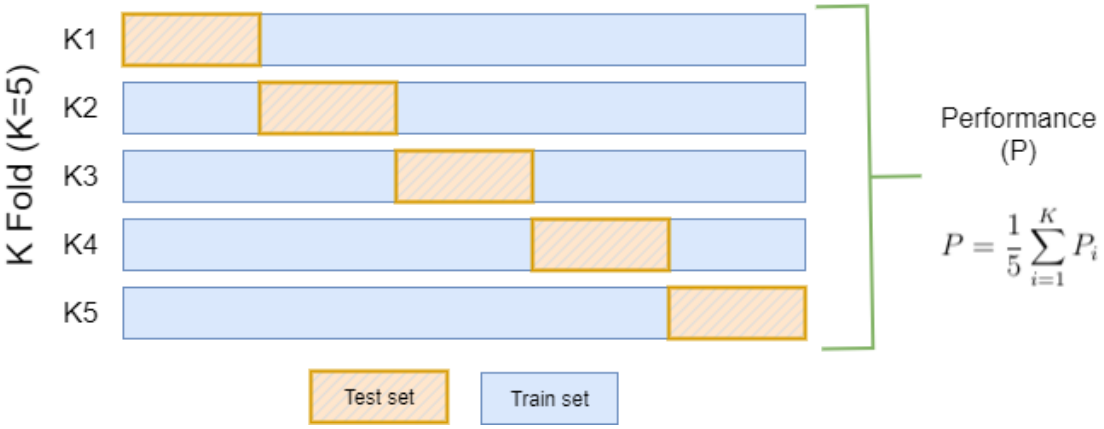


Figure 4.2: K fold cross-validation, when K = 5

Above figure shows the K_fold cross-validation process and the average formula. In this process, whole data are being utilised 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 each train set. The model learns via the Doc2Vec algorithm taken from[20d] 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 started the evaluation process next.

   a) Sklearn based DT (CART)

   b) LR induced DT (LR_ODT)

   c) Random sample split 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, if we give depth 1 then the prediction happens only at the root decision node where if give depth 19 then the model is considered as till the tree has 19th max depth. If in the case where some model has 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 analyse 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 six main information to extraction.

   a) Time complexity: Here, the general run time complexity of each algorithm is represented by BigO notation for both learning and prediction.

b) Tree structure: It shows the structures of the tree generation from given algorithms. The tree is categorised into 5 distinct types based on their inner nodes.

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

d) Min Max: This shows maximum and minimum accuracy results, including evaluation matrix, feature size, depth, inner nodes count, branch sizes and training time.

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

f) 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 domains 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 been calculated after basic text cleaning processes. It includes a 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 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 common words are always articles, prepositions and verbs. In the other part least common words don't form any meaning. Therefore, exploiting the 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

It explains how we created doc2vec representation for each dataset as well as how we store final representation. It includes 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 creating all embeddings. The datasets have a different variety of text file and comma-separated value files (CSV). First, we transform various formats in one CSV format by manipulating their original structure. Then we follow the given steps in Doc2Vec's official site to make feature embeddings. Below steps contain the necessary process of creating Doc2Vec embeddings.

1. The feature simple_preprocess in Gensim library creates tokens of words per document. At first, the process takes one document and tokenized then with a unique id same document is tagged. The tagged document is later used as a 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 is used only for train documents.

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 a given higher dimension as explained in chapter 1.1.2. Every iteration of the training process minimises 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 an infer_vector. It uses the learned model and tokenized corpus of train and test data.

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

4.4 Experimental Result

4.4.1 General time complexity

Runtime analysis of an algorithm depends on many factors. On a large scale, only particular operations make a drastic difference. Therefore, to know general algorithm performance we use BigO notation. Order of growth of algorithm performance is measured 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 measure the number of operations it takes to complete. We keep the assumptions of

1. A full-grown tree with fixed finite hyperparameter sizes.
2. Considering only the worst-case scenario using BigO but not BigΩ or Bigθ.
3. n is a number of training examples, m is a number of features and e is a number of iterations.
4. Global optimal solution is NP-hard problem.

Time complexity of CART

[SLN18] has explained the complexity of the decision tree, including CART. Even the conducted experiment supports 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 analyse 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 of 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 the opposite half of the tree branch will not be considered for each depth. While matching value takes constant time. Therefore, the estimated complexity for CART becomes $O(\log n)$.

Time complexity of LR_ODT

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

1. The estimated time for creating a subset of data based on the random selection of features is $O(n)$.

2. LR does matrix multiplication, inversion to find θ, regardless of various solver the time complexity becomes $O(m^2 n)$ [Iye]. Our random feature selection affects computation by a constant factor of time, hence final complexity would become $O(m^2 n)$.

3. Similarly to analyze recursive calls of the algorithm for each subset it takes $O(n \log n)$.

4. Hence sum of all operations and final complexity of written as

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

5. The recursive search takes $O(\log n)$ while every decision node multiply each weight (θ) by the value of feature and add bias hence $O(m + 1)$ therefore final estimated prediction time will be $O(m \log n)$.

Time complexity of RS_ODT

Similarly, this approach again follows the same procedure. The only difference is while creating a linear decision boundary. RS_ODT takes only one observation 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 $O(m^2 n)$ 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:

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

5. Again the recursive binary tree search takes $O(\log n)$ when matrix multiplication on θ with feature becomes $O(m \log n)$ which is the final complexity.

4.4.2 Model structure comparison

Let's understand the notion of tree type or structure. A type zero tree consists of a single rule at the root node only and consists of leaf nodes as output. A tree is perfectly balanced when both branches from the root node have equal inner nodes regardless of inner branch structures, as an example if the tree is unbalanced after n depth we do not consider those cases. We assumed an almost balanced is measured when both sides have a difference of less than 10 percent. Left and right trees skewed are measures when branches have more inner nodes in respective sides. The accuracy levels are categorised in five different groups of 5% of differences from less than 70 up to greater than 85.

Data \ *Algo*	Tree	20ng_CG_RM		IMDB		QUORA	
Accuracy	type	LR_ODT	RS_ODT	LR_ODT	RS_ODT	LR_ODT	RS_ODT
-70	zero	0	0	1	0	0	0
	perfect	3	0	7	0	0	0
	almost	20	0	95	0	0	0
	left	43	0	206	0	0	0
	right	45	0	206	0	0	0
70-75	zero	0	0	0	0	0	0
	perfect	9	0	4	0	0	0
	almost	41	0	92	10	0	0
	left	110	2	113	12	0	0
	right	111	1	546	21	0	0
75-80	zero	1	0	0	0	203	0
	perfect	18	0	6	1	0	0
	almost	53	6	182	205	0	0
	left	109	60	127	161	10	0
	right	182	24	120	181	24	0
80-85	zero	1	0	0	0	176	0
	perfect	14	2	8	3	16	0
	almost	67	20	332	329	10	3
	left	139	280	148	313	134	67
	right	253	97	305	427	1014	275
85+	zero	0	0	0	0	0	0
	perfect	206	40	60	3	0	2
	almost	145	179	249	420	2	233
	left	527	1809	54	491	2	423
	right	1028	605	264	548	1534	2122

Table 4.4: Tree structure analysis with train accuracy range

The above table gives an answers about what kind of tree was formed based on the level of training accuracy. We observe that LR_ODT is able to create range tree structures. Most of the trees are right skewed than left, and it also creates a comparably high amount of perfect and almost perfect trees. On the other hand, RS_ODT has not created a single zero depth tree. Similarly, perfect trees are also less compared to others. Interestingly, almost the perfect tree is around in the same range on both algorithms where there is no significant structure in skewness. The following table shows an overall combined result.

Tree type Dataset	Algo	Zero	Perfect	Almost	Left	Right
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 4.5: Overall tree structure analysis

The above table shows a combined look of the previous table. Therefore, following insights are equivalent to above. We observed RS_ODT doesn't have any zero-depth which confirms the approach performs a weak learning procedure. In the opposite LR_ODT creates a much more perfect balance tree than others. Both trees have a similar number of the tree when branch difference is less than 10 percent. We saw LR_ODT has generated many right skewed trees where RS_ODT was unpredictable. Finally, the result shows that the linear model is suitable at D2V representation because all the training is giving at least some result.

4.4.3 Overall comparison

Various factors heavily affected our experiment, such as hyperparameters and size of D2V. CART has only 25 total runs per fold and rest have each 625 runs due to its additional hyperparameters (epochs and combination of features). All the results are aim to compare CART vs LR_ODT vs RS_ODT. The below figure shows the average relationship between accuracy and depth concerning each dataset of all vector sizes.

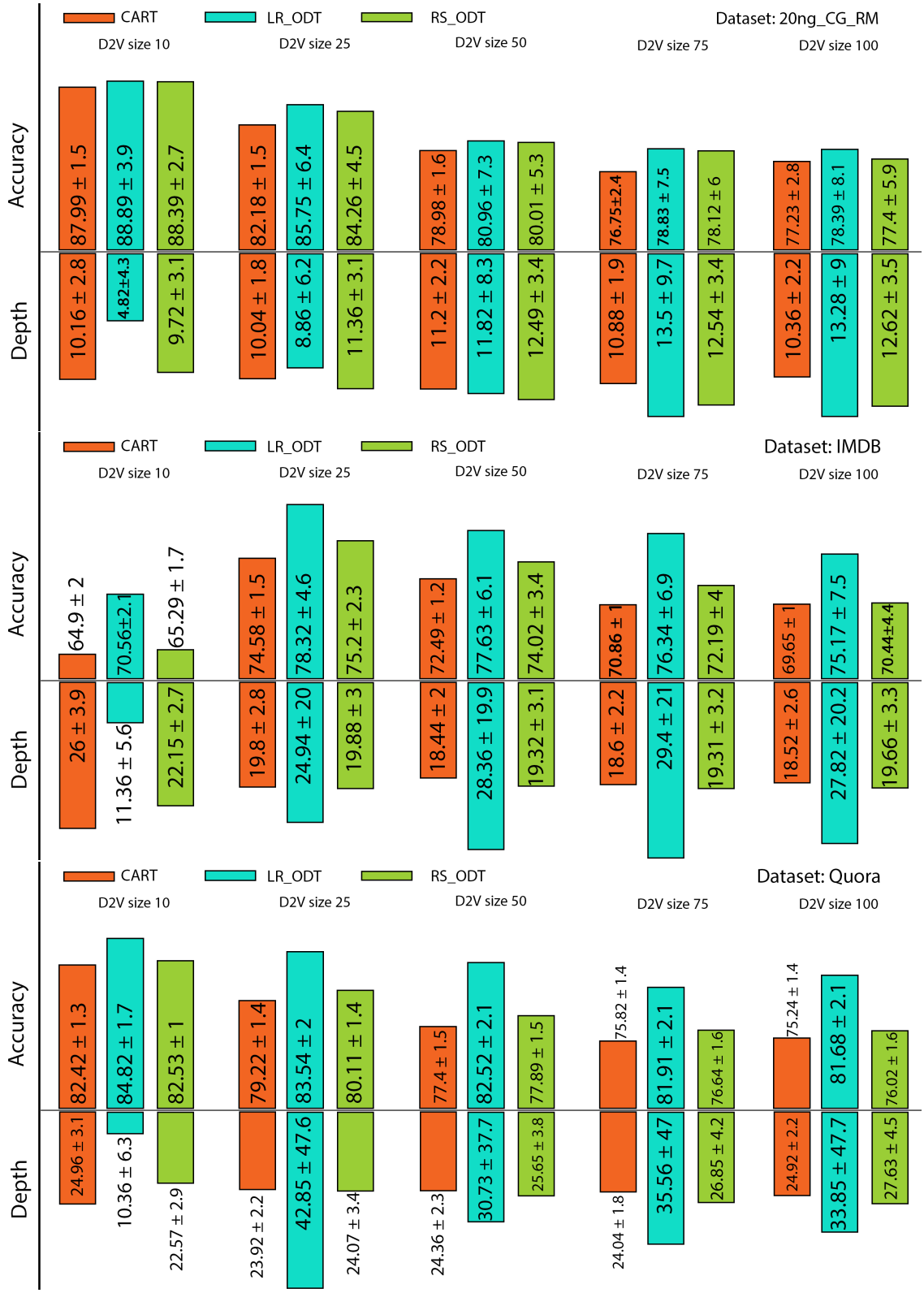


Figure 4.3: Overall average test accuracy and average depth for each algorithm on each datasets' embeddings

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 4.6: 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 a drastic difference in depth. For the smallest vector size 10 LR_ODT proved the best choice whereas for other vector sizes we compromise with depth.

2. Performance of RS_ODT is slightly above than CART for both accuracy and depth while still accuracy lacks behind LR_ODT.

3. If we look at standard deviation we can find that CART has pretty low compared to LR_ODT and slightly high to RS_ODT for both accuracy and depth.

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

5. We can see that the f1 of RS_ODT is great to other algorithms 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 predicted almost equal numbers of false positives and false negatives or algorithms are biased to someone class. It can also be possible since both datasets have around 50% of equal class distribution.

4.4.4 Min max comparison

Previously we analysed 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 a 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_	max	5	1k	10	93.15	91.75	94.18	92.95	5	5,8	0.35
	ODT	min	2	500	15	70.05	63.67	93.4	75.72	3	0,6	0.16
	RS_	max	20	100	15	93.65	91.84	95.24	93.51	16	19,1	2.14
	ODT	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_	max	20	100	5	94.16	92.35	95.77	94.03	6	7,5	3.25
	ODT	min	2	1k	15	65.48	63.2	74.11	68.22	12	48,0	0.5
	RS_	max	20	800	30	92.64	92.11	92.59	92.35	8	2,8	9.96
	ODT	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_	max	50	100	5	93.91	91.88	95.77	93.78	2	2,1	2.64
	ODT	min	2	800	10	60.81	60	78.95	68.18	6	0,11	0.18
	RS_	max	50	1k	15	90.36	89.12	91.01	90.05	15	5,18	18.37
	ODT	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_	max	75	100	5	91.37	87.08	96.3	91.46	2	2,2	5.64
	ODT	min	2	1k	30	60.05	64.77	54.55	59.22	10	17,14	0.4
	RS_	max	10	100	30	89.34	87.31	91.01	89.12	8	4,9	2.09
	ODT	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_	max	100	100	5	91.62	87.94	95.11	91.38	3	3,1	8.27
	ODT	min	2	100	15	58.38	55.81	63.49	59.41	9	13,11	0.37
	RS_	max	100	500	5	89.59	90	91.24	90.62	16	32,28	23.31
	ODT	min	2	800	5	60.15	60.1	60.41	60.25	18	21,92	26.49

Table 4.7: 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 4.8: 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 4.9: QUORA maximum and minimum test accuracy performance of each algorithms for all d2v sizes

Min Max evaluation, findings from Table 4.7, 4.8 and 4.9

1. For all dataset ODTs approach has higher accuracy than CART where LR_ODT obtained best for accuracy. In contrast, 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 the representation of the D2V model is in a somewhat linear form of higher dimension as proof most of the maximum performance is achieved when we take 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 at least min leaf point where other larger datasets have a 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 the other two datasets high accuracy is 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 an equal relation for increment and decrement.

6. Structure of trees doesn't provide much information alone, yet most of the time high accuracy is gained when the tree is almost balanced in ODTs. The same effect also applied for training time. As explained 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 analyse 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 4.10: 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 4.11: 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 4.12: 20ng min leaf point average test performance

n_feat	d2v	LR_ODT accuracy	LR_ODT f1	LR_ODT depth	RS_ODT accuracy	RS_ODT f1	RS_ODT 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 4.13: 20ng n feature wise performance

Hyperparameters wise evaluation of 20ng_CG_RM dataset, Findings from Table (4.10, 4.11, 4.12, 4.13)

1. In Table 4.10 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 4.11 evaluation based on 5 folds, Most of the times performance of RS_ODT is slightly higher than CART even though the number of training sessions is 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 models has the same effect in Table 4.12. There has been a small increment when choosing least min point for ODT where selecting the largest value reduces the depth.

4. In Table 4.13 evaluation by the number of feature combinations, 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 matrices.

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 4.14: imdb ephocs wise performance

k fold	algo. d2v	CART accuracy	CART f1	CART max depth	LR_ODT accuracy	LR_ODT f1	LR_ODT max depth	RS_ODT accuracy	RS_ODT f1	RS_ODT 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 4.15: 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 4.16: 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 4.17: imdb average test performance by n features

Hyperparameters wise evaluation of IMDB dataset, Findings from Table (4.14, 4.15, 4.16, 4.17)

1. Table 4.14 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 4.15 shows the similar result as Table 4.11, 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 models shown in Table 4.16. The low point has high depth and vice versa whereas low min point doesn't show high accuracy.

4. Feature selection results in Table 4.17 shows RS_ODT has performed slightly better performance when selecting only 2 features but the size of the tree is also high. For the rest of the feature selection, LR_ODT performs better keeping 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 4.18: 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 4.19: quora average test performance by k fold

min leaf	algo d2v	CART accuracy	CART f1	CART max depth	LR_ODT accuracy	LR_ODT f1	LR_ODT max depth	RS_ODT accuracy	RS_ODT f1	RS_ODT 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 4.20: quora average test performance by min leaf point

n_feat	d2v	LR_ODT accuracy	LR_ODT f1	LR_ODT depth	RS_ODT accuracy	RS_ODT f1	RS_ODT 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 4.21: quora average test performance by n feature

Hyperparameters wise evaluation of QUORA dataset, Findings from Table (4.18, 4.19, 4.20, 4.21)

1. In Table 4.18 shows the comparison based on epochs, for all epochs, LR_ODT performs high accuracy and depth are almost the same where RS_ODT has a higher f1 score than LR_ODT.

2. Table 4.19 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 is 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 a high f1 score.

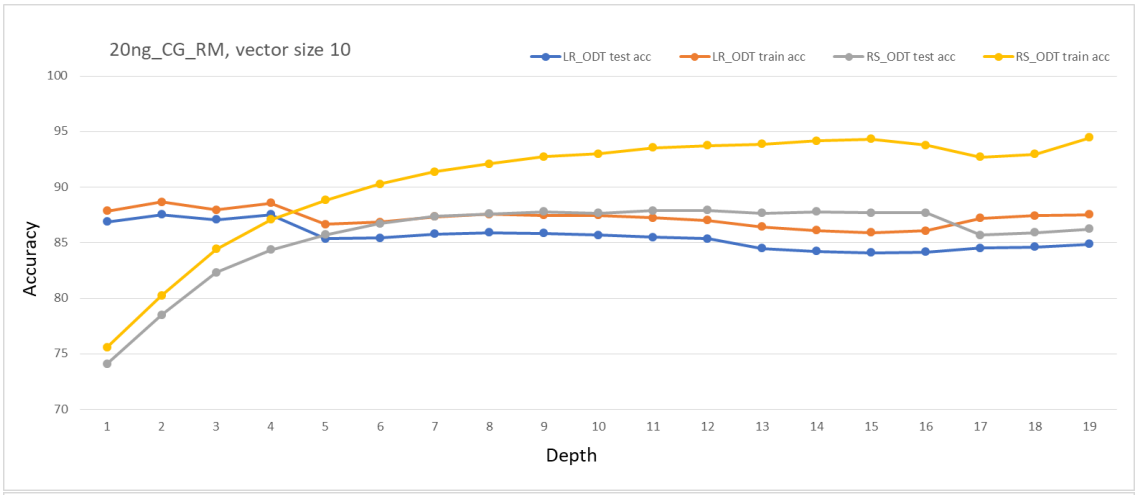
3. Again, Table 4.20 shows similar results as in Table 4.18 when analyzing with several min leaf points. CART and RS_ODT show the same range of f1 score while LR_ODT lacks in this evaluation.

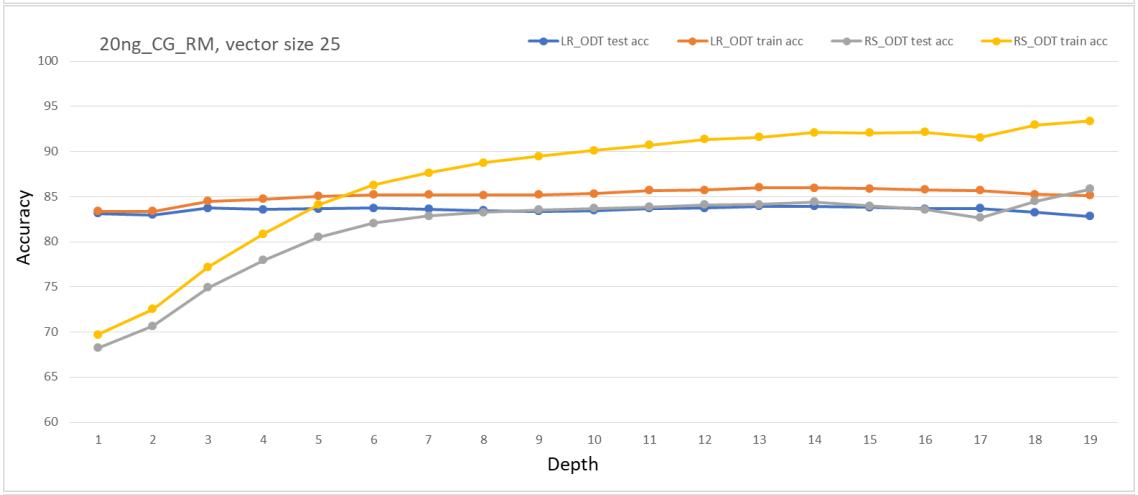
4. While comparing by various combinations of features in Table 4.21 shows the same effect as epochs wise comparison in Table 4.17. LR_ODT has the upper hand on accuracy and RS_ODT has a higher f1 score. Interestingly, LR_ODT has short depths.

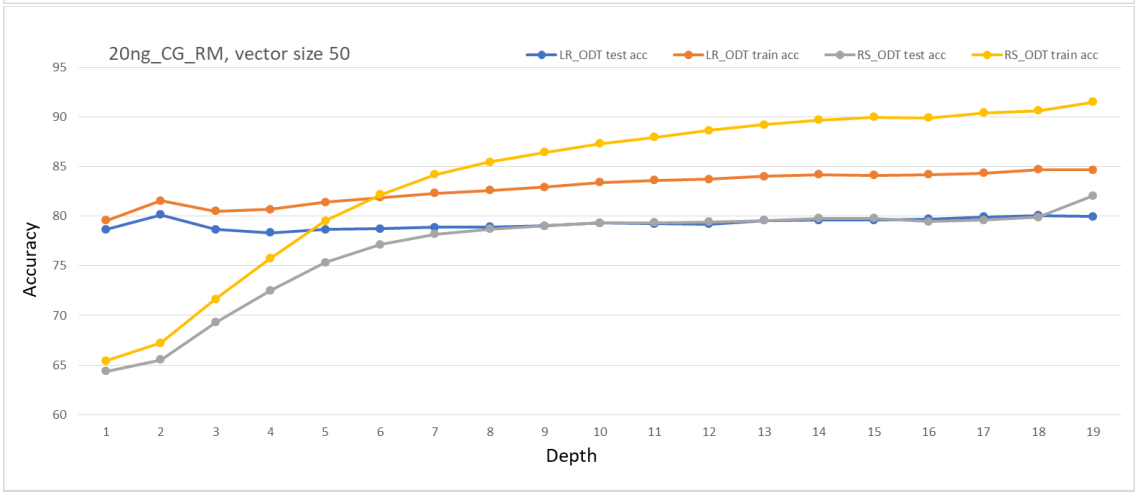
4.4.6 Intermediate accuracy vs depth comparison

In this section, we analyse the intermediate performance of our algorithms. We plot training and testing results of both accuracy and depth to analyze the intermediate behaviour of algorithms. Since CART has been taken as granted from scikit learn it becomes hard to modify internal processes, 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 the 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. On the other hand, it's interesting to observe the RS_ODT learning process because the only single instance is considered creating a decision boundary. The following line chart shows depth and accuracy score based 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

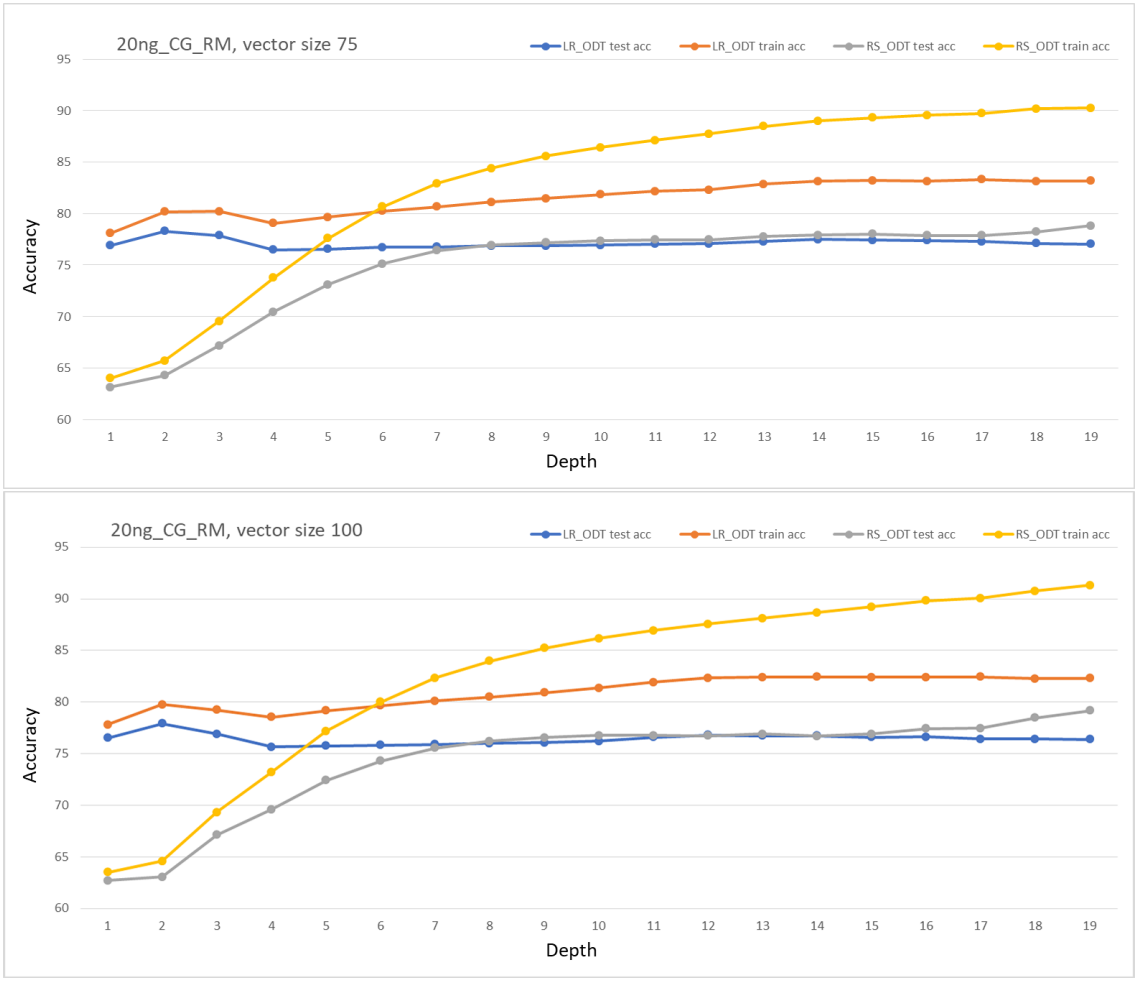
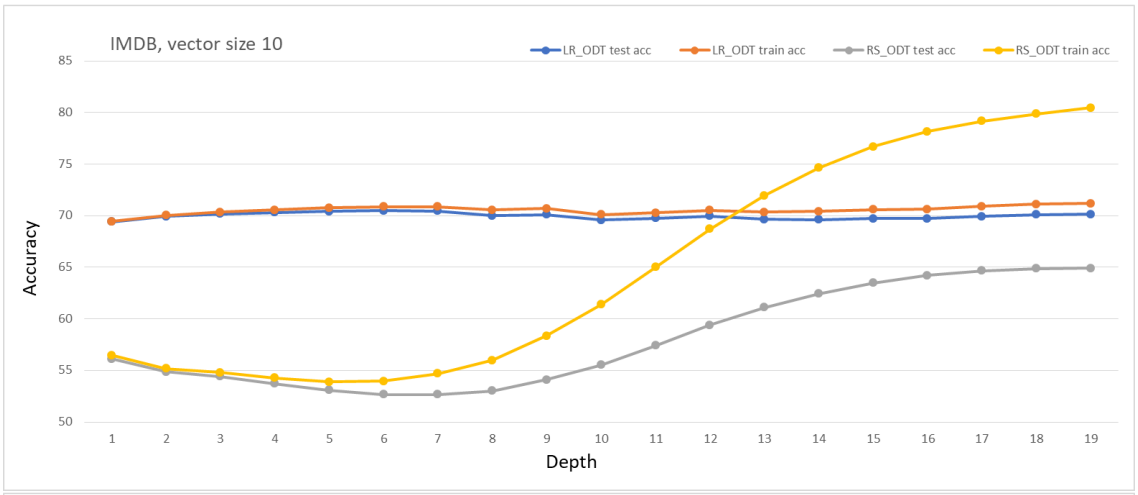
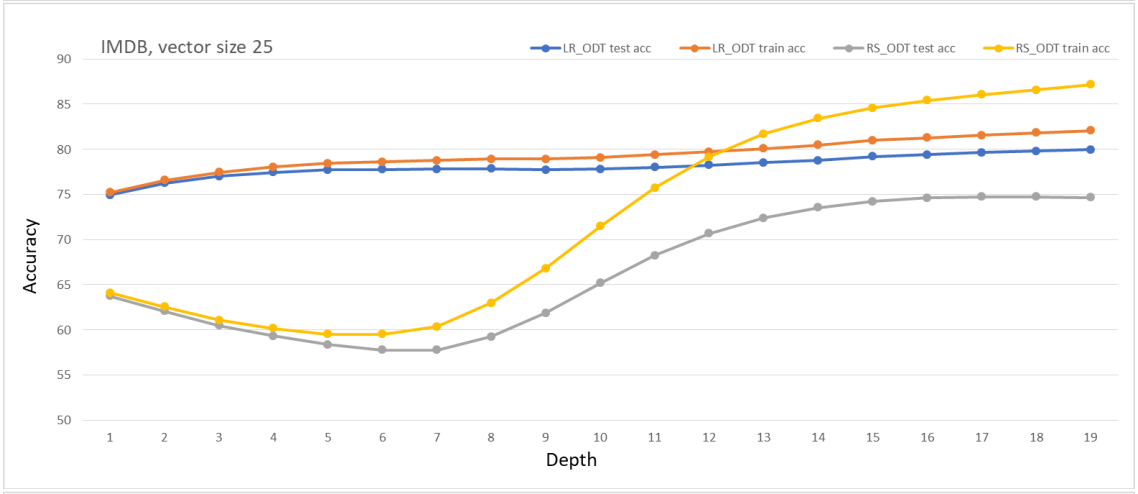


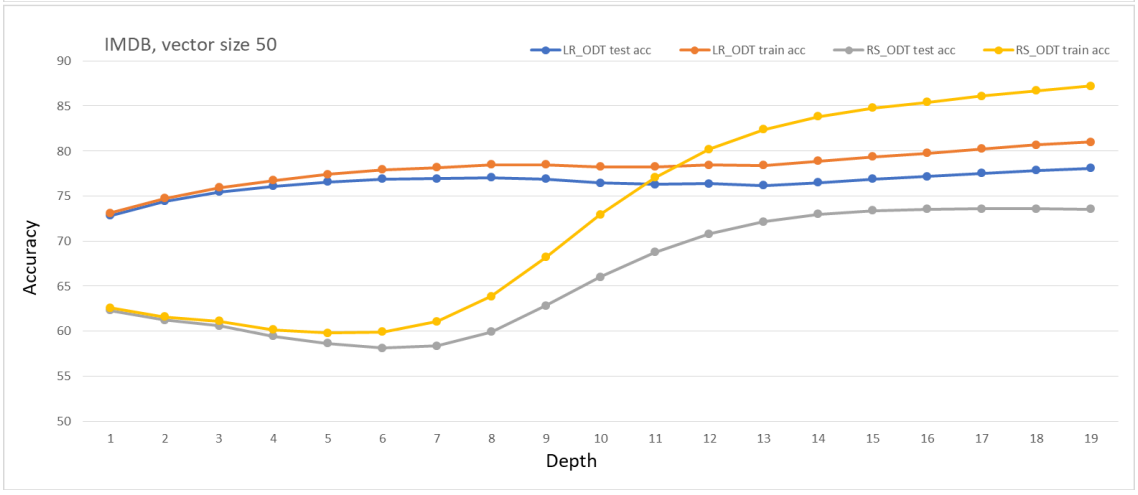
Figure 4.4: 20ng intermediate model evaluation; comparison of depth and corresponding accuracy for each D2V sizes

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, the rest of the change happens at the end. Since the result is an accumulated average result, consequently, we can see slight drops in accuracy even if depth increases for both train and test. It shows around 5 percent of the difference in LR_ODT.

The results show for all vector sizes RS_ODT has performed well in the 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

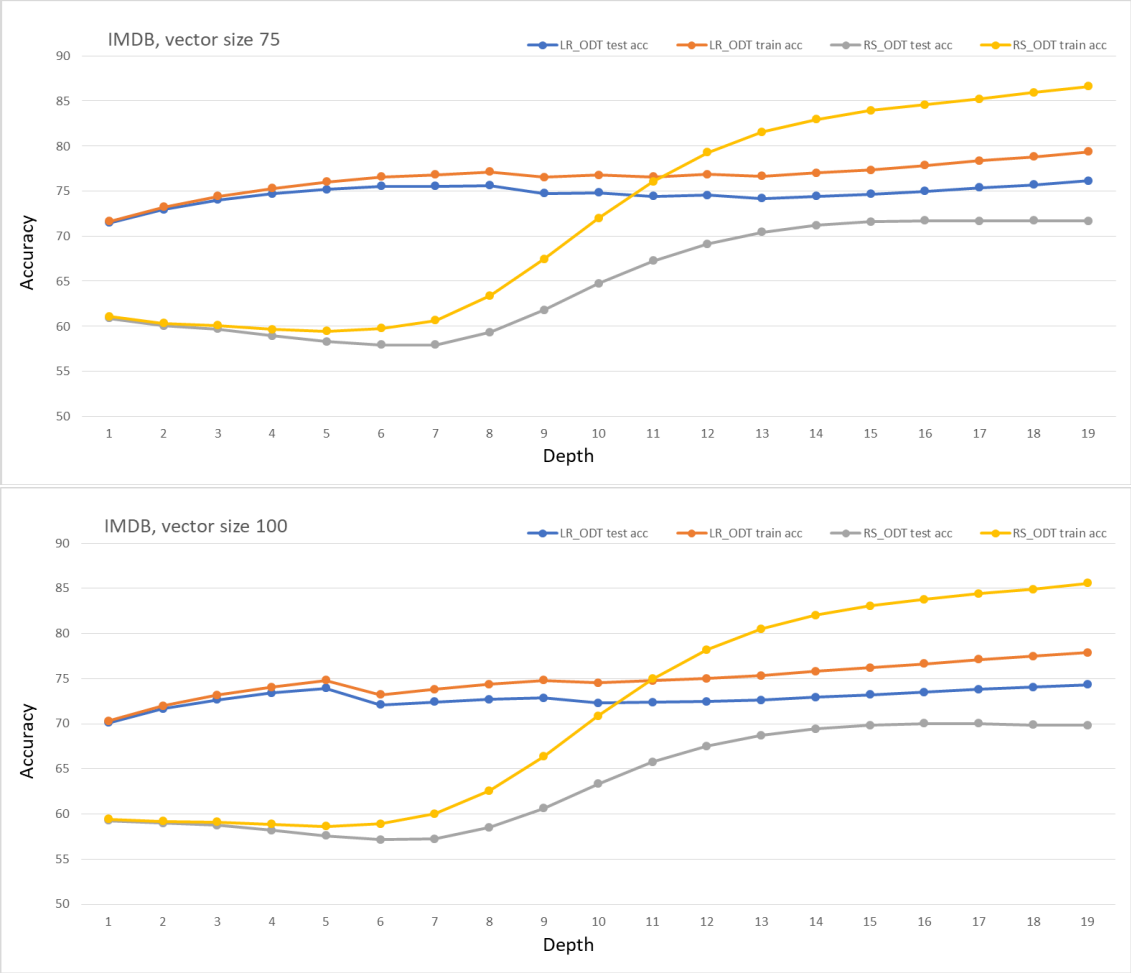
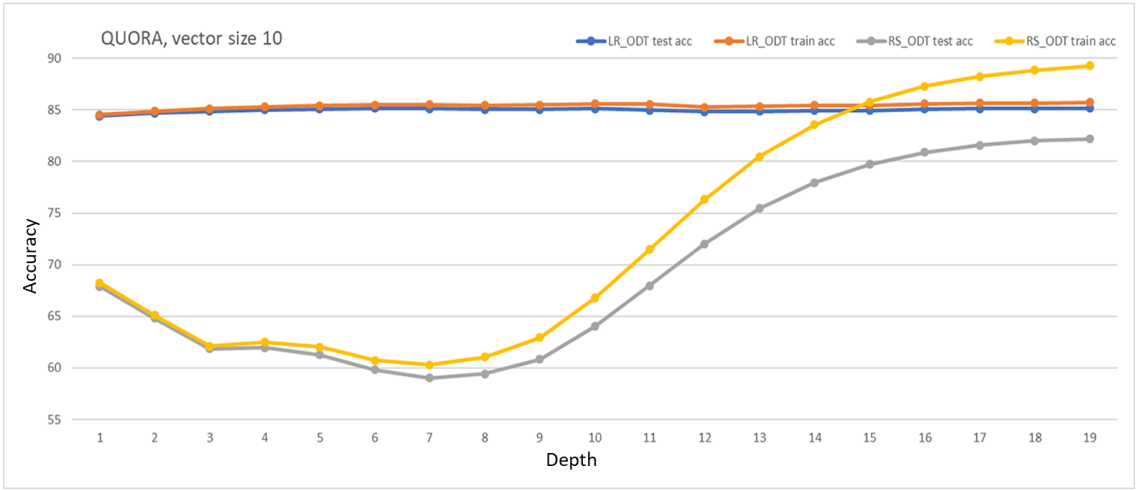
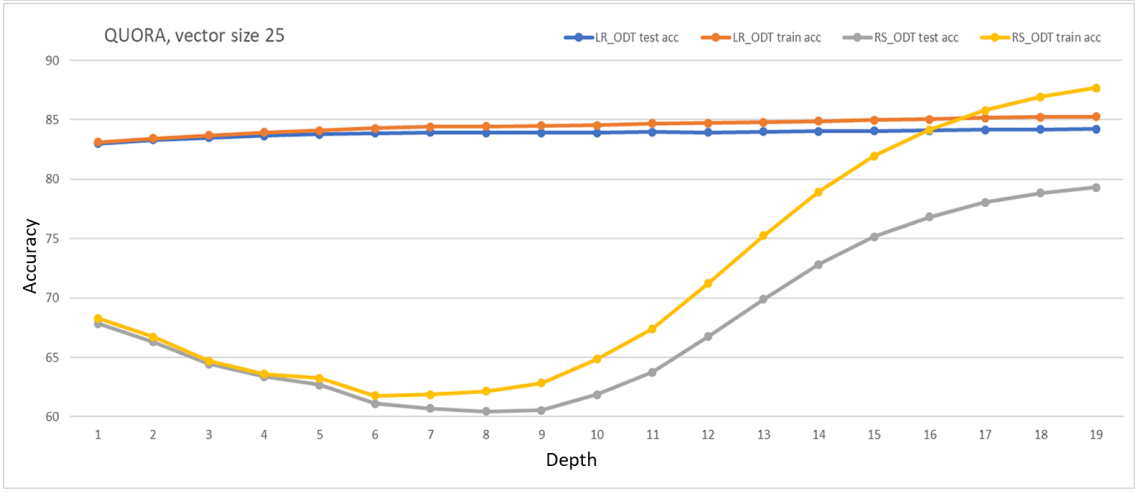


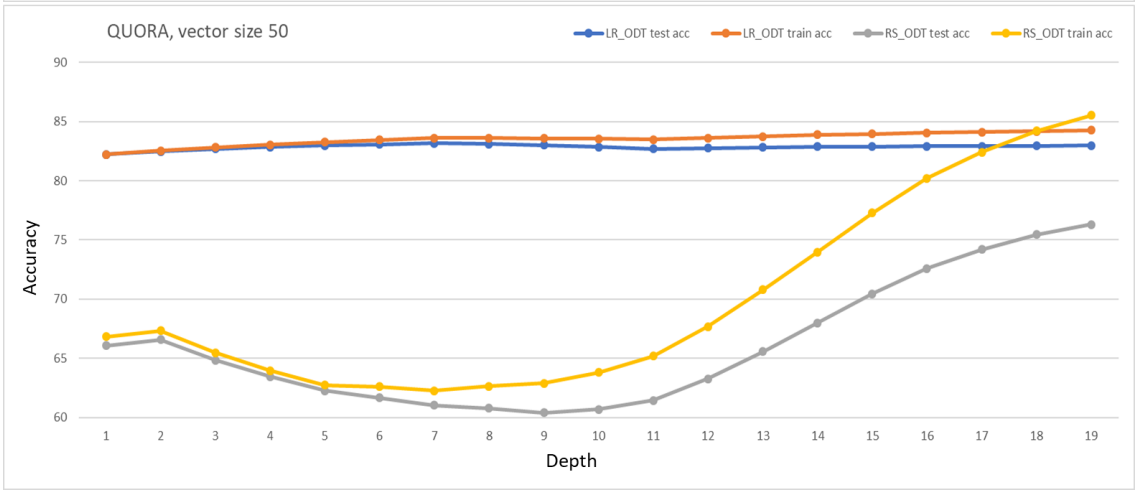
Figure 4.5: imdb intermediate model evaluation; comparison of depth and corresponding accuracy for each D2V sizes.

For IMDB LR_ODT shows 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

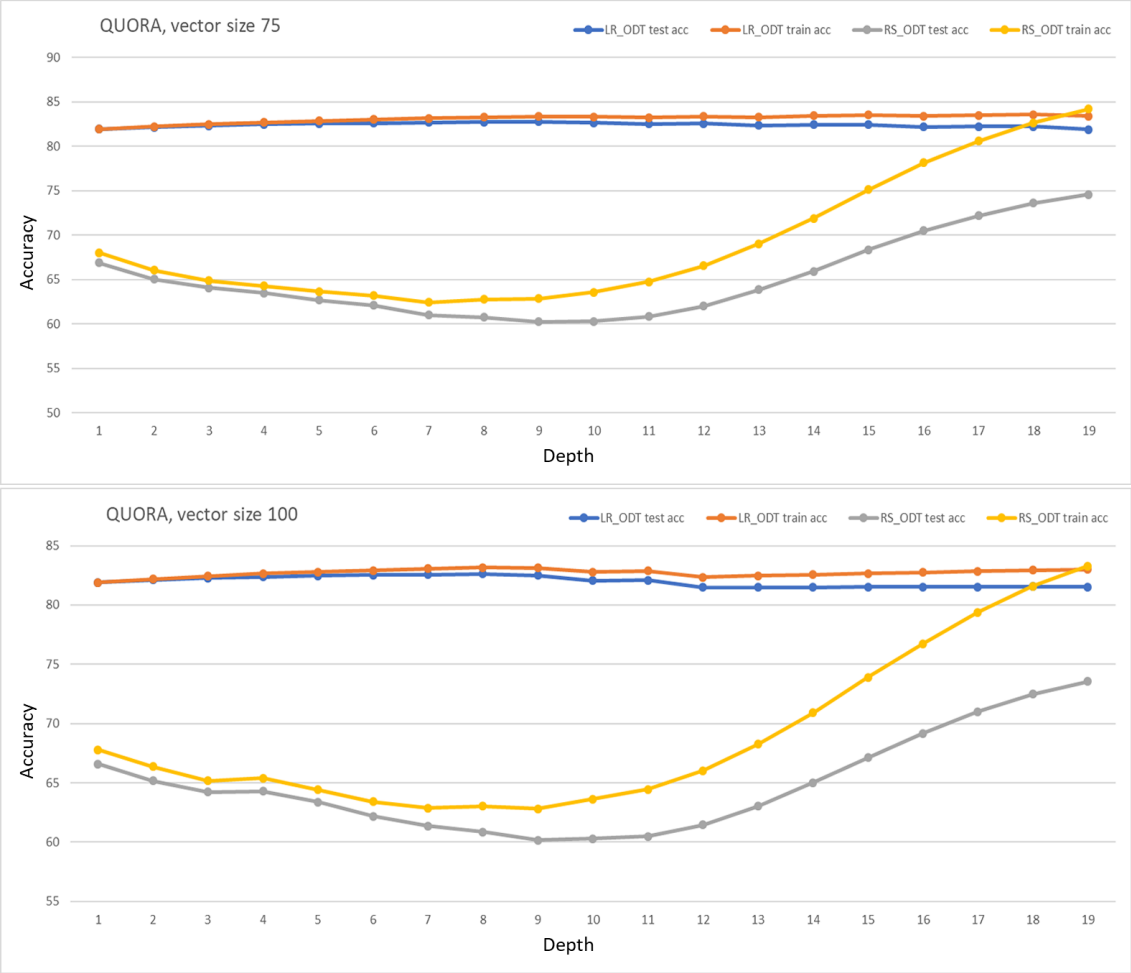


Figure 4.6: quora intermediate model evaluation; comparison of depth and corresponding accuracy for each D2V sizes.

In the QUORA dataset, LR_ODT shows a 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 a pace to uphill. This behaviour could form those models which have low accuracy in short depths.

5 Discussion

In this section, we will discuss our findings afterwards, point out 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. Most of the time pattern of each algorithm was alike. Combination of LR was more promising than random picking. Here we discussed experimental thoughts on topic comparison.

1. **Time complexity**
   We measured the runtime complexity of the train and test for each algorithm. Our ODTs take much more time to learn the model and predict than CART. At every inner node, a linear computation has to be performed by ODTs at both learnings and the predicted process. The length of time automatically increased by factor of that linear algorithm. There is no optimised way of fitting θ value in RS_ODT, hence, the process becomes faster by a constant factor.

2. **Tree structure**
   The shape of the tree doesn't give much information about the learning process yet LR fitting and random sample splitting approaches clearly indicate why LR_ODT has higher perfect balance than another. Additionally, RS_ODT has no zero tree meaning random processes actually find at least local optimal splits. Whereas sometimes LR cannot split the regions due to its feature representation, an example is available in the appendix section.

3. **Overall results**
   Here, we saw LR based approach outperforms others inaccuracy but also increase the depth. When classifiers find a very small data point to separate, it creates more depth. Post pruning or early stopping criteria could solve the problem. On 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 misclassifications in both classes, we confirm the issue arises 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 a different perspective 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 percent on both low and high accuracy where RS_ODT comes second. We observed major benefits received by 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 analyse based on a small set of lists.

5. **Hyperparameters findings**
   In the process of hyperparameters wise comparison, we more epochs give better accuracy in less vector size and hence opposite relation. Similarly, the 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 every fold 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 points help to reduce overfitting problems in the train phase. Our results show minimum value creates high depth which doesn't lead to high accuracy whereas most of the less depth which has high min leaf value show good accuracy. On the other hand higher the vector size shows an increment in depth.

   Selection of features was a crucial process of our experiment. While our experiment

doesn't provide any self-optimisation or feature pre-engineering process for taking the best set of features we choose a random method. In the random process, a fixed number of features is taken. We mostly want to explore the effect when we select the least number of features to all features. In our observation, we encounter less number of feature combinations that gives more accuracy than more or all combinations. By saying this trend is also affected by the size of the vector in the opposite direction.

6. **Intermediate observation**
   To examine model performance during full growth of the 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 is a powerful classification model, it tries to find the best split in the beginning. At the first depth, most of the data is 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 points could be hard to separate via a linear line.

   On 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 where we find a comparably large tree. Unlike logistic regression, it can be used for any shape of data. Due to the same reason training performance is better than testing performance. From the above observation, we find some aspects to improve and some limitations.

5.2 Limitation

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

- **Limitation on feature importance selection**
  Finding an optimal split requires a large number of iterations using brute force steps. The feature is either selected randomly or all considered. The selection

process becomes the most resource consuming part if partial combinations are used.

- **Limitation of LR_ODT approach**
  Classifier merging technique automatically increases time and storage. On the other hand, the merging classifier has its own hyperparameters to tune. Some of the hyperparameters of scikit learn based LR are max_iter, solver and class weight. Since it's a linear model, it performs best when feature space is linearly separable. After n splits, a subset of impure data could form in any shape in which case the model performs worse and keeps increasing its depth with any accuracy gain (Appendix Figure A.2).

- **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 right number of iterations that gives the best hyperplane. When a number of instances are extremely high, then smaller iteration couldn't split best.

5.3 Further Improvement

This section sparks some of the areas which can be improved as future work. Regardless of other problem domains, embedding techniques, tree induction approach, improvement in interpretability and regression problem. Our thesis raises the following unsolved questions. These questions 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 the text domain yet, we could perform the known technique to explore the benefit. Our experiment doesn't talk about the feature engineering process, but creating a valuable new feature and transformations concerning algorithmic suitable form is always beneficial. Creating numeric attributes like a specific word based count, sentiment probability, merging or experimenting with

other embedding techniques could be interesting to an observer in our given approach.

- **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 trees based 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. On 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. Embedding other linear classifiers like support vector machines to find the best hyperplane may increase the performance and also reduce the depth. Since logistic regression is a robust linear classifier but performs poorly 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 a better approximation in the training phase, but algorithm complexity also increases. Finally, a combined approach of artificial neural networks can discover a new area of research.

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. Final review of results, their limitation and further improvement expressed in Chapter 5. Here, we summarise the outcome of this thesis briefly.

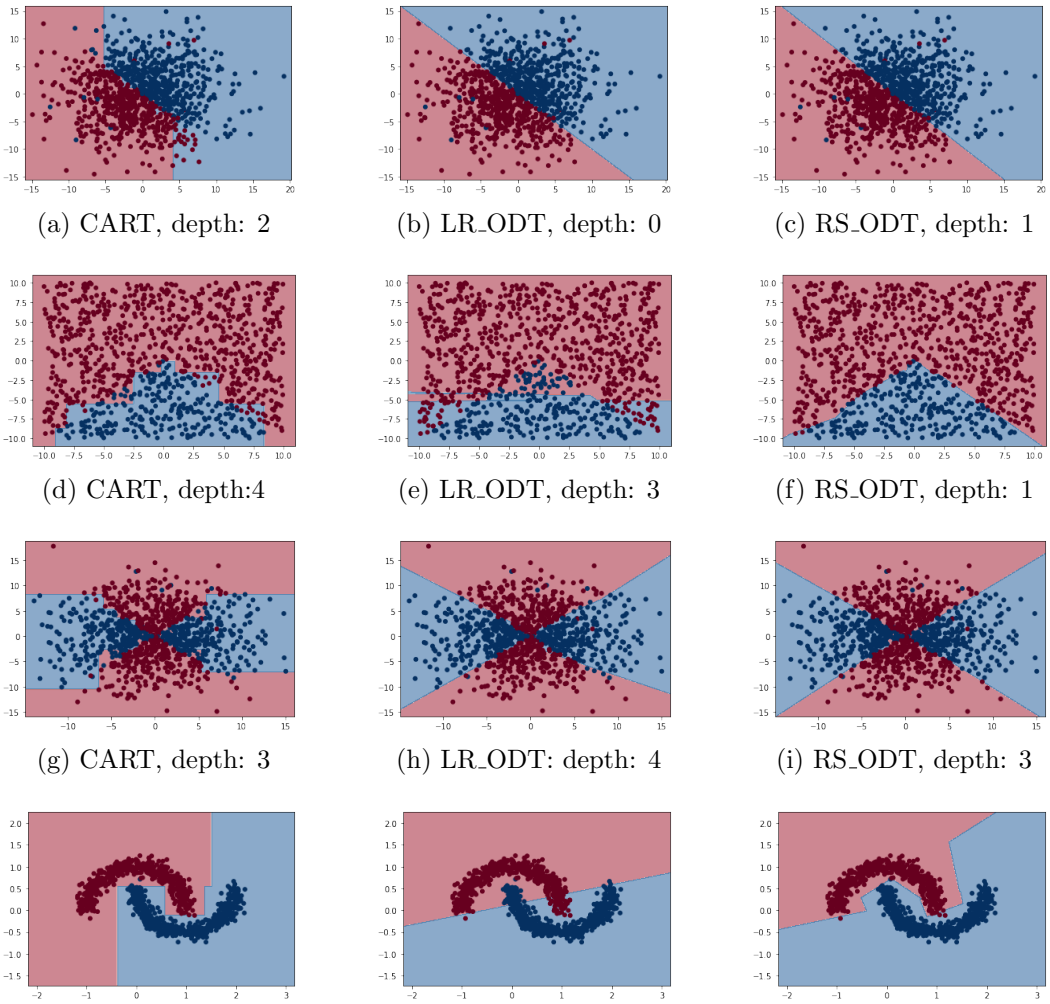
In this thesis, we performed two methods for finding oblique splits in the decision tree. One method was known as a LR classifier and the second was a random weak approximation approach. The experiment conducted in text classification focusing on the binary problem. The text data represented via document 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 features where identifying best sets are computationally expensive than searching as CART. Our second approach of RS_ODT mostly suffers single sample splits but LR_ODT utilises all feature values, yet the feature selection problem remains. A linear model's hyperparameters parameters increase run time. The overfitting problem can reduce by inducing a linear model but it highly depends on the behaviour of that model and we induced it. The possible improvements exist in both representation and learning processes.

We conclude that in text classification using D2V embedding oblique decision trees can perform better than a CART. We showed better in the sense of evaluation matrix and tree properties. Combining classifiers gives comparably excellent results by compromising interpretability and runtime of CART. Finally, a linearly induced tree model shows a better alternative approach for both decision tree and linear model.

A Appendix

A.1 Decision boundaries visualization



(a) CART, depth: 2 (b) LR_ODT, depth: 0 (c) RS_ODT, depth: 1

(d) CART, depth:4 (e) LR_ODT, depth: 3 (f) RS_ODT, depth: 1

(g) CART, depth: 3 (h) LR_ODT: depth: 4 (i) RS_ODT, depth: 3

(j) CART, depth:3 (k) LR_ODT, depth: 0 (l) RS_ODT, depth: 4

Figure A.1: 2D binary dataset consist of 1k instances, max_depth:4 & epoch:800 showing decision boundaries.

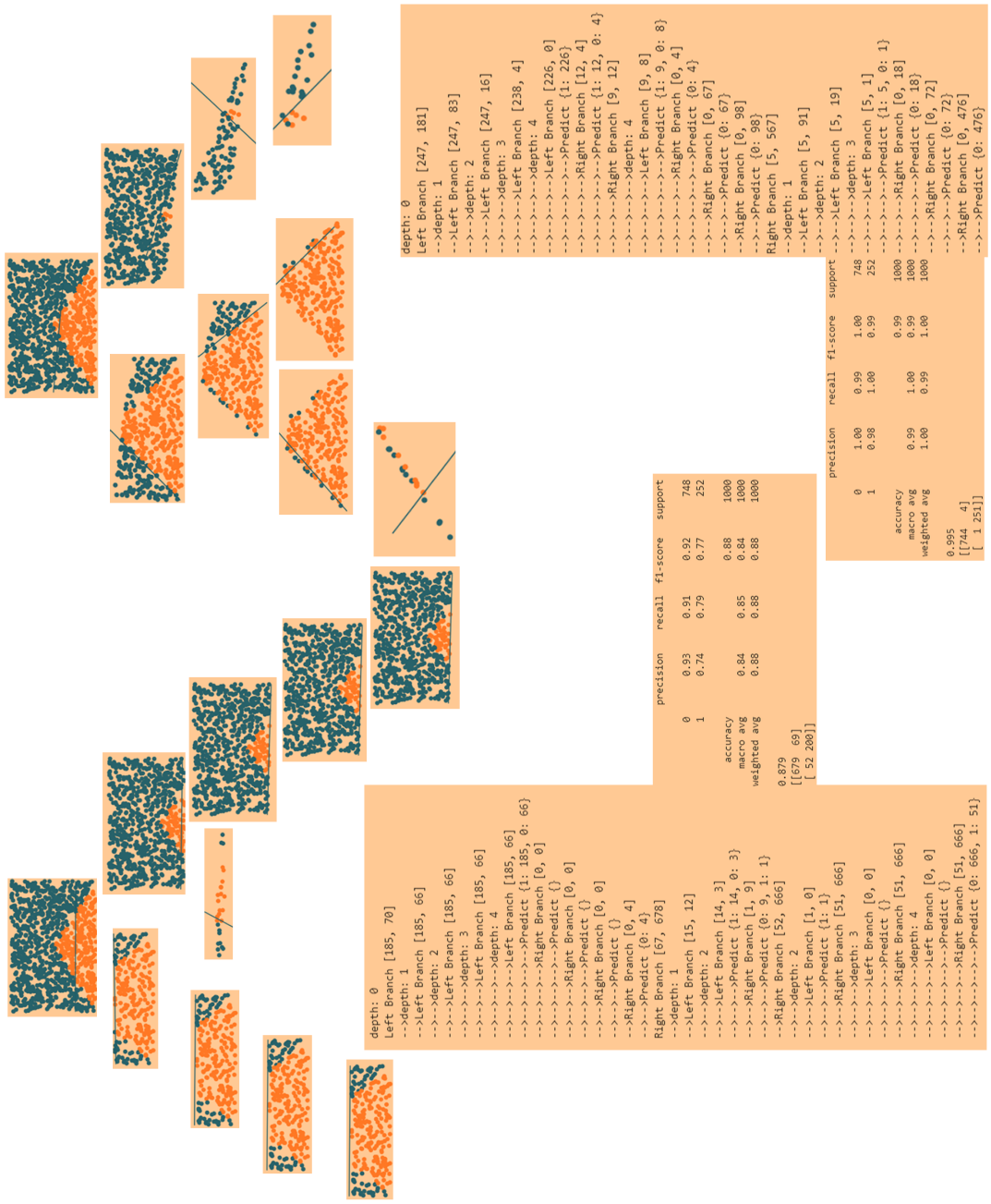


Figure A.2: Linearly separable 2D dataset consist of 1000 instances applied LR_ODT(before entropy condition) & RS_ODT shows learning process and evaluation when depth:4 & epoch:800

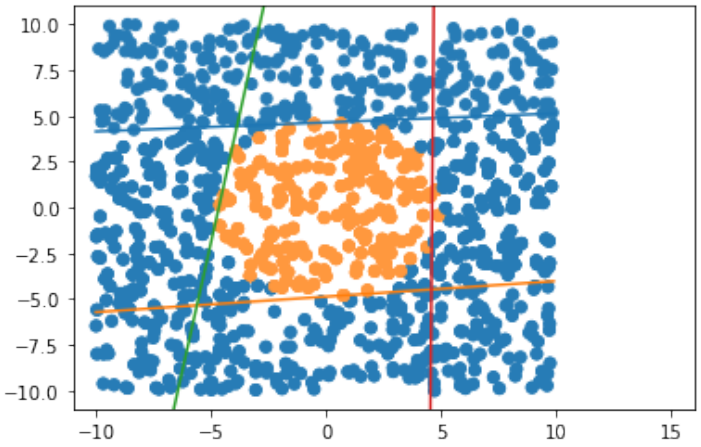


Figure A.3: An example where RS_ODT training on non linear dataset. A non linear 2D dataset consist of 1000 instances trained on RS_ODT shows decision boundaries when depth:4 & epoch:800. Our LR_ODT model cannot approximate this type of complex non linear form.

A.2 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.

Bibliography

[20a] en. Logistic Regression. Lecture note from CMU. 2020. URL: https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch12.pdf (cit. on p. 24).

[20b] *Artificial intelligence*. en. Page Version ID: 949523748. Apr. 2020. URL: https://en.wikipedia.org/w/index.php?title=Artificial_intelligence&oldid=949523748 (visited on 04/07/2020) (cit. on p. 1).

[BAC99] Z. Bandar, H. Al-Attar, and K. Crockett. "Genetic Algorithms For Decision Tree Induction". en. In: *Artificial Neural Nets and Genetic Algorithms*. Ed. by Andrej Dobnikar et al. Vienna: Springer, 1999, pp. 187–190. ISBN: 978-3-7091-6384-9. DOI: 10.1007/978-3-7091-6384-9_32 (cit. on p. 23).

[BA00] G. Baudat and F. Anouar. "Generalized Discriminant Analysis Using a Kernel Approach". en. In: *Neural Computation* 12.10 (Oct. 2000), pp. 2385–2404. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/089976600300014980. URL: http://www.mitpressjournals.org/doi/10.1162/089976600300014980 (visited on 04/14/2020) (cit. on p. 27).

[Ble] David M Blei. "Latent Dirichlet Allocation". en. In: (), p. 30 (cit. on p. 27).

[Bre+84] Leo Breiman et al. *Classification and Regression Trees*. en. Taylor & Francis, Jan. 1984. ISBN: 978-0-412-04841-8 (cit. on pp. 7, 20, 29).

[BU95] Carla E. Brodley and Paul E. Utgoff. "Multivariate decision trees". en. In: *Machine Learning* 19.1 (Apr. 1995), pp. 45–77. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/BF00994660. URL: http://link.springer.com/10.1007/BF00994660 (visited on 04/13/2020) (cit. on p. 27).

[Chi+] Eugene Chiu et al. "Mathematical Theory of Claude Shannon". en. In: (), p. 68 (cit. on p. 7).

[CE07] Ioannis T. Christou and Sofoklis Efremidis. "An Evolving Oblique Decision Tree Ensemble Architecture for Continuous Learning Applications". en. In: *Artificial Intelligence and Innovations 2007: from Theory to Applications*. Ed. by Christos Boukis, Aristodemos Pnevmatikakis, and Lazaros Polymenakos. Vol. 247. Series Title: IFIP The International Federation for Information Processing. Boston, MA: Springer US, 2007, pp. 3–11. ISBN: 978-0-387-74160-4. DOI: 10.1007/978-0-387-74161-1_1. URL: http://link.springer.com/10.1007/978-0-387-74161-1_1 (visited on 04/13/2020) (cit. on p. 23).

[DOL15] Andrew M. Dai, Christopher Olah, and Quoc V. Le. "Document Embedding with Paragraph Vectors". en. In: *arXiv:1507.07998 [cs]* (July 2015). arXiv: 1507.07998. URL: http://arxiv.org/abs/1507.07998 (visited on 04/10/2020) (cit. on pp. 12, 13).

[DS] Simon Kasif David Heat and Steven Salzberg. "Induction of Oblique Decision Tree". en. In: *Johns Hopkins University* () (cit. on pp. 19, 20).

[19] *Decision tree*. en. Page Version ID: 932260312. Dec. 2019. URL: https://en.wikipedia.org/w/index.php?title=Decision_tree&oldid=932260312 (visited on 04/06/2020) (cit. on p. 3).

[18] *Decision Tree Classification in Python*. Library Catalog: www.datacamp.com. Dec. 2018. URL: https://www.datacamp.com/community/tutorials/decision-tree-classification-python (visited on 04/23/2020) (cit. on p. 8).

[DC14] Mital Doshi and Setu K Chaturvedi. "Correlation Based Feature Selection (CFS) Technique to Predict Student Perfromance". en. In: *International journal of Computer Networks & Communications* 6.3 (May 2014), pp. 197–206. ISSN: 09752293, 09749322. DOI: 10.5121/ijcnc.2014.6315. URL: http://www.airccse.org/journal/cnc/6314cnc15.pdf (visited on 04/13/2020) (cit. on p. 27).

[Fan+] Rong-En Fan et al. "LIBLINEAR: A Library for Large Linear Classification". en. In: (), p. 31 (cit. on p. 25).

[FB81] Martin A. Fischler and Robert C. Bolles. *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*. June 1981. URL: https://doi.org/10.1145/358669.358692 (visited on 04/13/2020) (cit. on p. 25).

[Fur05] Johannes Furnkranz. *Logistic Model Tree*. English. 1 edition. Germany: Springer, Mar. 2005 (cit. on p. 23).

[20c] *gensim: topic modelling for humans*. en. Word embeddings, Topic modeling library. Library Catalog: radimrehurek.com. 2020. URL: `https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html` (cit. on p. 11).

[20d] *gensim: topic modelling for humans*. en. Natural language processing library. Library Catalog: radimrehurek.com. 2020. URL: `https://radimrehurek.com/gensim/index.html` (visited on 04/16/2020) (cit. on p. 35).

[Heh20] Thomas M Hehn. "End-to-End Learning of Decision Trees and Forests". en. In: *International Journal of Computer Vision* (2020), p. 15 (cit. on p. 23).

[Iye] Karthik Iyer. "The Pennsylvania State University". en. In: (), p. 99 (cit. on p. 42).

[KA13] Mohammad Khanbabaei and Mahmood Alborzi. "The Use of Genetic Algorithm, Clustering and Feature Selection Techniques in Construction of Decision Tree Models for Credit Scoring". en. In: *International Journal of Managing Information Technology* 5.4 (Nov. 2013), pp. 13–32. ISSN: 09755926, 09755586. DOI: `10.5121/ijmit.2013.5402`. URL: `http://www.airccse.org/journal/ijmit/papers/5413ijmit02.pdf` (visited on 04/13/2020) (cit. on p. 27).

[LM14] Quoc V. Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents". In: *arXiv:1405.4053 [cs]* (May 2014). arXiv: 1405.4053. URL: `http://arxiv.org/abs/1405.4053` (visited on 04/11/2020) (cit. on p. 13).

[LS98] Huan Liu and Rudy Setiono. "Feature Transformation and Multivariate Decision Tree Induction". en. In: *Discovey Science*. Ed. by G. Goos et al. Vol. 1532. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 279–291. ISBN: 978-3-540-65390-5 978-3-540-49292-4. DOI: `10.1007/3-540-49292-5_25`. URL: `http://link.springer.com/10.1007/3-540-49292-5_25` (visited on 04/13/2020) (cit. on pp. 27, 29).

[MS12] Naresh Manwani and P. S. Sastry. "Geometric Decision Tree". en. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42.1 (Feb. 2012). arXiv: 1009.3604, pp. 181–192. ISSN: 1083-4419, 1941-0492. DOI: 10.1109/TSMCB.2011.2163392. URL: http://arxiv.org/abs/1009.3604 (visited on 04/12/2020) (cit. on p. 21).

[Mik+13] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv:1301.3781 [cs]* (Sept. 2013). arXiv: 1301.3781. URL: http://arxiv.org/abs/1301.3781 (visited on 04/10/2020) (cit. on p. 12).

[Mit97] Tom M. Mitchell. *Machine Learning*. English. 1 edition. New York: McGraw-Hill Education, Mar. 1997. ISBN: 978-0-07-042807-2 (cit. on p. 1).

[MKS94] S. K. Murthy, S. Kasif, and S. Salzberg. "A System for Induction of Oblique Decision Trees". In: *arXiv:cs/9408103* (July 1994). arXiv: cs/9408103. URL: http://arxiv.org/abs/cs/9408103 (visited on 04/13/2020) (cit. on p. 20).

[Ngh12] Nghiaho12. *Multivariate decision tree (decision tree + linear classifier) — Nghia Ho*. Machine learning blog. 2012. URL: http://nghiaho.com/?p=1300 (visited on 04/14/2020) (cit. on p. 32).

[20e] *Niklaus Wirth Quotations at QuoteTab*. en. Blog. Library Catalog: www.quotetab.com. 2020. URL: https://www.quotetab.com/quotes/by-niklaus-wirth (cit. on p. 3).

[Nim11] Hanieh Poostchi Nima Salehi Moghaddami Hadi Sadoghi Yazdi. *Correlation based splitting criterion in multi branch decision tree*. English. 1 edition. USA: versita, Feb. 2011 (cit. on p. 23).

[00] *QUEST Node*. en-US. Library Catalog: www.ibm.com. Oct. 200. URL: www.ibm.com/support/knowledgecenter/en/ss3ra7_15.0.0/com.ibm.spss.modeler.help/questnode_general.htm (visited on 04/13/2020) (cit. on p. 21).

[Qui86] J. R. Quinlan. "Induction of decision trees". en. In: *Machine Learning* 1.1 (Mar. 1986), pp. 81–106. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/BF00116251. URL: http://link.springer.com/10.1007/BF00116251 (visited on 05/19/2020) (cit. on p. 7).

[RPR13] B. L. Robertson, C. J. Price, and M. Reale. "CARTopt: a random search method for nonsmooth unconstrained optimization". en. In: *Computational Optimization and Applications* 56.2 (Oct. 2013), pp. 291–315. ISSN: 0926-6003, 1573-2894. DOI: 10.1007/s10589-013-9560-9. URL: http://link.springer.com/10.1007/s10589-013-9560-9 (visited on 04/13/2020) (cit. on p. 21).

[Rud17] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv:1609.04747 [cs]* (June 2017). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747 (visited on 04/13/2020) (cit. on p. 25).

[SLN18] Habiba Muhammad Sani, Ci Lei, and Daniel Neagu. "Computational Complexity Analysis of Decision Tree Algorithms". en. In: *Artificial Intelligence XXXV*. Ed. by Max Bramer and Miltos Petridis. Vol. 11311. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 191–197. ISBN: 978-3-030-04190-8 978-3-030-04191-5. DOI: 10.1007/978-3-030-04191-5_17. URL: http://link.springer.com/10.1007/978-3-030-04191-5_17 (visited on 05/11/2020) (cit. on p. 42).

[20f] *scikit-learn: machine learning in Python — scikit-learn 0.22.2 documentation*. Open source machine learning and data science library. 2020. URL: https://scikit-learn.org/stable/index.html (cit. on p. 25).

[SWZ20] Hai Shu, Xiao Wang, and Hongtu Zhu. "D-CCA: A Decomposition-Based Canonical Correlation Analysis for High-Dimensional Datasets". en. In: *Journal of the American Statistical Association* 115.529 (Jan. 2020), pp. 292–306. ISSN: 0162-1459, 1537-274X. DOI: 10.1080/01621459.2018.1543599. URL: https://www.tandfonline.com/doi/full/10.1080/01621459.2018.1543599 (visited on 04/13/2020) (cit. on p. 26).

[SS09] Shailendra Singh and Sanjay Silakari. "Generalized Discriminant Analysis algorithm for feature reduction in Cyber Attack Detection System". en. In: 6.1 (2009), p. 8 (cit. on p. 27).

[Siu19] Chapman Siu. "Automatic Induction of Neural Network Decision Tree Algorithms". en. In: *Intelligent Computing*. Ed. by Kohei Arai, Rahul Bhatia, and Supriya Kapoor. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2019, pp. 697–704. ISBN: 978-3-030-22871-2. DOI: 10.1007/978-3-030-22871-2_48 (cit. on p. 23).

[skl20a] sklearn. *sklearn.linear_model.LogisticRegression — scikit-learn 0.22.2 documentation*. Machine Learning libraries. 2020. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logistic%5C%20regression#sklearn.linear_model.LogisticRegression (visited on 04/14/2020) (cit. on p. 30).

[skl20b] sklearn. *sklearn.linear_model.RANSACRegressor — scikit-learn 0.22.2 documentation*. en. Machine Learning libraries. 2020. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RANSACRegressor.html?highlight=ransac#sklearn.linear_model.RANSACRegressor (visited on 04/14/2020) (cit. on pp. 26, 32).

[skl20c] sklearn. *sklearn.tree.DecisionTreeClassifier — scikit-learn 0.22.2 documentation*. en. Machine Learning libraries. 2020. URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (visited on 04/14/2020) (cit. on pp. 8, 28).

[SG] Greg Ver Steeg and Aram Galstyan. "Discovering Structure in High-Dimensional Data Through Correlation Explanation". en. In: (), p. 9 (cit. on p. 27).

[Vla05] Vlado. *Oblique Decision Trees Using Embedded Support Vector Machines in Classifier Ensembles*. English. 1 edition. Germany: unknown, Mar. 2005 (cit. on p. 23).

[Wic+15] D. C. Wickramarachchi et al. "HHCART: An Oblique Decision Tree". en. In: *arXiv:1504.03415 [cs, stat]* (Apr. 2015). arXiv: 1504.03415. URL: http://arxiv.org/abs/1504.03415 (visited on 04/13/2020) (cit. on pp. 14, 22).

[20g] *Word2vec*. en. Page Version ID: 948380709. Mar. 2020. URL: https://en.wikipedia.org/w/index.php?title=Word2vec&oldid=948380709 (visited on 04/10/2020) (cit. on p. 12).

[Yan+14] Lijun Yan et al. "Genetic Generalized Discriminant Analysis and Its Applications". en. In: *Modern Advances in Applied Intelligence*. Ed. by David Hutchison et al. Vol. 8481. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 246–255. ISBN: 978-3-319-07454-2 978-3-319-07455-9. DOI: 10.1007/978-3-319-07455-9_26. URL: http://link.springer.com/10.1007/978-3-319-07455-9_26 (visited on 04/14/2020) (cit. on p. 27).

[YA12] Kazuyoshi Yata and Makoto Aoshima. "Effective PCA for high-dimension, low-sample-size data with noise reduction via geometric representations". en. In: *Journal of Multivariate Analysis* 105.1 (Feb. 2012), pp. 193–215. ISSN: 0047259X. DOI: 10.1016/j.jmva.2011.09.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S0047259X11001904 (visited on 04/13/2020) (cit. on p. 27).

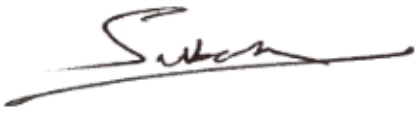
[YL] Lei Yu and Huan Liu. "Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution". en. In: (), p. 8 (cit. on p. 26).

[ZLY08] Defu Zhang, Stephen C. H. Leung, and Zhimei Ye. "A Decision Tree Scoring Model Based on Genetic Algorithm and K-Means Algorithm". en. In: *2008 Third International Conference on Convergence and Hybrid Information Technology*. Busan, Korea: IEEE, Nov. 2008, pp. 1043–1047. ISBN: 978-0-7695-3407-7. DOI: 10.1109/ICCIT.2008.110. URL: http://ieeexplore.ieee.org/document/4682170/ (visited on 04/13/2020) (cit. on p. 27).

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, June 6, 2020



AUTHOR