

Ensembles of example-dependent cost-sensitive decision trees

April 28, 2015

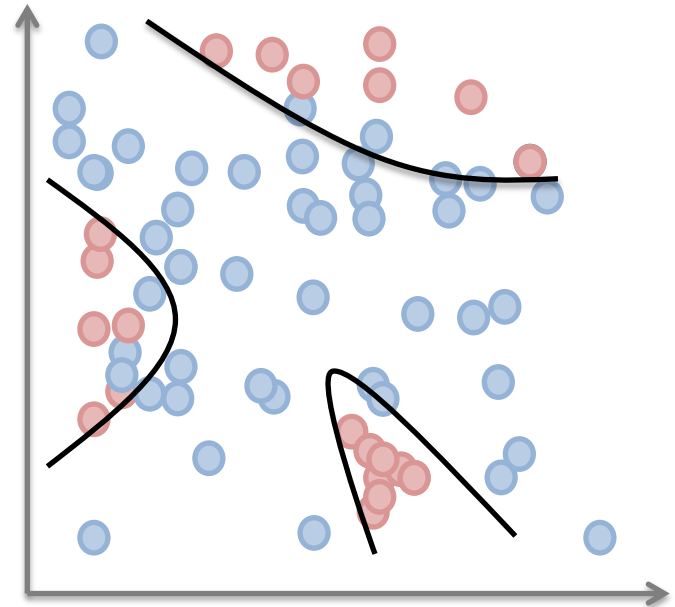
Alejandro Correa Bahnsen

with

Djamila Aouada, SNT
Björn Ottersten, SNT

Motivation

- Classification: **predicting the class of a set of examples given their features.**
- Standard classification methods aim at minimizing the errors
- Such a traditional framework assumes that all **misclassification errors carry the same cost**
- This is not the case in many real-world applications: **Credit card fraud detection, churn modeling, credit scoring, direct marketing.**



Agenda

- **Cost-sensitive classification**
Background, previous contributions
- **Cost-sensitive Ensembles**
Introduction, random inducers, combination methods, propose algorithms
- **Datasets**
Credit card fraud detection, churn modeling, credit scoring, direct marketing
- **Experiments**
Experimental setup, results
- **Conclusions**
Contributions

Background - Binary classification

predict the class of set of examples given their features

$$f : \mathcal{S} \rightarrow \{0, 1\}$$

Where each element of \mathcal{S} is composed by $\mathbf{X}_i = [x_i^1, x_i^2, \dots, x_i^k]$

It is usually evaluated using a traditional misclassification measure such as Accuracy, F1Score, AUC, among others.

However, these measures assumes that different misclassification errors carry the **same cost**

Background - Cost-sensitive evaluation

We define a cost measure based on the **cost matrix** [Elkan 2001]

	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	C_{TP_i}	C_{FP_i}
Predicted Negative $c_i = 0$	C_{FN_i}	C_{TN_i}

From which we calculate the **cost** of applying a classifier to a given set

$$Cost(f(\mathcal{S})) = \sum_{i=1}^N y_i(c_i C_{TP_i} + (1 - c_i) C_{FN_i}) + (1 - y_i)(c_i C_{FP_i} + (1 - c_i) C_{TN_i})$$

Background - Cost-sensitive evaluation

However, the total cost may not be easy to interpret. Therefore, we propose a **savings measure** as the cost vs. the cost of using no algorithm at all

$$\text{Savings}(f(\mathcal{S})) = \frac{\text{Cost}_l(\mathcal{S}) - \text{Cost}(f(\mathcal{S}))}{\text{Cost}_l(\mathcal{S})}$$

Where $\text{Cost}_l(\mathcal{S})$ is the cost of predicting the costless class

$$\text{Cost}_l(\mathcal{S}) = \min\{\text{Cost}(f_0(\mathcal{S})), \text{Cost}(f_1(\mathcal{S}))\}$$

Background - State-of-the-art methods

Research in example-dependent cost-sensitive classification has been narrow, mostly because of the **lack of publicly available datasets** [Aodha and Brostow 2013].

Standard approaches consist in **re-weighting the training examples** based on their costs:

- Cost-proportionate rejection sampling [Zadrozny et al. 2003]
- Cost-proportionate oversampling [Elkan 2001]

Previous contributions

- **Bayes minimum risk**

A. Correa Bahnsen, A. Stojanovic, D. Aouada, and B. Ottersten, “Cost Sensitive Credit Card Fraud Detection Using Bayes Minimum Risk,” in 2013 12th International Conference on Machine Learning and Applications. Miami, USA: IEEE, Dec. 2013, pp. 333–338.

- **Probability calibration for Bayes minimum risk (BMR)**

A. Correa Bahnsen, A. Stojanovic, D. Aouada, and B. Ottersten, “Improving Credit Card Fraud Detection with Calibrated Probabilities,” in Proceedings of the fourteenth SIAM International Conference on Data Mining, Philadelphia, USA, 2014, pp. 677 – 685.

- **Cost-sensitive logistic regression (CSLR)**

A. Correa Bahnsen, D. Aouada, and B. Ottersten, “Example-Dependent Cost-Sensitive Logistic Regression for Credit Scoring,” in 2014 13th International Conference on Machine Learning and Applications. Detroit, USA: IEEE, 2014, pp. 263–269.

- **Cost-sensitive decision trees (CSDT)**

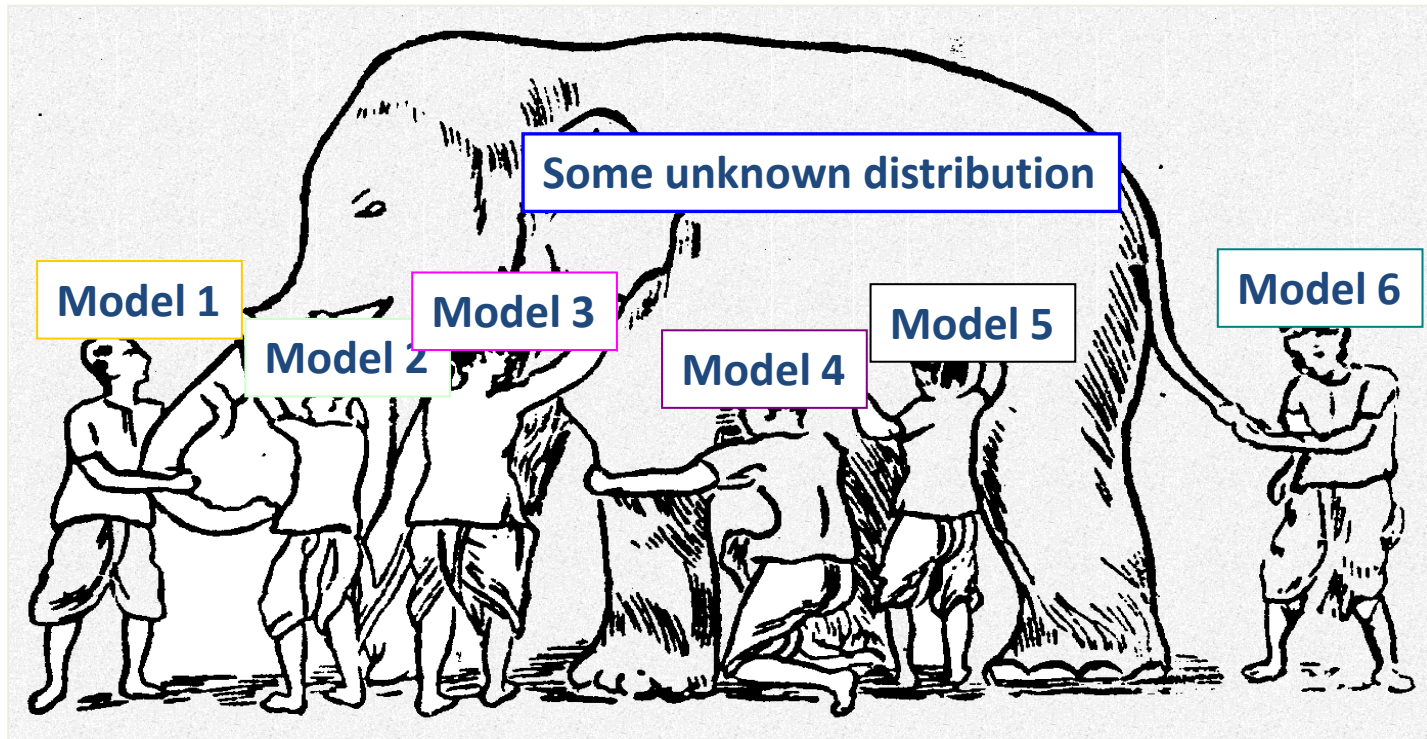
A. Correa Bahnsen, D. Aouada, and B. Ottersten, “Example-Dependent Cost-Sensitive Decision Trees,” Expert Systems with Applications, in press, 2015.

Agenda

- **Cost-sensitive classification**
Background, previous contributions
- **Cost-sensitive Ensembles**
Introduction, random inducers, combination methods, propose algorithms
- **Datasets**
Credit card fraud detection, churn modeling, credit scoring, direct marketing
- **Experiments**
Experimental setup, results
- **Conclusions**
Contributions

Introduction - Ensemble learning

The main idea behind the ensemble methodology is to **combine several individual base classifiers** in order to have a classifier that outperforms everyone of them

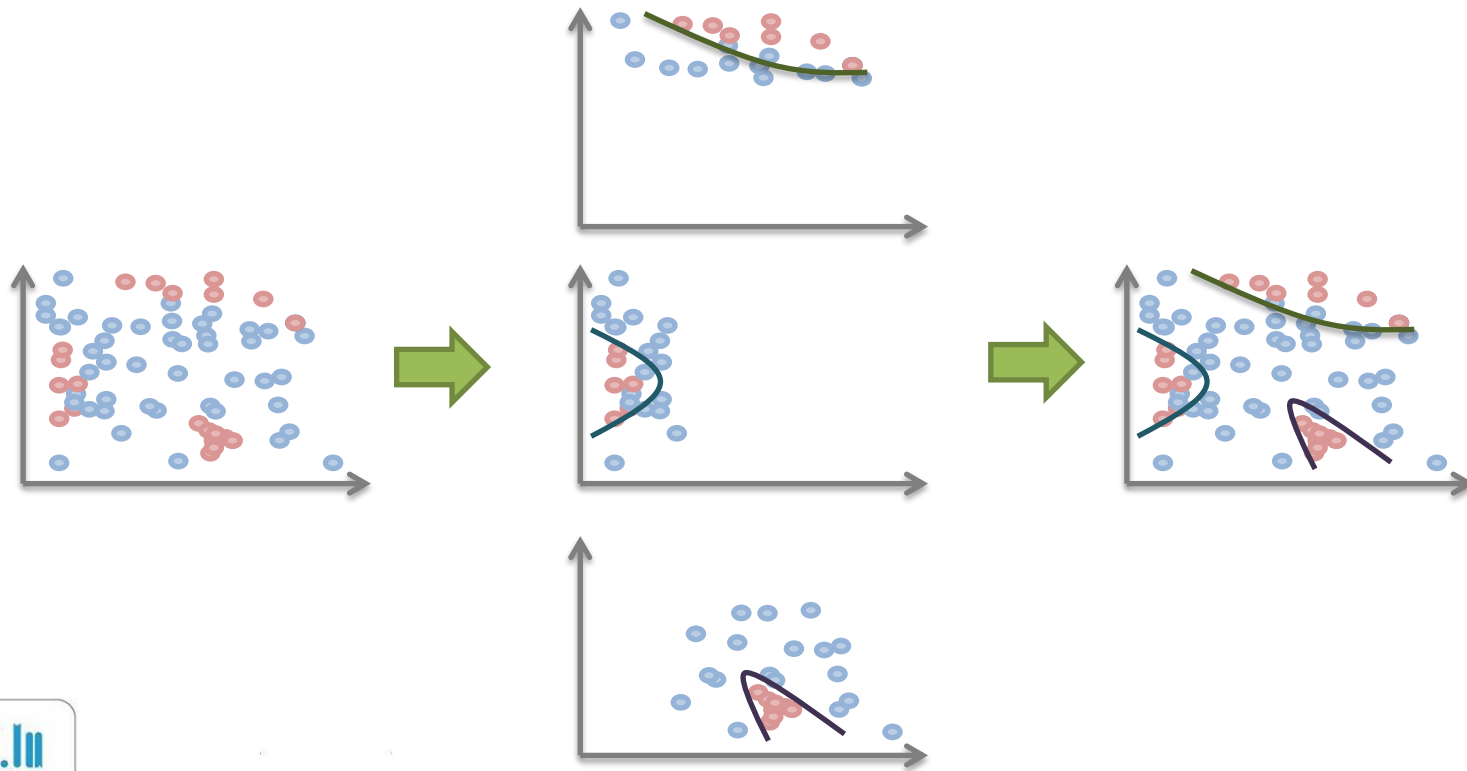


“The Blind Men and the Elephant”, Godfrey Saxe’s

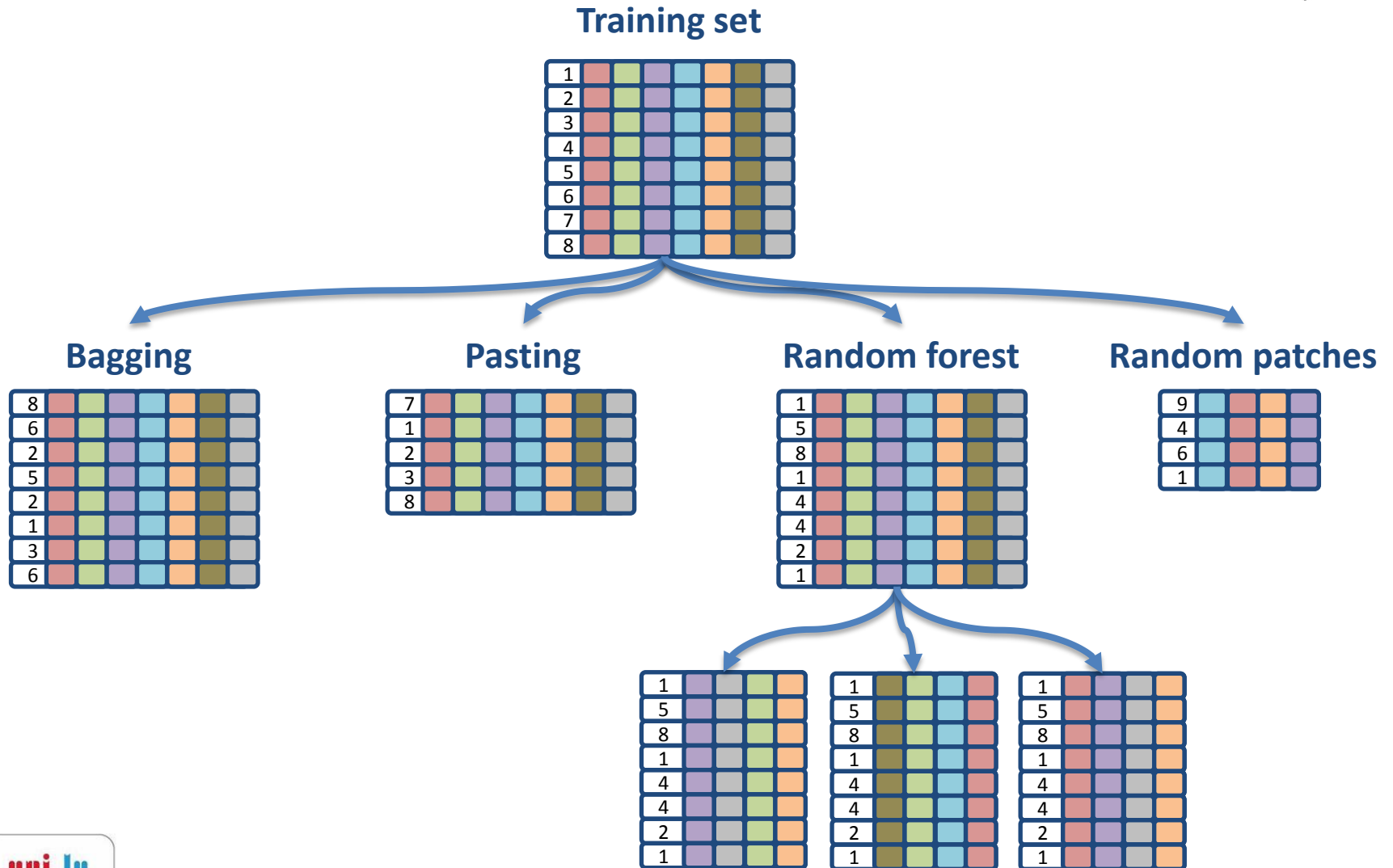
Introduction - Ensemble learning

Typical ensemble is made by combining T different **base classifiers**. Each base classifier is trained by applying algorithm M in a random subset

$$M_j \equiv M(\mathcal{S}_j) \quad \forall j \in \{1..T\}$$



Random inducers



Proposed combination methods

After the base classifiers are constructed they are typically combined using one of the following methods:

- **Majority voting**

$$H(\mathcal{S}) = f_{mv}(\mathcal{S}, \mathcal{M}) = \arg \max_{c \in \{0,1\}} \sum_{j=1}^T \mathbf{1}_c(M_j(\mathcal{S}))$$

- **Proposed cost-sensitive weighted voting**

$$H(\mathcal{S}) = f_{wv}(\mathcal{S}, \mathcal{M}, \alpha) = \arg \max_{c \in \{0,1\}} \sum_{j=1}^T \alpha_j \mathbf{1}_c(M_j(\mathcal{S}))$$

$$\alpha_j = \frac{1 - \epsilon(M_j(\mathcal{S}_j^{oob}))}{\sum_{j_1=1}^T 1 - \epsilon(M_{j_1}(\mathcal{S}_{j_1}^{oob}))} \quad \longrightarrow \quad \alpha_j = \frac{\text{Savings}(M_j(\mathcal{S}_j^{oob}))}{\sum_{j_1=1}^T \text{Savings}(M_{j_1}(\mathcal{S}_{j_1}^{oob}))}$$

$$\mathcal{S}_j^{oob} = \mathcal{S} - \mathcal{S}_j$$

Proposed combination methods

- **Proposed cost-sensitive stacking**

$$H(\mathcal{S}) = f_s(\mathcal{S}, \mathcal{M}, \beta) = \frac{1}{1 + e^{-\left(\sum_{j=1}^T \beta_j M_j(\mathcal{S})\right)}}$$

Using the cost-sensitive logistic regression [Correa et. al, 2014] model:

$$J(\mathcal{S}, \mathcal{M}, \beta) = \sum_{i=1}^N \left[y_i \left(f_s(\mathbf{X}_i, \mathcal{M}, \beta) \cdot (C_{TP_i} - C_{FN_i}) + C_{FN_i} \right) + \right. \\ \left. (1 - y_i) \left(f_s(\mathbf{X}_i, \mathcal{M}, \beta) \cdot (C_{FP_i} - C_{TN_i}) + C_{TN_i} \right) \right]$$

Then the weights are estimated using

$$\beta = \arg \min_{\beta \in \mathbb{R}^T} J(\mathcal{S}, \mathcal{M}, \beta)$$

Proposed algorithms

Base classifiers

For j in $1..T$:

1. Subsample from training set

$$S_j \leftarrow \text{Subsample}(S)$$

2. Train a CSDT on S_j

$$M_j \leftarrow M(S_j)$$

3. Estimate the weight

$$\alpha_j \leftarrow \text{savings} \left(M_j \left(S_j^{\text{oob}} \right) \right)$$



Combination

Select combination method:

1. Majority voting

$$H \leftarrow f_{mv}(S, M)$$

2. CS-Weighted voting

$$H \leftarrow f_{mv}(S, M, \alpha)$$

3. CS-Stacking

$$\beta \leftarrow \text{argmin} J(S, M, \beta)$$

$$H \leftarrow f_s(S, M, \beta)$$

The subsampling can be done either by: Bagging, pasting, random forest or random patches

Agenda

- **Cost-sensitive classification**
Background, previous contributions
- **Cost-sensitive Ensembles**
Introduction, random inducers, combination methods, propose algorithms
- **Datasets**
Credit card fraud detection, churn modeling, credit scoring, direct marketing
- **Experiments**
Experimental setup, results
- **Conclusions**
Contributions

Credit card fraud detection

Cost matrix

	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	C_a	C_a
Predicted Negative $c_i = 0$	Amt_i	0

Database

# Examples	% Positives	Cost (Euros)
1,638,772	0.21%	860,448

A. Correa Bahnsen, A. Stojanovic, D. Aouada, and B. Ottersten, "Cost Sensitive Credit Card Fraud Detection Using Bayes Minimum Risk," in 2013 12th International Conference on Machine Learning and Applications. Miami, USA: IEEE, Dec. 2013, pp. 333–338.

Churn modeling

Cost matrix

	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Pos $c_i = 1$	$C_{TP_i} = \gamma_i C_{o_i} + (1 - \gamma_i)(CLV_i + C_a)$	$C_{FP_i} = C_{o_i} + C_a$
Predicted Neg $c_i = 0$	$C_{FN_i} = CLV_i$	$C_{TN_i} = 0$

Database

# Examples	% Positives	Cost (Euros)
9,410	4.83%	580,884

A. Correa Bahnsen, A. Stojanovic, D. Aouada, and B. Ottersten, "A novel costsensitive framework for customer churn predictive modeling," Decision Analytics, vol. under review, 2015.

Credit scoring

Cost matrix

	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	0	$r_i + C_{FP}^a$
Predicted Negative $c_i = 0$	$Cl_i \cdot L_{gd}$	0

Database

	# Examples	% Positives	Cost (Euros)
Kaggle Credit	112,915	6.74%	83,740,181
PAKDD09 Credit	38,969	19.88%	3,117,960

A. Correa Bahnsen, D. Aouada, and B. Ottersten, "Example-Dependent Cost-Sensitive Logistic Regression for Credit Scoring," in 2014 13th International Conference on Machine Learning and Applications. Detroit, USA: IEEE, 2014, pp. 263–269.

Direct marketing

Cost matrix

	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	C_a	C_a
Predicted Negative $c_i = 0$	Int_i	0

Database

# Examples	% Positives	Cost (Euros)
37,931	12.62%	59,507

A. Correa Bahnsen, A. Stojanovic, D. Aouada, and B. Ottersten, "Improving Credit Card Fraud Detection with Calibrated Probabilities," in Proceedings of the fourteenth SIAM International Conference on Data Mining, Philadelphia, USA, 2014, pp. 677 – 685.

Agenda

- **Cost-sensitive classification**
Background, previous contributions
- **Cost-sensitive Ensembles**
Introduction, random inducers, combination methods, propose algorithms
- **Datasets**
Credit card fraud detection, churn modeling, credit scoring, direct marketing
- **Experiments**
Experimental setup, results
- **Conclusions**
Contributions

Experimental setup - Methods

- **Cost-insensitive (CI):**
 - Decision trees (DT)
 - Logistic regression (LR)
 - Random forest (RF)
 - Under-sampling (u)
- **Cost-proportionate sampling (CPS):**
 - Cost-proportionate rejection-sampling (r)
 - Cost-proportionate over-sampling (o)
- **Bayes minimum risk (BMR)**
- **Cost-sensitive training (CST):**
 - Cost-sensitive logistic regression (CSLR)
 - Cost-sensitive decision trees (CSDT)

Experimental setup - Methods

- **Ensemble cost-sensitive decision trees (ECSDT):**

Random inducers:

- Bagging (CSB)
- Pasting (CSP)
- Random forest (CSRF)
- Random patches (CSRП)

Combination:

- Majority voting (mv)
- Cost-sensitive weighted voting (wv)
- Cost-sensitive staking (s)

Experimental setup

- Each experiment was carry out **50 times**
- For the parameters of the algorithms a **grid search** was made
- Results are measured by **savings**
- Then the **Friedman ranking** is calculated for each method

Results

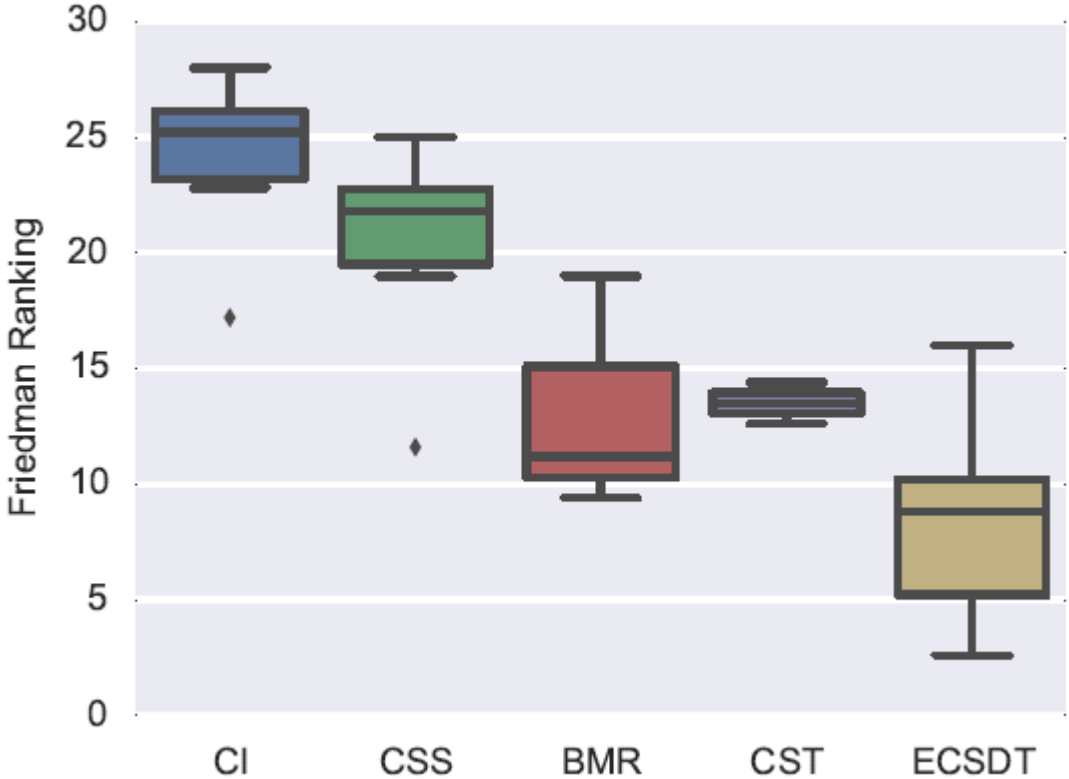
Results of the **Friedman rank** of the savings (1=best, 28=worst)

Family	Algorithm	Rank
ECSDT	CSRP-wv-t	2.6
ECSDT	CSRP-s-t	3.4
ECSDT	CSRP-mv-t	4
ECSDT	CSB-wv-t	5.6
ECSDT	CSP-wv-t	7.4
ECSDT	CSB-mv-t	8.2
ECSDT	CSRF-wv-t	9.4
BMR	RF-t-BMR	9.4
ECSDT	CSP-s-t	9.6
ECSDT	CSP-mv-t	10.2
ECSDT	CSB-s-t	10.2
BMR	LR-t-BMR	11.2
CPS	RF-r	11.6
CST	CSDT-t	12.6

Family	Algorithm	Rank
CST	CSLR-t	14.4
ECSDT	CSRF-mv-t	15.2
ECSDT	CSRF-s-t	16
CI	RF-u	17.2
CPS	LR-r	19
BMR	DT-t-BMR	19
CPS	LR-o	21
CPS	DT-r	22.6
CI	LR-u	22.8
CPS	RF-o	22.8
CI	DT-u	24.4
CPS	DT-o	25
CI	DT-t	26
CI	RF-t	26.2

Results

Results of the **Friedman rank** of the savings organized by family



Results

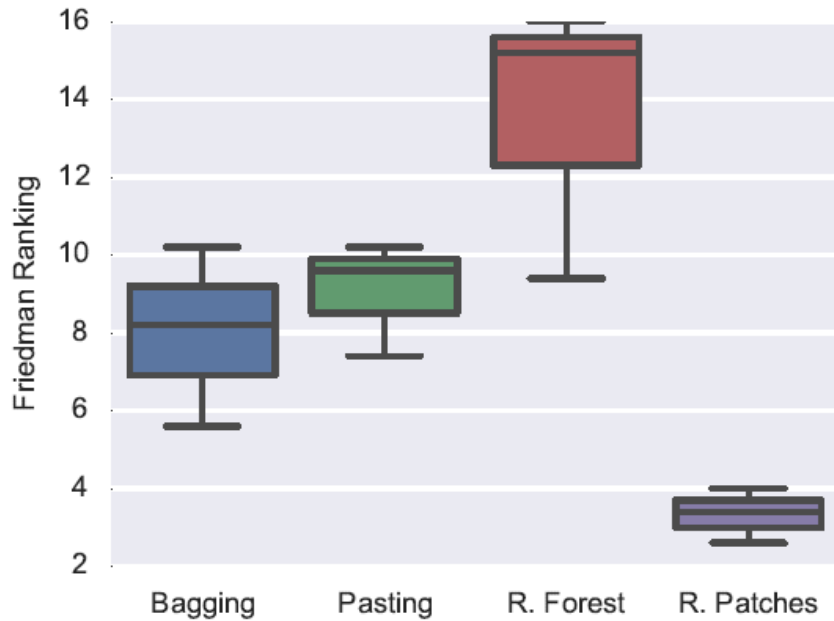
Database	Algorithm	Savings
Fraud	CSRP-wv-t	0.73
Churn	CSRP-s-t	0.17
Credit1	CSRP-mv-t	0.52
Credit2	LR-t-BMR	0.31
Marketing	LR-t-BMR	0.5

Percentage of the **highest savings**



Results within the ECSDT family

By random inducer



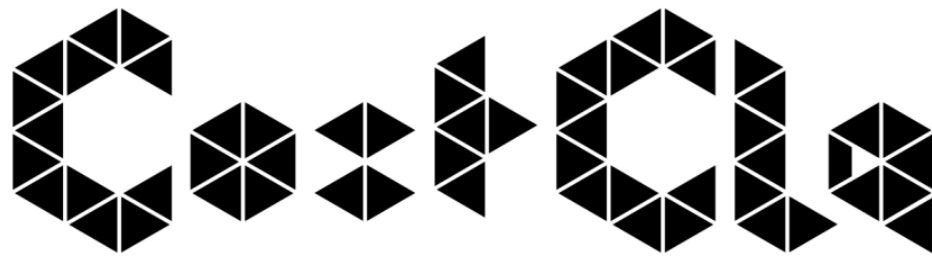
By combination method



Conclusions

- New framework for **ensembles of example dependent cost-sensitive decision trees**
- Using five databases, from four real-world applications: credit card fraud detection, churn modeling, credit scoring and direct marketing, we show that the **proposed algorithm significantly outperforms** the state-of-the-art cost-insensitive and example-dependent cost-sensitive algorithms
- Highlight the importance of using the **real example-dependent financial costs** associated with the real-world applications

Costcla - Software



CostCla is a Python module for **cost-sensitive machine learning** built on top of Scikit-Learn, SciPy and distributed under the 3-Clause BSD license.

In particular, it provides:

- A set of example-dependent cost-sensitive algorithms
- Different real-world example-dependent cost-sensitive datasets.

Installation

pip install costcla

Documentation: <https://pythonhosted.org/costcla/>

Development: <https://github.com/albahnsen/CostSensitiveClassification>

Prepare dataset and load libraries

```
In [38]: from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import train_test_split
from costcla.metrics import savings_score
from costcla.datasets import load_creditscoring2
from costcla.sampling import cost_sampling
from costcla import models
data = load_creditscoring2()
X_train, X_test, y_train, y_test,
cost_mat_train, cost_mat_test = \
train_test_split(data.data, data.target, data.cost_mat)
```

Random forest

```
In [19]: f_RF = RandomForestClassifier()
y_pred = f_RF.fit(X_train, y_train).predict(X_test)
print savings_score(y_test, y_pred, cost_mat_test)

0.042197359989
```

cost-proportionate rejection sampling

```
In [26]: X_cps_r, y_cps_r, cost_mat_cps_r = \
cost_sampling(X_train, y_train, cost_mat_train,
method='RejectionSampling')
y_pred = f_RF.fit(X_cps_r, y_cps_r).predict(X_test)
print savings_score(y_test, y_pred, cost_mat_test)

0.280743761779
```

Bayes minimum risk

```
In [30]: f_RF.fit(X_train, y_train)
y_prob_test = f_RF.predict_proba(X_test)
f_BMR = models.BayesMinimumRiskClassifier()
f_BMR.fit(y_test, y_prob_test)
y_pred = f_BMR.predict(y_prob_test, cost_mat_test)
print savings_score(y_test, y_pred, cost_mat_test)

0.285102564249
```

cost-sensitive decision tree

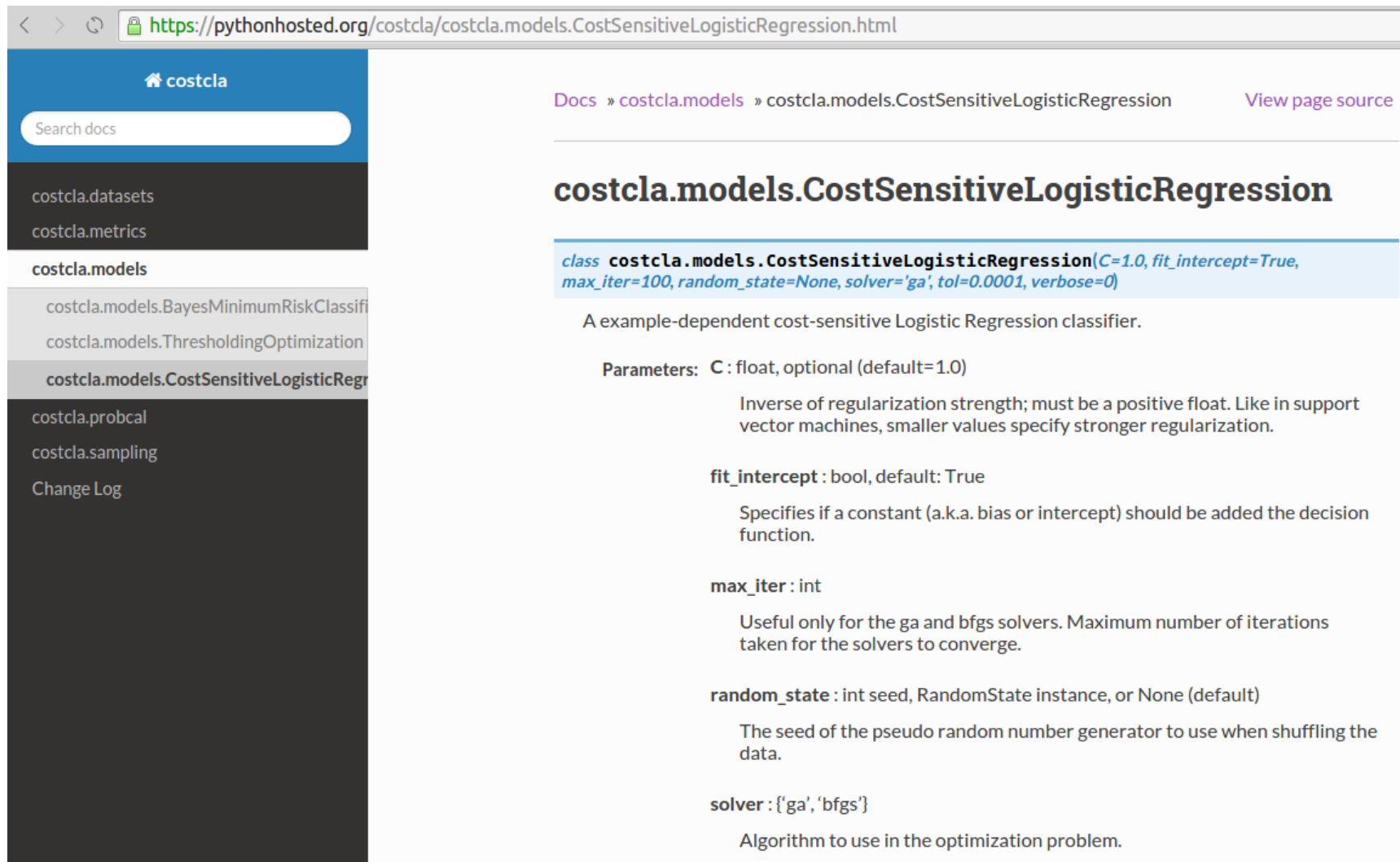
```
In [2]: f_CSDT = models.CSDecisionTreeClassifier()
f_CSDT.fit(data.data, data.target, data.cost_mat)
y_pred = f_CSDT.predict(data.data)
print savings_score(data.target, y_pred, data.cost_mat)

0.289489571352
```

cost-sensitive random patches

```
In [33]: f_CSRP = costcla.models.CSRandomPatchesClassifier()
f_CSRP.fit(data.data, data.target, data.cost_mat)
y_pred = f_CSRP.predict(data.data)
print savings_score(data.target, y_pred, data.cost_mat)

0.306607400467
```



Docs » [costcla.models](#) » [costcla.models.CostSensitiveLogisticRegression](#) [View page source](#)

costcla.models.CostSensitiveLogisticRegression

```
class costcla.models.CostSensitiveLogisticRegression(C=1.0, fit_intercept=True,
max_iter=100, random_state=None, solver='ga', tol=0.0001, verbose=0)
```

A example-dependent cost-sensitive Logistic Regression classifier.

Parameters:

- C** : float, optional (default=1.0)
Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.
- fit_intercept** : bool, default: True
Specifies if a constant (a.k.a. bias or intercept) should be added the decision function.
- max_iter** : int
Useful only for the ga and bfgs solvers. Maximum number of iterations taken for the solvers to converge.
- random_state** : int seed, RandomState instance, or None (default)
The seed of the pseudo random number generator to use when shuffling the data.
- solver** : {'ga', 'bfgs'}
Algorithm to use in the optimization problem.

Thank You!!

Alejandro Correa Bahnsen