STUDIEN ARBEIT

Submitted to

Intelligent Systems Group
University of Siegen

In partial fulfillment of the requirements for the degree of

MASTERS IN MECHATRONICS

By

Gireesh Guntupalli

Consistency of Algorithm-Implementations of different Machine Learning Frameworks

Primary Supervisor:	Prof. Joran Beel
Secondary Supervisor:	Tobias Vente
Author:	Gireesh Guntupalli
Matriculation Number:	1586967

January 2023

Contents

1. Introduction 1
2. Background 2
   2.1 How are Random Forests random? 2
   2.2 Datasets 3
3. Methodology 4
   3.1 Random Forests in Scikit-Learn, H2o and PySpark Machine Learning Frameworks: 5
4. Results 9
5. Discussions 12
   5.3 Prediction 12
   5.4 H2O's special parameters: 14
   5.5 Differences in the Feature Importances: 15
6. Conclusion: 17
6. References 18

List of Figures:

Figure 1: Simple representation of random forest algorithm implementation. 2
Figure 2: Formation of H2O training frame by concatenating pandas train and test frames and converting it to H2O dataFrame. 4
Figure 3: Process of producing vectorised Inependent_features using PySpark Vector assembler. 5
Figure 4: Bar chart representing comparison among precision, recall and f1 scores of 4 different classifiers implemented in scikit-learn, H20 and PySpark machine leaning frameworks. 10
Figure 5: Precision-Recall curves of Scikit-Learn and PySpark random forest classifiers for creditcard_fraud_detection task. 10
Figure 6: Precision-Recall curves of H2O random forest classifiers for creditcard_fraud_detection task. 11
Figure 7: Console output showing performance of H2O Random Forest classifier built on term_deposit_subscription data. 13
Figure 8: Bar chart representing accuracies of H2O model with and without binomial_double_trees. 14
Figure 9: Feature importances of Bank_loan_classification dataset computed using sklearn random forest algorithm. 15
Figure 10: Feature importances of Bank_loan_classification dataset computed using H20 random forest algorithm 16
Figure 11: Feature importances of Bank_loan_classification dataset computed using PySpark random forest algorithm. 16

List of Tables:

Table 1: List of similar algorithm parameters across frameworks and their descriptions. 7
Table 2: Performance scores of each random forest classifier. 9
Table 3: Performance scores of random forest regressor for California house price prediction task. 11
Table 4: Comparison table of actual and predicted labels of classifier built on term_deposit_subscription dataset. 13

Abstract

Machine learning frameworks are interfaces that enable faster implementation and deployment of machine learning models with ease. Most of the machine learning frameworks implement algorithms for supervised and unsupervised learning. Many industries are already incorporating machine learning into their work. Companies are unsure about which machine learning frameworks to use. This study tries to resolve this uncertainty by answering the research question "how consistent are the algorithm implementations in different machine learning frameworks?". To meet the study goal, this project creates random forest models in the machine learning libraries Scikit-Learn, H20, and PySpark and compares their performances to assess differences and determine what is causing these variances. The results reveal that there are few differences in the way random forests are implemented in these frameworks. The algorithm-implementation is almost same for Scikit-Learn and PySpark frameworks expect for the ways in which they make predictions. Because of the way H2O forests are implemented it is seen that they perform well on both classification and regression tasks. This study also showcases the differences in the feature importances of given data, computed using random forests algorithm in these frameworks.

1. Introduction

A subset of artificial intelligence, machine learning is composed of algorithms that train the data to make classifications or predictions in response to new data. The surge in the growth of Machine Learning adoption in companies in recent years is remarkable. 67% of enterprises now employ machine learning, and 97% either already use it or expect to in the upcoming year, says a 2020 Deloitte survey. In response to the pandemic in 2021, 41% of firms boosted their deployment of AI, as per IBM's most recent Global AI Adoption Index. Today, according to one-third of IT professionals, AI is deployed in their companies.

A machine learning framework is a tool that gives a user the interface to build machine learning models effortlessly without having to delve into the intricate mathematical or statistical principles that underlie the machine learning algorithms. This is accomplished by allowing the user to import built-in algorithm functions. Most of the machine learning frameworks implement algorithms for supervised and unsupervised learning. As it is said and done that many enterprises are implementing machine learning projects, it is now quite a challenge for them to choose a machine learning framework that will work for them. Consequently, the question arises: Are these algorithm implementations among different machine learning libraries the same? If used on the same data, do they produce the same results?

The decision of which machine learning framework to choose, is determined by the availability of the algorithm in that framework, but if only a few of these frameworks share an algorithm, then the selection is based on the performance of the model built. A model whose evaluation metrics are superior to those of models' evaluations in other frameworks can be a deterministic factor that aids in selecting the framework. The following research question is at the centre track of my work: How consistent are algorithm-implementations of different machine learning frameworks?

The objective of my research is to find out if different machine learning frameworks implement the same algorithms differently. This research project aims to identify how an algorithm is implemented in a framework and compare it to another. The idea is to use two or more machine learning frameworks that implement the same algorithm and build learning models in each framework on the same data sets and analyze whether the algorithms' predictions differ.

To accomplish the purpose of this study, I selected the machine learning frameworks Scikit-Learn, H2o, and PySpark (a Python interface for Apache Spark) to investigate the consistency of “Random Forest” algorithm implementation among these three frameworks. A random forest[1] model is trained in each framework on the same dataset and the results are then analyzed using evaluation metrics such as accuracy, precision, recall, F1 score, and mean square error (MSE). Random forest is an ensembled supervised learning algorithm that can be used for both classification and regression problems.

This project demonstrates to a machine learning engineer or data scientist how random forest implementation varies from framework to framework. This report presents clear disparities in feature significance evaluation across all three frameworks.

2. Background

2.1 How are Random Forests random?

Decision trees are highly sensitive to the training data which results in high variance. To overcome this problem, random forests are introduced. A random forest algorithm is a collection of multiple uncorrelated random decision trees.

Random forests are random because of the two random processes involved in it:

1. Bootstrapping (Row sampling with replacement.)
2. Random feature selection.

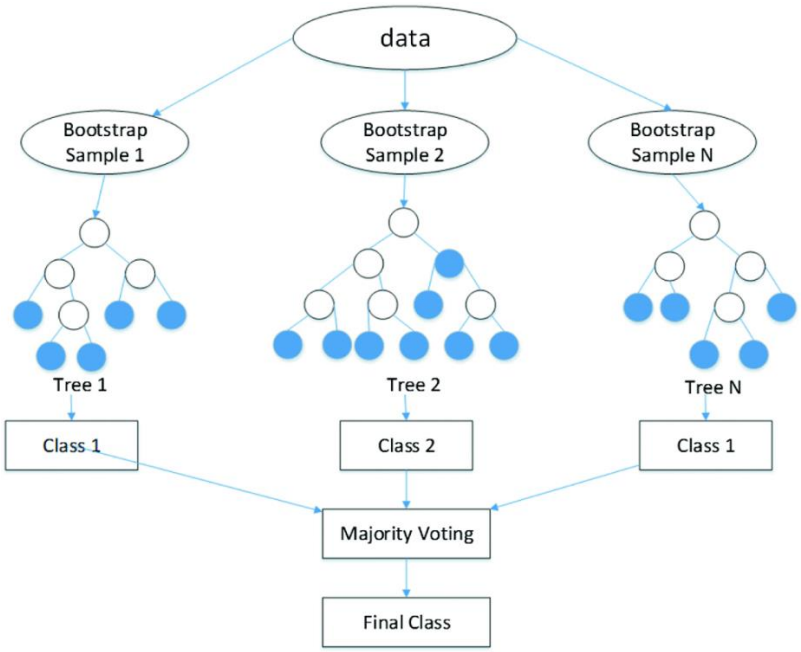

Figure 1: Simple representation of random forest algorithm implementation.

Internally, the random forest algorithm generates new datasets from the original data. This is known as bootstrapping. Bootstrapping is the process of randomly selecting rows (notice that the same row can be sampled numerous times) from the original data to generate a new dataset with the same number of instances as the original dataset, but with features randomly picked as a subset of the original features. These new randomly generated datasets are used to train the decision trees, and predictions from each tree are aggregated to determine the Random Forest model's overall prediction result. In a regression problem, the prediction from each tree is typically averaged to obtain the final prediction, whereas in a classification task, the most frequently predicted category (majority vote) is used to determine the final predicted class. This entire process of bootstrapping and aggregating is called Bagging[2].

It is statistically proven that during bootstrapping approximately one third of the original data instances may not be sampled at all. These records are called Out of the Bag records (OOB records). These OOB records are used as validation dataset and the accuracy score obtained on this valid dataset is called OOB Score.

2.2 Datasets

This section presents datasets used in this project to build random forest algorithms. To examine how the model evaluation metrics differ from one library's model to another and to see if there is any trend in how these metrics differ from one another, a random forest model is built with each framework on five different datasets ranging in size from 5000 to approximately 388000 observations. Each dataset is briefly described in the following description.

Bank_loan_classification: This dataset contains information about 5000 Universal Bank clients gathered over several years. This collection has 14 features. The aim is to estimate whether a costumer would be approved for a personal loan based on his or her degree, age, work, family status, and a few other attributes. The target column contains 9.6% instances of class - '1' and the remainder are class - '0' items.

Term_deposit_subscription: This dataset comprised of Portuguese banking institution's phone call collected data from May 2008 to November 2010. This dataset contains 41188 occurrences with 20 characteristics. The categorization task is to determine whether a client will subscribe to the term deposit. "Subscribed" (Binary: 'yes' or 'no') is the target variable. The target column has 36548 'No' categories and 4640 'Yes' categories.

Rain_in_Australia: This dataset comprises around ten years of daily weather measurements from various places throughout Australia. The data set has 145460 records over 23 columns. After training the model with temperature, rainfall, clouds, wind speed, and humidity features, the classification aim is to predict whether it will rain the next day in Australia. The label column is called "RainTomorrow", and it contains 110316 "yes" and 31877 "no" classes.

Creditcard_fraud_detection: The binary classification task requires identifying fraudulent credit card transactions. This file comprises credit card transactions from European cardholders, with 491 frauds out of a total of 284807 transactions. As a result, the model constructed on this basis produces very high accuracy by default. Due to confidentiality concerns, the dataset's characteristics have been PCA reduced. This collection has 31 characteristics, and the features v1, v2, ..., v28 are PCA principal components. The target column is "class," and a '1' indicates that the transaction is fraudulent, while a '0' indicates that it is not.

Covid_detection: The shape of the data frame is (388878, 17). The main goal is to determine if the patient is at high risk of being attacked by Covid-19 based on the patient's symptoms and prior health concerns, which are provided as dataset features. This dataset is provided by Mexican government for research purposes. In the target column, there are 59979 patients that are at high risk of getting Covid-19 and 328899 entries that are not.

California_housing_rates: The data contains information from the 1990 California census. The shape of the dataset is (20433, 10). The regression task is to fit the housing data and predict the price of a new house with given attributes such as total_rooms, ocean_proximity, total_bedrooms, population of that region, housing_median_range and median incomes of residents in the area.

3. Methodology

To assess how consistent the random forest algorithm implementation is in Scikit-Learn[3], H20[4], and PySpark[5], a python code pipeline is created for each library in such a way that just the algorithm implementation portion of the pipeline is varied based on the model train and predict methods of that library, while keeping rest of the pipeline the same for all three libraries. This ensures that the data is partitioned consistently and that the same train and test sets are utilized for all three random forests models constructed in all three frameworks. Then the algorithm phase is implemented differently for each library and the predictions are taken, to evaluate Accuracy, Precision, Recall and F1 Score, with the help of Scikit-Learn model metrics. These evaluation metrics are then compared to one another for more insights.

The pipeline can be divided into 5 steps. The template of the pipeline is as follows:

Step 1: Importing dataset as pandas dataframe.

Step 2: Data wrangling using pandas and Scikit-Learn modules.

Step 3: Splitting Dataframe into train and test sets using Sklearn K-Folds cross validation.

Step 4: Implementing random forest algorithm and making predictions.

Step 5: Evaluating model performance using Scikit-learn metrics and scoring.

The step 4 of the pipeline is the only segment that is different for different libraries. The real algorithm implementation occurs in each pipeline's 'build()' function. The build() method is comprised up of a 'for loop' that iterates for K-Fold times. Within this for loop, NumPy arrays ‘x_train, y_train and, x_test, y_test’, are constructed from the data set.

Arrays of this kind are accepted by the Scikit-Learn model, however the methods in the H2O and PySpark libraries cannot be trained on NumPy arrays. H2O requires an H2O data frame, while PySpark requires a Spark type data frame for training the model.

The next step is to convert the train and test sets into H2O and PySpark compatible data formats. As a result, in the PySpark and H2O pipeline's build function, these x _train, y_train, and x_test, y_test NumPy arrays are first encoded into Pandas frames. Then "x train + y train" and "x test + y test" are concatenated to generate the train and test sets.

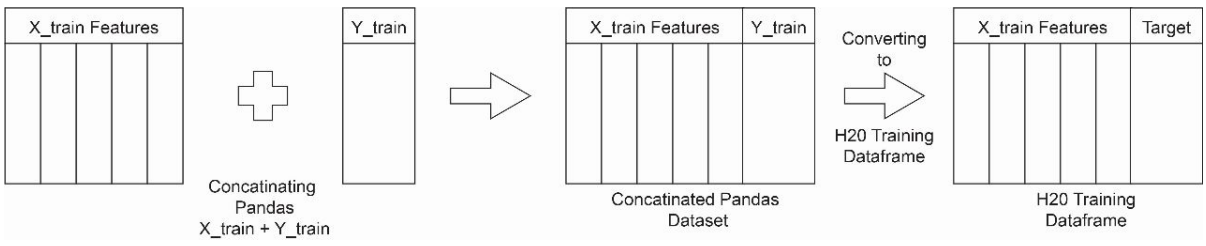

Figure 2: Formation of H2O training frame by concatenating pandas train and test frames and converting it to H2O dataframe.

These train and test sets are converted to H2O data frames in the H2O pipeline. Figure 2 demonstrates the formation of H20 train set. Similarly, H20 test set frame is formed. These H2O data frames may now be used to train the model, while the converted H2O test set can be utilized to generate predictions.

Before utilizing 'NumPy to Spark converted' data frames for training and testing, the PySpark pipeline necessitates a further vectorization procedure. A vector assembler is a PySpark module that converts the numerical features of a spark dataframe into a single vector that machine learning models may utilize for training and prediction. After concatenating NumPy arrays and converting them to spark frameworks, an additional column called 'Independent_features' is shaped in the spark train and test sets by vectorizing all the independent variables into a single vector column using a vector assembler. Only the 'Independent_features' and 'target' columns are chosen from the vector-assembled data frame for training and testing the model.

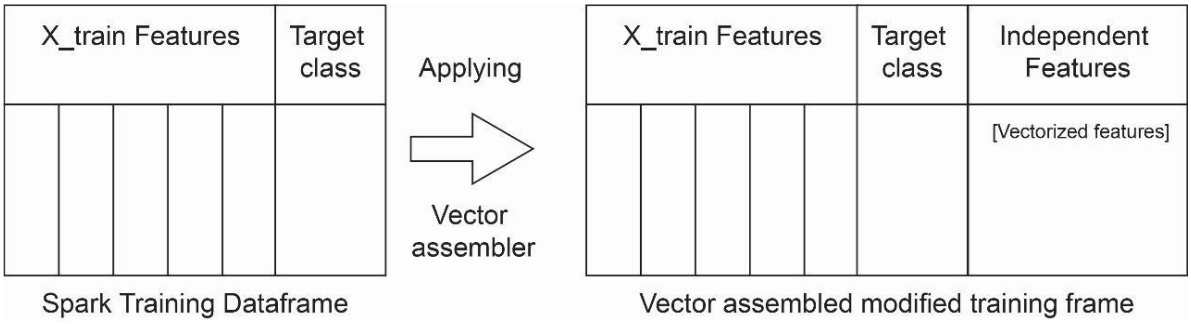

Figure 3: Process of producing vectorised Inependent_features using PySpark Vector assembler.

After training is completed and predictions are acquired from the models in H2O and PySpark, these predictions are transformed back to Pandas dataframes or NumPy arrays since only these types of data are compatible for computing the Scikit-Learn metric assessment.

3.1 Random Forests in Scikit-Learn, H2o and PySpark Machine Learning Frameworks:

Decision trees are the building blocks of random forests. Hence node splitting criteria is one of the important parameters of the algorithm which must be taken into consideration for comparison. The measure of information of a class provided by a feature is called information gain. Information Gain aids in determining the order of attributes in decision tree nodes. The feature with maximum information gain is the node that is split first.

Let us now compare forest classifier algorithms' basic features across all the three frameworks. Gini Impurity is the criterion set in Scikit-Learn and PySpark libraries' Random Forest classifiers[6] for information gain calculation. The mathematical formulation used in both libraries are same. Likewise, there are few other parameters that influence the performance of the model. The section below discusses about the algorithm parameters.

There are few parameters that are common among these models. All the three Random forests algorithms' hyperparameters that are identical in these machine learning frameworks are tuned to the same value to perform a reliable comparison study of the algorithm implementation across these three libraries. Given below are the Random Forest classifiers' initializations in each framework.

Scikit-Learn Random Forest Classifier:

```
model = RandomForestClassifier(n_estimators=100,  max_features="sqrt", max_depth=4,
                               max_samples= None, criterion="gini", min_impurity_decrease=0,
                               min_samples_leaf=1, ## for H2O comparison (set to default)
                               random_state=42)
```

H2O Random Forest Estimator:

```
model = H2ORandomForestEstimator(ntrees=100, mtries= -1, max_depth=4, sample_rate = 1,
                                 min_split_improvement = 0,
                                 min_rows=1, ## for Sklearn comparison
                                 seed=42)
```

PySpark Random Forest Classifier:

```
model = RandomForestClassifier(featuresCol='Independent_features', labelCol=y_names,
                               numTrees=100, featureSubsetStrategy="sqrt", maxDepth=4,
                               subsamplingRate= 1, impurity="gini",
                               minInfoGain=0, seed=42)
```

The parameters utilized for these models are listed in detail in the table below.

Library	Parameter	Description
Scikit- Learn	n_estimators	The number of trees in the forest. For all 3 models, this number is set to 100 trees. The performance of the model will typically improve with increasing number of trees, but this requires longer computational times, and after a certain point, there won't be any noticeable increases in the performance.
H2O	ntrees	
PySpark	numTrees	
Scikit- Learn	max_features	A parameter that lets user select the number of randomly sampled features. Researchers discovered values near to the square root or log of the total number of features in the original data set as an approximation of the number of features sampled. These parameters are set to "sqrt" in all the three models. In H2O 'mtries = -1' returns square root of number of columns in the dataset.
H2O	mtries	
PySpark	featureSubsetStrategy	
Scikit- Learn	max_depth	This parameter limits growth of a tree. If its value is 1, it means tree has one internal node and 2 leaf nodes. These parameters set to "4" in each model.
H2O	max_depth	
PySpark	maxDepth	

Scikit- Learn	max_samples	This parameter value prescribes the number of instances to use from the train set for training the model. These parameters are set in a such a way that all models use all the instances of the training set for training the model.
H2O	sample_rate	
PySpark	subsamplingRate	
Scikit- Learn	criterion	Specifies the type of criterion to be used for calculating the gain. PySpark supports "gini" for gini impurity criterion and "entropy" for information gain calculation. In addition to these Scikit-Learn also supports "log-loss" for Shannon information gain calculation. "gini" criterion is used for both the models to measure the quality of split while building estimators.
H2O	-	
PySpark	impurity	
Scikit- Learn	min_impurity_decrease	Node will be split if and only if the value of impurity is greater than or equal to the value specified to these parameters. These are set to '0' in all three models.
H2O	min_split_improvement	
PySpark	minInfoGain	
Scikit- Learn	class_weight	This specifies a column of weights associated with classes. In all the three frameworks this parameter is set to "None" which means all the classes are assumed to be of equal weight 1.0.
H2O	weights_column	
PySpark	weightCol	
Scikit- Learn	min_samples_leaf	This parameter value set the minimum number of observations for a leaf. This value is set to '1' in our Scikit-Learn(default) and H2O random forest models.
H2O	min_rows	
PySpark	-	
Scikit- Learn	random_state	This is set to an integer value "42", so that the model outputs same values every time it is run.
H2O	seed	
PySpark	seed	

Table 1: List of similar algorithm parameters across frameworks and their descriptions.

Besides Random Forest Classifiers, Scikit-Learn, H2o and PySpark also provide Random Forest Regressors for solving regression problems. The function to measure the quality of split varies from that of classifiers. Sklearn supports various criterions such as "squared_error"**,** "absolute_error"**,** "friedman_mse", and "poisson"**,** whereas in PySpark "variance" is the only criterion used for information gain calculation.

H2O trains regression trees regardless of whether the task is classification or regression. In H2O the decision of which feature to split-on is based on which feature reduces a node's squared error the most.

The histogram binning process is used to quickly compute the potential MSE of each possible split in H2O trees. Hence the histogram parameters are also of important consideration in H2O random forest hyper parameter tuning. The histogram type defaults to ‘auto’ which forms bins from minimum to maximum in the steps of (maximum – minimum)/N. This parameter can also be set to ‘random split points’ or ‘uniform adaptive’ or ‘quantiles global’ or ‘random robin’. The number of bins in the histogram is controlled with the help of parameters ‘nbins_cats’ for categoricals, ‘nbins’ and ‘nbins_top_level’. All these values are set to default for our random forest classifiers.

These three random forest pipelines are trained on five distinct classification datasets of different shapes, and a comparison table of metrics evaluation values on test sets is produced. The next section goes through this comparative study.

4. Results

All the random forest classifiers are implemented for 10 folds and the classification metrics are computed by averaging the values from 10-fold evaluations. The results of these metrics are tabulated as follows.

Library	Metrics	DATSETS				
		bank_loan_classification	Term_deposit_subscription	Rain_in_australia	Creditcard_fraud_detection	Covid_detection
Scikit-Learn	Accuracy	0.7418	0.889351	0.813218	0.999266	0.826403
	Precision	0.812021	0.354967	0.834866	0.910726	0.262916
	F1_Score	0.822693	0.032277	0.33549	0.73053	0.30235
	Recall	0.161294	0.0169072	0.209924	0.609863	0.355703
H20	Accuracy	0.4022	0.886415	0.796874	0.999378	0.885396
	Precision	0.5978	0.465244	0.577298	0.849381	0.348334
	F1_Score	0.927629	0.546240	0.300513	0.813593	0.490736
	Recall	0.945529	0.661383	0.203125	0.780700	0.830077
PySpark	Accuracy	0.7414	0.889667	0.813056	0.999252	0.829350
	Precision	0.798479	0.357180	0.829761	0.916125	0.205596
	F1_Score	0.854037	0.392839	0.335445	0.743895	0.247302
	Recall	0.163309	0.020785	0.210214	0.626176	0.310235

Table 2: Performance scores of each random forest classifier.

It can be observed that there are not many differences in the accuracy scores of these models. The trend is almost similar. But, except for the rain_in_australia task, H2O's recall score is very high in all other tasks when compared to other framework models' recall scores. To draw a clear comparison, a bar graph is plotted in Figure 4 to represent the variations in Precision, F1_score and Recall scores of classification task bank_loan_classification, term_deposit_subscription, creditcard_fraud_detection and covid_detection. From figure it can be inferred that the trend of scores for Scikit-Learn and PySpark are almost similar. H2O models always show higher recall scores implying H2O random forests are successful in predicting correctly large number of positive classes out of actual positives.

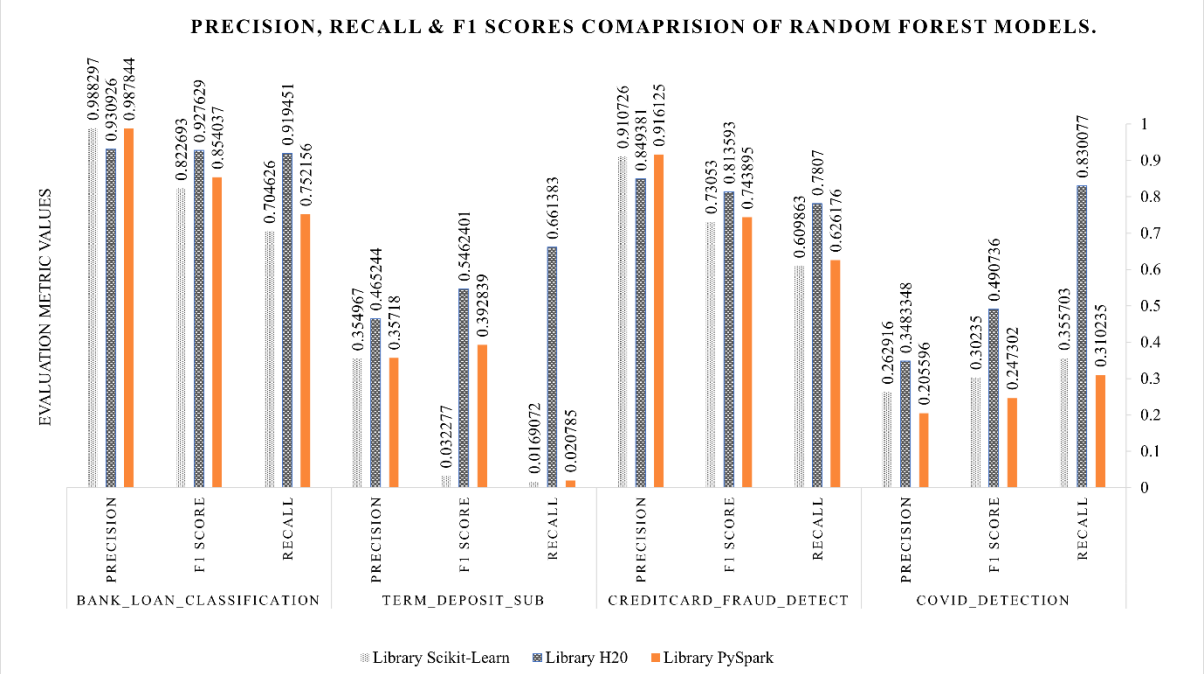

Figure 4: Bar chart representing comparison among precision, recall and f1 scores of 4 different classifiers implemented in scikit-learn, H20 and PySpark machine leaning frameworks.

For the creditcard fraud detection task, the target column 'Class' has only 492 fraudulent items, which is only about 0.17% of the total data, making it a highly imbalanced dataset. Hence, to evaluate the performance of the random forest model trained on this data, Precision-Recall curve is drawn for all the 10 folds and the area under the curve is taken for the overall resultant characteristic. Figure 5 represents the precision-recall curve of Sklearn and PySpark models plotted for 10 folds. H20 classifier outperformed the Scikit-Learn and PySpark classifier by around 6%.

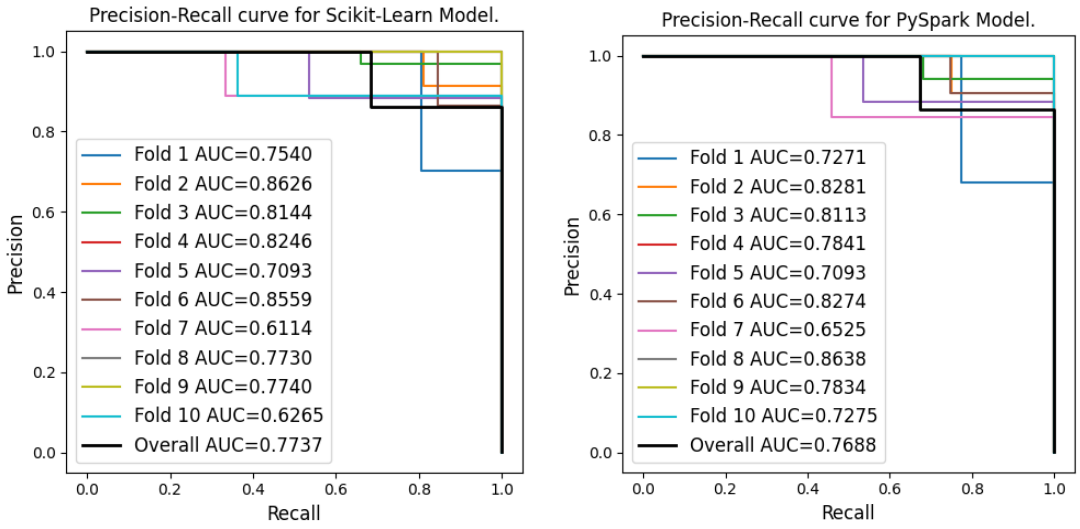

Figure 5: Precision-Recall curves of Scikit-Learn and PySpark random forest classifiers for creditcard_fraud_detection task.

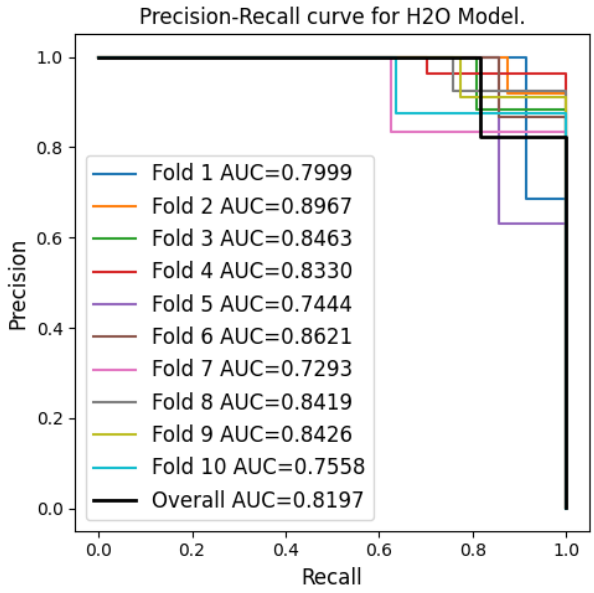

Figure 6: Precision-Recall curves of H2O random forest classifiers for creditcard_fraud_detection task.

Also, a random forest regressor is built in each framework on the popular California housing rates dataset. The data is split with 20-fold Sklearn cross fold validator. The forest is built with 100 trees and each tree having a depth of 8. The results of the evaluation metrics are tabulated as follows.

Metrics	Scikit-Learn Library	H20 Library	PySpark Library
Mean Absolute Error (MAE)	51815.8272	52103.9158	50274.0036
Root Mean Squared Error (RMSE)	68655.5268	69163.19607	68014.7319
R2 Score	0.382701	0.373161	0.4102567

Table 3: Performance scores of random forest regressor for California house price prediction task.

H2o and Scikit-Learn Forest regressors use squared error criteria to choose best split at the node. Whereas PySpark model is implemented using "variance' criterion. From table 3 it is evident that squared error criterion is better than variance for tree splitting while constructing trees for random forest regressors as Scikit-Learn and H2O regressors lead to less RMSE when compared to PySpark regressor. Despite being a robust algorithm, random forest regressors do not really perform good when given a new data that is beyond the range of the data on which it was trained. It is incapable of extrapolating the data, but a linear regression model can.

5. Discussions

Overall, there are few differences in the implementation of random forest algorithm among the machine learning libraries Scikit-Learn, H2O and PySpark. This section discusses about Random Forest algorithm implementation across these frameworks and points out differences among them.

5.1 Bootstrapping and Feature Sampling:

Three frameworks' random forests perform bootstrapping and features sampling. For classification problems, by default, all the libraries sample square root of original number of features (n_features) and for regression 1/3 of the predictors are sampled. In addition to these, log2(n_features) can be implemented in Sklearn and PySpark. PySpark also allows user to specify a number between 0 and 1 and number of features which is equivalent to product of given fraction and n_features are drawn for feature sampling. It is also possible to use all the attributes for training in all the three libraries' algorithms.

5.2 Tree node split criterion:

Choosing the best split while building trees is one criterion which marks H2O different from others. Scikit-Learn and PySpark can implement gini impurity and entropy for information gain calculation. Sklearn also supports logloss criterion. But the way H2O's random forests chooses the best split is different from these criterions. H2O random forest algorithm splits on the attribute and the level that results in the greatest reduction in residual sum of squares (RSS) of the subtree at that point to find the best level. H2O adopts histogram binning process to quickly compute mean squared error.

5.3 Prediction:

In H20 and Scikit-Learn random forests, both classifiers and regressors compute the mean prediction of all their trees to give a final prediction whether predicting for a class or a numeric value. But, conventionally for classification problems the majority vote from all the trees is taken as final prediction. This is the case with Spark's Forests. For classification tasks in PySpark each tree's prediction is counted as a vote for one class. The label is predicted to be the class which receives the most votes. For spark forest regression the aggregation is averaging of all tree predictions.

For predicting a class usually, the class with the highest predicted probability values are chosen as the final label in any classification problem but H2O adopts a different probability threshold selection for choosing a class from the predicted probabilities. For a binary classification problem in H2O all predicted probabilities greater than or equal to the maximum F1 (F1 max) threshold are labelled with positive class and for a multiclass problem class with highest predicted probability is taken as final prediction.

The threshold at F1 MAX which is set to find the class from predicted probabilities is one of the reasons which causes major differences in case of H2O random forests' classification metrics when compared to others. From the Figure 4 see how h2O's recall, precision and F1 score are varied from other model's scores. Except for the rain_in_australia task, in rest of the

tasks, H2O's recall score is always the highest. It can be inferred that H2O random forest classifier classifies maximum number of classes correctly.

Consider the binary classification task of 'term_deposit_subscription', where there is major difference of recall score to that of others. Given below, in the following table, are the list of first ten predictions of random forest model built on 10th fold of term_deposit_subscription dataset:

Serial no.	P0	P1	Actual Label	Predicted Label
1	0.619570	0.380430	0	1
2	0.899882	0.100118	1	0
3	0.890192	0.100118	0	1
4	0.901478	0.098522	1	0
5	0.908837	0.091163	0	0
6	0.874354	0.125646	1	1
7	0.883407	0.116593	0	1
8	0.883266	0.116734	0	1
9	0.896392	0.103608	1	0
10	0.901478	0.098522	0	0

Table 4: Comparison table of actual and predicted labels of classifier built on term_deposit_subscription dataset.

"model.**model_performance(test)**" module in H2o library gives performance of the model on the test data. For the 10th fold, the performance of classifier can be extracted as follows (only a portion of output on console is shown in the Figure 7):

```
Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.10871746370325919:
       0     1     Error    Rate
-----  ----  ----  -------  ---------------
0      894   1332  0.5984   (1332.0/2226.0)
1      281   1611  0.1485   (281.0/1892.0)
Total  1175  2943  0.3917   (1613.0/4118.0)

Maximum Metrics: Maximum metrics at their respective thresholds
metric                       threshold    value     idx
---------------------------  -----------  --------  -----
max f1                       0.108717     0.666391  319
max f2                       0.0316893    0.809585  398
max f0point5                 0.285517     0.669395  162
```

Figure 7: Console output showing performance of H2O Random Forest classifier built on term_deposit_subscription data.

Figure 7 shows values of maximum metrics at their respective thresholds. The threshold at maximum f1 score is 0.10871746370325919. This is the threshold used to determine the positive class. From the Table 4 observe highlighted rows, even though the probability of class-0 is higher than class-1, classifier model predicted the label as class-1, as probability of Class-1 (P1) value is greater than the threshold at maximum f1 score (0.10871746370325919).

5.4 H2O's special parameters:

H2O goes a step ahead and provides two additional parameters for random forest algorithm. One of them is 'binomial_double_trees' parameter. By default, H2O random forest estimator builds half as many trees for binomial problems. The algorithm forms a single tree to estimate class '0' and then to estimate class '1' it computes "1 – probability (class zero)", instead of building a tree again for class '1'. This way the algorithm can be less computationally expensive. If binomial_double_trees parameter is set to true, the algorithm calculates trees for both the classes '0' and '1'. This option can increase the accuracy of the model but takes more time for training the model.

To check the performances of binomial_double_trees parameter of H20 Random forests, three binary classification models are built with and without binomial_double_trees parameter using 4 folds (k=4) cross validation split. Accuracy score results of these models are plotted below in Figure 8. It can be observed that there is no much difference in predictive accuracy score of the model even when the binomial_double_trees is set to 'True.' There is only very small increase in accuracy, that too change can be noticed only after three or four decimal points, which contributes less than 0.5% to 1% increase of accuracy depending on the dataset. On the other hand, it is noticed that time taken to train the model has been increased to 40% to 70% of the time taken without binomial_double_trees.

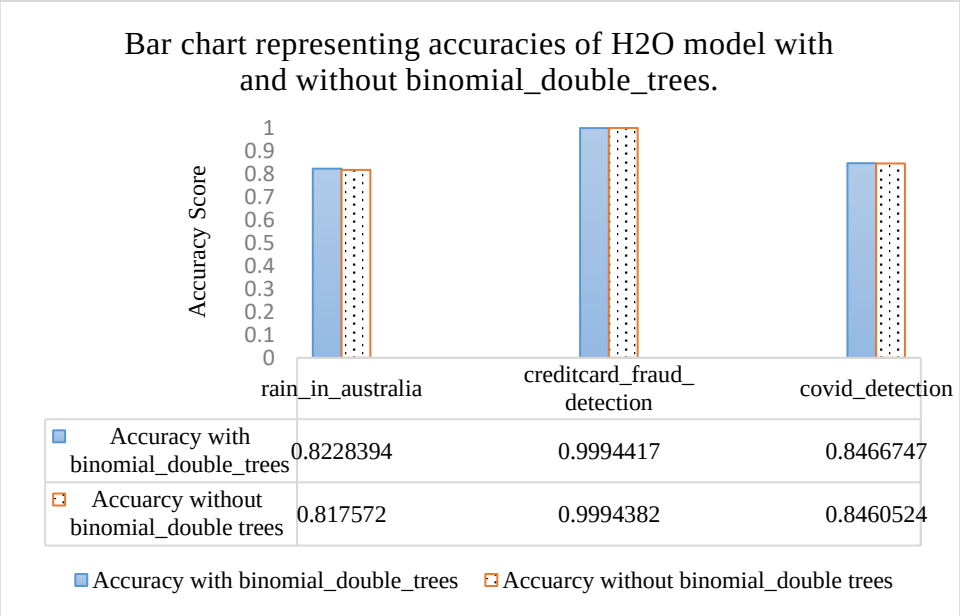

Figure 8: Bar chart representing accuracies of H2O model with and without binomial_double_trees.

The second additional parameter provided in H2O random forests is 'early stopping' of the training in the H2O forests. In Sklearn and PySpark the recursive tree construction is stopped when the maximum depth is reached or when no split condition leads to information at the note. But H2O forests provide early stopping parameters such as stopping_rounds, stopping_metrics and stopping_tolerance. When these stopping criteria are met the model stops building the tree any further.

5.5 Differences in the Feature Importances:

Using significant features to train a model is one of the keys to build a machine learning model that performs very well. Feature importances are the scores that rank significance of features used to build a model. Random forests algorithms can be used to evaluate feature importance scores. Figures 9, 10, 11 show the features importances taken from Bank_loan_classification dataset's first fold evaluation out of 10 folds, using random forest classifier.

Sklearn analyzes feature relevance using a method known as mean decrease in impurity (MDI)[7]. The top-of-the-tree features influence the final prediction choice of a bigger proportion of the input samples. The proportion of samples that a feature contributes to is paired with impurity reduction from splitting them to get a normalized estimate of that feature's predictive power. MDI averages predictive ability estimates from numerous randomized trees to decrease volatility and utilize them for feature selection.

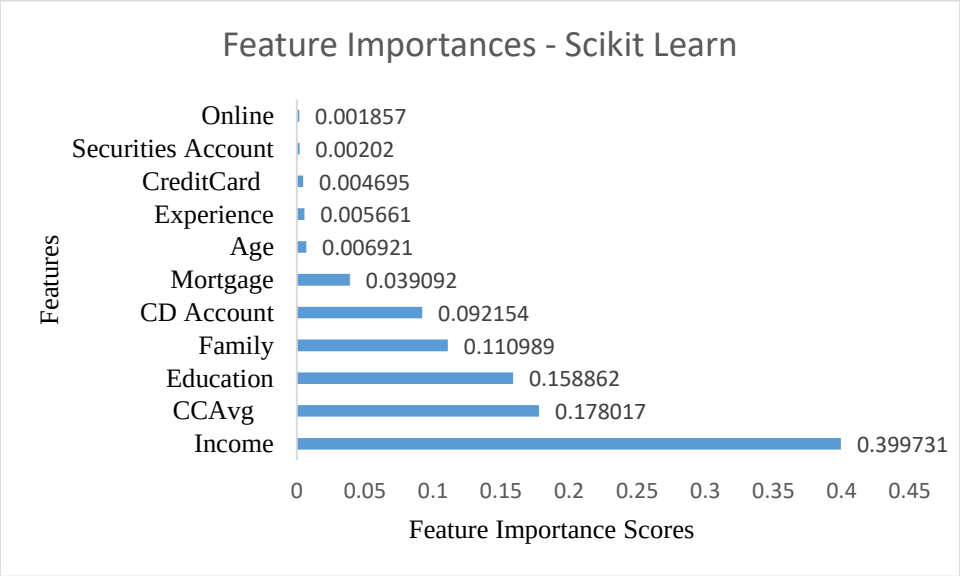

Figure 9: Feature importances of Bank_loan_classification dataset computed using sklearn random forest algorithm.

H2O compute the differences in the improvement of squared error before and after the split of each feature. Each features improvement is then summed up at the end to get its total feature importance

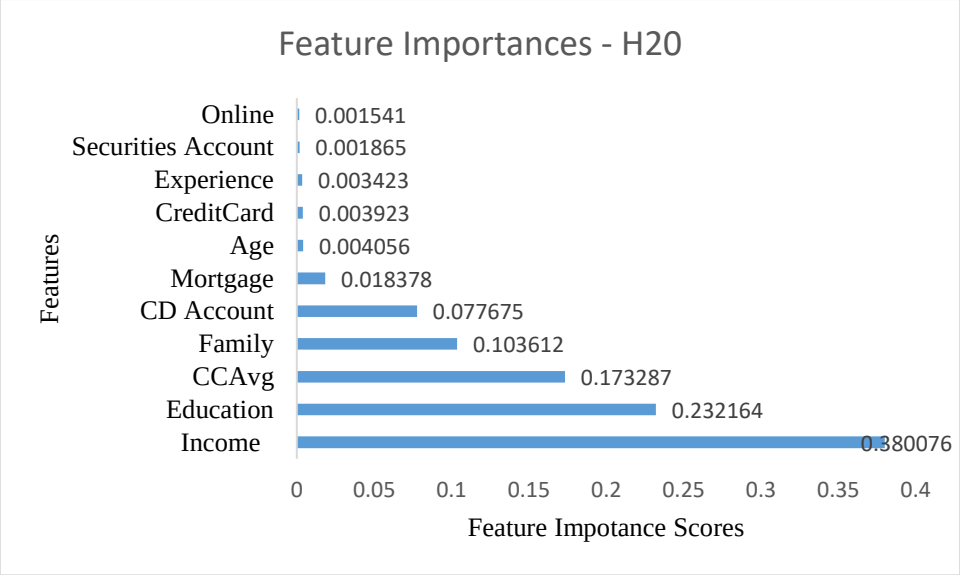

Figure 10: Feature importances of Bank_loan_classification dataset computed using H20 random forest algorithm

Spark evaluates the importance of a feature for each decision tree by summing the information gain scaled by the number of samples passing through the node.

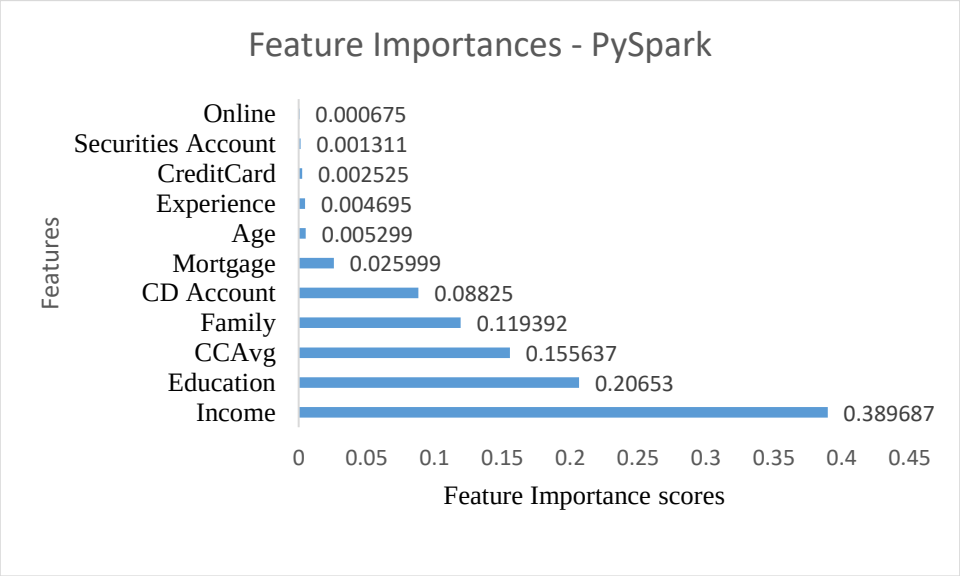

Figure 11: Feature importances of Bank_loan_classification dataset computed using PySpark random forest algorithm

It can be seen from the plots that there are differences in the evaluations of feature importances using Random forests among the frameworks Scikit-Learn, H2o and PySpark. The order of feature significances derived from these models differs between libraries.

6. Conclusion:

This study presented the differences in Random Forest algorithm implementation in Scikit-Learn, H2O and PySpark machine learning frameworks. This research project implemented Random Forest classifiers for five different binary classification problems and a Random Forest regressor to evaluate consistency of algorithm-implementations among these three machine learning frameworks. The results showed that Scikit-Learn and PySpark models' performances are almost similar even though they follow different aggregation methods to get predictions. H2O random forests outperformed other forest models in both classification and regression tasks. This study also proved that the feature importances computed using random forests is different in different frameworks.

6. References

1. Breiman, L., 2015. Random forests leo breiman and adele cutler. Random Forests-Classification Description, 106.
2. L. Breiman, “Bagging predictors”, Machine Learning, 24(2), 123-140, 1996
3. Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P.Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E (2011) Scikit-learn: Machine Learning in {P}ython. https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
4. Cook, D., 2016. *Practical machine learning with H2O: powerful, scalable techniques for deep learning and AI*. " O'Reilly Media, Inc.".
5. Ranganathan, G., 2020. Real time anomaly detection techniques using pyspark frame work. *Journal of Artificial Intelligence*, *2*(01), pp.20-30.
6. Singh, P., 2022. Random Forests Using PySpark. In *Machine Learning with PySpark* (pp. 105-126). Apress, Berkeley, CA.
7. Louppe, G., 2014. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*.