A Meta-Learning Analysis for evaluating the performance of Neural Networks vs. Boosted Trees on Tabular Data

Bachelor's Thesis project

Ramyasri Palti
Indian Institute of Technology, Bombay
Guides: Prof. Ganesh Ramakrishnan, Vishak Prasad, Abacus.AI

Novemeber 2022

Contents

1 Introduction

Heterogeneous tabular data are the most commonly used form of data and are essential for numerous applications. On homogeneous data sets, deep neural networks have repeatedly shown excellent performance. Although deep learning methods perform outstandingly well for classification or data generation tasks on homogeneous data, tabular data still pose a challenge to deep learning models.
However, despite recent advances in neural network architectures attuned to tabular data, there is still an active discussion on whether or not deep learning methods generally outperform gradient-boosted trees on tabular data.
Average performances of deep learning methods and boosted trees vary from dataset to dataset, while it is observed, algorithms based on gradient-boosted trees mostly outperform neural networks. Performance comparison varies from dataset to dataset, due to the variation in features a dataset has.
So, our direction of analysis is observing whether certain properties of a tabular dataset make deep learning methods outperform gradient-boosted trees or vice-versa. We conduct the analysis of tabular data by comparing 7 approaches and up to 30 hyperparameters across 200 datasets, while testing over 300 meta-features. We find that certain features are predictive of deep learning performance. Next, using our metadata set, we train an algorithm selection approach to select the best method and hyperparameters for new, unseen datasets.

2 Tabular Data vs. Homogeneous Data

Tabular Data is organized in terms of rows and columns, with each row depicting a data point and each column depicting an attribute corresponding to the data point. It consists of continuous, discrete, categorical, and ordinal values for the attributes. In contrast, homogeneous data such as text, image, or audio data, consists of a single feature type that is either continuous or discrete.

Age	Education	Occupation	Sex	Income
39	Bachelors	Adm-clerical	Male	$\leq$50K
50	Bachelors	Exec-managerial	Male	$>$50K
38	HS-grad	Handlers-cleaners	Male	$\leq$50K
53	11th	Handlers-cleaners	Male	$\leq$50K
28	Bachelors	Prof-specialty	Female	$>$50K

Figure 1: Example of Tabular Data, having attributes - Age, Education, Occupation, Sex, Income

Figure 1 depicts a tabular dataset. Attribute types are

- Discrete - a variable with a specified range Ex: Age
- Categorical - a variable with finite categories/groups Ex: Sex
- Ordinal - a categorical variable with an ordering among values Ex: Education

Tabular data are heterogeneous, leading to dense numerical and sparse categorical features. Furthermore, the correlation among the features is weaker than the one introduced through spatial or semantic relationships in homogeneous data. Hence, it is necessary to discover and exploit relations without relying on spatial information.
As mentioned earlier, deep neural networks often perform less favorably compared to decision trees for tabular data. However, it is often unclear why deep learning cannot achieve the same level of predictive quality as in other domains. The possible reasons include

- **Low-quality training data** - Real-world tabular datasets may have missing values, outliers, or inconsistent data
- **Missing or Complex Irregular Spatial Dependencies** - There is often no spatial correlation between the variables in tabular data sets, or the dependencies between features are rather complex and irregular
- **Preprocessing** - Handling the categorical features remains particularly challenging and can easily lead to a very sparse feature matrix (e.g., one-hot encoding) or introduce a synthetic ordering of previously unordered values (e.g., using an ordinal encoding)
- **Importance of Single features** - the smallest possible change of a categorical feature can entirely flip a prediction on tabular data. In contrast to deep neural networks, decision-tree algorithms can handle varying feature importance exceptionally well by selecting a single feature and appropriate threshold values and ignoring the rest of the data sample

3 Traditional Machine Learning Methods for Tabular Data

3.1 Logistic Regression

Find the best fitting model to describe the relationship between the dependent output and the independent input variable

3.2 KNN

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity (calculated by distance functions) to make classifications or predictions about the grouping of an individual data point(represented by an n-dimensional vector). Some of the Distance functions used are:

- Minkowski distance
$$d(\vec{a}, \vec{b}) = \left(\sum_{i=1}^{n} |a_i - b_i|^p \right)^{1/p}$$
- Euclidean distance
$$d(\vec{a}, \vec{b}) = \sqrt{\sum_{i=1}^{n} |a_i - b_i|^2}$$

- Manhattan distance

$$d(\vec{a}, \vec{b}) = \sum_{i=1}^{n} |a_i - b_i|$$

- Cosine distance

$$d(\vec{a}, \vec{b}) = 1 - \left(\frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \cdot \left\|\vec{b}\right\|} \right)$$

3.3 SVM

Performs classification by finding the optimal hyper-plane that differentiates the two classes very well. For multiclass classification, the same principle is used after breaking down the multi-classification problem into binary classification problems.
The data can be either linear or nonlinear. There are different kernels that can be set in an SVM Classifier. SVM uses Kernel, which transforms the data so that a non-linear decision surface is able to transform to a linear equation in a higher number of dimension spaces, in which they become separable.
For a linear dataset, we can set the kernel as 'linear'. For non-linear data,

- RBF kernel

$$k(x, y) = e^{-\gamma \|x-y\|^2}$$

- Sigmoid Kernel

$$k(x, y) = tanh\left(\gamma \cdot x^T y + r\right)$$

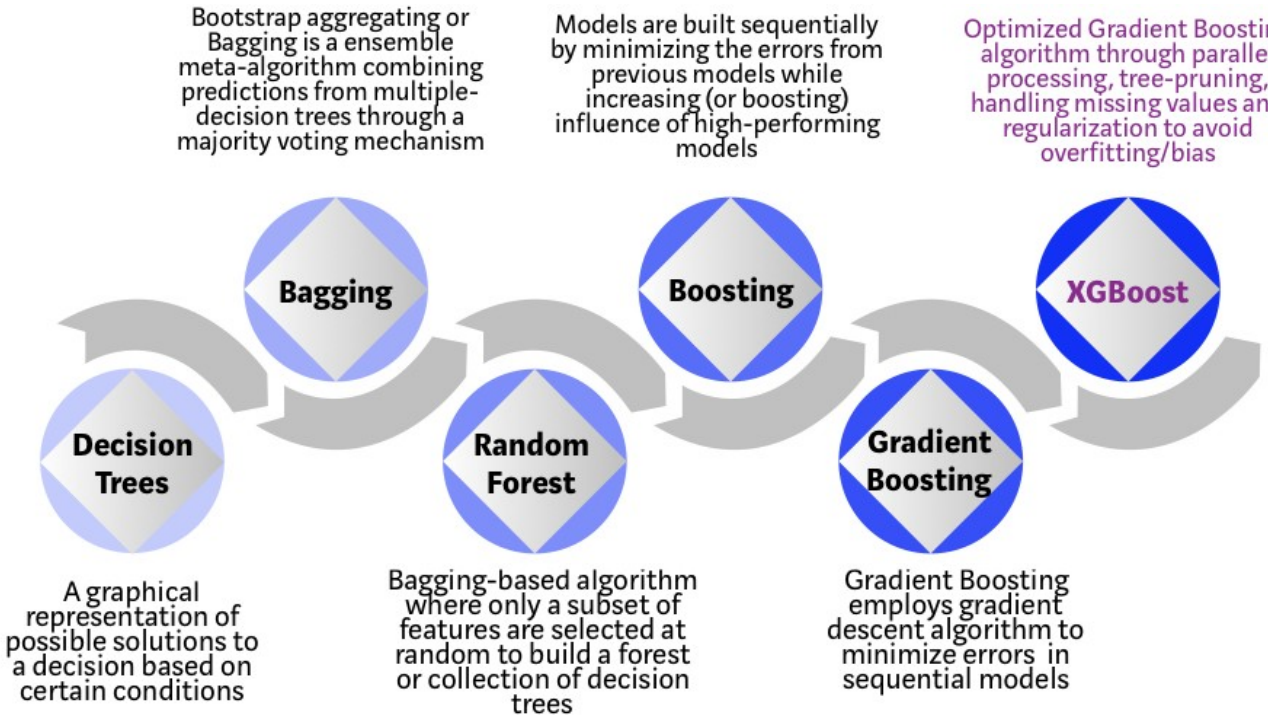
- Polynomial Kernel

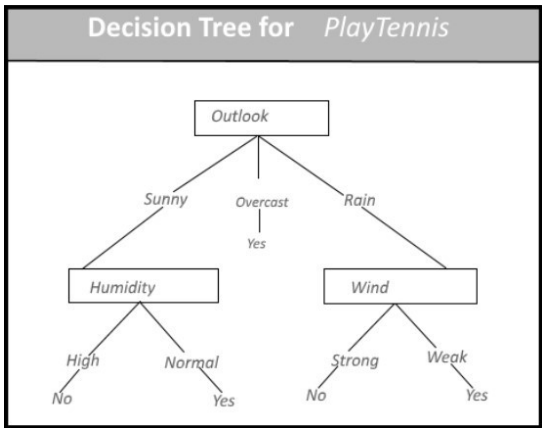
$$k(x, y) = tanh\left(\gamma \cdot x^T y + r\right)^d$$

4 Decision Trees for Tabular Data

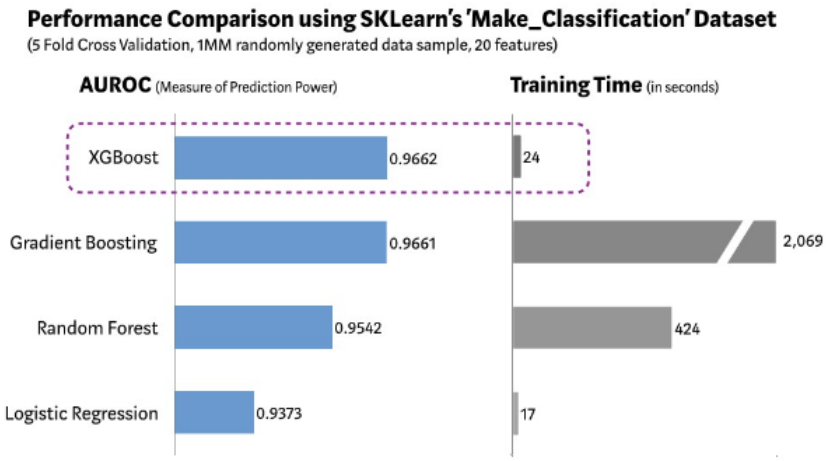
A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions.





Boosting: At each step, we alter the evaluation criterion based on the result of the previous step. This 'boosts' the efficiency of the algorithm by deploying a more dynamic evaluation process.
Gradient Boosting: A special case of boosting where errors are minimized by gradient descent algorithm.
XGBoost: It is a combination of software and hardware optimization techniques to yield superior results using fewer computation resources in the shortest amount of time. It uses parallelized tree building, Tree pruning using depth-first approach, cache awareness, and out-of-core computing, Regularization to avoid overfitting, efficient handling of missing data, and in-built cross-validation capabilities.



There are also other types of boosting like **LGBoost**(Light Gradient Boosting), **NGBoost**(Natural Gradient Boosting, which uses natural gradient in order to enable uncertainty estimates of predictions, which can be used to update the training dynamics), **CatBoost**(Categorical Boosting, it uses a permutation driven variant of boosting), it performs gradient boosting on oblivious decision trees (decision tables), which makes inference very efficient, and the method is quite resistant to overfitting

5 Neural Networks for Tabular Data

5.1 TabNet

TabNet inputs raw tabular data without any preprocessing and is trained using gradient descent-based optimization. TabNet uses sequential attention to choose which features to reason from at each decision step, enabling interpretability and better learning. This feature selection is instance-wise, e.g. it can be different for each input, and TabNet employs a single deep learning architecture for feature selection and reasoning. The sequential attention mechanism improves the performance of the model.

5.2 NODE

Neural Oblivious Decision Ensembles generalize ensembles of oblivious decision trees. Performance is enhanced with gradient-based optimization and hierarchical representation learning. In its essence, the proposed architecture generalizes CatBoost, making the splitting feature choice and decision tree routing differentiable. As a result, the NODE architecture is fully differentiable. Furthermore, NODE allows constructing multi-layer architectures, which resemble "deep" GBDT that is trained end-to-end.

5.3 Hopular

A recent architecture designed for tabular data, where each layer is equipped with continuous modern Hopfield networks. The modern Hopfield networks use stored data to identify feature-feature, feature-target, and sample-sample dependencies. Every layer can directly access the original input as well as the whole training set via stored data in the Hopfield networks. Therefore, Hopular can step-wise update its current model and the resulting prediction at every layer like standard iterative learning algorithms.

6 Analysis of Algorithms on Tabular Data

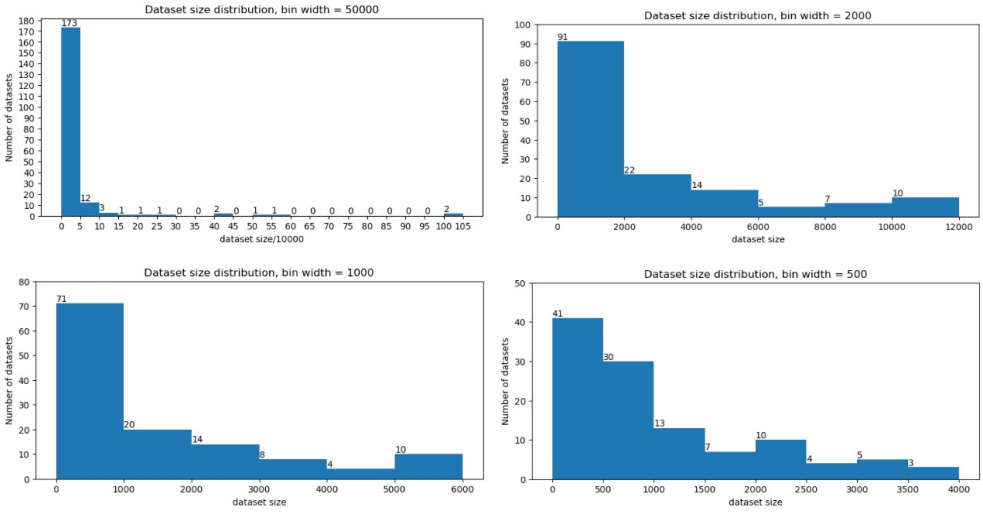
6.1 Datasets

We focussed on OpenML datasets. For each dataset, there are specific problems that have to be solved (Ex: predicting a specific attribute value). Each such specific problem is called a task for the given dataset. A task specifies train/test splits, a target feature that has to be predicted, and metadata such as the size of the dataset, number of features, number of classes, etc.
We focused on supervised classification tasks for which there was a 10-fold cross-validation split provided within OpenML.
Since we wanted to generate train/validation/test splits to be able to tune algorithms (choose their hyperparameters) and then estimate their performance on unseen data, we used a procedure to generate such splits. For each fold i, we keep the test set the same as it was in the original split. The validation set will be the test set in fold i + 1(modulo the number of folds), and the new training set is created by removing these samples from the original training set. In this way, each sample is included in training, testing, and validation at least once.

6.2 Metadata Analysis

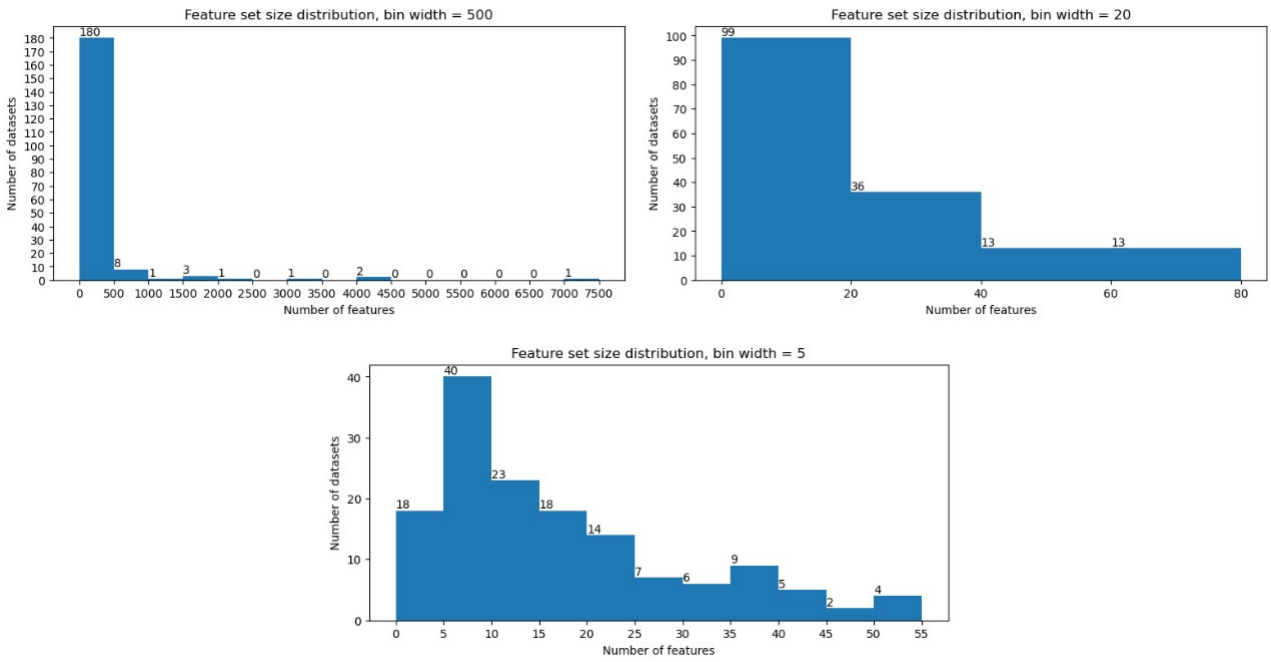
Metadata for a dataset include target feature type(regression/ binary classification/ multi-class classification), number of features, size of the dataset, etc.
Performed analysis on metadata to observe the distributions of data over this properties.
Target type distribution over datasets:
Binary Classification - 95
Multi-classification - 83
Other target types(regression, etc.) - 19

Distribution of dataset size across 200 datasets



185/197 datasets had 50k number of rows(dataset size)

Distribution of feature set size across 200 datasets



180/197 datasets had less than 500 features

6.3 Extracting Meta Features

We extracted meta-features using the Python library PyMFE. Within this library, meta-features are divided into groups. The meta-features we used are

- General - number of classes, number of numeric features, type of classification
- Statistical - mean and range of each feature
- Information theory - such as the Shannon entropy of the target
- Landmarking - relative and sub-sampling with algorithms such as Naive Bayes, 1-Nearest neighbor
- Model-based - summary statistics for some model fit on the data, such as number of leaf nodes in a decision tree model

7 Experiments

For each algorithm, and for each dataset split, we tune the algorithm on the validation set and evaluate its performance on the test set.
To investigate how well meta-features can explain the difference in performance between pairs of algorithms, we compute the difference in tuned performance between all pairs of algorithms across all datasets (for which both algorithms in the pair resulted in successful experiments). We compute the correlations between each meta feature and the performance difference across all datasets. Later, for new unseen datasets, we devise an algorithm selection approach to choose the best algorithm and hyperparameters using our meta-dataset

The results for the F1 metric: Top 2 Most correlated features for each algorithm

Algorithm	Metafeature 1	Metafeature 2
CatBoost	statistical.can cor.quantiles., score:0.46	info-theory.attr ent.histogram, score:-0.45
KNN	statistical.can cor.quantiles, score:0.39	statistical.can cor.mean, score:0.39
LinearModel	statistical.can cor.quantiles, score:0.47	statistical.can cor.median, score:0.44
MLP	statistical.can cor.skewness, score:-0.39	statistical.can cor.quantiles, score:0.38
XGBoost	info-theory.attr ent.histogram, score:-0.42	statistical.can cor.mean, score:0.38
SVM	info-theory.attr ent.histogram.7, score:-0.56	statistical.can cor.mean, score:0.45
TabNet	info-theory.attr ent.histogram, score:-0.52	statistical.can cor.mean, score:0.49

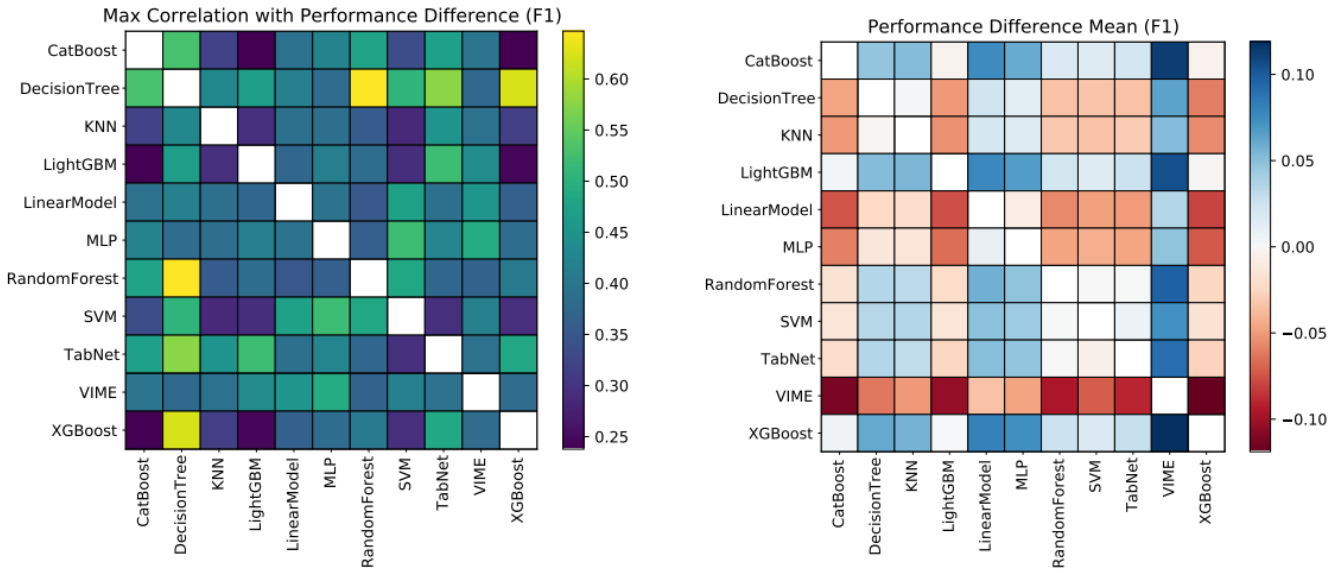


Figure 2: Left plot shows the maximum absolute correlation between any meta feature and the performance difference, and the right plot shows the mean performance differences. For the latter, the performance of the algorithm on the x-axis is subtracted from that on the y-axis

8 Relative Algorithm Performance

Performance of algorithms varies from dataset to dataset. For each dataset split and for each algorithm, we first "tune" the algorithm by selecting the hyperparameter sample that has the best performance on the validation set. We use 3 performance metrics and tune each algorithm to each metric. We compute mean performance by averaging out all the 10-folds performances and rank the algorithms accordingly.

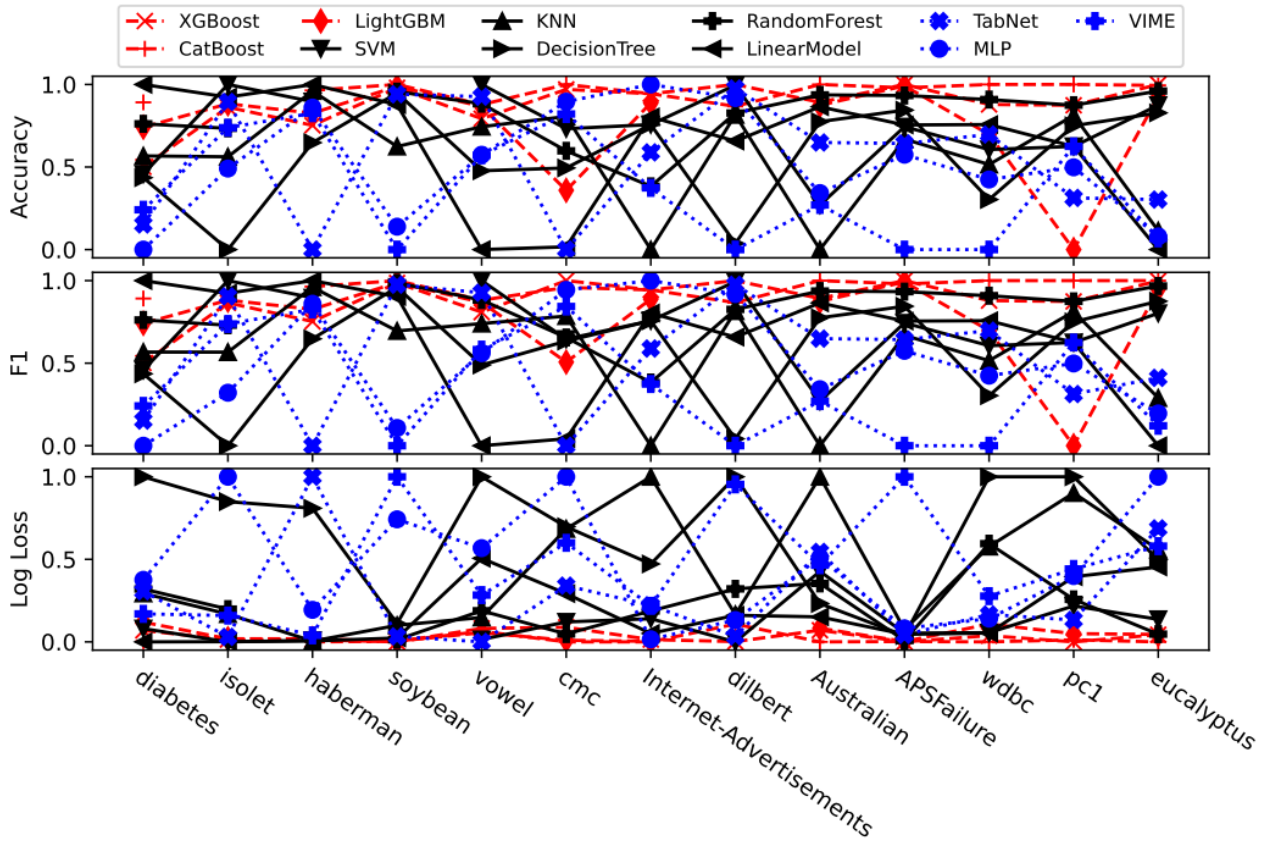


Figure 3: Relative performance of all algorithms, over a subset of datasets. All three metrics are normalized to be in range [0.0, 1.0]

9 Adding Kernel as a hyperparameter to SVM Algorithm

Earlier, SVM Classification/Regression is trained and tested with the default "rbf" kernel. But, the possible kernels are linear(for linear data), rbf, sigmoid, and polynomial Also, we can give a custom precomputed kernel function, for the model to classify data points. Now, the SVM algorithm also has kernel function as a hyper-parameter which has to be optimised for best performance.

10 Adding Similarity metric as a hyperparameter to KNN Algorithm

Earlier, KNN(K-Nearest Neighbors) is trained and tested with the default distance metric function "minkowski" or the c-norm. But, the possible distance metric functions are cosine distance, euclidean or the L2-norm, and manhattan or the L1-norm. Now, the KNN algorithm also has distance metric function as a hyper-parameter which has to be optimised for best performance.

11 References

- Deep Learning and Tabular Data : A Survey
- Well Tuned Simple Nets Excel on Tabular Datasets
- Tabular Data: Deep Learning is not all you need
- Hopular: Modern Hopfiled Networks for Tabular Data