Machine learning – Final Project

Doron Laadan
Inbal Roshanski

1. Algorithm Name

ECSDT – Ensemble of Example-Dependent Cost-Sensitive Decision Trees.

2. Reference

Bahnsen, A. C., Aouada, D., & Ottersten, B. (2015). Ensemble of example-dependent cost-sensitive decision trees. *arXiv preprint arXiv:1505.04637*.

3. Motivation for the algorithm (or which problems it tries to solve?)

Most classification algorithms aim at minimizing the misclassification of examples, in which an example is misclassified if the predicted class is different from the true class. Such a traditional framework assumes that all misclassification errors carry the same cost. This is not the case in many real-world applications. Methods that use different misclassification costs are known as cost-sensitive classifiers. Typical cost sensitive approaches assume a constant cost for each type of error, in the sense that, the cost depends on the class and is the same among examples, but this approach is not realistic in failing to detect a fraudulent transaction may have an economic impact from a few to thousands of Euros, depending on the particular transaction and card holder.
In order to deal with these specific types of cost-sensitive problems, called example-dependent cost-sensitive several methods were suggested by the authors such as cost-sensitive decision tree (CSDT). However, the CSDT algorithm only creates one tree in order to make a classification, and individual decision trees typically suffer from high variance. A very efficient and simple way to address this flaw is to use them in the context of ensemble method (ECSDT).

4. Short Description:

ECSDT is an ensemble of example-dependent cost-sensitive decision-trees, built by training example-dependent cost-sensitive decision trees using four different random inducer methods (bagging, pasting, random forest, random patches) and then blending them using three different combination approaches, majority voting and two others that are new cost-sensitive combination approaches suggested by the authors (cost-sensitive weighted voting and cost sensitive stacking).

5. Pseudo-Code

$\underline{\boldsymbol{ECSDT - Building\ the\ ensemble(fit)}}$

$input$:

$S - a\ labeled\ training\ set$

$I - Inducer\ method$

$T - number\ of\ iterations$

$N_e - number\ of\ samples\ for\ each\ base\ classifier$

$N_f - number\ of\ features\ for\ each\ base\ classifier$

$C - Combiner\ methods$

$\underline{step\ 1: Create\ the\ set\ of\ base\ classifiers}$

for j=1…T do:

switch(inducer):

case Bagging:

$S_j = sample\ N_e\ examples\ from\ S\ with\ replacment.$

case Pasting:

$S_j = sample\ N_e\ examples\ from\ S\ without\ replacment.$

case Random Forest:

$S_j = sample\ N_e examples\ \ from\ S\ with\ replacment.$

case Random Patches:

$S_j = sample\ N_e\ examples\ and\ N_f$

$features\ from\ S\ with\ replacment.$

End switch

$M_j = CSDT(S_J)$

$S_J^{oob} = S - S_j$

$\alpha_j = Saving(M_j(S_j^{oob}))$

End for

Set combiner to C

If combiner is stacking:

$\beta = argmin_{\beta \in R^T} J(S, M, \beta)$

stacking_clf = $g(\sum_{j=1}^{T} \beta_j (M_j(S)))$

Output: ensemble of cost sensitive decision trees

$\underline{\boldsymbol{ECSDT - Classify\ an\ instance(predict)}}$

$input$:

$x - an\ instance\ needed\ to\ be\ labeld$

$\underline{step\ 2: Combine\ the\ base\ classifiers}$

switch(combiner):

case Majority voting:

$H(x) = argmax_{c \in \{0,1\}} \sum_{j=1}^{T} 1_c(M_j(x))$

case Cost-sensitive-weighted voting:

$H(x) = argmax_{c \in \{0,1\}} \sum_{j=1}^{T} \alpha_j 1_c(M_j(x))$

Case Cost-sensitive-stacking:

$H(x) = g(\sum_{j=1}^{T} \beta_j (M_j(x)))$

Return H(x)

6. Algorithm Explanation:

Creating the ensemble(fit):
In line 1, the for loop is set to executed T times, meaning there will be T trees in the final ensemble, for each tree the following happens:
In lines 2-11 depending on the inducer method, a subset of examples and features from the training set is chosen.
In line 12 a cost sensitive decision tree classifier is trained on the subset created before.
In line 13 a subset of "out of bag" examples is created by removing the examples that were used to train the classifier from the original training set. Then in line 14 the classifier adjusted weight is calculated by measuring the Saving score of the classifier on the out of bag set.
In line 16 the ensemble combiner method is set according to the user input.
If the method is "stacking" the coefficients for the cost-sensitive regression classifier are learned (lines 17 and 18).
In line 19 the second level classifier is trained using the entire training set and the coefficients.

Classify an instance (predict):
In line 1, depending on the combiner that was set for the ensemble in training phase The prediction is made.
in line 2 if the combiner majority voting is used to predict the label for the new instance as can be seen in line 3 where 1_c is an indicator function that return 1 for the class the model predicted.
In lines 4 and 5 Cost-sensitive-weighted voting is used, which is a the same as majority voting while considering the weight of the classifiers in the ensemble as calculated in the training phase.
In lines 6 and 7 cost-sensitive stacking is used which means each of the classifiers in the ensemble predict the value for the instances and those predictions are then used as features for the second level classifier model.

7. Illustration

To illustrate the essence of the algorithm we will use one of the datasets that were used in the original article and is available through the *costcla* python package. We will use the credit scoring Kaggle Credit competition dataset (classification).
It has 1129150 records and 10 features and it's a binary classification problem. For the purpose of this illustration we will show only a sample of the data in each phase.
First, for the purpose of the algorithm each dataset is comprised of 3 parts.
Data. which contains the records with all the features.
Target. which is the label for each record.
Cost matrix. which represent the example dependent cost of each record.
For the credit score data set it looks like that:

```
dataset = {Bunch} {'data': array([[ 0.76612661, 45.      , 2.      , ..., 6.      ,\n
  ▶ 'data' (1620334300216) = {ndarray} ...View as Array
  ▶ 'target' (1620334180872) = {ndarray} ...View as Array
  ▶ 'cost_mat' (1620382836208) = {ndarray} ...View as Array
```

An example of the records as seen if use dataset["data"]:

	0	1	2	3	4	5	6	7	8	9
0	0.488790684	49.0	0.0	0.732504728	3700.0	7.0	0.0	1.0	0.0	0.0
1	0.441222424	41.0	1.0	0.353097935	1500.0	5.0	0.0	0.0	2.0	0.0
2	0.020732642	40.0	0.0	0.462585951	7416.0	9.0	0.0	2.0	0.0	2.0
3	0.0	62.0	0.0	0.06570675299999999	4930.0	9.0	0.0	0.0	0.0	0.0
4	0.03018754	37.0	0.0	0.406038756	8875.0	5.0	0.0	3.0	0.0	5.0
5	0.013155287	62.0	0.0	0.311135775	8108.0	7.0	0.0	1.0	0.0	0.0
6	0.274180158	24.0	0.0	0.383122611	3400.0	8.0	0.0	1.0	0.0	3.0
7	0.056631445999999995	55.0	0.0	0.15161696800000002	4761.0	3.0	0.0	1.0	0.0	0.0
8	0.46286342799999997	28.0	0.0	0.21176160300000002	3791.0	3.0	0.0	0.0	0.0	0.0
9	0.664078892	57.0	0.0	0.022833712000000003	49750.0	17.0	0.0	0.0	0.0	0.0
10	0.037529234	67.0	0.0	0.462218089	4300.0	10.0	0.0	1.0	0.0	0.0

An example of the label for the same records as seen if we use dataset["target"]:

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	1	0	0	0	0	0	0	0	0

And the cost matrix for the records as seen if we use dataset["cost_mat]:

	0	1	2	3
0	757.4798465535023	8324.25	0.0	0.0
1	631.0677478822363	3374.25	0.0	0.0
2	971.0013732182415	16685.25	0.0	0.0
3	828.1557017197138	11091.75	0.0	0.0
4	1023.7305410427907	18750.0	0.0	0.0
5	1010.7637242548408	18242.25	0.0	0.0
6	740.241833098331	7649.25	0.0	0.0
7	818.4449541399645	10711.5	0.0	0.0
8	762.7087106349063	8529.0	0.0	0.0
9	1023.7305410427907	18750.0	0.0	0.0
10	791.9558734638504	9674.25	0.0	0.0

The cost matrix represents the TP/FP table as seen here:

	Actual Positive ($y_i = 1$)	Actual Negative ($y_i = 0$)
Predicted Positive ($c_i = 1$)	$C_{TP_i} = 0$	$C_{FP_i} = r_i + C^a_{FP}$
Predicted Negative ($c_i = 0$)	$C_{FN_i} = Cl_i \cdot L_{gd}$	$C_{TN_i} = 0$

For all data set the cost of a TP or TN is set to 0 and the cost of a FP and FN is set for each example in the context of the dataset. For this dataset the authors used a cost metric as explain here : https://nbviewer.jupyter.org/github/albahnsen/CostSensitiveClassification/blob/master/doc/tutorials/tutorial_edcs_credit_scoring.ipynb.

Next in the fit function for each iteration, as describe in the algorithm we sample $N_e record\ and\ N_f\ features$, according to the inducer method.

So, for bagging for example (selecting random features without replacement) we get the next features out of 10:

	0	1	2	3	4	5	6
0	1	3	9	8	4	5	7

And the next rows out of the training set:

	0	1	2	3	4	5	6	7	8	9	10
0	55758	70626	13067	29941	40099	21643	73139	11518	51051	25215	8481

We then create the training set for the tree by using the records and features chosen before from the original training set. (The number in the header aren't the original feature numbers) as well as their corresponding label and cost matrix

	0	1	2	3	4	5	6
0	27.0	0.265306122	0.0	0.0	2400.0	4.0	0.0
1	24.0	0.130217446	0.0	0.0	4000.0	6.0	0.0
2	36.0	0.47421638	1.0	0.0	5933.0	10.0	1.0
3	84.0	0.000965484	0.0	0.0	4142.0	1.0	0.0
4	47.0	0.20060944600000002	0.0	0.0	3937.0	10.0	0.0
5	65.0	0.20062819	0.0	0.0	10187.0	4.0	2.0
6	62.0	0.8832634629999999	0.0	0.0	3100.0	12.0	2.0
7	44.0	0.558384317	3.0	0.0	15200.0	29.0	4.0
8	64.0	0.266273373	0.0	0.0	10000.0	5.0	0.0
9	54.0	0.41228919	1.0	0.0	14467.0	22.0	3.0
10	50.0	0.348772239	1.0	1.0	6800.0	29.0	0.0

We also create the out of bag set which uses all the examples not chosen for the training set on the same features (they will be used as test set to compute the saving score of the tree later). This process is repeated for each tree in the ensemble.

So for 5 iteration we get the following ensemble as a list which can be accessed by using self.models:

```
▶ 0 = {CostSensitiveDecisionTreeClassifier} CostSensitiveDecisionTreeClassifier(c
▶ 1 = {CostSensitiveDecisionTreeClassifier} CostSensitiveDecisionTreeClassifier(c
▶ 2 = {CostSensitiveDecisionTreeClassifier} CostSensitiveDecisionTreeClassifier(c
▶ 3 = {CostSensitiveDecisionTreeClassifier} CostSensitiveDecisionTreeClassifier(c
▶ 4 = {CostSensitiveDecisionTreeClassifier} CostSensitiveDecisionTreeClassifier(c
```

For each model we save the features use to train it:

```
▶ features_drawn = {list} <class 'list'>: [array([1, 3, 9, 8, 4, 5, 7]), array([7, 4, 9, 3, 2, 6, 8]), array([6, 1, 8, 4, 7, 9, 0]), array([1, 0, 9, 8, 5, 4, 3]), array([3, 7, 1, 5, 8, 4, 0])]
```

Where each feature list is corresponding to the model in the same index.

We also save the weights or α_j for each model as can be seen here:

	0	1	2	3	4
0	0.1765214107432489	0.42120269754032746	0.42922655993319836	0.3949226853696748	0.3388307708216538

According to the paper we then normalize the weights using the proposed formula:

$$\alpha_j = \frac{Savings(M_j(\mathcal{S}_j^{oob}))}{\sum_{j_1=1}^{T} Savings(M_{j_1}(\mathcal{S}_j^{oob}))}.$$

And get the normalized weights:

	0	1	2	3	4
0	0.10025614655874689	0.23922400799846105	0.24378119752374164	0.22429815429803468	0.1924404936210158

If the combiner method selected is "stacking" we need to build the second level model. we used, as in the paper, a cost sensitive logistic regression which we train on the original dataset (all the records) while the features are the results of the prediction of each tree in the ensemble for each record. For our example we will get a matrix with 5 features as can be seen next.

	0	1	2	3	4
0	1.0	0.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	1.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0
8	1.0	0.0	0.0	0.0	0.0
9	0.0	0.0	1.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0

The logistic regression is then trained to evaluate the coefficient of the model for each tree as can be seen here

	0	1	2	3	4
0	-0.03238692326223753	-0.0457493515797758	-0.0050951708529222905	0.024754859217559325	-0.019997991622983997

We then return the fitted ensemble model.

For the prediction of the ensemble we first create a matrix with number of rows equal to the number of records in the test set with number of columns corresponding to the number of expected classes.

According to the combiner method for the ensemble we then aggregate the prediction per model for each record.

For Majority voting we get:

	0	1
0	5.0	0.0
1	5.0	0.0
2	2.0	3.0
3	1.0	4.0
4	5.0	0.0
5	2.0	3.0
6	4.0	1.0
7	3.0	2.0
8	5.0	0.0
9	5.0	0.0
10	5.0	0.0

So, the final classification will be:

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	1	1	0	1	0	0	0	0	0

For Cost-sensitive-weighted voting we take into account the weight of each model, so we get:

	0	1
0	0.9999999999999999	0.0
1	0.8098311461898318	0.19016885381016813
2	0.5877946562236058	0.41220534377639406
3	0.1765730082634481	0.8234269917365519
4	0.9999999999999999	0.0
5	0.37184130594744436	0.6281586940525555
6	0.9999999999999999	0.0
7	0.6145628485058355	0.3854371514941644
8	0.9999999999999999	0.0
9	0.9999999999999999	0.0
10	0.8098311461898318	0.19016885381016813

So, the final classification will be:

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	1	0	1	0	0	0	0	0

For Cost-sensitive-stacking we get the testing set need to be classified and then create a matrix similar for the one we used for the fitting stage to create the set for the regression model.

	0	1	2	3	4
0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0	0.0
3	1.0	1.0	1.0	1.0	0.0
4	0.0	0.0	0.0	0.0	0.0
5	1.0	0.0	1.0	1.0	0.0
6	0.0	0.0	0.0	0.0	0.0
7	1.0	1.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0

And then we use the regression model predict method to predict the classes.

8. Strengths

- "Combining individual cost-sensitive classifiers achieves better results in the sense of higher savings."
- "The proposed method outperforms state-of-the-art example-dependent cost-sensitive methods."
- "the proposed algorithm is simple."
- The algorithm reduces the variance of a single CSDT.

9. Drawbacks

- " it is important to use the real practical financial costs of each context" but such cost metrices are not easily obtainable for each dataset. For our experiments we used a random cost matrix for each dataset in which the numbers were distributes from the same distribution we saw at the *costcla* data sets.
- "the methods covered in this work are all batch, in the sense that the batch algorithms keep the system weights constant while calculating the evaluation measures. However, in some applications such as fraud detection, the evolving patters due to change in the fraudsters behavior is not capture by using batch methods."
- The running time CSDT is quite long and so the running time for the ensemble is even larger with more iterations. Moreover, when stacking is used as the combiner method the fitting time increases even further on big datasets.
- The method only works for binary classification now as we understand because of the saving metric being used to build the underline CSDT and calculating the weights.
- In our implementation: scores are not as good as reported in the paper.

10. Experimental Results

A. Measures (Accuracy and Runtime):

1. F1Score
2. Saving
3. Fit Time
4. Prediction time

B. Ensemble size (or any other important parameters of the algorithm)

1. 10
2. 20
3. 30

C. Combiner method

1. MV (Majority Voting)
2. CSMV (Cost Sensitive Weighted Voting)
3. CSS (Cost Sensitive Stacking)

D. Inducer Methods

1. Bagging
2. Pasting
3. Random Forest
4. Random Patches

E. Baselines:

1. Random Forest
2. Cost Sensitive Decision Tree (CSDT)

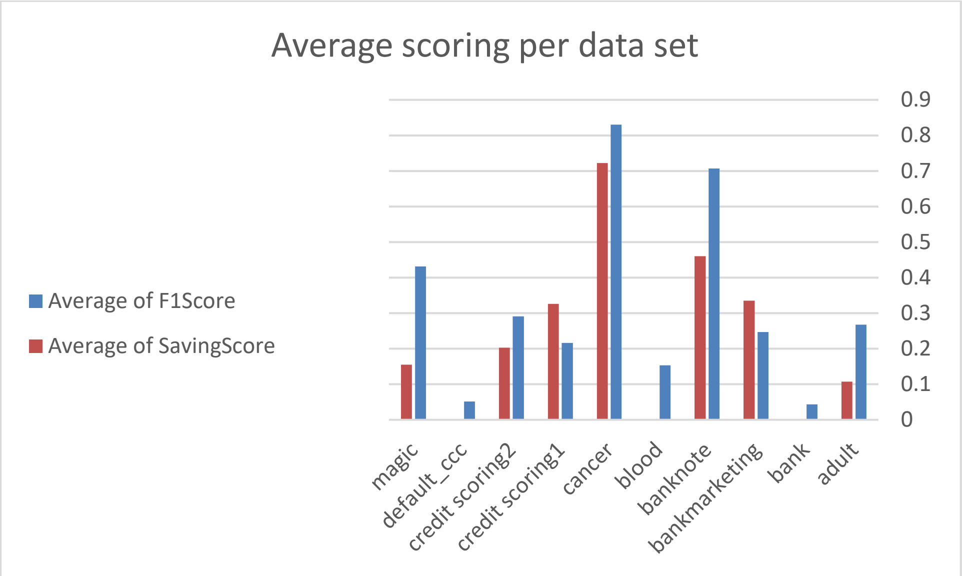
F. Datasets

Dataset name	Number of Instances	Number of Attributes	Short description
Adult	48842	14	Predict whether income exceeds $50K/yr based on census data.
Kaggle credit (costcla)	112,915	10	The objective is to identify customers of personal loans that will experience financial distress in the next two years
PAKDD credit (costcla)	38,969	30	The objective is to identify which credit card applicants were likely to default and by doing so deciding whether to approve their applications.
Bank marketing (costcla)	37931	31	The dataset consists of fraudulent and legitimate transactions made with credit and debit cards between January 2012 and June 2013.
MAGIC Gamma Telescope	19020	11	Data are MC generated to simulate registration of high energy gamma particles in an atmospheric Cherenkov telescope
Bank marketing	45211	17	direct marketing campaigns of a Portuguese banking institution
Blood Transfusion	748	5	Data taken from the Blood Transfusion Service Center in Hsin-Chu City in Taiwan
Breast Cancer Wisconsin (Original)	699	10	Information about 699 breast cancer patients. The objective is to decide whether a tumor is benign or malignant
default of credit card clients	30000	24	This research aimed at the case of customers default payments in Taiwan and its objective is to distinguish between credible or not credible clients
banknote authentication	1372	5	Data were extracted from images that were taken for the evaluation of an authentication procedure for banknotes.

Results:

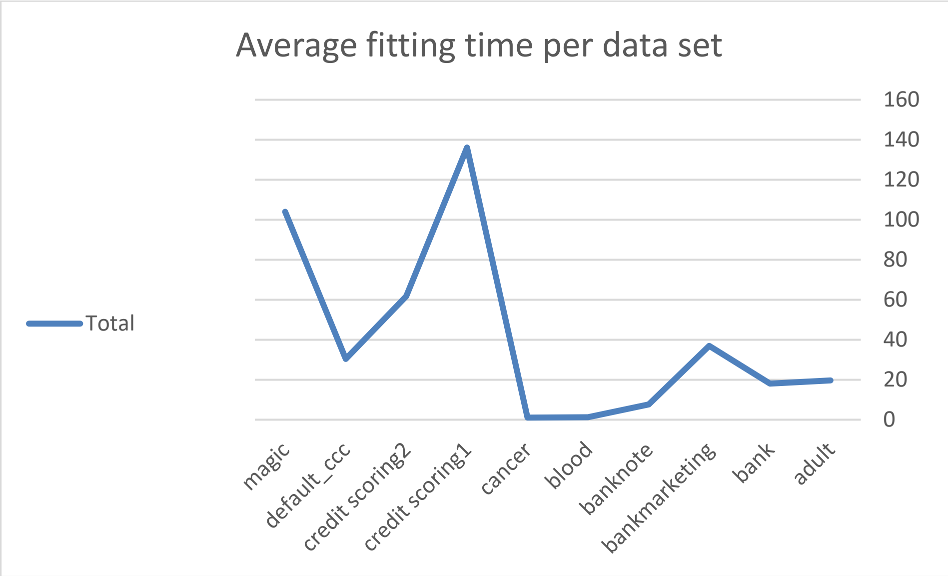
<u>Total average of scores across all dataset and all combinations:</u>

Average of SavingScore	Average of F1Score	Row Labels
0.107634355	0.267508638	adult
0	0.043140596	bank
0.33487469	0.247171175	bankmarketing
0.46022298	0.707147335	banknote
0.000458329	0.153478499	blood
0.722222222	0.83014523	cancer
0.32644642	0.21615725	credit scoring1
0.202423103	0.290693314	credit scoring2
0	0.051264608	default_ccc
0.154998968	0.431483759	magic
0.230928107	**0.32381904**	**Grand Total**



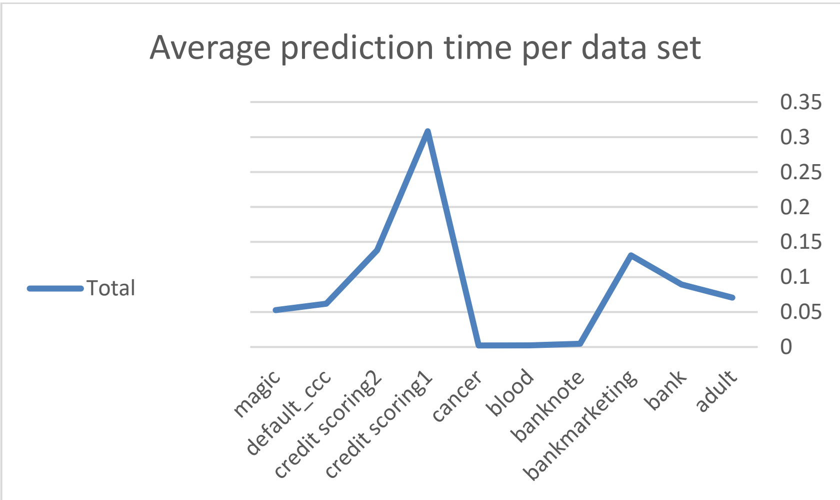
<u>Average fit time:</u>

Average of FitTIme	Row Labels
19.64344649	adult
18.00969951	bank
36.94582357	bankmarketing
7.597493874	banknote
1.183896588	blood
1.053685129	cancer
136.1063169	credit scoring1
61.80836965	credit scoring2
30.32044195	default_ccc
104.029957	magic
41.66991306	**Grand Total**



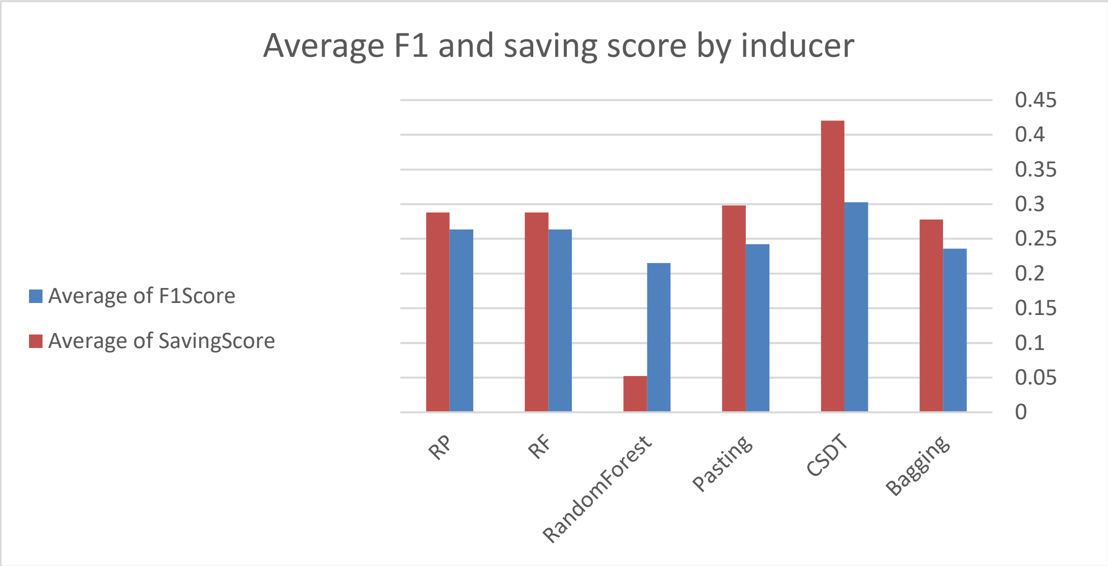
Average prediction time:

Average of PredictTIme	Row Labels
0.070708176	adult
0.08914519	bank
0.130778399	bankmarketing
0.004844679	banknote
0.002232677	blood
0.002225982	cancer
0.308298912	credit scoring1
0.138218847	credit scoring2
0.06205574	default_ccc
0.052863095	magic
0.08613717	**Grand Total**



There were 3 data sets that their cost matrix were given as part of the data set, the matrix was created using experts, so the matrix is a lot more accurate then the one we created in the process. We wanted to see only those data set compering the f1 and saving scores:

Average of SavingScore	Average of F1Score	Row Labels
0.277882165	0.23575955	Bagging
0.420380924	0.302639467	CSDT
0.298063902	0.242418107	Pasting
0.052333803	0.215060995	RandomForest
0.287795303	0.263511171	RF
0.287917581	0.263673491	RP
0.285201192	**0.251735825**	**Grand Total**

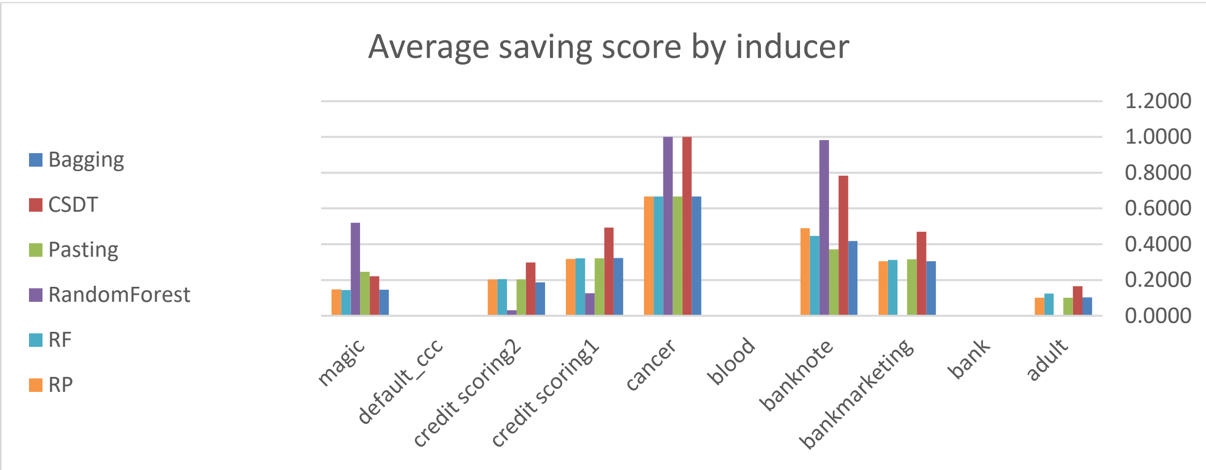


<u>Results for ensemble size 10:</u>

Legend by inducer:

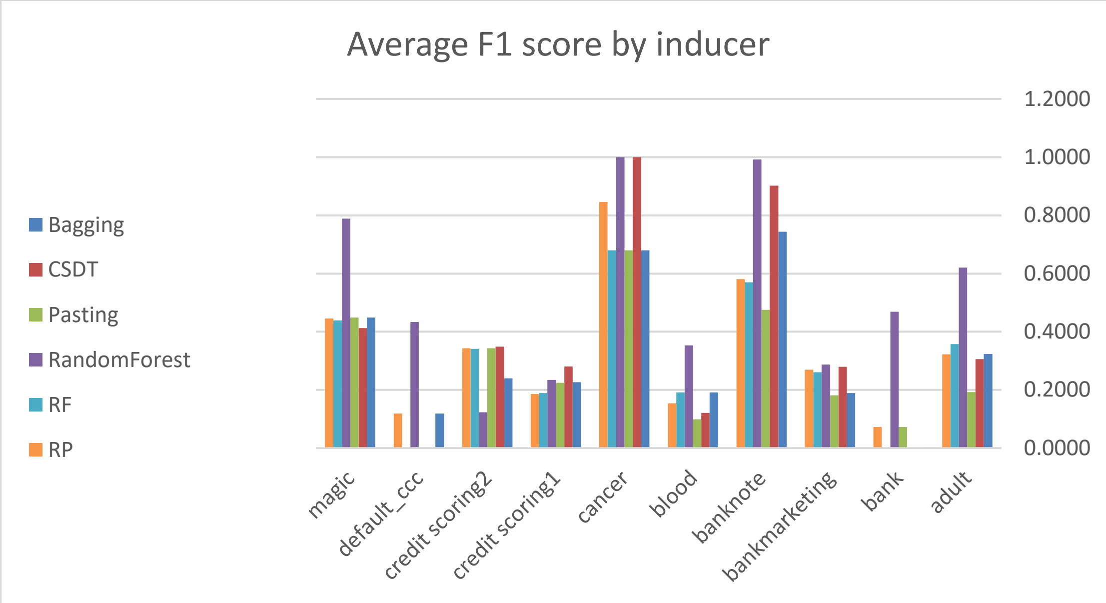
<u>Saving score</u>:

Grand Total	RP	RF	RandomForest	Pasting	CSDT	Bagging	Row Labels
0.1039	0.1017	0.1249	0.0000	0.1017	0.1645	0.1018	adult
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	bank
0.2988	0.3047	0.3128	0.0000	0.3153	0.4700	0.3051	bankmarketing
0.4957	0.4898	0.4468	0.9826	0.3708	0.7836	0.4172	banknote
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	blood
0.7143	0.6667	0.6667	1.0000	0.6667	1.0000	0.6667	cancer
0.3192	0.3181	0.3205	0.1254	0.3218	0.4924	0.3234	credit scoring1
0.1949	0.2036	0.2051	0.0316	0.2035	0.2987	0.1872	credit scoring2
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	default_ccc
0.1995	0.1482	0.1435	0.5196	0.2460	0.2218	0.1462	magic
0.2326	**0.2233**	**0.2220**	**0.2659**	**0.2226**	**0.3431**	**0.2148**	**Grand Total**

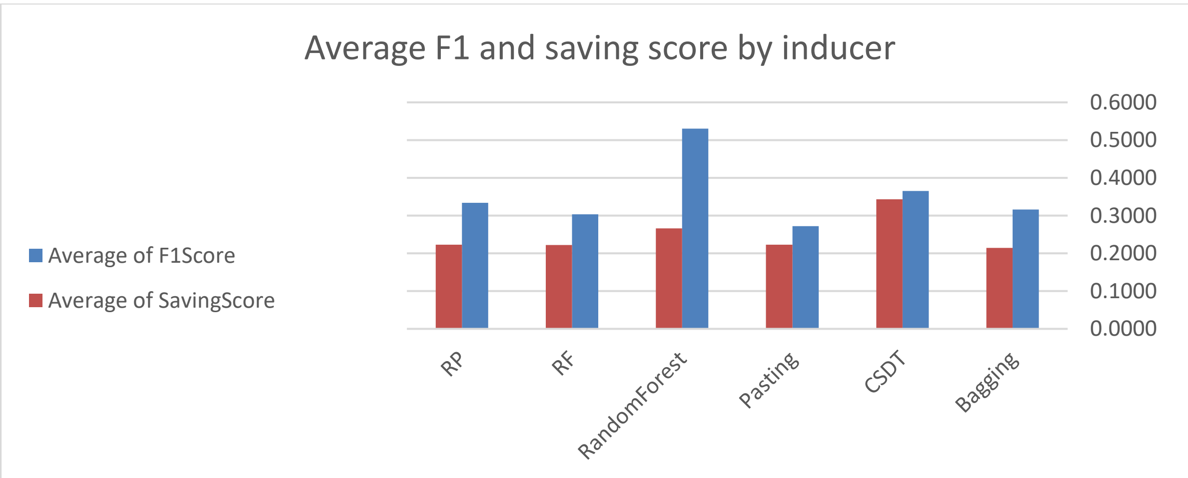


<u>F1-Score:</u>

Grand Total	RP	RF	RandomForest	Pasting	CSDT	Bagging	Row Labels
0.3224	0.3226	0.3579	0.6206	0.1924	0.3062	0.3229	adult
0.0655	0.0726	0.0018	0.4683	0.0726	0.0018	0.0018	bank
0.2335	0.2691	0.2611	0.2871	0.1820	0.2791	0.1888	bankmarketing
0.6430	0.5813	0.5697	0.9922	0.4750	0.9016	0.7437	banknote
0.1700	0.1535	0.1913	0.3529	0.0993	0.1212	0.1913	blood
0.7609	0.8463	0.6792	1.0000	0.6792	1.0000	0.6792	cancer
0.2137	0.1859	0.1891	0.2345	0.2243	0.2807	0.2264	credit scoring1
0.3051	0.3434	0.3411	0.1236	0.3426	0.3481	0.2396	credit scoring2
0.0827	0.1188	0.0016	0.4336	0.0016	0.0016	0.1188	default_ccc
0.4675	0.4450	0.4390	0.7883	0.4485	0.4119	0.4492	magic
0.3264	**0.3338**	**0.3032**	**0.5301**	**0.2718**	**0.3652**	**0.3162**	**Grand Total**

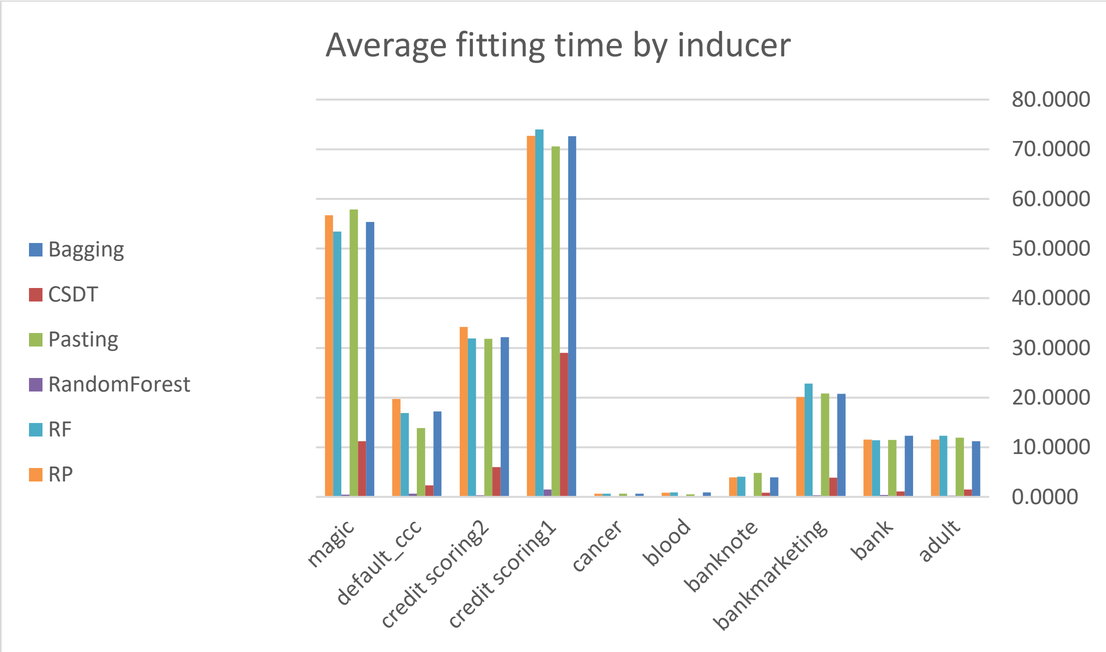


Average of SavingScore	Average of F1Score	Row Labels
0.2148	0.3162	Bagging
0.3431	0.3652	CSDT
0.2226	0.2718	Pasting
0.2659	0.5301	RandomForest
0.2220	0.3032	RF
0.2233	0.3338	RP
0.2326	**0.3264**	**Grand Total**



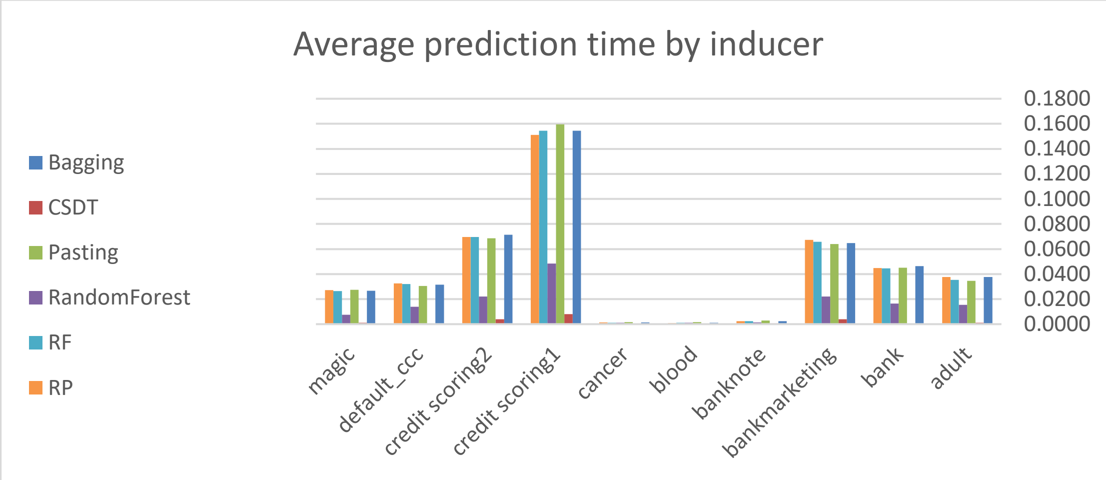
Fitting time:

Grand Total	RP	RF	RandomForest	Pasting	CSDT	Bagging	Row Labels
10.1946	11.5247	12.3319	0.2455	11.9010	1.5137	11.2308	adult
10.1229	11.5272	11.4128	0.3913	11.4807	1.0680	12.3330	bank
18.4026	20.1364	22.8259	0.2965	20.8060	3.8718	20.7209	bankmarketing
3.6607	3.9146	4.0921	0.0210	4.8358	0.8559	3.9485	banknote
0.6865	0.8220	0.8741	0.0175	0.5458	0.1432	0.9082	blood
0.5424	0.6278	0.6278	0.0135	0.6380	0.0170	0.6272	cancer
64.2890	72.6795	73.9560	1.4648	70.5723	29.0056	72.6507	credit scoring1
28.3489	34.2505	31.9032	0.3464	31.8676	6.0063	32.1560	credit scoring2
14.7110	19.7210	16.9097	0.6203	13.8682	2.3057	17.1770	default_ccc
48.7083	56.7195	53.4380	0.4586	57.8744	11.1847	55.3922	magic
19.9667	**23.1923**	**22.8372**	**0.3875**	**22.4390**	**5.5972**	**22.7144**	**Grand Total**

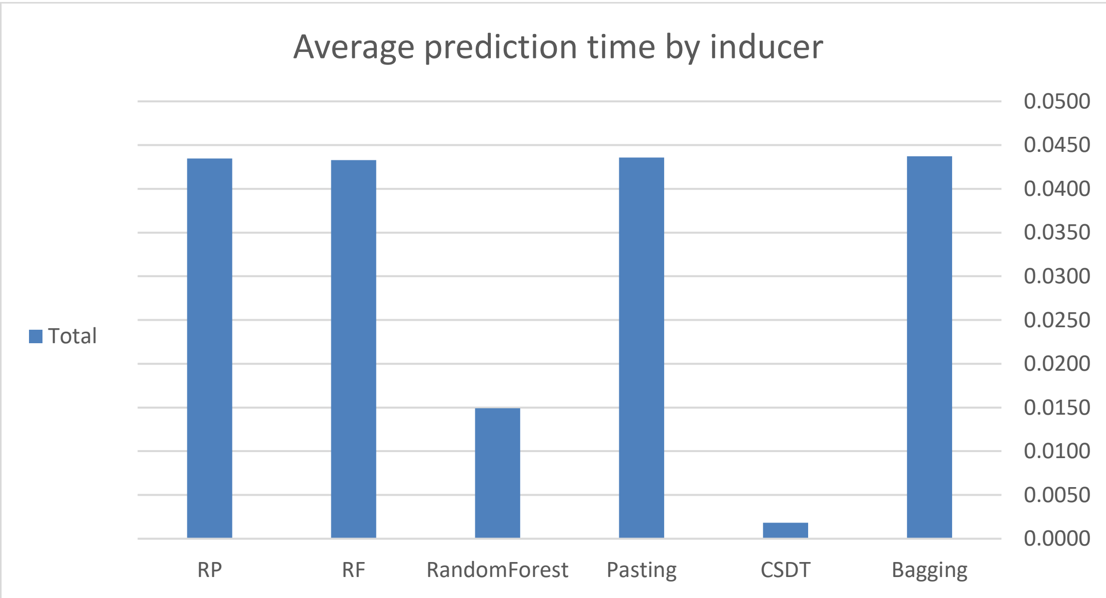


!!!!!!! timeEEEEE!!!!!!!!!!!!
Prediction time:

Grand Total	RP	RF	RandomForest	Pasting	CSDT	Bagging	Row Labels
0.0323	0.0376	0.0354	0.0155	0.0346	0.0010	0.0376	adult
0.0399	0.0449	0.0446	0.0165	0.0451	0.0000	0.0463	bank
0.0579	0.0672	0.0657	0.0220	0.0639	0.0040	0.0647	bankmarketing
0.0023	0.0023	0.0025	0.0015	0.0028	0.0005	0.0023	banknote
0.0011	0.0008	0.0011	0.0010	0.0016	0.0000	0.0011	blood
0.0012	0.0013	0.0012	0.0010	0.0015	0.0000	0.0013	cancer
0.1368	0.1511	0.1544	0.0484	0.1597	0.0080	0.1546	credit scoring1
0.0617	0.0697	0.0695	0.0220	0.0685	0.0040	0.0714	credit scoring2
0.0281	0.0324	0.0321	0.0140	0.0306	0.0000	0.0314	default_ccc
0.0237	0.0273	0.0263	0.0075	0.0275	0.0010	0.0266	magic
0.0385	**0.0435**	**0.0433**	**0.0149**	**0.0436**	**0.0018**	**0.0437**	**Grand Total**



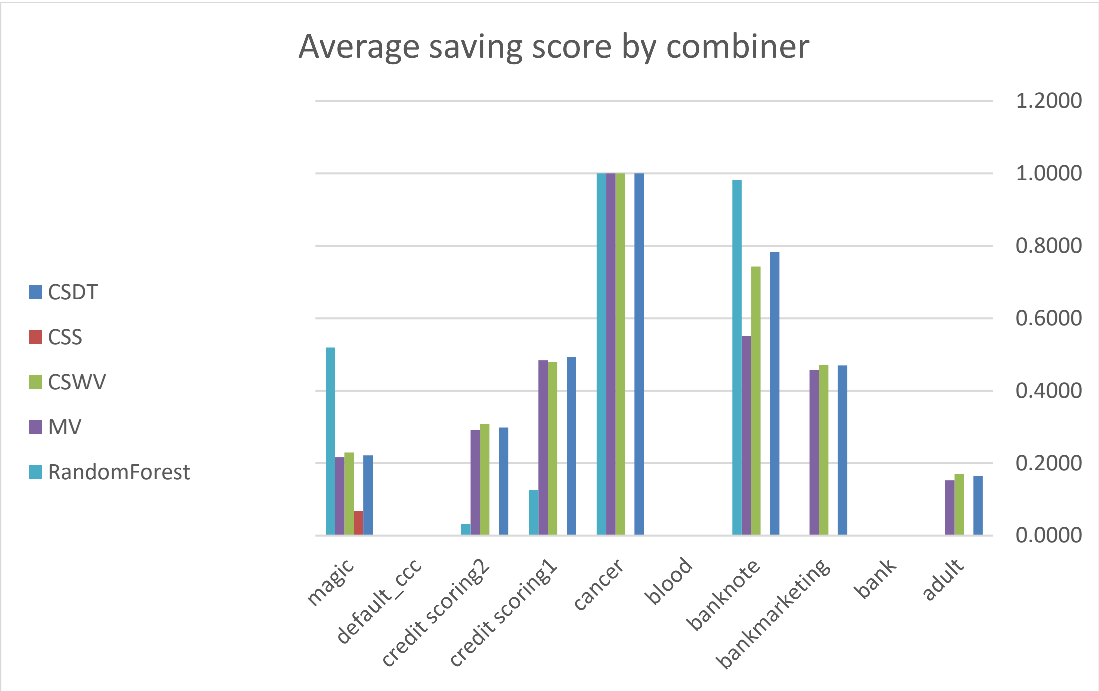
Average of PredictTIme	Row Labels
0.0437	Bagging
0.0018	CSDT
0.0436	Pasting
0.0149	RandomForest
0.0433	RF
0.0435	RP
0.0385	**Grand Total**



Legend by combiner:

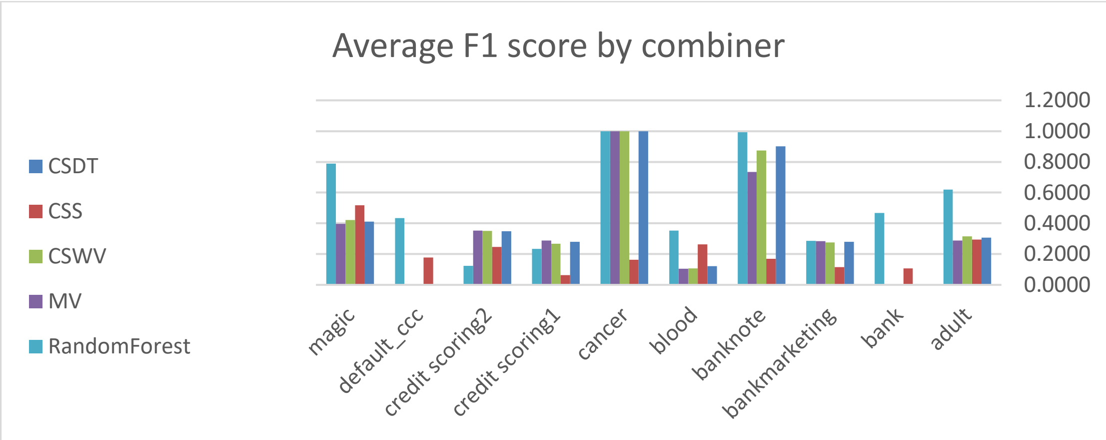
<u>Saving score</u>:

Grand Total	RandomForest	MV	CSWV	CSS	CSDT	Row Labels
0.1039	0.0000	0.1526	0.1700	0.0000	0.1645	adult
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	bank
0.2988	0.0000	0.4565	0.4719	0.0000	0.4700	bankmarketing
0.4957	0.9826	0.5508	0.7428	0.0000	0.7836	banknote
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	blood
0.7143	1.0000	1.0000	1.0000	0.0000	1.0000	cancer
0.3192	0.1254	0.4843	0.4786	0.0000	0.4924	credit scoring1
0.1949	0.0316	0.2911	0.3084	0.0000	0.2987	credit scoring2
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	default_ccc
0.1995	0.5196	0.2159	0.2298	0.0673	0.2218	magic
0.2326	**0.2659**	**0.3151**	**0.3401**	**0.0067**	**0.3431**	**Grand Total**

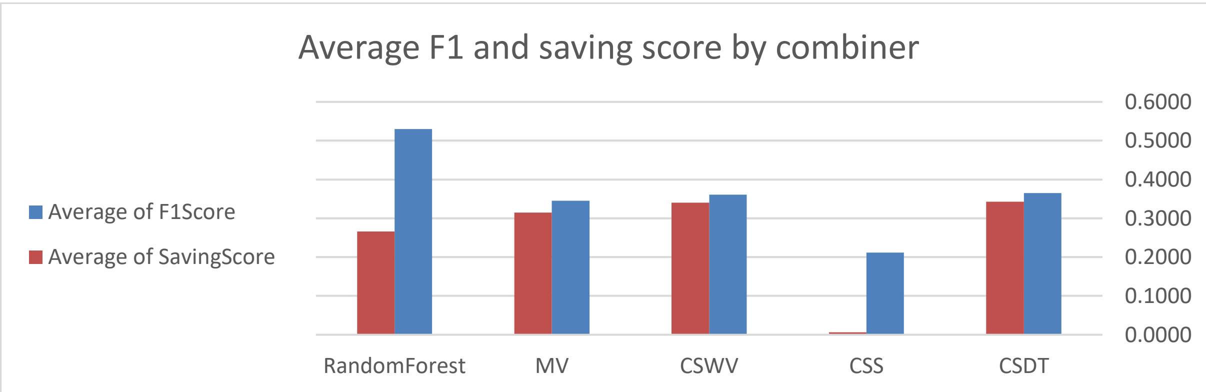


F1-Score:

Grand Total	RandomForest	MV	CSWV	CSS	CSDT	Row Labels
0.3224	0.6206	0.2880	0.3147	0.2942	0.3062	adult
0.0655	0.4683	0.0018	0.0018	0.1080	0.0018	bank
0.2335	0.2871	0.2851	0.2750	0.1156	0.2791	bankmarketing
0.6430	0.9922	0.7337	0.8735	0.1700	0.9016	banknote
0.1700	0.3529	0.1053	0.1070	0.2642	0.1212	blood
0.7609	1.0000	1.0000	1.0000	0.1630	1.0000	cancer
0.2137	0.2345	0.2892	0.2667	0.0634	0.2807	credit scoring1
0.3051	0.1236	0.3523	0.3508	0.2470	0.3481	credit scoring2
0.0827	0.4336	0.0016	0.0016	0.1775	0.0016	default_ccc
0.4675	0.7883	0.3972	0.4217	0.5174	0.4119	magic
0.3264	**0.5301**	**0.3454**	**0.3613**	**0.2120**	**0.3652**	**Grand Total**

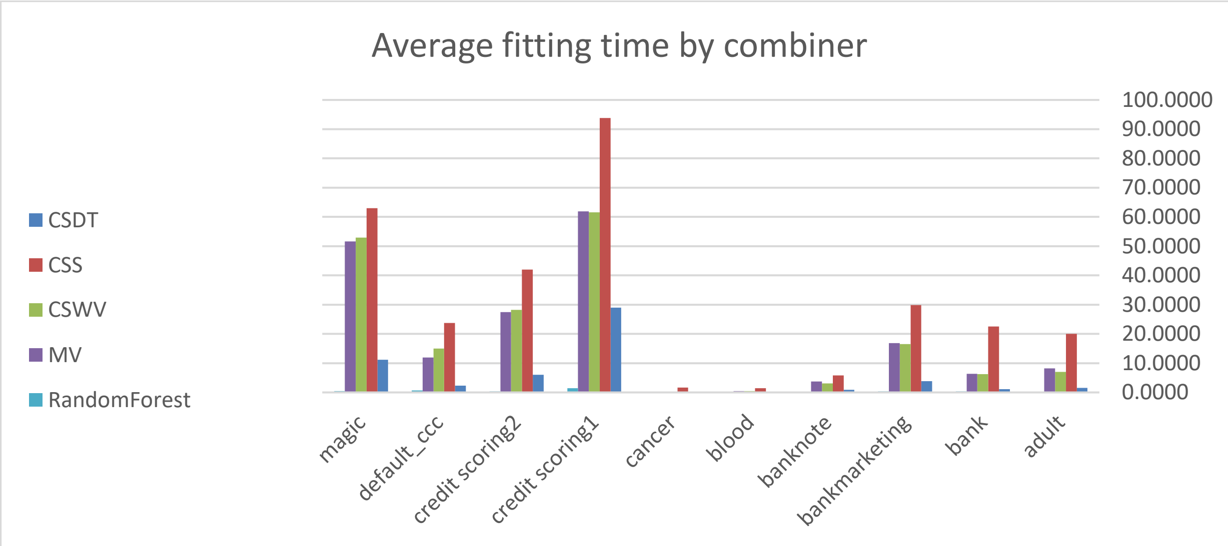


Average of SavingScore	Average of F1Score	Row Labels
0.3431	0.3652	CSDT
0.0067	0.2120	CSS
0.3401	0.3613	CSWV
0.3151	0.3454	MV
0.2659	0.5301	RandomForest
0.2326	**0.3264**	**Grand Total**

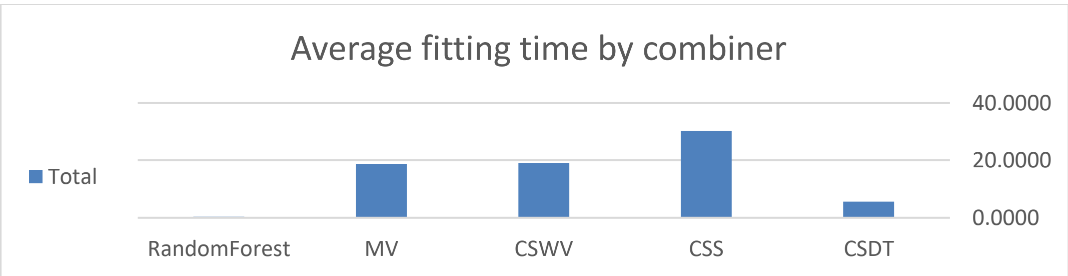


Fitting time:

Grand Total	RandomForest	MV	CSWV	CSS	CSDT	Row Labels
10.1946	0.2455	8.2278	7.0006	20.0129	1.5137	adult
10.1229	0.3913	6.3583	6.2083	22.4986	1.0680	bank
18.4026	0.2965	16.9093	16.5519	29.9057	3.8718	bankmarketing
3.6607	0.0210	3.7230	3.0626	5.8077	0.8559	banknote
0.6865	0.0175	0.4860	0.4100	1.4665	0.1432	blood
0.5424	0.0135	0.1053	0.1051	1.6803	0.0170	cancer
64.2890	1.4648	61.9393	61.5785	93.8761	29.0056	credit scoring1
28.3489	0.3464	27.4135	28.2658	41.9537	6.0063	credit scoring2
14.7110	0.6203	11.9704	15.0267	23.7599	2.3057	default_ccc
48.7083	0.4586	51.6185	52.9620	62.9875	11.1847	magic
19.9667	**0.3875**	**18.8751**	**19.1172**	**30.3949**	**5.5972**	**Grand Total**

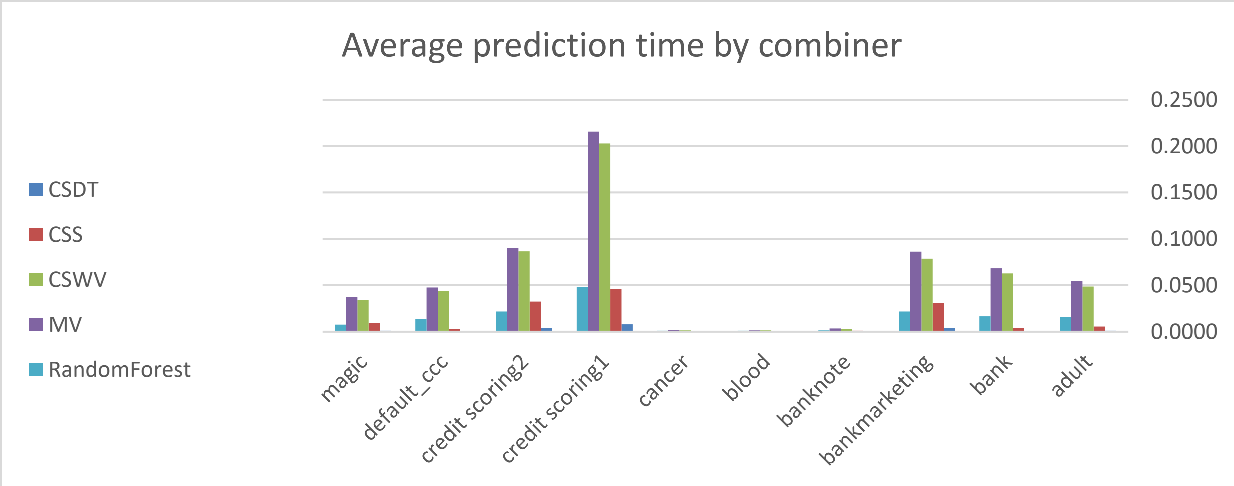


Average of FitTIme	Row Labels
5.5972	CSDT
30.3949	CSS
19.1172	CSWV
18.8751	MV
0.3875	RandomForest
19.9667	**Grand Total**

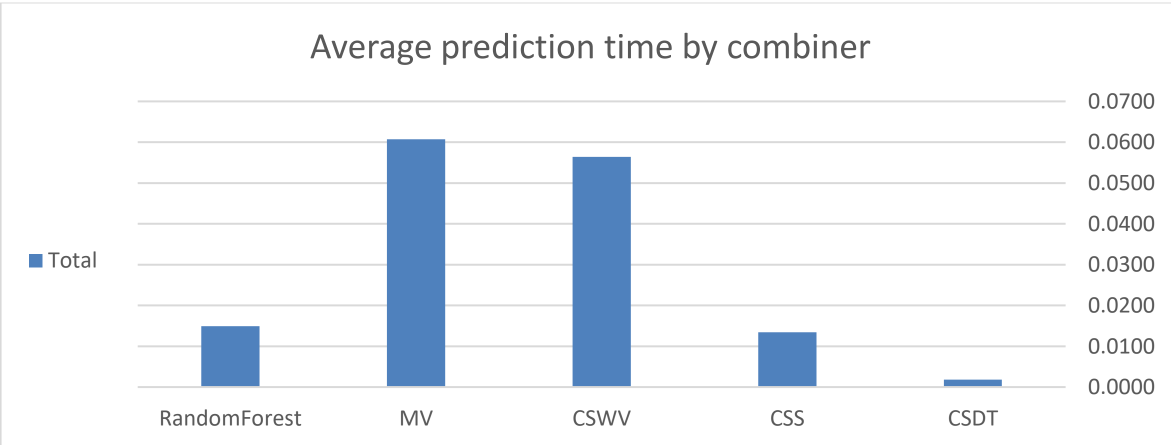


Prediction time:

Grand Total	RandomForest	MV	CSWV	CSS	CSDT	Row Labels
0.0323	0.0155	0.0547	0.0488	0.0055	0.0010	adult
0.0399	0.0165	0.0685	0.0629	0.0042	0.0000	bank
0.0579	0.0220	0.0863	0.0787	0.0310	0.0040	bankmarketing
0.0023	0.0015	0.0034	0.0030	0.0011	0.0005	banknote
0.0011	0.0010	0.0015	0.0014	0.0007	0.0000	blood
0.0012	0.0010	0.0016	0.0015	0.0009	0.0000	cancer
0.1368	0.0484	0.2157	0.2030	0.0460	0.0080	credit scoring1
0.0617	0.0220	0.0902	0.0866	0.0325	0.0040	credit scoring2
0.0281	0.0140	0.0477	0.0440	0.0032	0.0000	default_ccc
0.0237	0.0075	0.0374	0.0341	0.0092	0.0010	magic
0.0385	**0.0149**	**0.0607**	**0.0564**	**0.0135**	**0.0018**	**Grand Total**



Average of PredictTIme	Row Labels
0.0018	CSDT
0.0135	CSS
0.0564	CSWV
0.0607	MV
0.0149	RandomForest
0.0385	**Grand Total**

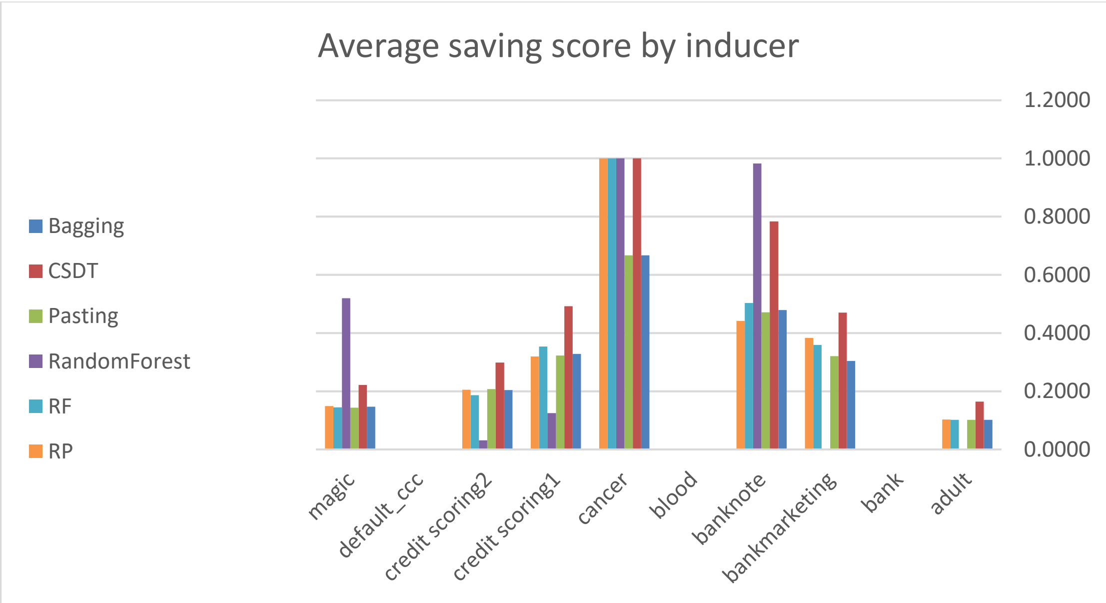


<u>Results for ensemble size 20:</u>

Legend by inducer:

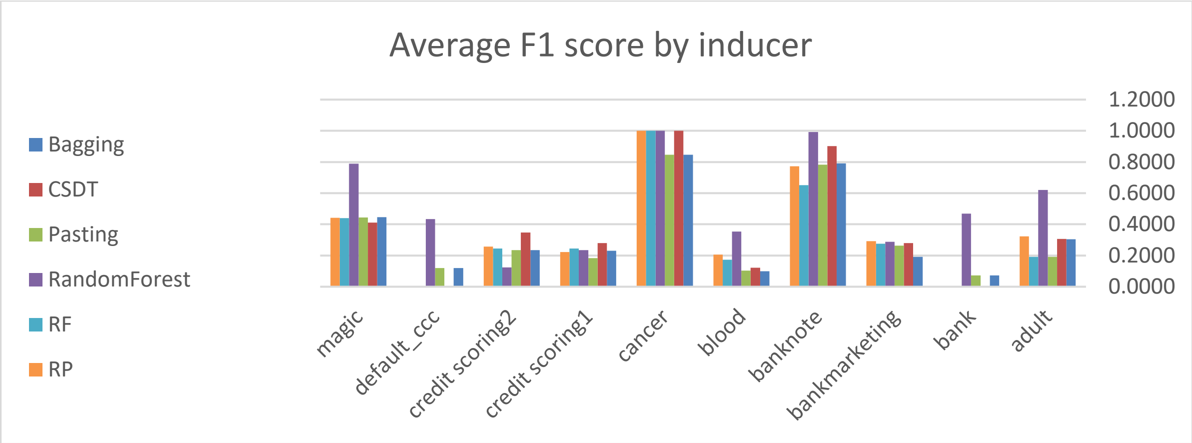
<u>Saving score:</u>

Grand Total	RP	RF	RandomForest	Pasting	CSDT	Bagging	Row Labels
0.0992	0.1029	0.1017	0.0000	0.1017	0.1645	0.1017	adult
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	bank
0.3265	0.3829	0.3594	0.0000	0.3202	0.4700	0.3043	bankmarketing
0.5325	0.4417	0.5029	0.9826	0.4718	0.7836	0.4796	banknote
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	blood
0.8571	1.0000	1.0000	1.0000	0.6667	1.0000	0.6667	cancer
0.3280	0.3200	0.3532	0.1254	0.3225	0.4924	0.3289	credit scoring1
0.1956	0.2052	0.1864	0.0316	0.2075	0.2987	0.2037	credit scoring2
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	default_ccc
0.1781	0.1486	0.1444	0.5196	0.1439	0.2218	0.1470	magic
0.2517	**0.2601**	**0.2648**	**0.2659**	**0.2234**	**0.3431**	**0.2232**	**Grand Total**

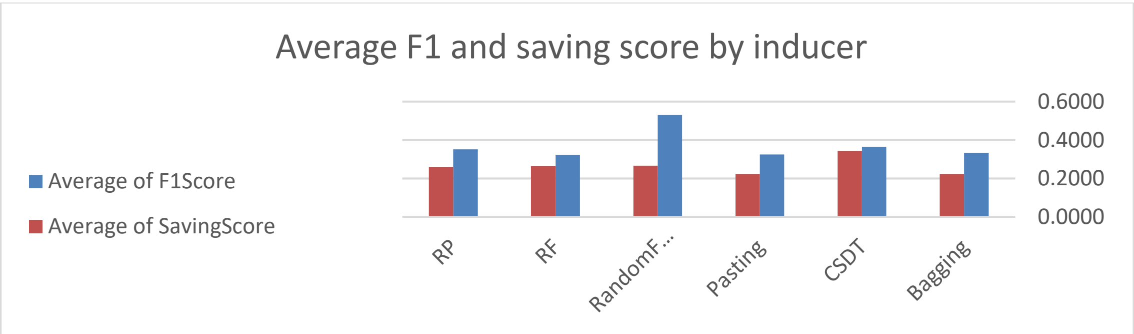


F1-Score:

Grand Total	RP	RF	RandomForest	Pasting	CSDT	Bagging	Row Labels
0.2828	0.3221	0.1924	0.6206	0.1924	0.3062	0.3036	adult
0.0655	0.0018	0.0018	0.4683	0.0726	0.0018	0.0726	bank
0.2595	0.2920	0.2764	0.2871	0.2630	0.2791	0.1911	bankmarketing
0.7774	0.7730	0.6511	0.9922	0.7819	0.9016	0.7904	banknote
0.1584	0.2065	0.1724	0.3529	0.1023	0.1212	0.1000	blood
0.9341	1.0000	1.0000	1.0000	0.8463	1.0000	0.8463	cancer
0.2259	0.2231	0.2447	0.2345	0.1842	0.2807	0.2306	credit scoring1
0.2422	0.2575	0.2449	0.1236	0.2352	0.3481	0.2355	credit scoring2
0.0827	0.0016	0.0016	0.4336	0.1188	0.0016	0.1188	default_ccc
0.4657	0.4420	0.4406	0.7883	0.4445	0.4119	0.4460	magic
0.3494	**0.3520**	**0.3226**	**0.5301**	**0.3241**	**0.3652**	**0.3335**	**Grand Total**

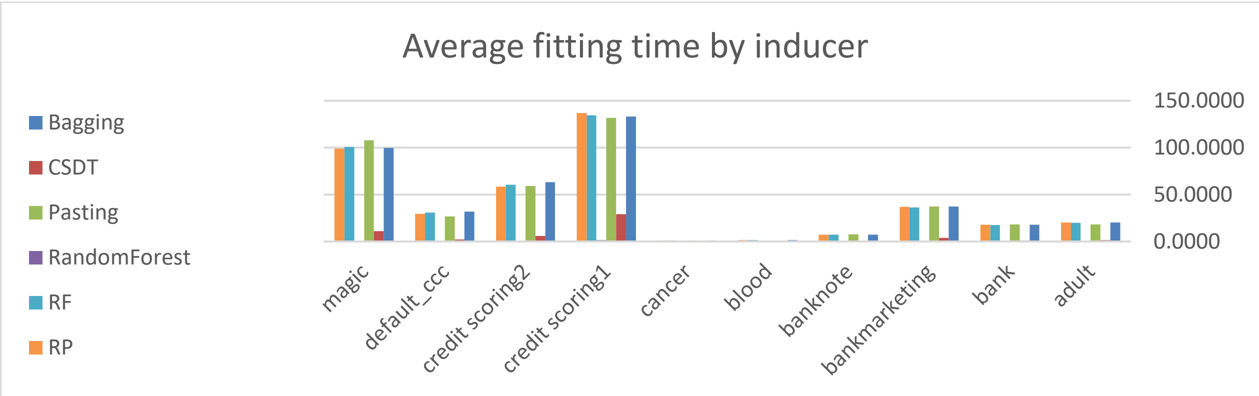


Average of SavingScore	Average of F1Score	Row Labels
0.2232	0.3335	Bagging
0.3431	0.3652	CSDT
0.2234	0.3241	Pasting
0.2659	0.5301	RandomForest
0.2648	0.3226	RF
0.2601	0.3520	RP
0.2517	**0.3494**	**Grand Total**

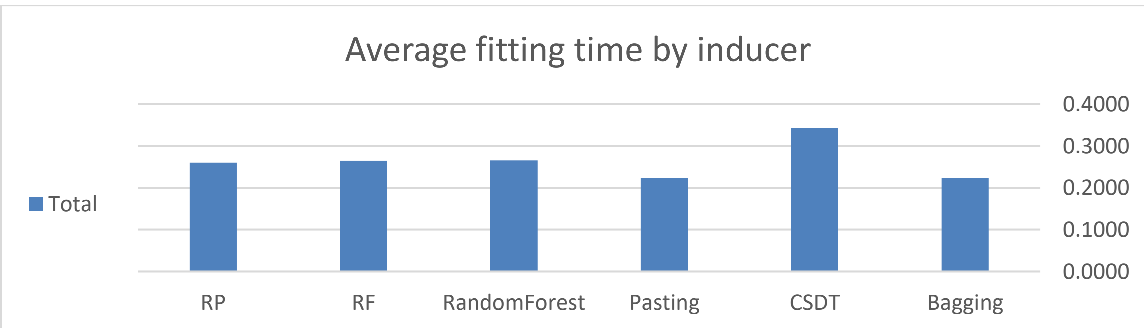


Fitting time:

Grand Total	RP	RF	RandomForest	Pasting	CSDT	Bagging	Row Labels
17.0077	20.1871	19.9644	0.2455	18.3276	1.5137	20.3038	adult
15.3632	17.7198	17.4527	0.3913	18.0255	1.0680	18.0106	bank
31.9822	37.0712	36.1682	0.2965	37.2168	3.8718	37.4045	bankmarketing
6.3672	7.1392	7.4292	0.0210	7.6149	0.8559	7.2382	banknote
1.0875	1.3141	1.4119	0.0175	0.9376	0.1432	1.3577	blood
0.9952	1.1593	1.1579	0.0135	1.1554	0.0170	1.1615	cancer
117.1153	136.9359	134.5098	1.4648	131.6794	29.0056	133.2563	credit scoring1
52.1324	58.4137	60.3434	0.3464	59.0869	6.0063	63.3231	credit scoring2
25.7205	29.6201	30.7978	0.6203	26.6814	2.3057	31.9544	default_ccc
88.0722	98.8970	100.7234	0.4586	107.8774	11.1847	99.6246	magic
35.5843	**40.8457**	**40.9959**	**0.3875**	**40.8603**	**5.5972**	**41.3635**	**Grand Total**

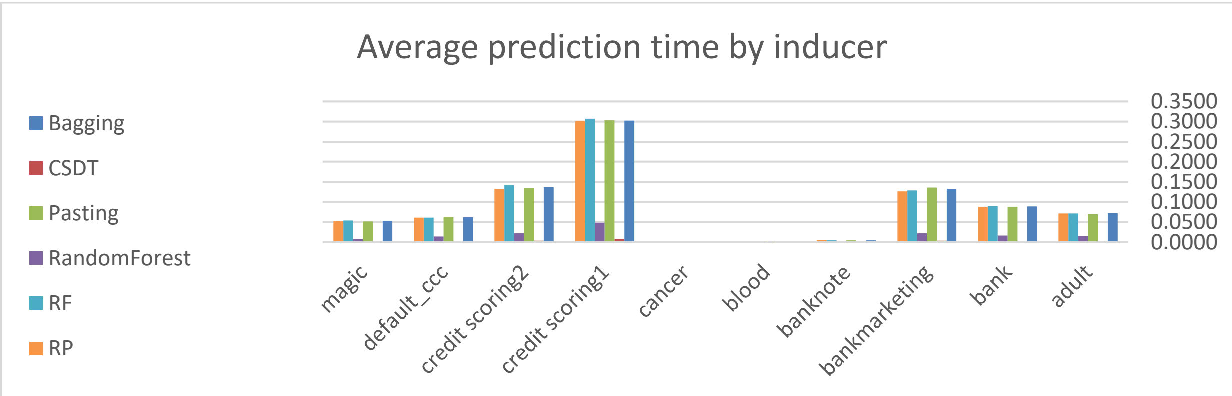


Average of SavingScore	Row Labels
0.2232	Bagging
0.3431	CSDT
0.2234	Pasting
0.2659	RandomForest
0.2648	RF
0.2601	RP
0.2517	**Grand Total**

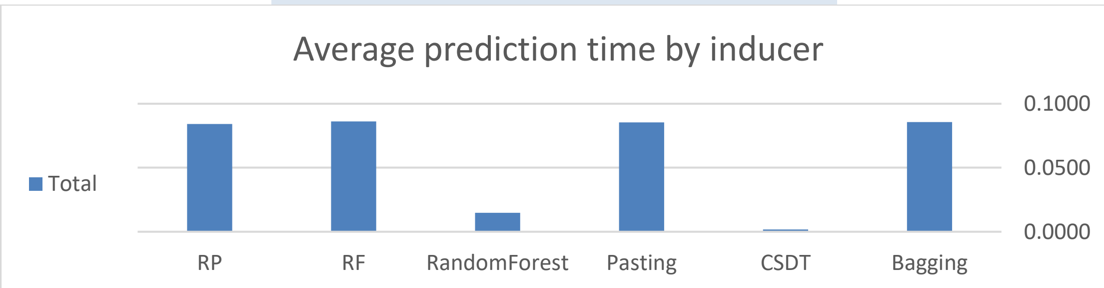


Prediction time:

Grand Total	RP	RF	RandomForest	Pasting	CSDT	Bagging	Row Labels
0.0620	0.0710	0.0710	0.0155	0.0697	0.0010	0.0720	adult
0.0773	0.0883	0.0897	0.0165	0.0883	0.0000	0.0891	bank
0.1139	0.1261	0.1284	0.0220	0.1356	0.0040	0.1327	bankmarketing
0.0042	0.0050	0.0048	0.0015	0.0047	0.0005	0.0047	banknote
0.0018	0.0018	0.0018	0.0010	0.0026	0.0000	0.0018	blood
0.0020	0.0022	0.0022	0.0010	0.0021	0.0000	0.0023	cancer
0.2639	0.3009	0.3068	0.0484	0.3028	0.0080	0.3024	credit scoring1
0.1188	0.1329	0.1412	0.0220	0.1349	0.0040	0.1366	credit scoring2
0.0537	0.0609	0.0610	0.0140	0.0617	0.0000	0.0622	default_ccc
0.0458	0.0526	0.0539	0.0075	0.0516	0.0010	0.0531	magic
0.0743	**0.0842**	**0.0861**	**0.0149**	**0.0854**	**0.0018**	**0.0857**	**Grand Total**



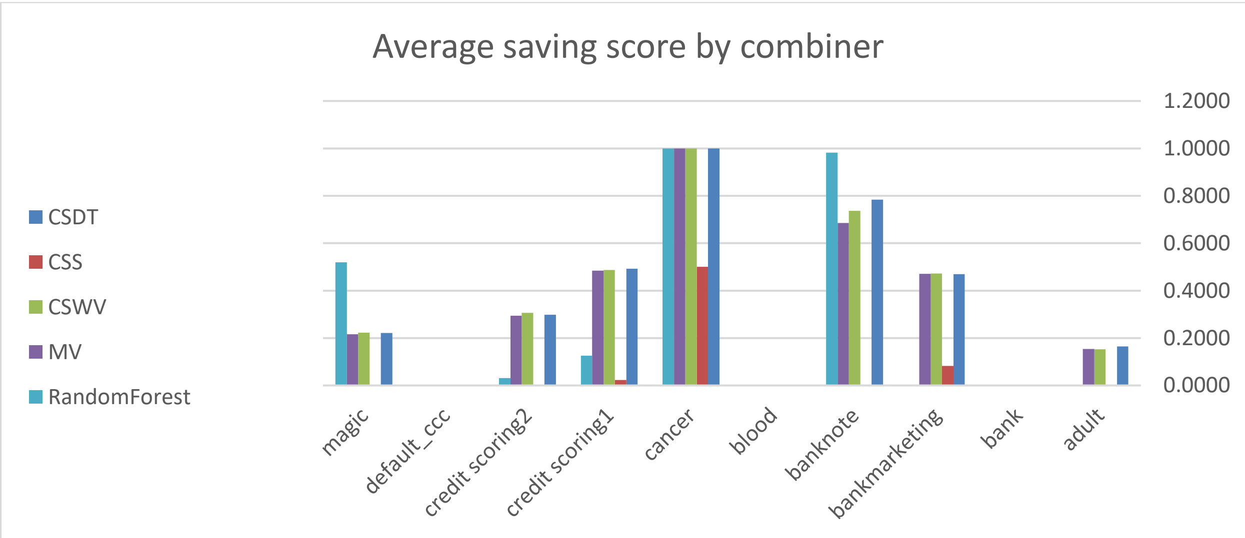
Average of PredictTIme	Row Labels
0.0857	Bagging
0.0018	CSDT
0.0854	Pasting
0.0149	RandomForest
0.0861	RF
0.0842	RP
0.0743	**Grand Total**



Legend by combiner:

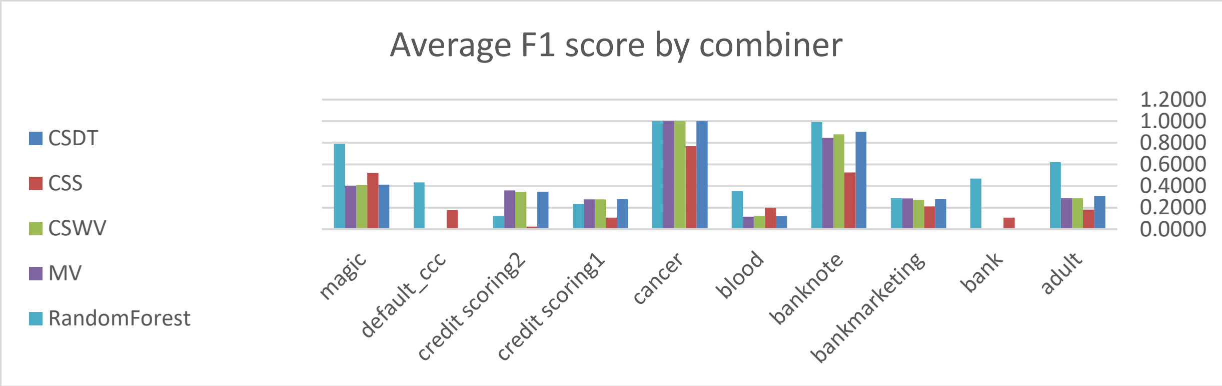
<u>Saving score</u>:

Grand Total	RandomForest	MV	CSWV	CSS	CSDT	Row Labels
0.0992	0.0000	0.1535	0.1526	0.0000	0.1645	adult
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	bank
0.3265	0.0000	0.4711	0.4716	0.0825	0.4700	bankmarketing
0.5325	0.9826	0.6859	0.7362	0.0000	0.7836	banknote
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	blood
0.8571	1.0000	1.0000	1.0000	0.5000	1.0000	cancer
0.3280	0.1254	0.4844	0.4867	0.0224	0.4924	credit scoring1
0.1956	0.0316	0.2947	0.3059	0.0016	0.2987	credit scoring2
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	default_ccc
0.1781	0.5196	0.2155	0.2224	0.0000	0.2218	magic
0.2517	**0.2659**	**0.3305**	**0.3375**	**0.0606**	**0.3431**	**Grand Total**

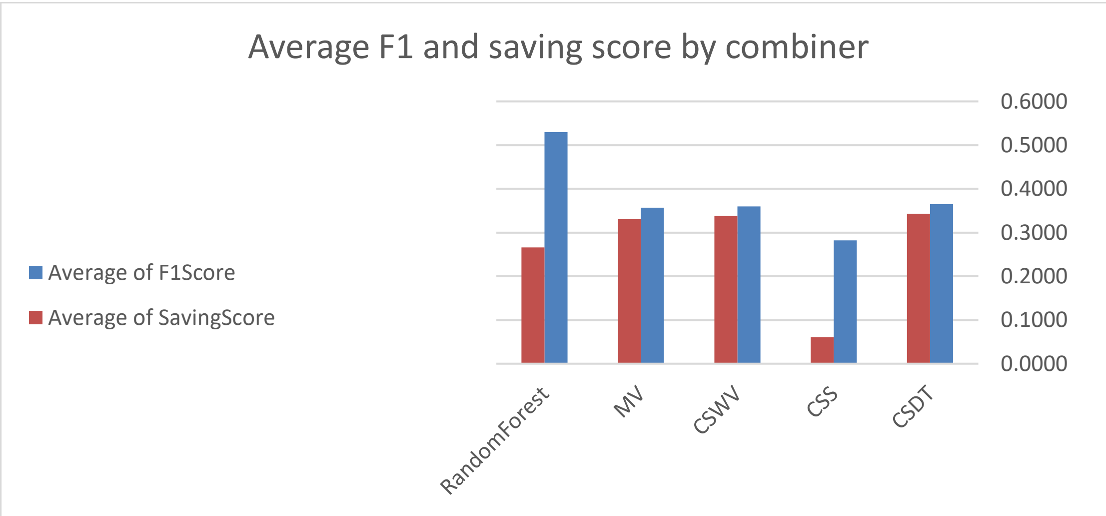


F1-Score:

Grand Total	RandomForest	MV	CSWV	CSS	CSDT	Row Labels
0.2828	0.6206	0.2876	0.2880	0.1823	0.3062	adult
0.0655	0.4683	0.0018	0.0018	0.1080	0.0018	bank
0.2595	0.2871	0.2849	0.2707	0.2112	0.2791	bankmarketing
0.7774	0.9922	0.8458	0.8772	0.5243	0.9016	banknote
0.1584	0.3529	0.1167	0.1212	0.1980	0.1212	blood
0.9341	1.0000	1.0000	1.0000	0.7694	1.0000	cancer
0.2259	0.2345	0.2770	0.2765	0.1084	0.2807	credit scoring1
0.2422	0.1236	0.3583	0.3481	0.0234	0.3481	credit scoring2
0.0827	0.4336	0.0016	0.0016	0.1775	0.0016	default_ccc
0.4657	0.7883	0.3981	0.4104	0.5214	0.4119	magic
0.3494	**0.5301**	**0.3572**	**0.3596**	**0.2824**	**0.3652**	**Grand Total**

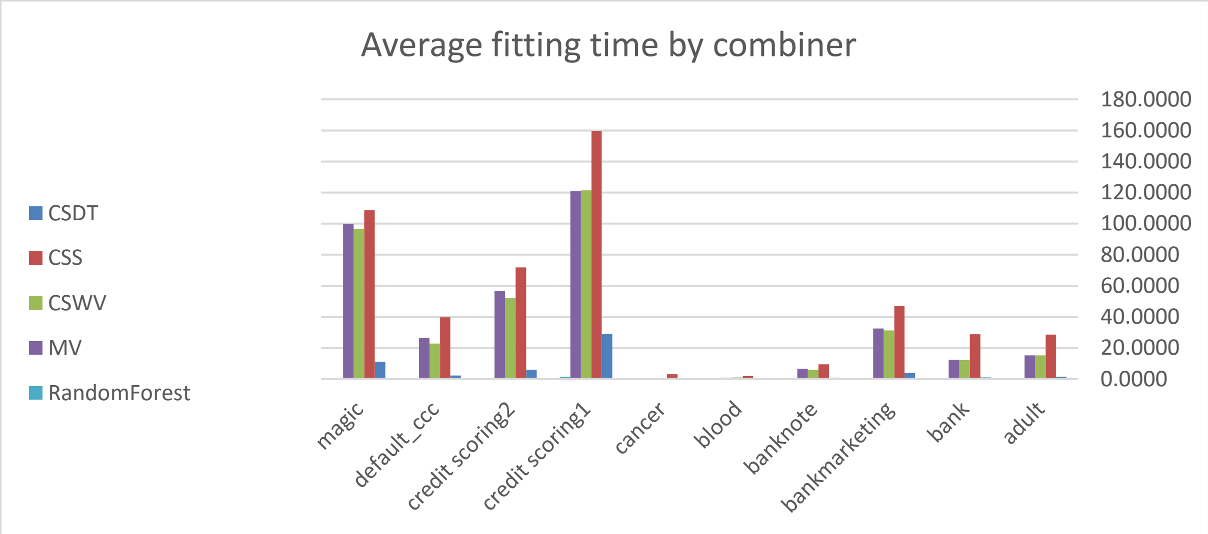


Average of SavingScore	Average of F1Score	Row Labels
0.3431	0.3652	CSDT
0.0606	0.2824	CSS
0.3375	0.3596	CSWV
0.3305	0.3572	MV
0.2659	0.5301	RandomForest
0.2517	**0.3494**	**Grand Total**

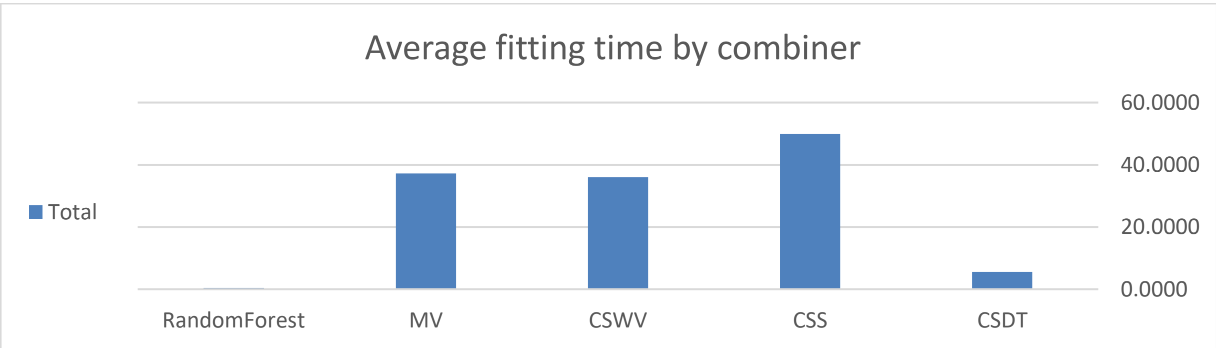


Fitting time:

Grand Total	RandomForest	MV	CSWV	CSS	CSDT	Row Labels
17.0077	0.2455	15.2123	15.2076	28.6672	1.5137	adult
15.3632	0.3913	12.4568	12.1867	28.7630	1.0680	bank
31.9822	0.2965	32.5566	31.3155	47.0234	3.8718	bankmarketing
6.3672	0.0210	6.6051	6.0468	9.4142	0.8559	banknote
1.0875	0.0175	0.9187	0.9570	1.8902	0.1432	blood
0.9952	0.0135	0.2003	0.2051	3.0702	0.0170	cancer
117.1153	1.4648	121.0590	121.4450	159.7820	29.0056	credit scoring1
52.1324	0.3464	56.8889	52.1280	71.8584	6.0063	credit scoring2
25.7205	0.6203	26.6136	22.8951	39.7816	2.3057	default_ccc
88.0722	0.4586	99.9044	96.8346	108.6028	11.1847	magic
35.5843	**0.3875**	**37.2416**	**35.9221**	**49.8853**	**5.5972**	**Grand Total**

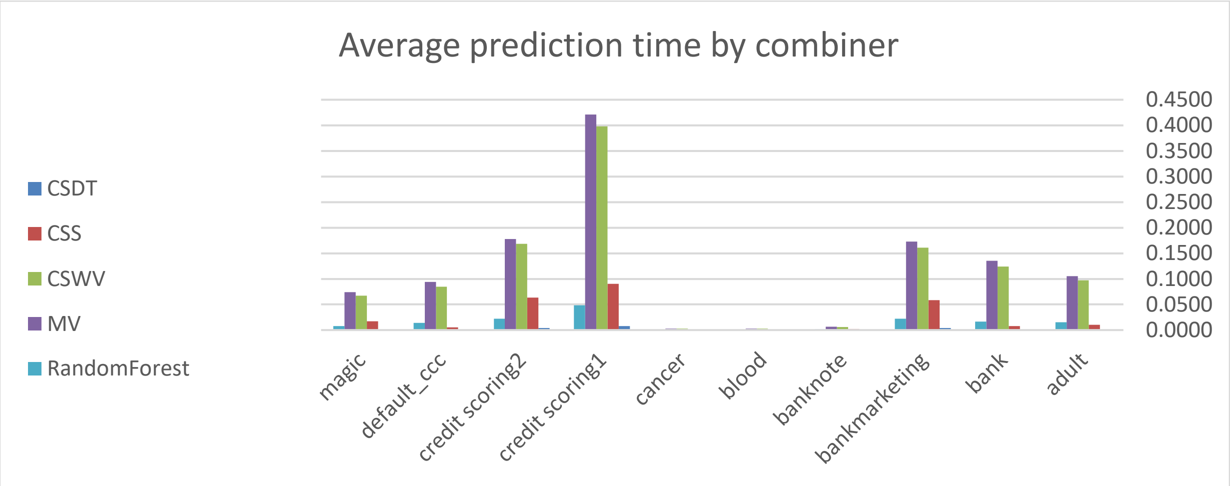


Average of FitTIme	Row Labels
5.5972	CSDT
49.8853	CSS
35.9221	CSWV
37.2416	MV
0.3875	RandomForest
35.5843	**Grand Total**

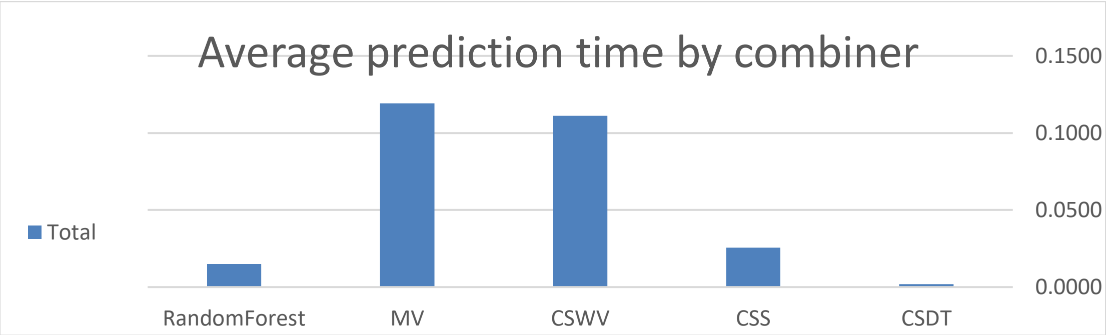


Prediction time:

Grand Total	RandomForest	MV	CSWV	CSS	CSDT	Row Labels
0.0620	0.0155	0.1056	0.0969	0.0104	0.0010	adult
0.0773	0.0165	0.1351	0.1240	0.0075	0.0000	bank
0.1139	0.0220	0.1727	0.1610	0.0585	0.0040	bankmarketing
0.0042	0.0015	0.0062	0.0058	0.0022	0.0005	banknote
0.0018	0.0010	0.0027	0.0026	0.0007	0.0000	blood
0.0020	0.0010	0.0030	0.0026	0.0010	0.0000	cancer
0.2639	0.0484	0.4212	0.3979	0.0906	0.0080	credit scoring1
0.1188	0.0220	0.1778	0.1682	0.0632	0.0040	credit scoring2
0.0537	0.0140	0.0943	0.0847	0.0054	0.0000	default_ccc
0.0458	0.0075	0.0740	0.0672	0.0171	0.0010	magic
0.0743	**0.0149**	**0.1193**	**0.1111**	**0.0257**	**0.0018**	**Grand Total**



Average of PredictTIme	Row Labels
0.0018	CSDT
0.0257	CSS
0.1111	CSWV
0.1193	MV
0.0149	RandomForest
0.0743	**Grand Total**

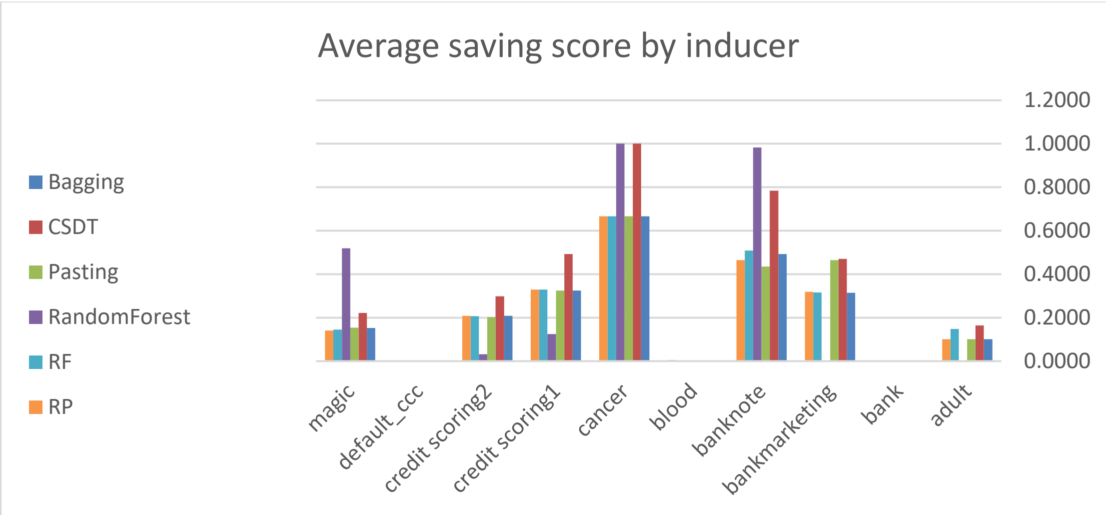


<u>Results for ensemble size 30:</u>

Legend by inducer:

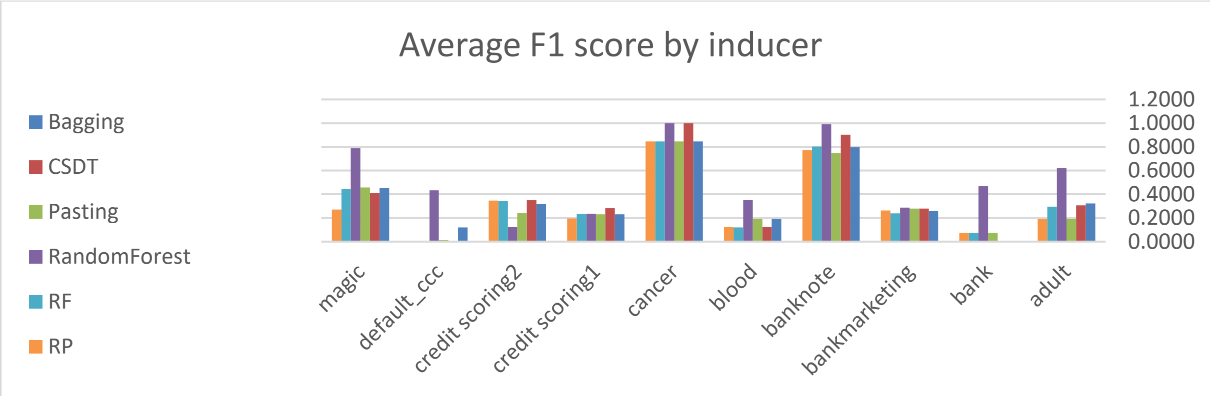
<u>Saving score</u>:

Grand Total	RP	RF	RandomForest	Pasting	CSDT	Bagging	Row Labels
0.1089	0.1017	0.1480	0.0000	0.1017	0.1645	0.1017	adult
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	bank
0.3365	0.3192	0.3155	0.0000	0.4645	0.4700	0.3146	bankmarketing
0.5337	0.4641	0.5085	0.9826	0.4360	0.7836	0.4932	banknote
0.0012	0.0055	0.0000	0.0000	0.0000	0.0000	0.0000	blood
0.7143	0.6667	0.6667	1.0000	0.6667	1.0000	0.6667	cancer
0.3246	0.3295	0.3298	0.1254	0.3247	0.4924	0.3248	credit scoring1
0.2008	0.2081	0.2074	0.0316	0.2024	0.2987	0.2089	credit scoring2
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	default_ccc
0.1799	0.1411	0.1448	0.5196	0.1538	0.2218	0.1527	magic
0.2400	**0.2236**	**0.2321**	**0.2659**	**0.2350**	**0.3431**	**0.2263**	**Grand Total**

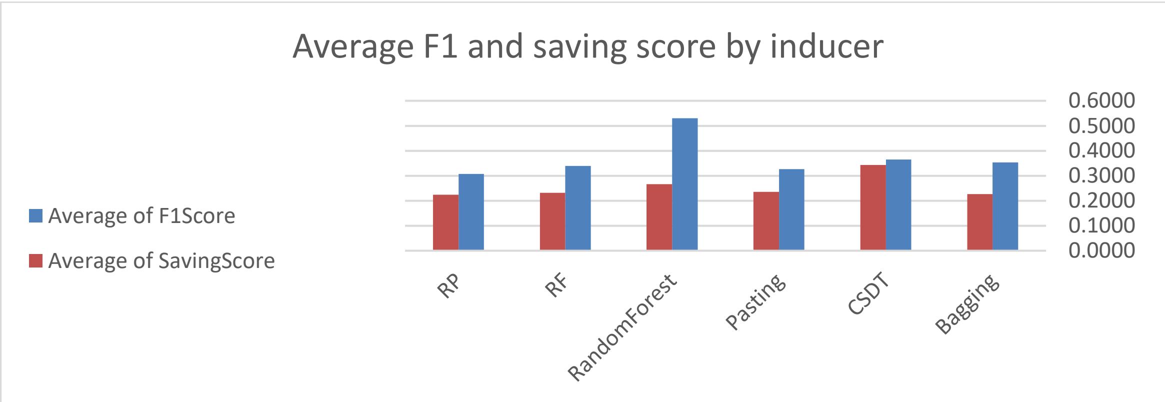


F1-Score:

Grand Total	RP	RF	RandomForest	Pasting	CSDT	Bagging	Row Labels
0.2813	0.1924	0.2962	0.6206	0.1924	0.3062	0.3226	adult
0.0807	0.0726	0.0726	0.4683	0.0726	0.0018	0.0018	bank
0.2639	0.2627	0.2394	0.2871	0.2795	0.2791	0.2610	bankmarketing
0.8038	0.7717	0.8031	0.9922	0.7487	0.9016	0.7963	banknote
0.1678	0.1225	0.1200	0.3529	0.1913	0.1212	0.1913	blood
0.8682	0.8463	0.8463	1.0000	0.8463	1.0000	0.8463	cancer
0.2266	0.1940	0.2322	0.2345	0.2298	0.2807	0.2296	credit scoring1
0.3012	0.3454	0.3428	0.1236	0.2411	0.3481	0.3191	credit scoring2
0.0597	0.0016	0.0016	0.4336	0.0117	0.0016	0.1188	default_ccc
0.4335	0.2699	0.4438	0.7883	0.4566	0.4119	0.4525	magic
0.3487	**0.3079**	**0.3398**	**0.5301**	**0.3270**	**0.3652**	**0.3540**	**Grand Total**

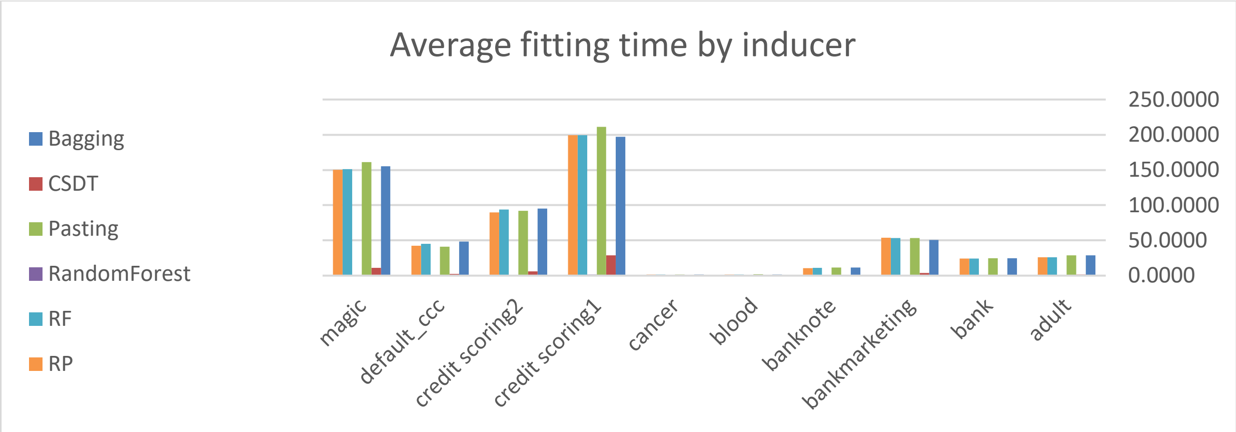


Average of SavingScore	Average of F1Score	Row Labels
0.2263	0.3540	Bagging
0.3431	0.3652	CSDT
0.2350	0.3270	Pasting
0.2659	0.5301	RandomForest
0.2321	0.3398	RF
0.2236	0.3079	RP
0.2400	**0.3487**	**Grand Total**

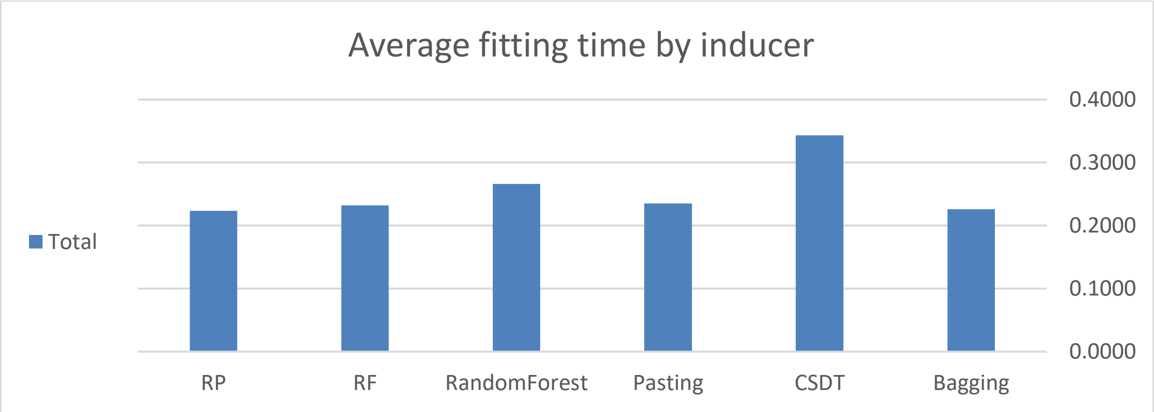


Fitting time:

Grand Total	RP	RF	RandomForest	Pasting	CSDT	Bagging	Row Labels
23.6864	26.2709	26.2251	0.2455	28.6118	1.5137	28.8422	adult
21.1372	24.3907	24.4097	0.3913	24.5100	1.0680	24.8437	bank
45.5120	53.8112	53.3891	0.2965	53.1394	3.8718	50.6604	bankmarketing
9.6964	10.7943	11.0203	0.0210	11.5783	0.8559	11.5645	banknote
1.3048	1.2984	1.3360	0.0175	1.8807	0.1432	1.5204	blood
1.1785	1.3720	1.3716	0.0135	1.3741	0.0170	1.3716	cancer
175.1127	199.5040	199.3645	1.4648	210.9686	29.0056	197.1987	credit scoring1
79.8158	89.7930	93.6861	0.3464	91.7961	6.0063	95.0809	credit scoring2
38.1623	42.4578	45.2286	0.6203	41.0196	2.3057	48.4096	default_ccc
133.2202	150.3926	151.2325	0.4586	161.1459	11.1847	155.0421	magic
52.8826	**60.0085**	**60.7263**	**0.3875**	**62.6025**	**5.5972**	**61.4534**	**Grand Total**

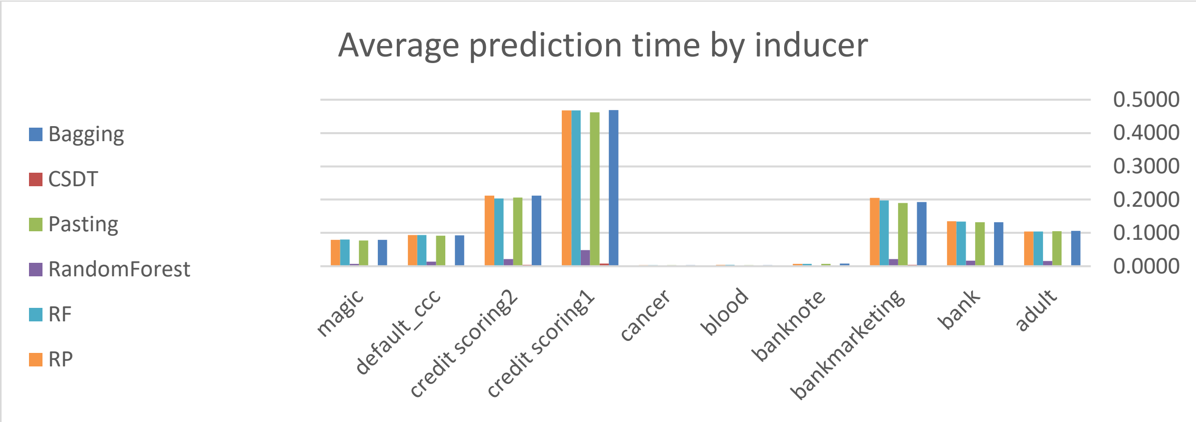


Average of SavingScore	Row Labels
0.2263	Bagging
0.3431	CSDT
0.2350	Pasting
0.2659	RandomForest
0.2321	RF
0.2236	RP
0.2400	**Grand Total**

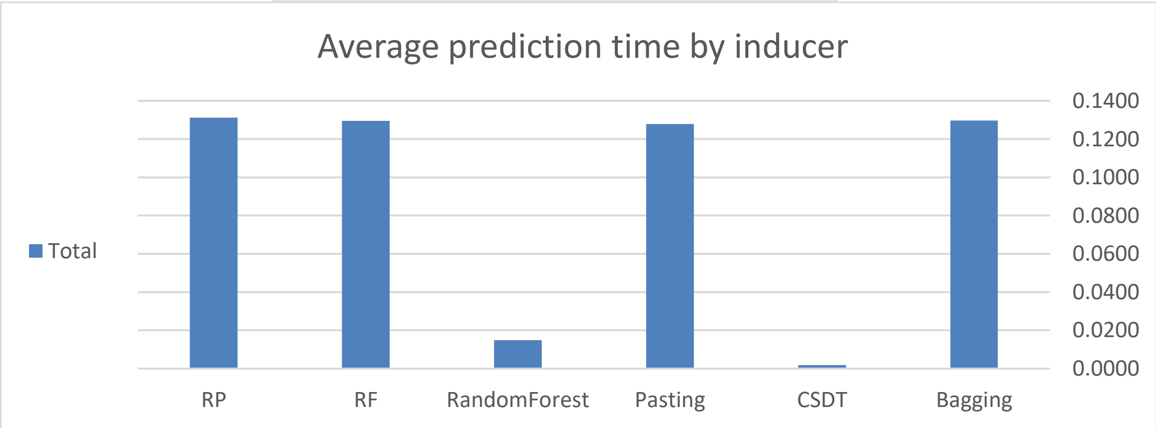


Prediction time:

Grand Total	RP	RF	RandomForest	Pasting	CSDT	Bagging	Row Labels
0.0911	0.1040	0.1041	0.0155	0.1050	0.0010	0.1065	adult
0.1155	0.1352	0.1341	0.0165	0.1321	0.0000	0.1321	bank
0.1701	0.2051	0.1973	0.0220	0.1902	0.0040	0.1925	bankmarketing
0.0064	0.0071	0.0073	0.0015	0.0070	0.0005	0.0077	banknote
0.0031	0.0043	0.0040	0.0010	0.0028	0.0000	0.0028	blood
0.0028	0.0032	0.0032	0.0010	0.0032	0.0000	0.0031	cancer
0.4041	0.4683	0.4681	0.0484	0.4620	0.0080	0.4686	credit scoring1
0.1806	0.2122	0.2036	0.0220	0.2064	0.0040	0.2116	credit scoring2
0.0808	0.0938	0.0936	0.0140	0.0918	0.0000	0.0930	default_ccc
0.0682	0.0789	0.0798	0.0075	0.0777	0.0010	0.0792	magic
0.1122	**0.1312**	**0.1295**	**0.0149**	**0.1278**	**0.0018**	**0.1297**	**Grand Total**



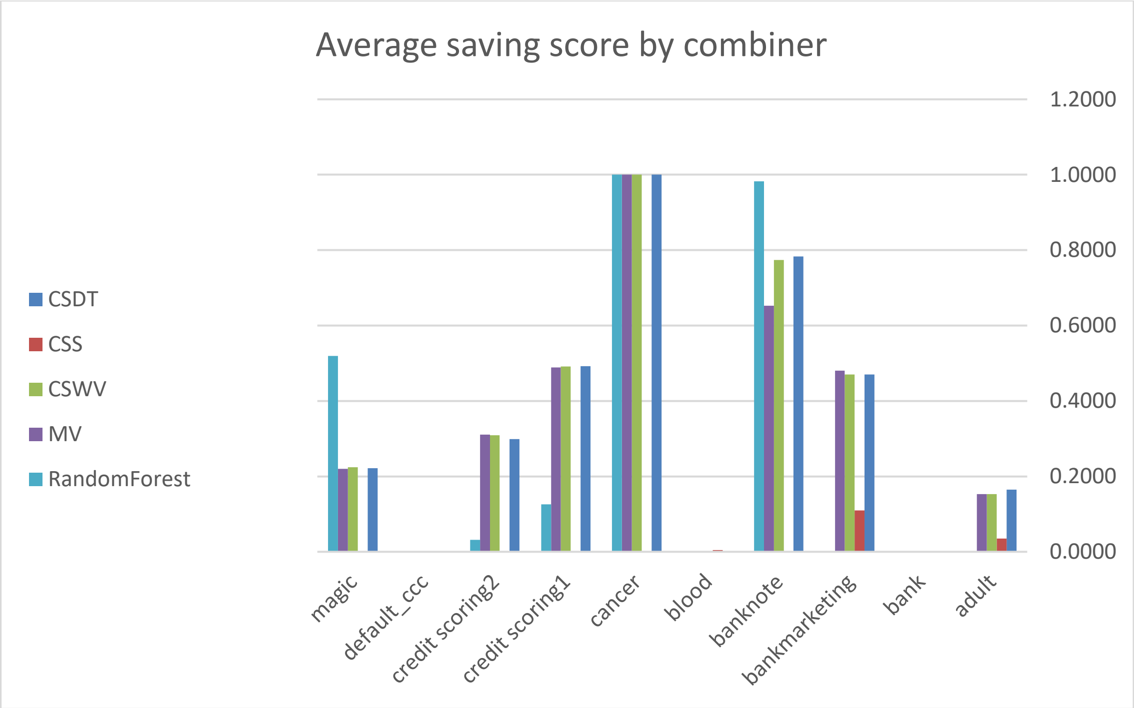
Average of PredictTIme	Row Labels
0.1297	Bagging
0.0018	CSDT
0.1278	Pasting
0.0149	RandomForest
0.1295	RF
0.1312	RP
0.1122	**Grand Total**



Legend by combiner:

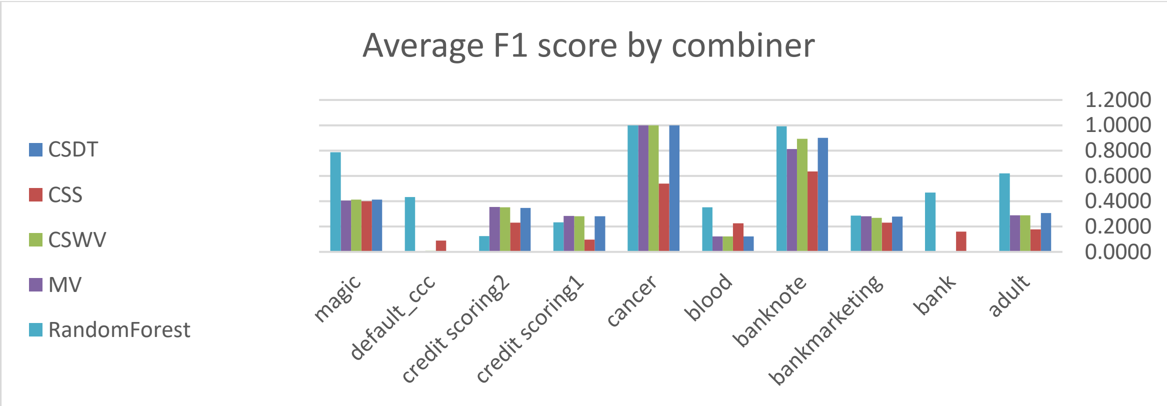
Saving score:

Grand Total	RandomForest	MV	CSWV	CSS	CSDT	Row Labels
0.1089	0.0000	0.1526	0.1526	0.0347	0.1645	adult
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	bank
0.3365	0.0000	0.4804	0.4702	0.1097	0.4700	bankmarketing
0.5337	0.9826	0.6523	0.7741	0.0000	0.7836	banknote
0.0012	0.0000	0.0000	0.0000	0.0041	0.0000	blood
0.7143	1.0000	1.0000	1.0000	0.0000	1.0000	cancer
0.3246	0.1254	0.4890	0.4917	0.0009	0.4924	credit scoring1
0.2008	0.0316	0.3109	0.3092	0.0000	0.2987	credit scoring2
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	default_ccc
0.1799	0.5196	0.2197	0.2242	0.0003	0.2218	magic
0.2400	**0.2659**	**0.3305**	**0.3422**	**0.0150**	**0.3431**	**Grand Total**

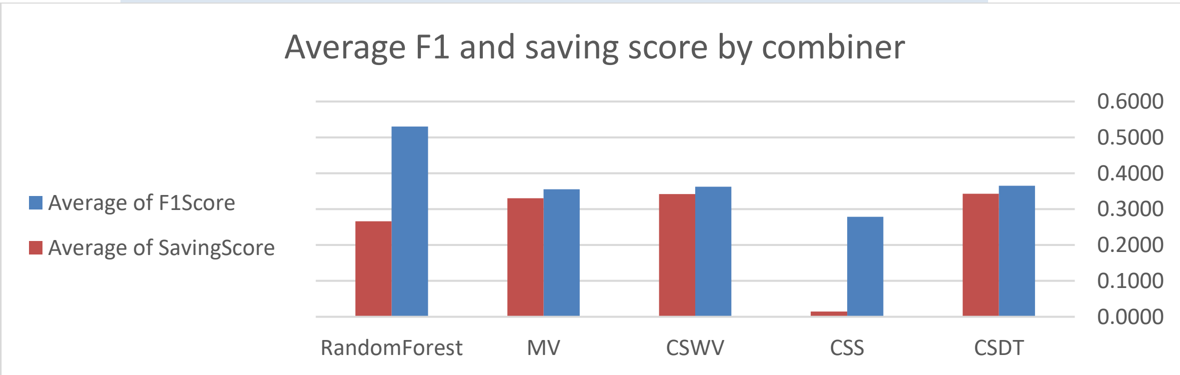


F1-Score:

Grand Total	RandomForest	MV	CSWV	CSS	CSDT	Row Labels
0.2813	0.6206	0.2880	0.2880	0.1767	0.3062	adult
0.0807	0.4683	0.0018	0.0018	0.1611	0.0018	bank
0.2639	0.2871	0.2812	0.2693	0.2315	0.2791	bankmarketing
0.8038	0.9922	0.8124	0.8927	0.6348	0.9016	banknote
0.1678	0.3529	0.1212	0.1212	0.2264	0.1212	blood
0.8682	1.0000	1.0000	1.0000	0.5389	1.0000	cancer
0.2266	0.2345	0.2845	0.2827	0.0970	0.2807	credit scoring1
0.3012	0.1236	0.3536	0.3515	0.2312	0.3481	credit scoring2
0.0597	0.4336	0.0016	0.0091	0.0895	0.0016	default_ccc
0.4335	0.7883	0.4057	0.4121	0.3994	0.4119	magic
0.3487	**0.5301**	**0.3550**	**0.3628**	**0.2787**	**0.3652**	**Grand Total**

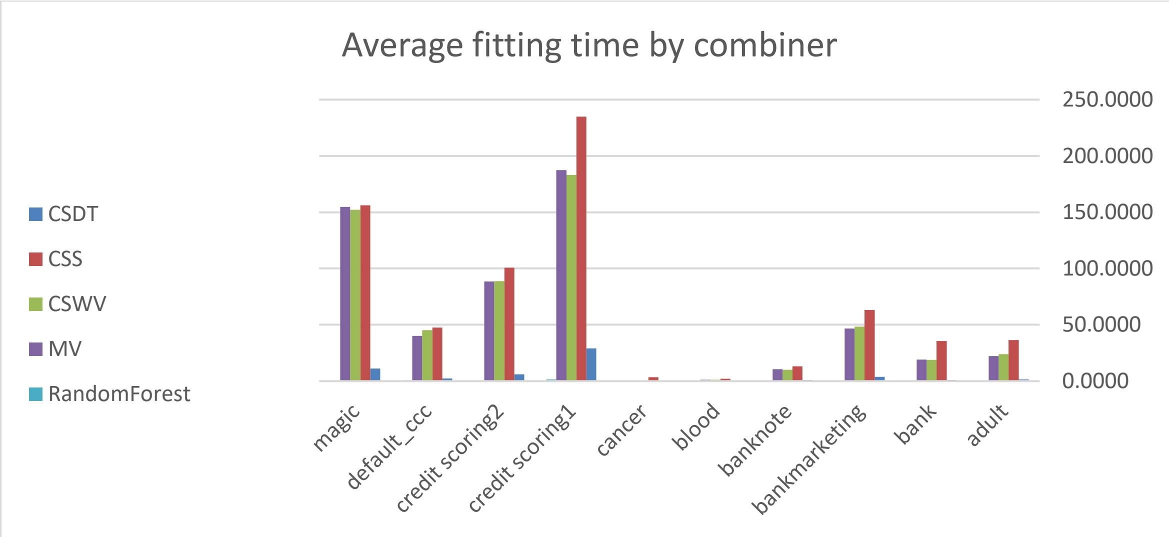


Average of SavingScore	Average of F1Score	Row Labels
0.3431	0.3652	CSDT
0.0150	0.2787	CSS
0.3422	0.3628	CSWV
0.3305	0.3550	MV
0.2659	0.5301	RandomForest
0.2400	**0.3487**	**Grand Total**

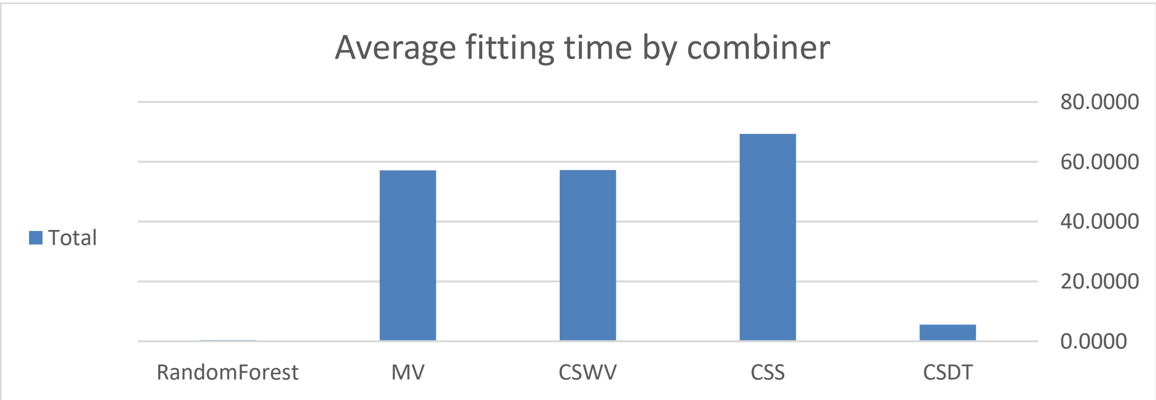


Fitting time:

Grand Total	RandomForest	MV	CSWV	CSS	CSDT	Row Labels
23.6864	0.2455	22.1396	23.9864	36.3365	1.5137	adult
21.1372	0.3913	19.2023	18.7718	35.6414	1.0680	bank
45.5120	0.2965	46.7858	48.3740	63.0902	3.8718	bankmarketing
9.6964	0.0210	10.5815	9.9204	13.2162	0.8559	banknote
1.3048	0.0175	1.2836	1.2942	1.9488	0.1432	blood
1.1785	0.0135	0.2986	0.3036	3.5148	0.0170	cancer
175.1127	1.4648	187.3158	183.1988	234.7622	29.0056	credit scoring1
79.8158	0.3464	88.3782	88.6586	100.7302	6.0063	credit scoring2
38.1623	0.6203	40.0389	45.1678	47.6299	2.3057	default_ccc
133.2202	0.4586	154.8403	152.2637	156.2558	11.1847	magic
52.8826	**0.3875**	**57.0864**	**57.1939**	**69.3126**	**5.5972**	**Grand Total**

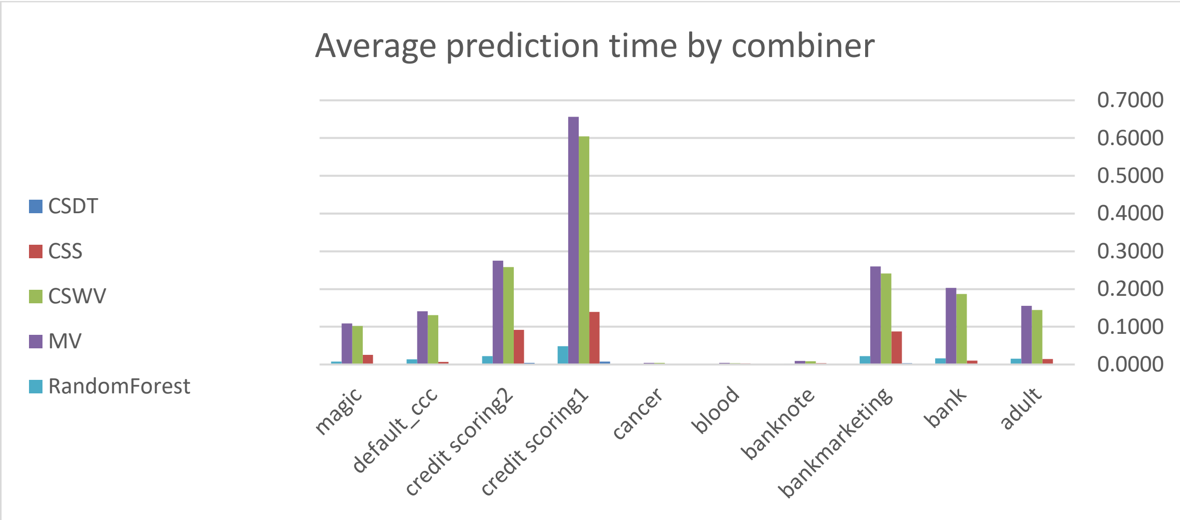


Average of FitTIme	Row Labels
5.5972	CSDT
69.3126	CSS
57.1939	CSWV
57.0864	MV
0.3875	RandomForest
52.8826	**Grand Total**

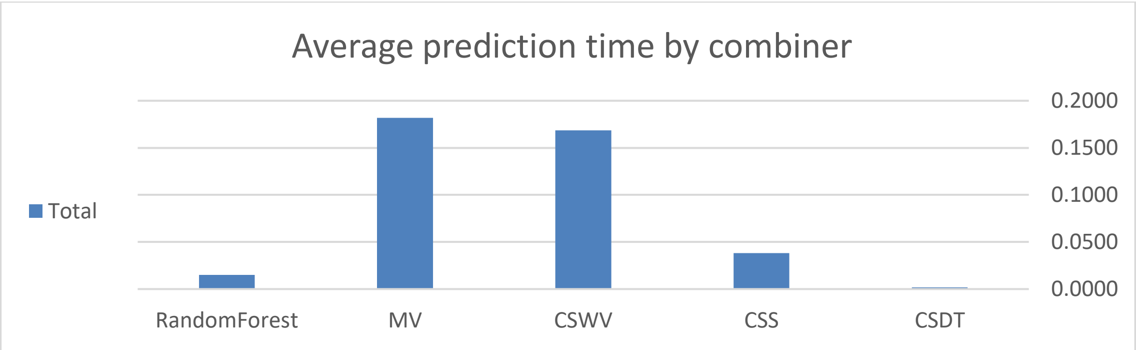


Prediction time:

Grand Total	RandomForest	MV	CSWV	CSS	CSDT	Row Labels
0.0911	0.0155	0.1553	0.1444	0.0149	0.0010	adult
0.1155	0.0165	0.2034	0.1865	0.0102	0.0000	bank
0.1701	0.0220	0.2596	0.2414	0.0877	0.0040	bankmarketing
0.0064	0.0015	0.0095	0.0090	0.0034	0.0005	banknote
0.0031	0.0010	0.0044	0.0037	0.0024	0.0000	blood
0.0028	0.0010	0.0042	0.0040	0.0012	0.0000	cancer
0.4041	0.0484	0.6564	0.6049	0.1390	0.0080	credit scoring1
0.1806	0.0220	0.2751	0.2581	0.0922	0.0040	credit scoring2
0.0808	0.0140	0.1415	0.1311	0.0066	0.0000	default_ccc
0.0682	0.0075	0.1090	0.1023	0.0253	0.0010	magic
0.1122	**0.0149**	**0.1818**	**0.1685**	**0.0383**	**0.0018**	**Grand Total**



Average of PredictTIme	Row Labels
0.0018	CSDT
0.0383	CSS
0.1685	CSWV
0.1818	MV
0.0149	RandomForest
0.1122	**Grand Total**



11. Conclusions

From the results of our experiment we had the following conclusions

1. The major drawback we mention before for the need of a domain expert to create the cost matrix is very clear as the results of our various algorithms which used a custom cost matrix are worse on the custom datasets than on the 3 costcla datasets.
2. The fitting time of the ensemble using the stacking combiner is significantly higher then any other methods and its results are always inferior in our implementation so we can say it's the least preferable method to use for the algorithm.
3. Random forest is always faster than the cost sensitive methods. We think this is because of the costcla implantation of the CSDT.
4. The different ensemble sizes did not show any significance change in the results but the runtime increases with each added tree so we can say it is best to use minimal size for the ensemble.
5. Normal random forest shows the best result on average across all data sets with regards to the F1-score. Which is reasonable because our algorithms are mainly used to optimize the saving score.
6. Single cost sensitive decision tree outperforms all the other algorithms on average in regards of saving score on each data set including the ones used in the original article.
7. The best combiner methods on average in regards of maximizing the saving score out of the three suggested is CSMW. As mention before CSS is the worst combiner method.
8. The best inducer methods on average in regards of maximizing the saving score out of the four suggested is random patches.
9. All the inducer methods gave similar results on the f1-score.
10. There is some obvious fault in the stacking and cost sensitive regression model as we can see that it got bad results across all data sets. We tried using *costcla* implementations of the stacking to check if the problem persists and it has. Which led us to believe either the *costcla* underline regression model has a problem.

12. Citations

At the moment of writing this report there are 5 citations for this work (one of them being a book citing a paper so overall 4 citations) all of cited this paper because they used one of the data sets the authors used in this paper or as an example of a method for solving cost sensitive problem.