Practical Work 2: Decision Forests

Victor Badenas Crespo

Supervised and Experiene Learning
Master in Artificial Intelligence (FIB-MAI)

May 2021

Contents

Chapter 1

Introduction

1.1 Introduction

In this work, whe have implemented and tested a version of *RandomForestClassifier* and *DecisionForestClassifier*. Both approaches are quite similar and have a lot of common concepts, that is why they have been developed in an OOP style following some SOLID principles. The algorithms have been tested for the required configurations which as a reminder are:

1. Random Forest Classifier
   (a) $NT = 1, 10, 25, 50, 75, 100$
   (b) $F = 1, 3, log2(M + 1), \sqrt{M}$
2. Decision Forest Classifier
   (a) $NT = 1, 10, 25, 50, 75, 100$
   (b) $F = int(M/4), int(M/2), int(3 * M/4), RU(1, M)$

Where RU stands for a random uniform value between 1 and M. The previous combinations have been tested on UCI Datasets. The datasets chosen for this experiments are:

1. iris (< 500)
2. car ($500 < n\_instances < 2000$)
3. kr-vs-kp (> 2000)

The datasets will be explained later on in the report. Finally we will comment the results extracted from the models, the times for each task, accuracies and the most relevant features.

Chapter 2

Classifiers

In this chapter we will discuss the base classes for all Classifiers as well as the particularities of each of the two classifiers implemented for this work. The implementation of the Classifiers is segmented in several parts:

1. BaseClassifier (*./src/base_classifier.py*)
2. ForestInterpreter (*./src/forest_interpreter.py*)
3. Node, Tree, Leaf: all of them inherit from BaseClassifier (*./src/tree_units.py*)
4. BaseForest: inherits from BaseClassifier and ForestInterpreter (*./src/base_forest.py*)
5. RandomForestClassifier: inherits from BaseForest (*./src/random_forest.py*)
6. DecisionForestClassifier: inherits from BaseForest (*./src/decision_forest.py*)

The *BaseClassifier* is an abstract class implementing the fit, predict and fit_predict methods for the classes that inherit from it. *ForestInterpreter* implements the loading and inference methods to create a Forest from a model previously saved as json. The *Node, Tree and Leaf* classes are the main units of the building of a tree structure and they all inherit from *BaseClassifier* as they need fit and predict methods. Then the *BaseForest* class inherits from *BaseClassifier* and *ForestInterpreter*. The first for the basic train methods and the later for inference. Finally the two *ForestClassifiers* inherit from *BaseForest* and will each implement their distinguishable features.

2.1 Leaf

The class *Leaf* implements the basic termination of a Tree structure. The main concept behind it is that once a branch of the tree has been terminated, will imply that a prediction has to be made for that instance and so, when in training we reach a leaf, we need to compute the probability for the instance to be of a class. For that we store the probability ccomputed as the count of class instances for the instances that reach the Leaf in the node divided by the number of instances in the leaf. When predicting, a dictionary containing the class names and the probability will be returned.

2.2 Node/Tree

The *Node* and *Tree* class are equivalent. However, the tree class is instantiated by the Classifiers while the Node is only instantiated by the Tree or another Node. The class is responsible for multiple actions as defined by the following methods:

2.2.1 fit

The goal of this method is to determine the best split of the node and determine the two branches if is not feasible to split the data anymore, a return statement forces the parent node to terminate that Node attempt with a Leaf. The main process for the training of the node is as follows:

Algorithm 1: Node/Tree fit method

```
Result: self
X := data for the node;
F := number of random features;
attributes := dataset attributes;
node_gini := gini_index(X);
best_gain := 0;
best_feature, best_value := None, None;
if F < 0 or len(attributes) < F then
    features := attributes
else
    features := K random attributes
end
for feature in features do
    for unique value in feature column in X do
        true_split, false_split = try to split by the condition. feature == value for
         categorical and feature >= value for numerical;
        if len(true_split) == 0 or len(false_split) == 0 then
            skip to next value as it does not split the data;
        end
        gain := node_gini - split_gain(true_split, false_split);
        if gain > best_gain then
            best_feature := feature;
            best_value := value;
            best_gain := gain;
        end
    end
end
true_split, false_split := split(X, best_value, best_feature);
// initialize both branches from the node. If the split only contains one item, create leaf
 instead and finally call recursively the nodefit method. If the method returns a None,
 replace the newly created node with a Leaf instead to terminate the process.;
branches[True] := initialize_branch(true_split);
branches[False] := initialize_branch(false_split);
```

The algorithm above is implemented on the Node/Tree. First, the data for the node X, the number of random features to F and dataset attributes *attributes*. Then we compute the gini_index for the instances in X. The gini_index is computed efficiently using pandas following the following expression:

$$Gini(X) = 1 - \sum_{i=0}^{N} (\frac{X_{x \in C_i}}{len(X)})^2$$

after computing the gini_index of the data in the node, a set of F features are chosen from the dataset attributes. If there are not enough attributes, all attributes will be used. Also, if $F < 0$, all attributes will be considered. For each feature considered, all (feature, value) pairs in the dataset are tested and the gini index gain is computed for all and the split condition with the highest gain is chosen to be the condition for this node.

Once the condition is set, the branches are then initialized. There are 3 scenarios:

1. the number of instances is 1 for a given split. Then the branch is initialized to a Leaf.

2. the branch is initialized as a Node but no gain is found in further splitting the data. Then the branch is initialized to a Leaf.

3. the branch is initialized to a Node and it fits appropppiately. We then continue creating nodes recursively.

2.2.2 predict

The predict method in the Node will evaluate the condition and then return whatever the branch returns recursively. This way we can return the dictionary containing the probabilities provided by the Leaf of the tree.

2.3 ForestInterpreter

The forest interpreter class contains the main loading and prediction methods for any BaseForest type of Classifier. The most interesting method for the class is the *predict* method. The predict method will ask the prediction for each instance to each tree and then average the probabilities for each of the class in order to get the most probable class for each instance of the dataset. It will call the predict method for each tree in the classifier. The class was optimized using the python package *multiprocessing* which allows us to generate a pool of threads that run a function's code for each item in a list. This way we can paralellize the computation of the predictions for each three among the desired number of jobs using:

```python
import multiprocessing as mp
from functools import partial
with mp.Pool(self.n_jobs) as p:
    predictions = list(p.map(partial(self._predict_tree, X=X), self.trees))
```

2.4 BaseForest

The main class for the ForestClassifiers. Implements the main fit methods for generating the trees from the train dataset. Also implements the methods to save the Classifiers as json objects. The fit method initializes the dataset characteristics and then creates NT trees and calls the fit method for each one of them. The same optimization that was done in the predict function in the ForestInterpreter class was done here, where each tree is fitted in a different thread to be able to paralellize the computation.

2.5 DecisionForestClassifier

The first final object is the DecisionForestClassifier, which inherits from BaseForest all the base methods for inference and training. The method is based on the [1] paper. The functions that the class implements is the *_fit_tree* method. In the case of the DecisionForestClassifier, the method first generates a dataset with F columns chosen at random from the dataset attributes. Once this dataset is generated, a tree is fitted with the newly constructed dataset where the per-node selection parameter F is set to -1 to exhaustively search for all (feature, value) pairs of the instances at each node.

2.6 RandomForestClassifier

Finally the RandomForestClassifier, which also inherits from BaseForest for the same reasons as DecisionForestClassifier, is implemented. The method is based on the [2] paper. Similarly to the DecisionForestClassifier, it only implements the *_fit_tree* method, which generates a bootstrapped dataset with the same number of instances as the original dataset. Then initializes the tree where the F value is set to the F value of the RandomForestClassifier to explore only F features at each node.

Chapter 3

Datasets

3.1 Datasets

The datasets used for the comparison were retrieved from the UCI dataset repository [3]. The datasets chosen are all exclusively categorical without any missing values, as those are the two aspects that prism cannot handle well. The datasets chosen are the Car Evaluation Data Set, kr-vs-kp and hayes-roth datasets.

3.1.1 Car Evaluation Data Set

Car Evaluation Database[4] was derived from a simple hierarchical decision model originally developed for the demonstration of DEX, M. Bohanec, V. Rajkovic: Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990.).

Input attributes are printed in lowercase. Besides the target concept (CAR), the model includes three intermediate concepts: PRICE, TECH, COMFORT. Every concept is in the original model related to its lower level descendants by a set of examples.

The Car Evaluation Database contains examples with the structural information removed, i.e., directly relates CAR to the six input attributes: buying, maint, doors, persons, lug_boot, safety.

Because of known underlying concept structure, this database may be particularly useful for testing constructive induction and structure discovery methods.

The characteristics of the dataset are as shown in 3.1

Data Set Characteristics:	Multivariate	Number of Instances:	1728	Area:	N/A
Attribute Characteristics:	Categorical	Number of Attributes:	6	Date Donated	1997-06-01
Associated Tasks:	Classification	Missing Values	No	Number of Web Hits:	1347720

Table 3.1: Car Evaluation Characteristics

3.1.2 Chess (King-Rook vs. King-Pawn) Data Set

The last dataset used in the project is the Chess (King-Rook vs. King-Pawn) Data Set [5], which consists of chess data from the match. The Dataset characteristics are shown in 3.2.

Data Set Characteristics:	Multivariate	Number of Instances:	3196	Area:	Game
Attribute Characteristics:	Categorical	Number of Attributes:	36	Date Donated	1989-08-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	125000

Table 3.2: Chess (King-Rook vs. King-Pawn) Characteristics

3.1.3 Iris Data Set

The last dataset used in the project is the Iris Data Set [6], which consists of measurements of iris flowers' petal and sepals. The Dataset characteristics are not shown because the website at the time of writing was down.

Chapter 4

Results Tables

Car DF

F	NT	fit_time	predict_time	fit_accuracy	predict_accuracy	buying	maint	doors	persons	lug_boot	safety
1	1	0.041	0.047	0.701	0.698	0	0	0	0	0	2
1	10	0.063	0.05	0.701	0.698	3	0	3	4	6	6
1	25	0.118	0.082	0.701	0.698	9	12	9	6	10	14
1	50	0.101	0.155	0.701	0.698	15	18	27	20	16	24
1	75	0.13	0.17	0.701	0.698	33	36	39	20	28	30
1	100	0.166	0.14	0.701	0.698	51	51	57	26	30	38
3	1	0.362	0.058	0.714	0.662	0	35	8	0	0	4
3	10	0.51	0.073	0.767	0.748	53	68	71	52	69	67
3	25	0.644	0.155	0.735	0.717	135	165	194	173	159	151
3	50	1.236	0.476	0.729	0.71	334	395	492	259	250	357
3	75	1.768	0.959	0.714	0.7	616	603	761	422	359	452
3	100	2.309	1.028	0.709	0.698	844	903	917	501	552	565
4	1	1.068	0.05	0.826	0.775	0	56	9	74	0	4
4	10	1.691	0.116	0.914	0.829	274	385	279	276	111	93
4	25	2.528	0.543	0.935	0.854	857	705	575	586	462	366
4	50	4.359	1.835	0.917	0.813	1505	1399	1284	1024	1128	1014
4	75	6.41	3.855	0.922	0.813	2042	1910	1884	1967	1615	1643
4	100	8.45	3.951	0.93	0.819	2732	2710	2326	2582	2166	2216
random	1	10.739	0.077	1.0	0.838	401	78	10	147	567	4
random	10	11.849	0.293	1.0	0.887	1547	572	908	394	639	341
random	25	12.286	1.339	1.0	0.871	2615	928	2526	2046	1030	1367
random	50	15.031	6.183	1.0	0.856	4282	2244	3419	4895	2752	2701
random	75	21.818	13.499	1.0	0.817	5294	3386	4369	6730	3775	3993
random	100	32.096	12.662	1.0	0.81	6788	5235	5708	8016	5284	5776

Car RF

F	NT	fit_time	predict_time	fit_accuracy	predict_accuracy	buying	maint	doors	persons	lug_boot	safety
1	1	0.357	0.047	0.718	0.71	10	18	11	15	10	12
1	10	0.612	0.087	0.723	0.712	130	130	106	139	92	107
1	25	0.693	0.244	0.706	0.702	292	271	249	277	220	237
1	50	1.221	0.594	0.711	0.7	564	551	525	494	426	488
1	75	1.701	1.498	0.708	0.702	831	846	807	727	654	723
1	100	2.342	1.381	0.716	0.7	1101	1109	1079	967	899	986
2	1	1.424	0.054	0.776	0.723	52	51	64	2	45	45
2	10	2.158	0.21	0.95	0.844	601	519	501	431	407	464
2	25	3.829	0.901	0.971	0.844	1454	1146	1269	1037	1065	1187
2	50	6.119	3.88	0.988	0.829	2790	2458	2581	2027	2000	2161
2	75	8.699	8.701	0.988	0.837	4074	3820	3837	3085	3020	3205
2	100	11.838	9.678	0.988	0.848	5468	4980	5111	4130	3987	4253
3	1	3.054	0.057	0.819	0.675	46	114	71	106	42	90
3	10	3.794	0.299	0.985	0.819	863	950	787	900	569	811
3	25	7.765	1.656	0.998	0.858	2351	2153	1979	2152	1698	1979
3	50	13.191	7.71	1.0	0.883	4637	4600	4318	3946	3594	3711
3	75	18.113	17.485	1.0	0.888	7059	7153	6496	5506	5371	5547
3	100	26.213	16.234	1.0	0.888	9524	9360	8830	7221	7101	7482

Iris DF

F	NT	fit_time	predict_time	fit_accuracy	predict_accuracy	sepal_length	sepal_width	petal_length	petal_width
1	1	0.385	0.038	0.962	0.911	0	0	38	0
1	10	0.394	0.044	0.99	0.889	96	21	152	36
1	25	0.467	0.145	0.981	0.867	256	84	266	108
1	50	0.877	0.316	0.981	0.867	448	210	532	216
1	75	1.218	0.604	0.981	0.889	736	294	912	252
1	100	1.716	0.563	0.981	0.867	960	462	1140	324
2	1	0.701	0.032	0.99	0.911	0	0	43	34
2	10	1.016	0.075	1.0	0.889	104	281	224	204
2	25	1.748	0.179	1.0	0.911	374	593	566	500
2	50	2.919	0.788	1.0	0.933	949	989	1284	892
2	75	4.23	1.919	1.0	0.911	1549	1514	1815	1320
2	100	6.303	1.851	1.0	0.911	2173	2084	2342	1692
3	1	1.289	0.032	1.0	0.933	0	23	43	35
3	10	1.755	0.075	1.0	0.911	193	343	232	224
3	25	3.125	0.357	1.0	0.911	604	687	641	548
3	50	5.181	1.017	1.0	0.889	1283	1191	1459	1039
3	75	7.213	2.142	1.0	0.933	1978	1817	2095	1566
3	100	10.577	2.484	1.0	0.889	2708	2485	2695	2047
random	1	1.688	0.037	1.0	0.933	0	23	43	35
random	10	1.962	0.065	1.0	0.933	139	325	235	168
random	25	2.451	0.233	1.0	0.933	467	643	569	436
random	50	4.983	0.884	1.0	0.911	960	1048	1301	790
random	75	6.91	1.721	1.0	0.911	1529	1496	1878	1164
random	100	9.341	1.921	1.0	0.889	2135	2073	2391	1617

Iris RF

F	NT	fit_time	predict_time	fit_accuracy	predict_accuracy	sepal_length	sepal_width	petal_length	petal_width
1	1	0.317	0.032	0.962	0.756	14	13	13	12
1	10	0.52	0.054	1.0	0.911	145	112	153	118
1	25	0.882	0.197	1.0	0.956	352	317	350	310
1	50	1.344	0.771	1.0	0.933	724	652	677	604
1	75	1.992	1.268	1.0	0.933	1051	1003	1023	924
1	100	2.806	1.139	1.0	0.911	1403	1364	1344	1234
2	1	0.634	0.04	0.971	0.911	17	17	17	13
2	10	1.005	0.063	1.0	0.933	154	149	158	149
2	25	1.606	0.184	1.0	0.911	401	396	398	361
2	50	2.576	0.624	1.0	0.889	818	779	812	712
2	75	3.7	1.446	1.0	0.911	1220	1174	1247	1060
2	100	5.387	1.245	1.0	0.911	1650	1578	1646	1388
3	1	0.856	0.04	1.0	0.911	20	12	18	14
3	10	1.195	0.055	0.99	0.956	175	154	161	135
3	25	2.454	0.248	1.0	0.956	421	416	416	335
3	50	3.979	0.611	1.0	0.956	867	814	841	673
3	75	5.316	1.823	1.0	0.911	1306	1222	1277	1012
3	100	7.694	1.387	1.0	0.911	1719	1627	1713	1347

Chess RF

F	NT	fit_time	predict_time	fit_accuracy	predict_accuracy	bkblk	bknwy	bkon8	bkona	bkspr	bkxbq	bkxcr	bkxwp	blxwp	bxqsq	cntxt	dsopp	dwipd	hdchk	katri	mulch	qxmsq	r2ar8	reskd	reskr	rimmx	rkxwp	rxmsq	simpl	skach	skewr	skrxp	spcop	stlmt	thrsk	wkcti	wkna8	wknck	wkovl	wkpos	wtoeg
1	1	0.181	0.064	0.714	0.702	1	0	1	1	1	0	1	0	1	1	0	0	0	1	0	0	1	0	0	0	0	1	1	0	1	1	1	0	1	0	2	1	1	1	1	0
1	10	0.23	0.114	0.802	0.828	6	6	3	8	13	8	11	4	9	9	10	4	12	2	5	8	5	9	2	6	6	5	2	11	6	8	6	0	5	5	11	5	11	6	7	8
1	25	0.363	0.218	0.859	0.86	17	17	11	14	25	23	24	14	22	27	21	15	22	4	18	20	14	23	7	23	17	13	9	28	9	17	18	0	12	19	24	17	24	20	18	15
1	50	0.632	0.5	0.873	0.885	37	39	24	38	55	50	44	32	47	54	47	38	42	10	46	37	24	49	17	46	45	30	23	57	19	31	39	2	17	33	47	30	56	50	51	30
1	75	0.787	1.243	0.896	0.908	61	62	42	58	75	74	64	51	78	82	69	64	69	17	85	57	33	70	25	69	67	53	40	87	23	50	60	4	32	52	74	47	79	79	72	53
1	100	1.042	1.106	0.903	0.911	79	84	54	82	95	107	92	74	103	109	98	91	93	27	109	77	45	99	34	94	88	72	62	122	34	66	78	7	43	66	102	64	107	96	101	78
3	1	0.884	0.075	0.703	0.692	2	1	2	5	10	9	6	2	2	6	5	2	7	0	4	3	1	7	1	7	5	3	4	7	1	6	5	0	0	0	2	3	3	5	3	4
3	10	1.748	0.243	0.946	0.929	41	36	27	71	95	64	62	52	57	80	70	55	92	6	80	41	17	99	12	74	62	55	38	114	8	50	37	1	14	30	48	27	82	85	62	55
3	25	2.726	1.102	0.979	0.959	120	108	81	192	244	216	167	141	150	214	170	136	200	15	226	112	66	224	26	179	180	162	91	288	27	157	101	2	32	87	148	90	209	227	184	125
3	50	4.853	4.163	0.98	0.965	255	212	167	364	484	453	388	282	320	414	348	278	412	34	464	240	133	470	56	341	381	308	183	578	49	307	202	7	67	174	302	172	455	461	391	277
3	75	7.136	11.535	0.982	0.962	408	342	253	527	714	728	604	434	508	635	513	434	601	54	687	352	197	707	77	501	562	501	274	837	76	479	300	11	97	269	449	257	682	699	582	438
3	100	10.183	10.093	0.983	0.967	548	456	331	678	913	955	800	584	693	836	719	569	801	72	916	475	264	916	106	669	750	687	361	1098	103	638	400	16	133	349	609	342	922	909	783	589
5	1	3.752	0.076	0.819	0.775	10	9	9	11	21	20	16	9	10	23	1	8	24	1	6	5	7	18	3	14	9	17	6	28	1	15	7	0	0	2	15	1	21	22	12	1
5	10	4.56	0.426	0.968	0.935	108	89	55	140	182	186	177	120	157	201	145	111	209	11	185	81	52	193	14	135	130	145	69	248	13	176	81	4	22	61	137	47	228	173	171	137
5	25	8.633	2.446	0.989	0.96	247	199	142	325	493	552	411	269	370	443	326	281	491	25	459	190	114	445	43	338	318	363	163	647	38	404	214	9	44	165	309	119	628	431	364	371
5	50	14.203	11.391	0.992	0.966	495	422	299	668	944	1157	880	568	715	922	679	540	999	45	897	398	228	893	87	684	668	690	300	1300	73	791	428	16	92	327	577	248	1243	897	761	718
5	75	19.246	27.777	0.994	0.974	755	626	434	975	1415	1710	1333	814	1014	1393	978	785	1443	66	1342	609	342	1384	126	978	1046	963	482	1930	115	1173	610	25	139	493	836	371	1716	1402	1154	1062
5	100	27.992	24.501	0.996	0.972	1042	842	585	1313	1906	2362	1819	1104	1371	1850	1312	1048	1924	88	1819	812	440	1781	177	1304	1416	1298	656	2517	158	1529	806	32	189	665	1136	489	2259	1834	1515	1472
6	1	3.964	0.085	0.83	0.793	10	11	7	14	28	29	25	10	20	17	18	8	30	1	6	14	4	27	1	15	19	12	1	28	1	22	7	0	3	3	15	8	24	14	13	3
6	10	5.381	0.471	0.987	0.957	114	90	77	185	244	257	214	113	163	226	135	165	289	14	212	112	49	233	23	169	161	134	58	382	13	209	90	3	21	74	162	47	272	218	193	173
6	25	11.538	3.54	0.994	0.967	277	230	179	458	607	692	562	328	419	555	391	386	617	31	590	246	135	599	50	361	465	345	145	873	38	528	270	7	53	198	346	138	690	555	505	418
6	50	19.698	17.305	0.994	0.972	595	457	346	861	1229	1324	1063	660	836	1105	813	751	1226	58	1161	507	270	1136	98	773	943	709	294	1700	80	1039	506	15	104	377	611	307	1426	1101	994	822
6	75	27.073	31.77	0.996	0.974	886	690	501	1278	1802	2093	1579	1001	1248	1599	1242	1153	1804	94	1711	747	396	1714	148	1149	1406	1064	506	2529	128	1533	714	22	150	596	953	470	2061	1677	1515	1224
6	100	37.926	33.883	0.996	0.98	1209	910	668	1651	2449	2786	2043	1334	1694	2130	1652	1452	2427	128	2277	1003	530	2294	206	1516	1934	1468	682	3308	165	1991	942	31	208	777	1334	603	2784	2269	1986	1661

Chess DF

F	NT	fit_time	predict_time	fit_accuracy	predict_accuracy	bkblk	bknwy	bkon8	bkona	bkspr	bkxbq	bkxcr	bkxwp	blxwp	bxqsq	cntxt	dsopp	dwipd	hdchk	katri	mulch	qxmsq	r2ar8	reskd	reskr	rimmx	rkxwp	rxmsq	simpl	skach	skewr	skrxp	spcop	stlmt	thrsk	wkcti	wkna8	wknck	wkovl	wkpos	wtoeg
9	1	0.753	0.062	0.682	0.694	0	0	9	0	0	0	0	0	0	0	6	16	0	0	0	6	1	0	0	0	6	0	6	0	0	0	0	0	0	0	4	4	0	0	0	0
9	10	1.607	0.121	0.912	0.902	4	9	22	33	3	9	0	16	0	17	6	23	4	14	0	37	2	35	8	37	23	40	22	66	11	4	18	2	15	32	43	20	19	0	26	0
9	25	2.081	0.397	0.901	0.895	5	42	32	115	16	32	17	38	37	46	34	60	94	25	25	110	5	89	33	56	35	100	45	129	20	42	66	5	40	50	76	37	72	46	54	4
9	50	3.77	0.91	0.941	0.933	18	93	67	164	120	108	65	160	89	180	119	134	221	35	140	136	28	139	52	83	115	180	85	160	25	71	96	11	62	101	191	82	163	137	74	20
9	75	5.312	1.898	0.934	0.922	23	111	132	248	206	148	114	225	119	195	132	168	325	55	275	161	48	259	82	172	163	281	127	307	37	122	133	20	113	114	246	118	255	208	139	43
9	100	7.815	1.977	0.936	0.915	72	148	199	291	209	228	214	272	146	314	242	244	464	63	378	189	79	326	97	216	221	346	201	475	48	186	163	25	141	185	276	173	344	318	162	80
18	1	6.543	0.082	0.866	0.872	0	0	9	0	0	88	0	0	59	0	7	18	0	4	0	12	1	0	0	0	8	0	8	0	0	0	0	1	6	16	3	7	56	40	0	20
18	10	11.831	0.418	0.995	0.966	36	118	88	150	242	334	193	70	59	96	115	181	81	22	369	65	31	175	14	183	59	144	45	259	30	36	69	4	20	107	213	53	351	258	248	208
18	25	21.988	2.441	0.996	0.969	92	330	142	539	484	882	353	168	168	418	322	371	237	47	854	217	98	616	51	495	302	339	145	1215	57	298	198	12	63	237	462	168	837	677	654	410
18	50	38.967	10.083	0.995	0.974	275	512	352	845	1002	1882	667	453	599	1032	611	729	813	91	1407	622	176	1059	86	976	588	651	369	1538	101	827	549	23	131	427	881	344	1455	1422	994	936
18	75	57.152	21.748	0.995	0.974	484	651	621	1243	1643	2403	991	774	945	1588	1130	940	1330	126	1978	816	288	1493	155	1347	899	933	515	2126	154	1383	742	37	235	634	1170	563	2032	2147	1465	1450
18	100	79.206	22.955	0.996	0.973	571	788	890	1673	2420	3037	1478	1029	1310	2386	1566	1410	2258	165	2493	1020	449	2287	215	1540	1181	1281	773	3161	196	2212	966	47	307	830	1512	710	2940	2764	2063	1796
27	1	28.116	0.106	0.932	0.893	0	31	14	0	0	140	102	38	78	0	9	23	149	4	79	12	1	107	0	0	8	0	9	0	8	147	24	1	7	18	3	7	72	37	0	23
27	10	47.927	1.343	1.0	0.961	88	164	141	278	606	620	915	344	134	372	473	253	1239	21	835	166	74	603	27	194	395	803	72	749	57	715	123	5	24	268	354	112	1152	626	286	659
27	25	77.836	8.75	1.0	0.97	414	424	361	846	1736	2297	1497	494	363	1202	1141	838	1627	50	1869	540	246	1676	88	809	1214	1487	266	2328	113	1434	366	15	80	533	878	236	2875	1465	1156	1469
27	50	132.166	37.175	1.0	0.979	903	918	707	1605	2944	3769	2433	1006	1499	2231	1788	1853	3014	110	3758	1404	327	3012	195	1519	2356	2271	729	5197	220	3018	975	34	165	937	1598	533	4813	2891	2574	2741
27	75	206.855	76.441	1.0	0.977	1387	1333	1171	2563	4778	5577	3648	1672	2368	3538	2736	2740	5125	153	5163	1895	578	4146	298	2280	3380	3058	960	7545	299	4175	1658	54	294	1546	2241	896	7012	4199	3859	4259
27	100	287.775	81.341	1.0	0.976	1788	1719	1543	3563	6601	7310	4699	2590	3218	5133	3646	4014	7264	205	6662	2354	928	5970	383	2999	4593	4094	1292	9872	400	5340	2076	69	372	1977	3326	1197	9460	5185	5327	5749
random	1	28.102	0.098	0.911	0.878	0	32	10	0	0	140	0	37	78	0	9	23	138	4	78	12	1	98	0	0	8	0	9	0	0	117	24	1	7	18	3	7	79	38	0	22
random	10	74.725	0.689	1.0	0.942	45	161	64	105	149	455	197	113	78	221	162	305	627	17	429	95	63	434	20	87	197	310	61	401	30	428	96	3	21	135	210	61	415	370	234	538
random	25	84.951	4.729	1.0	0.958	185	411	190	383	856	958	286	239	169	708	683	545	1265	37	866	260	177	944	70	531	801	633	199	1132	64	869	221	8	49	346	544	115	1120	962	849	1207
random	50	118.186	22.508	1.0	0.975	642	751	424	1036	1980	2461	1260	616	798	1576	1312	1230	2234	70	2165	766	250	1999	161	1162	1617	1211	466	2608	142	1775	637	22	113	671	1066	320	2586	1565	1835	2314
random	75	148.477	54.12	1.0	0.973	946	911	668	1507	3255	4251	1800	992	1120	2458	1869	1710	3628	97	2999	1097	350	2781	211	1793	2059	1552	724	4090	207	2398	1017	37	198	987	1482	503	3631	2579	2424	3232
random	100	262.086	45.772	1.0	0.972	1162	1187	916	2223	4504	5510	2368	1641	1522	3156	2477	2361	4411	140	3816	1451	602	3950	272	2399	2926	2073	963	5378	267	3337	1294	47	253	1328	2199	724	5112	3237	3315	4216

Bibliography

[1] Tin Kam Ho. “The random subspace method for constructing decision forests”. In: *IEEE transactions on pattern analysis and machine intelligence* 20.8 (1998), pp. 832–844.

[2] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[3] *UCI Machine Learning Repository*. URL: `https://archive.ics.uci.edu/ml/index.php`.

[4] *Car Evaluation Data Set*. URL: `https://archive.ics.uci.edu/ml/datasets/car+evaluation`.

[5] *Chess (King-Rook vs. King-Pawn) Data Set*. URL: `https://archive.ics.uci.edu/ml/datasets/Chess+(King-Rook+vs.+King-Pawn)`.

[6] *Iris Data Set*. URL: `https://archive.ics.uci.edu/ml/datasets/iris`.