BoostClean: Automated Error Detection and Repair for Machine Learning

SANJAY KRISHNAN, MICHAEL J. FRANKLIN, KEN GOLDBERG,

EUGENE WU

Presented by Christopher Buss

Introduction > Contributions

What we'll see in this paper:

1. **Cleaning as Boosting**
   - Using **statistical boosting (ensembles)**, automatically select the best cleaning operations from a library in order to maximize the predictive performance of a downstream model
2. **Automatic Model Improvements**
   - Evaluate BoostClean on 12 different datasets
   - Improved prediction accuracy of up to 9% compared to non-ensembled approaches
3. **Error Detection Library**
   - Built-in error detection library
   - Includes Word2Vec featurization
   - Achieves 81% accuracy of all errors found by hand-written rules
4. **Optimizations**
   - Parallelism, materialization, and indexing
   - 22.2x end-to-end speedup on a 16-core machine

Introduction > Problem Overview

Dirty data can bias predictions and degrade model accuracy.

- Recall Simpson's Paradox (discussed in ActiveClean paper)

Introduction > Problem Overview

Dirty data can bias predictions and degrade model accuracy.

- Recall Simpson's Paradox (discussed in ActiveClean paper)

Garbage in, garbage out

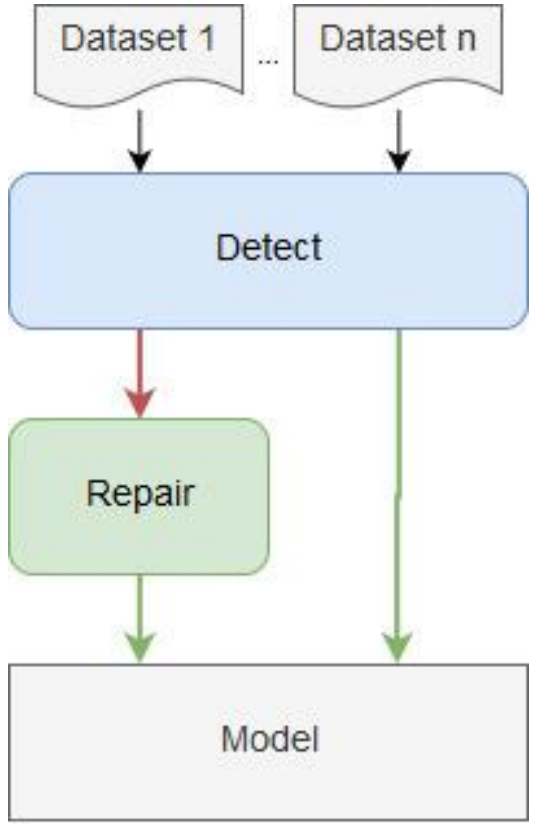
- Must clean data in the correct way for a good model

Cleaning is a necessity! How is this normally done?

Introduction > The General Pipeline

Pipeline of scripts to downstream model

1. Take in **n** datasets
2. Use Booleans to **detect** a subset of dirty data
3. Apply **repairs** to fix those records



Introduction > The General Pipeline

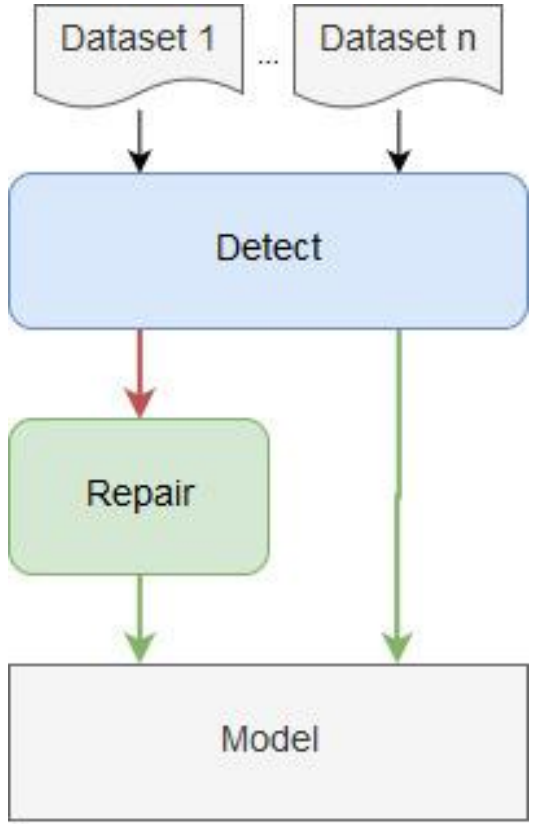
Pipeline of scripts to downstream model

1. Take in **n** datasets
2. Use Booleans to **detect** a subset of dirty data
3. Apply **repairs** to fix those records

More data, more problems

- New data means new (unforeseen) problems
- Must update **detect** and **repair** scripts regularly

Q: How can human effort be reduced?



Introduction > The General Pipeline

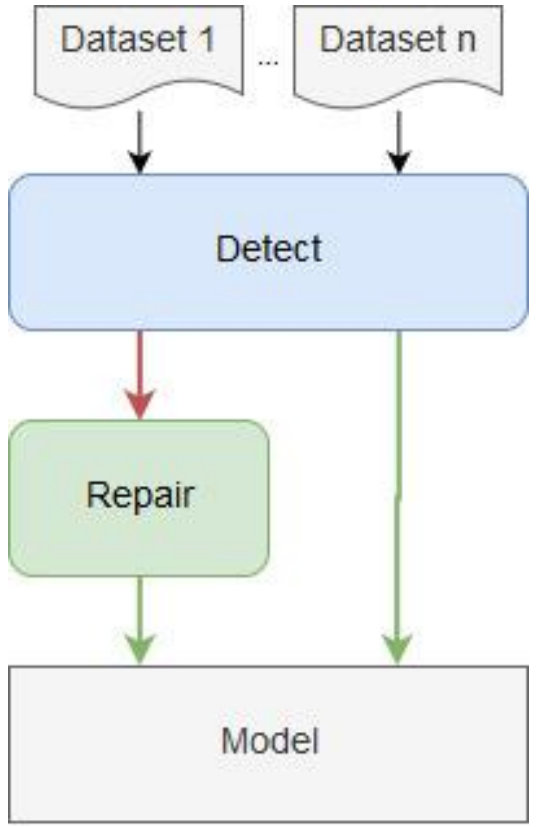
Pipeline of scripts to downstream model

1. Take in **n** datasets
2. Use Booleans to **detect** a subset of dirty data
3. Apply **repairs** to fix those records

More data, more problems

- New data means new (unforeseen) problems
- Must update **detect** and **repair** scripts regularly

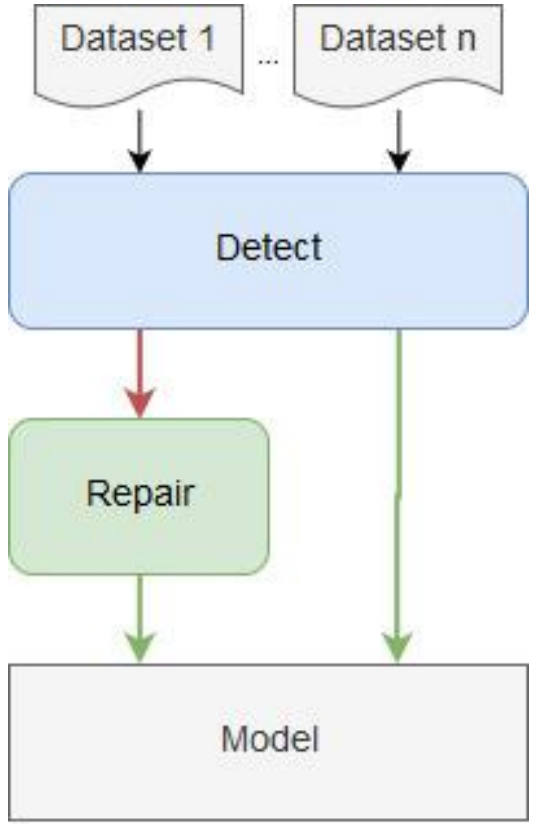
Q: How can human effort be reduced?
A: Automate the **detection** and **repair** of a common class of data errors called **domain value violations** (values outside the domain of the given type)



Introduction > BoostClean

BoostClean

Goal: Find the best ensemble of detect and repair operations in order to repair **domain value violations** and maximize the predictive performance of model



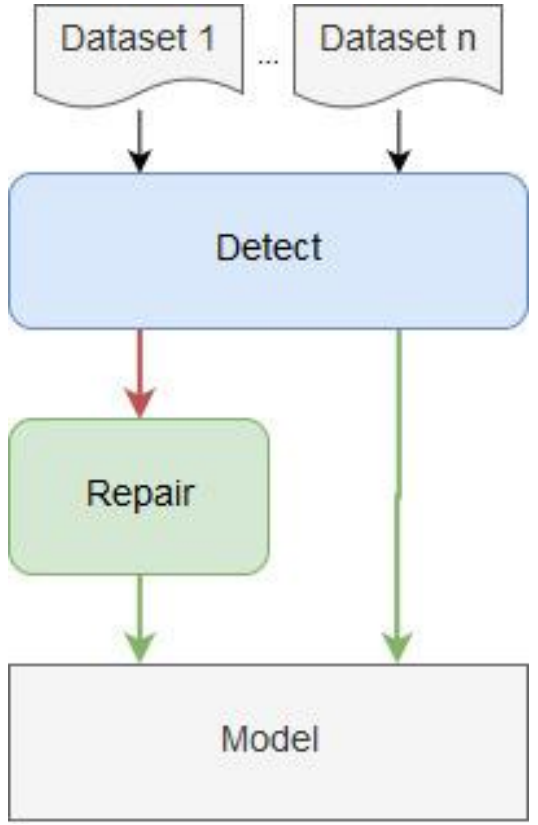
Introduction > BoostClean

BoostClean

Goal: Find the best ensemble of detect and repair operations in order to repair **domain value violations** and maximize the predictive performance of model

Components:

1. Error Detector **(detect)**
   - Generate predicates (rules) using **Isolation forests** on **featurized** values
2. Repair Selector **(repair)**
   - Use **boosting** to generate a sequence of repair scripts



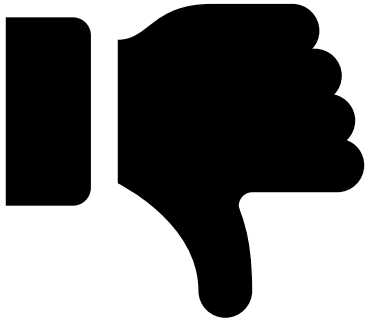
Background > Some Notes

What BoostClean does:

- Address systemic errors due to invariant violations that lead to unforeseen biases in the model
  - Think ActiveClean

What BoostClean does not do:

- Repair the entire relation
  - this could actually degrade performance!
- Address more difficult errors
  - Leave things like deduplication up to engineers



Problem Statement > Setup

BoostClean takes…

1. A set of test and train data
   - $(X_{train}, Y_{train}), (X_{test}, Y_{test})$ (Y_{test} must be non-dirty!)
2. A user defined function that returns a classifier C
   - $train(X_{train}, Y_{train})$
     - BoostClean will repair data and pass it to $train()$
     - No need to be concerned with detect/repair pipeline
3. A library of detectors* to generate predicates p_i
   - $\mathcal{D} = \{d_1, \dots\}$
4. A library of repair functions* to apply to data
   - $\mathcal{F} = \{f_1, \dots\}$

*Optional. BoostClean has default $\mathcal{D}$ and $\mathcal{F}$, but user can extend library with domain-specific functions

Problem Statement > Setup > Lead Prediction Example

Leads: *R(id, name, num_emp, industry, region, successful)*

Goal: Predict whether a lead is **successful**

Provide:

Problem Statement > Setup > Lead Prediction Example

Leads: *R(id, name, num_emp, industry, region, successful)*

Goal: Predict whether a lead is **successful**

Provide:

1. $X_{train,\,test} = \{name, num\_emp, \text{industry}, \text{region}\}$
2. $Y_{train,\,test} = \{successful\}$

Problem Statement > Setup > Lead Prediction Example

Leads: *R(id, name, num_emp, industry, region, successful)*

Goal: Predict whether a lead is **successful**

Provide:

1. $X_{train,\,test} = \{name, num\_emp, \text{industry}, \text{region}\}$
2. $Y_{train,\,test} = \{successful\}$
3. $train(.)$
   - Returns an SVM C that predicts y_i (true, false) given row x_i

Problem Statement > Setup > Lead Prediction Example

Leads: *R(id, name, num_emp, industry, region, successful)*

Goal: Predict whether a lead is **successful**

Provide:

1. $X_{train,\,test} = \{name, num\_emp, \text{industry}, \text{region}\}$
2. $Y_{train,\,test} = \{successful\}$
3. $train(.)$
   - Returns an SVM C that predicts y_i (true, false) given row x_i

Evaluate C :

$$acc(C) = \frac{|\{\forall x, y \in (X_{test}, Y_{test}) : C\big((x, null)\big).y = y\}|}{|Y_{test}|}$$

Problem Statement > Setup

BoostClean takes…

1. A set of test and train data
   - $(X_{train}, Y_{train}), (X_{test}, Y_{test})$ (Y_{test} must be non-dirty!)
2. A user defined function that returns a classifier C
   - $train(X_{train}, Y_{train})$
     - BoostClean will repair data and pass it to $train()$
     - No need to be concerned with detect/repair pipeline
3. A library of detectors* to generate predicates p_i
   - $\mathcal{D} = \{d_1, \dots\}$
4. A library of repair functions* to apply to data
   - $\mathcal{F} = \{f_1, \dots\}$

*Optional. BoostClean has default $\mathcal{D}$ and $\mathcal{F}$, but user can extend library with domain-specific functions

Problem Statement > Detection Generators and Predicates

Use $\mathrm{d_i} \in \mathcal{D}$ to generate predicate

- $d_i(data)$ returns predicate p_i
  - Example: $p_i(r) = r.n\_emp \leq 0$ returns $\{n\_emp\}$ if true, otherwise $\emptyset$

Use $f_i \in \mathcal{F}$ to repair data

- *Data repairs*
  - modify labels, delete records
- *Prediction repairs*
  - default prediction if data is too corrupt

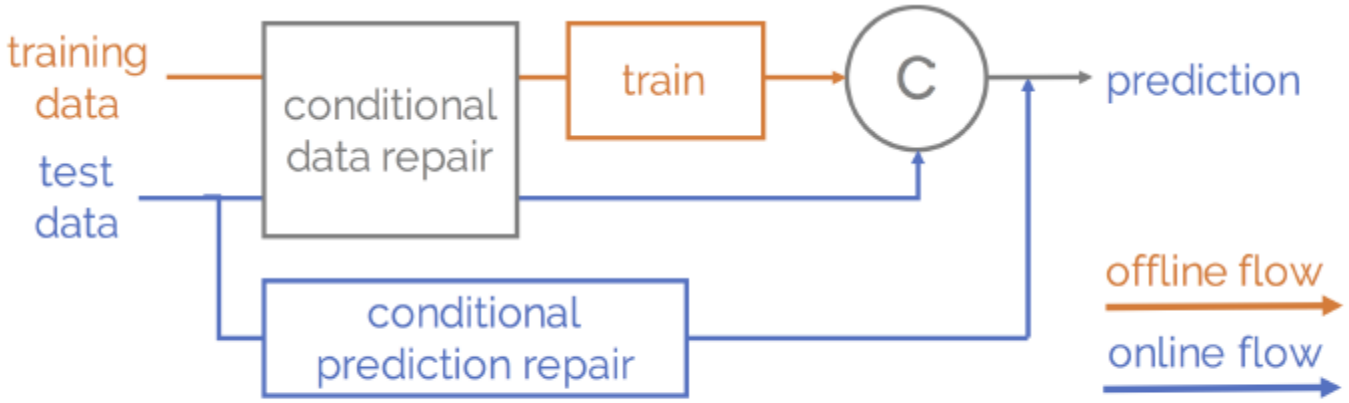


Figure 2: Offline (orange) and online (blue) workflows.

Problem Statement > Detection Generators and Predicates

Putting it together with *conditional repairs*

1. Combine p_i and f_i into repair functions:

```
def generate_repair(p, f):
  def repair(r, r_orig):
    if p(r): r = f(r)
    return r
  return apply
```

NOTE: Repair functions take r AND r_{orig} (we'll see why)

Problem Statement > Detection Generators and Predicates > Examples

Data repair:

```
def repair(r, r_orig):
  if r.region in ('USWest', 'USWESTERN'):
    r.region = 'USW'
  return r
```

Prediction repair:

```
def repair(r, r_orig):
  if r_orig.name == None:
    r.y = False
    return r
  return C(r)
```

Problem Statement > Detection Generators and Predicates

Some notation…

$L = (l_1, \ldots l_n)$ is a sequence of generated repairs s.t. $L \in \mathcal{D} \times \mathcal{F}$

- L^d - data repairs, L^p - prediction repairs

Problem Statement > Detection Generators and Predicates

Some notation…

$L = (l_1, \ldots l_n)$ is a sequence of generated repairs s.t. $L \in \mathcal{D} \times \mathcal{F}$

- L^d - data repairs, L^p - prediction repairs

Run data through sequence of **data** repair functions:

$$(X'_{train}, Y'_{train}) = \left\{ L^d(r, r) \middle| r \in (X_{train}, Y_{train}) \right\}$$
$$C = train(X'_{train}, Y'_{train})$$
$$L^d(r, r) = l_k(l_{k-1}(\ldots l_1(r, r), r), r), r) | l_i \in L^d$$

Problem Statement > Detection Generators and Predicates

Some notation…

$L = (l_1, \ldots l_n)$ is a sequence of generated repairs s.t. $L \in \mathcal{D} \times \mathcal{F}$

- L^d - data repairs, L^p - prediction repairs

Run data through sequence of **data** repair functions:

$$(X'_{train}, Y'_{train}) = \left\{ L^d(r, r) \middle| r \in (X_{train}, Y_{train}) \right\}$$
$$C = train(X'_{train}, Y'_{train})$$
$$L^d(r, r) = l_k(l_{k-1}(\ldots l_1(r, r), r), r), r) | l_i \in L^d$$

Choose last prediction repair:

$$l^* = argmax_{\ l_i \in L^p \wedge l_i(r) = true}\ i$$

Problem Statement > Detection Generators and Predicates

Some notation…

$L = (l_1, \ldots l_n)$ is a sequence of generated repairs s.t. $L \in \mathcal{D} \times \mathcal{F}$

- L^d - data repairs, L^p - prediction repairs

Run data through sequence of **data** repair functions:

$$(X'_{train}, Y'_{train}) = \{L^d(r, r) | r \in (X_{train}, Y_{train})\}$$
$$C = train(X'_{train}, Y'_{train})$$
$$L^d(r, r) = l_k(l_{k-1}(\ldots l_1(r, r), r), r), r) | l_i \in L^d$$

Choose last prediction repair:

$$l^* = argmax_{\ l_i \in L^p \wedge l_i(r) = true}\ i$$

Put it together:

$$C_L(r) = \begin{cases} C(L^d(r, r)) & if\ l^*\ not\ found \\ l^*(L^d(r, r), r) & otherwise \end{cases}$$

We want the optimal sequence of detect/repair operations:

$$L^* = argmax_{L \in \mathcal{D} \times \mathcal{F}} \, acc(C_L)$$

We want the optimal sequence of detect/repair operations:

$$L^* = argmax_{L \in \mathcal{D} \times \mathcal{F}} \, acc(C_L)$$

Q: Can we cheat?

We want the optimal sequence of detect/repair operations:

$$L^* = argmax_{L \in \mathcal{D} \times \mathcal{F}} \, acc(C_L)$$

Q: Can we cheat?

A: Yes! Be greedy and make “repairs” that are highly correlated with C’s accuracy!

We want the optimal sequence of detect/repair operations:

$$L^* = argmax_{L \in \mathcal{D} \times \mathcal{F}} \, acc(C_L)$$

Q: Can we cheat?

A: Yes! Be greedy and make "repairs" that are highly correlated with C's accuracy!

What if we take mispredictions from previous conditional repairs into account?

- BoostClean does this by using a modified version of the **Adaboost** algorithm

Repair Selection Algorithm > AdaBoost Overview

Key Concepts of AdaBoost:

1. Iteratively create "weak learners" to add to an ensemble of learners
   - Weak learners: > 50% accuracy (weak learners are cheap!)
2. Concentrate on data misclassified by earlier learners
   - Weight the data s.t. high weight = misclassified often by previous learners
   - Misclassifying highly weighted data means a higher error in model
3. When testing, have learners "vote" on classes
   - Learners are also weighted based on how well they predict
4. We want to **maximize** the weighted accuracy by **minimizing** the weighted error

$$acc(C, W) = \frac{\sum_{x,y} W(x, y)(C\big((x, null)\big).y = y)}{\sum_{x,y} W(x, y)}$$

Repair Selection Algorithm > Boost-and-Clean Algorithm

Algorithm 2: Boost-and-Clean Algorithm

Data: (X, Y)

1 Initialize $W_i^{(1)} = \frac{1}{N}$

2 $\mathcal{L}$ generates a set of classifiers $\mathcal{C}\{C^{(0)}, C^{(1)}, ..., C^{(k)}\}$ where $C^{(0)}$ is the base classifier and $C^{(1)}, ..., C^{(k)}$ are derived from the cleaning operations.

3 **for** $t \in [1, T]$ **do**

4 C_t = Find $C_t \in \mathcal{C}$ that maximizes the weighted accuracy on the test set. ϵ_t = Calculate weighted classification error on the test set $\alpha_t = \ln(\frac{1-\epsilon_t}{\epsilon_t})$ $W_i^{(t+1)} \propto W_i^{(t)} e^{-\alpha_t y_i C_t(x_i)}$: down-weight correct predictions, up-weight incorrectly predictions.

5 **return** $C(x) = \text{sign}(\sum_t^T \alpha_t C_t(x))$

Initialize weights to be uniform

Repair Selection Algorithm > Boost-and-Clean Algorithm

Algorithm 2: Boost-and-Clean Algorithm

Data: (X, Y)

1 Initialize $W_i^{(1)} = \frac{1}{N}$

2 $\mathcal{L}$ generates a set of classifiers $\mathcal{C}\{C^{(0)}, C^{(1)}, ..., C^{(k)}\}$ where $C^{(0)}$ is the base classifier and $C^{(1)}, ..., C^{(k)}$ are derived from the cleaning operations.

3 **for** $t \in [1, T]$ **do**

4 C_t = Find $C_t \in \mathcal{C}$ that maximizes the weighted accuracy on the test set. ϵ_t = Calculate weighted classification error on the test set $\alpha_t = \ln(\frac{1-\epsilon_t}{\epsilon_t})$ $W_i^{(t+1)} \propto W_i^{(t)} e^{-\alpha_t y_i C_t(x_i)}$: down-weight correct predictions, up-weight incorrectly predictions.

5 **return** $C(x) = \text{sign}(\sum_t^T \alpha_t C_t(x))$

Initialize weights to be uniform

Consider all the different enumerations of detecting/repairing we could do

Repair Selection Algorithm > Boost-and-Clean Algorithm

Algorithm 2: Boost-and-Clean Algorithm

Data: (X, Y)

1 Initialize $W_i^{(1)} = \frac{1}{N}$

2 $\mathcal{L}$ generates a set of classifiers $\mathcal{C}\{C^{(0)}, C^{(1)}, ..., C^{(k)}\}$ where $C^{(0)}$ is the base classifier and $C^{(1)}, ..., C^{(k)}$ are derived from the cleaning operations.

3 **for** $t \in [1, T]$ **do**

4 C_t = Find $C_t \in \mathcal{C}$ that maximizes the weighted accuracy on the test set. ϵ_t = Calculate weighted classification error on the test set $\alpha_t = \ln(\frac{1-\epsilon_t}{\epsilon_t})$

$W_i^{(t+1)} \propto W_i^{(t)} e^{-\alpha_t y_i C_t(x_i)}$: down-weight correct predictions, up-weight incorrectly predictions.

5 **return** $C(x) = \text{sign}(\sum_t^T \alpha_t C_t(x))$

Initialize weights to be uniform

Consider all the different enumerations of detecting/repairing we could do

Choose the one that has the best accuracy on the weighted test set

Repair Selection Algorithm > Boost-and-Clean Algorithm

Algorithm 2: Boost-and-Clean Algorithm

Data: (X, Y)

1 Initialize $W_i^{(1)} = \frac{1}{N}$

2 $\mathcal{L}$ generates a set of classifiers $\mathcal{C}\{C^{(0)}, C^{(1)}, ..., C^{(k)}\}$ where $C^{(0)}$ is the base classifier and $C^{(1)}, ..., C^{(k)}$ are derived from the cleaning operations.

3 **for** $t \in [1, T]$ **do**

4 C_t = Find $C_t \in \mathcal{C}$ that maximizes the weighted accuracy on the test set. ϵ_t = Calculate weighted classification error on the test set $\alpha_t = \ln(\frac{1-\epsilon_t}{\epsilon_t})$ $W_i^{(t+1)} \propto W_i^{(t)} e^{-\alpha_t y_i C_t(x_i)}$: down-weight correct predictions, up-weight incorrectly predictions.

5 **return** $C(x) = \text{sign}(\sum_t^T \alpha_t C_t(x))$

Initialize weights to be uniform

Consider all the different enumerations of detecting/repairing we could do

Choose the one that has the best accuracy on the weighted test set

Update weights for future classifiers

Repair Selection Algorithm > Boost-and-Clean Algorithm

Algorithm 2: Boost-and-Clean Algorithm

Data: (X, Y)

1 Initialize $W_i^{(1)} = \frac{1}{N}$

2 $\mathcal{L}$ generates a set of classifiers $\mathcal{C}\{C^{(0)}, C^{(1)}, ..., C^{(k)}\}$ where $C^{(0)}$ is the base classifier and $C^{(1)}, ..., C^{(k)}$ are derived from the cleaning operations.

3 **for** $t \in [1, T]$ **do**

4 C_t = Find $C_t \in \mathcal{C}$ that maximizes the weighted accuracy on the test set. ϵ_t = Calculate weighted classification error on the test set $\alpha_t = \ln(\frac{1-\epsilon_t}{\epsilon_t})$

$W_i^{(t+1)} \propto W_i^{(t)} e^{-\alpha_t y_i C_t(x_i)}$: down-weight correct predictions, up-weight incorrectly predictions.

5 **return** $C(x) = \text{sign}(\sum_t^T \alpha_t C_t(x))$

Initialize weights to be uniform

Consider all the different enumerations of detecting/repairing we could do

Choose the one that has the best accuracy on the weighted test set

Update weights for future classifiers

Return entire ensemble that classifies via weighted voting

Repair Selection Algorithm > Boost-and-Clean Algorithm > Complexity

Complexity
$O(k^2 N_{test} + kN_{train})$

k = # of data cleaning operations

N_{test} = # of test tuples

N_{train} = # of training tuples

Speeding up the enumeration step

- Prediction Materialization
  - Pre-train classifier C_i with cleaning operation l_i
  - When searching, just run pre-trained C_i on reweighted test data

- Prediction Indexing
  - Cache <test record, prediction> pairs for C_i
  - Use cached pairs when computing test accuracy

- Parallelization
  - Use multiple processors when choosing next cleaning operations

Repair Selection Algorithm > Why Boosting?

Why boosting?

1. Can make fewer assumptions about the classifier and data cleaning operations
2. It prioritizes cleaning operations that improve performance
3. If no such operation exists it does no worse than the base classifier
4. It is agnostic to the implementation of the classifiers
5. Boosting over B cleaning operations gives us an error rate of O(e^-2B)

The BoostClean System

Blue: Manages detector and repair libraries

Orange: Execute Boost-and-Clean algorithm

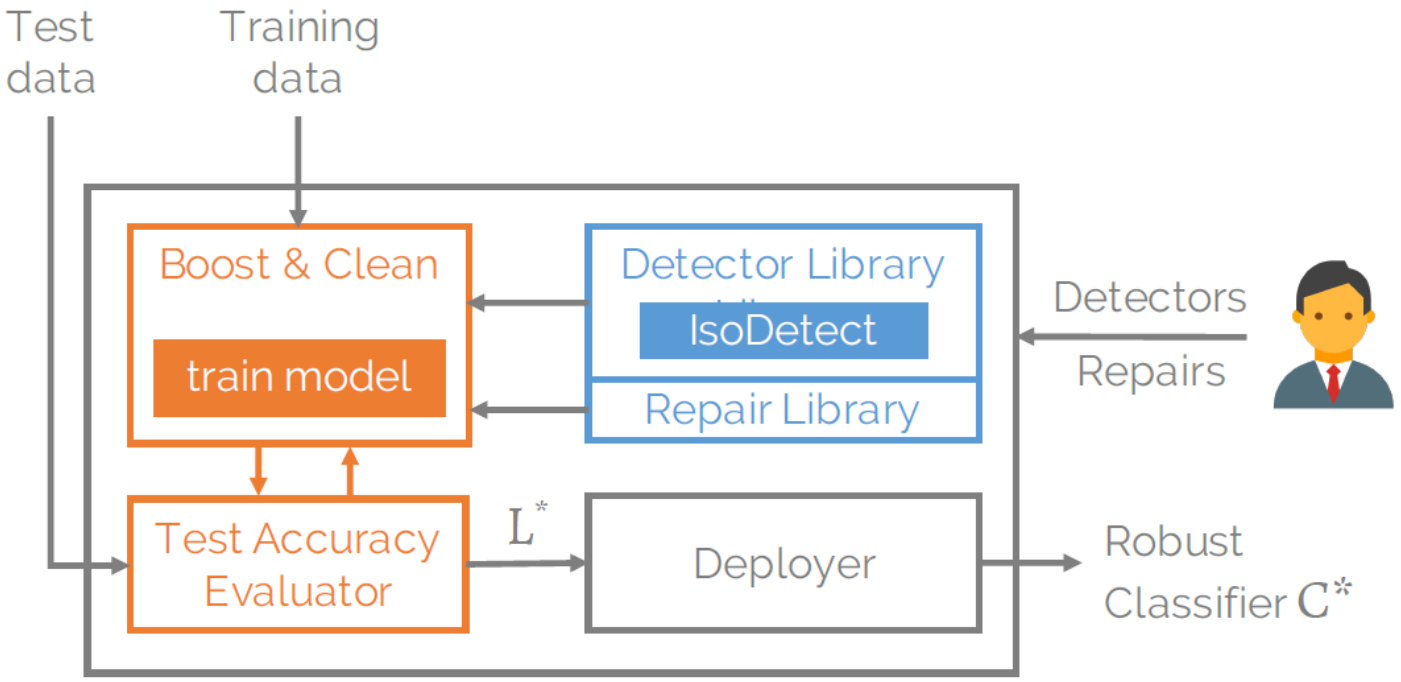


Figure 3: BoostClean system architecture.

The BoostClean System

Blue: Manages detector and repair libraries

Orange: Execute Boost-and-Clean algorithm

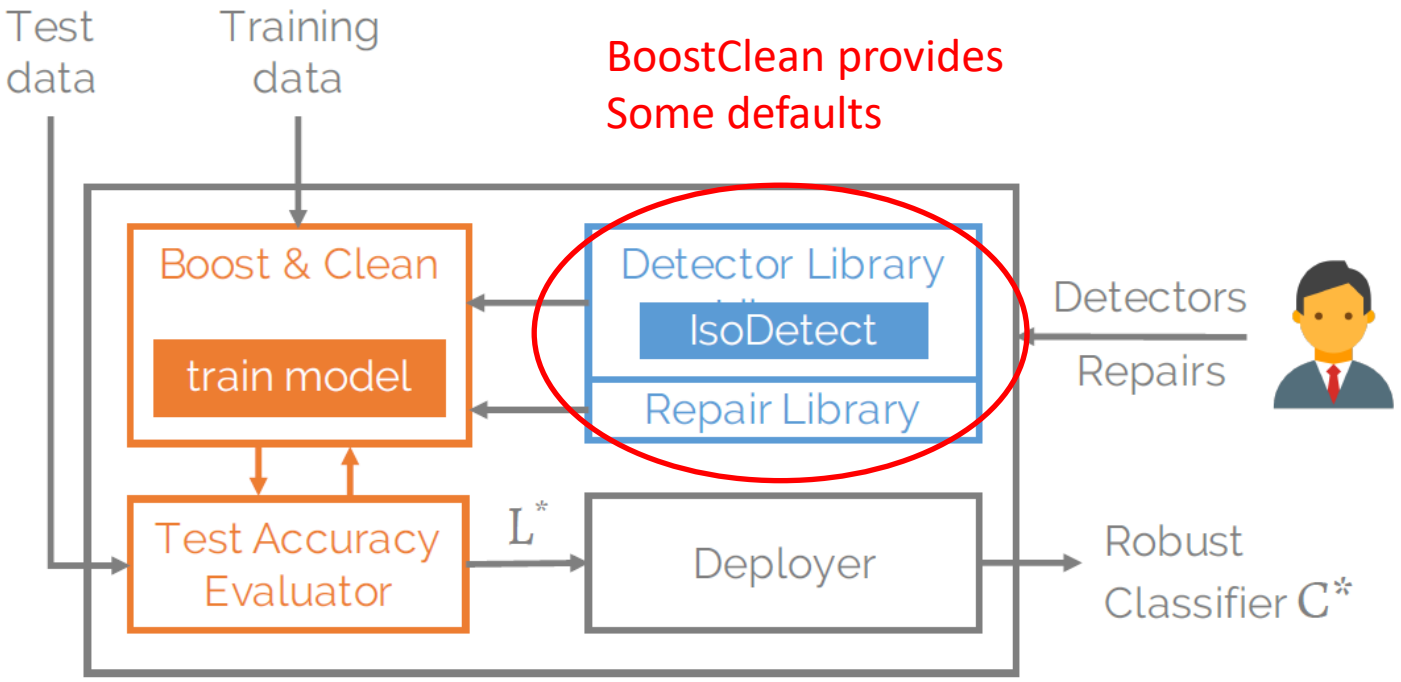


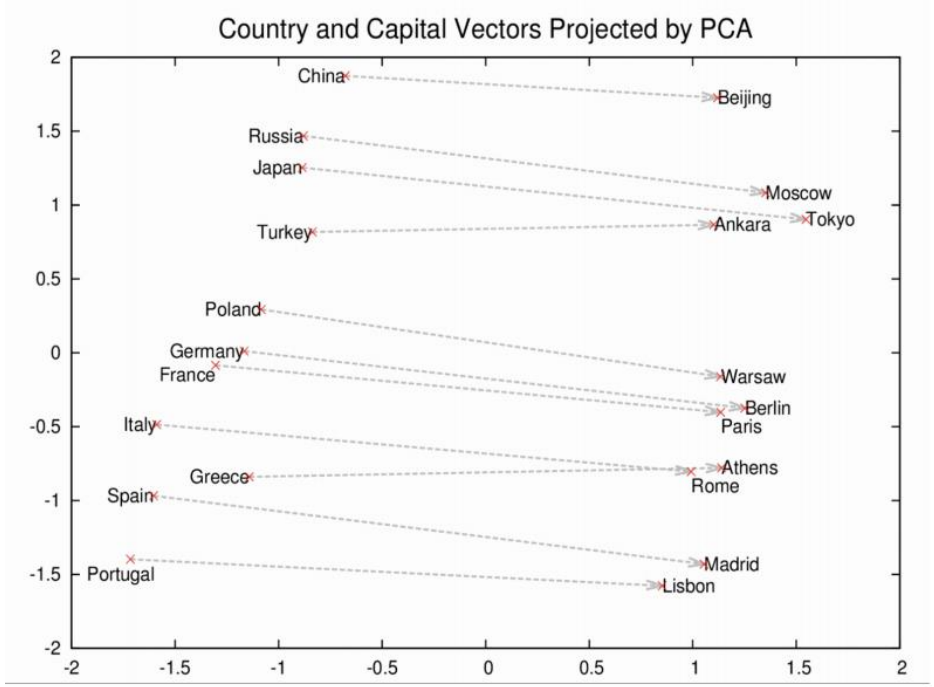
Figure 3: BoostClean system architecture.

The BoostClean System > Detectors

Detectors

- Missing values
  - Check for “NULL”, “NaN”, “Inf”
- Parsing/Type errors
  - Compare attribute’s type against common libraries (dates, addresses, etc.)
- Numerical attributes
- Anomalies in text
  - Using **text2vec** neural network and outlier detection

China is to **Bejing** as **Russia** is to …



The BoostClean System > Detectors

Detect outliers using *IsoDetect*

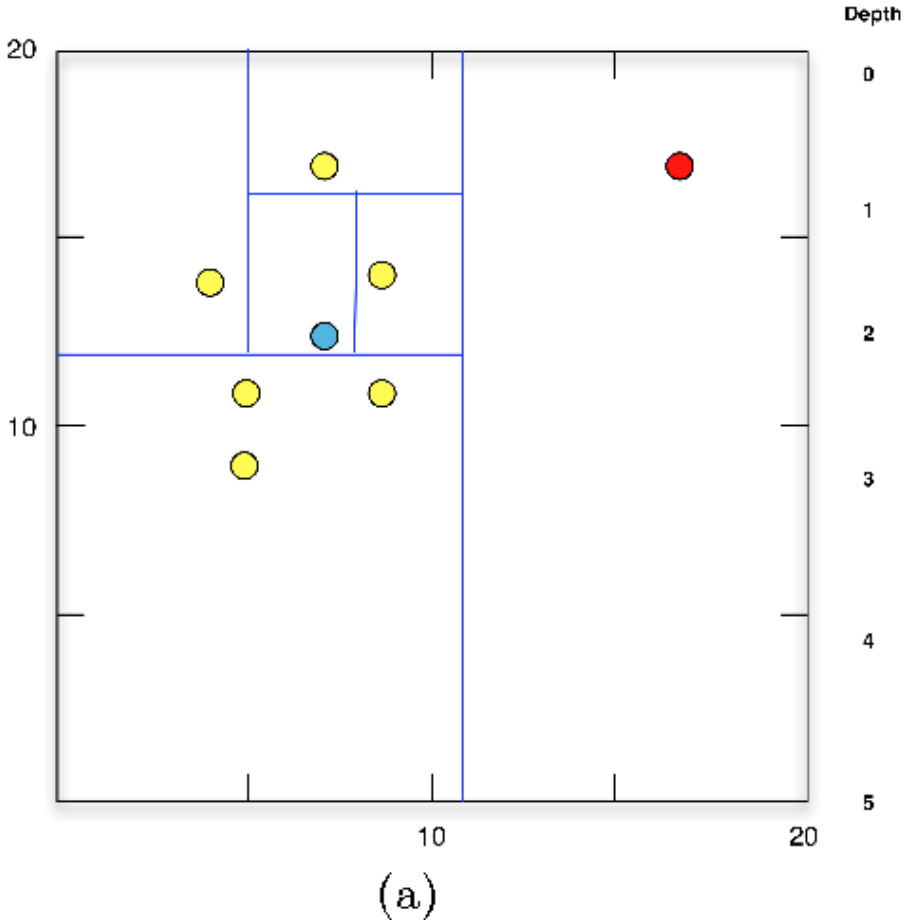
- Developers can define featurization functions f
  - Takes some attribute value and turns it into a numerical vector

$$Ex\colon\ f('Oregon') = < 0.43, 0.91 >$$

- *Featurizes* attributes and runs an **isolation forest** in order to determine the correct threshold
  - Isolation forests are made up of many decision trees that make random splits
  - Points that are isolated closer to the root (leaves) on average are more likely to be outliers
- If a featurized attribute value exceeds that threshold, then it is an outlier

The BoostClean System > Detectors > Isolation Forest

Make n trees and calculate average distance from root to isolation for points. We can establish a threshold.



Data and **prediction** repairs:

- Mean Imputation (**data** and **prediction**)
  - Mean computed over non-violated training cells
- Median Imputation (**data** and **prediction**)
  - Median computed over non-violated training cells
- Mode Imputation (**data** and **prediction**)
  - Mode computed over non-violated training cells
  - Includes non-numeric values
- Discard Record (**data**)
  - Discard training data record
- Default Prediction (**prediction**)
  - Default prediction to most common label in training data

Experiments > Setup and Methods

Method	Description
No Cleaning (NC)	Train a model with NO cleaning of train or test
Quantitative (Q)	Only use isolation forests to impute mean values
Integrity Constraint (IC)	Defined ICs and repaired to minimize distortion from dome ideal distribution
Quantitative + IC (Q+IC)	Same as Q, but impute with most common value for IC
Best Single (Best-1)	Run BoostClean with B=1 and identify single best conditional repair
Worst Single (Worst-1)	Run BoostClean with B=1 and identify single worst conditional repair
BC-3	Run BoostClean with B=3
BC-5	Run BoostClean with B=5

Test Data

- 60-20-20 Dataset split (train, validation, test)

Models:

- Sklearn random forest classifier with default parameters (mostly) and its own featurizers

Timing:

- Amazon EC2 m4.16xlarge instance (64 virtual cpus 256 memory)

Experiments > Datasets

ML Competition

- Datasets from Kaggle competitions with defined prediction goals
- Even though they were published, they contain missing values, numerical outliers, and pattern errors
- No parameter tuning

Data Analytics

- Used as benchmarks for previous data cleaning papers
- Significant errors
- Tuned classifier and detector hyperparameters for each dataset

Company X

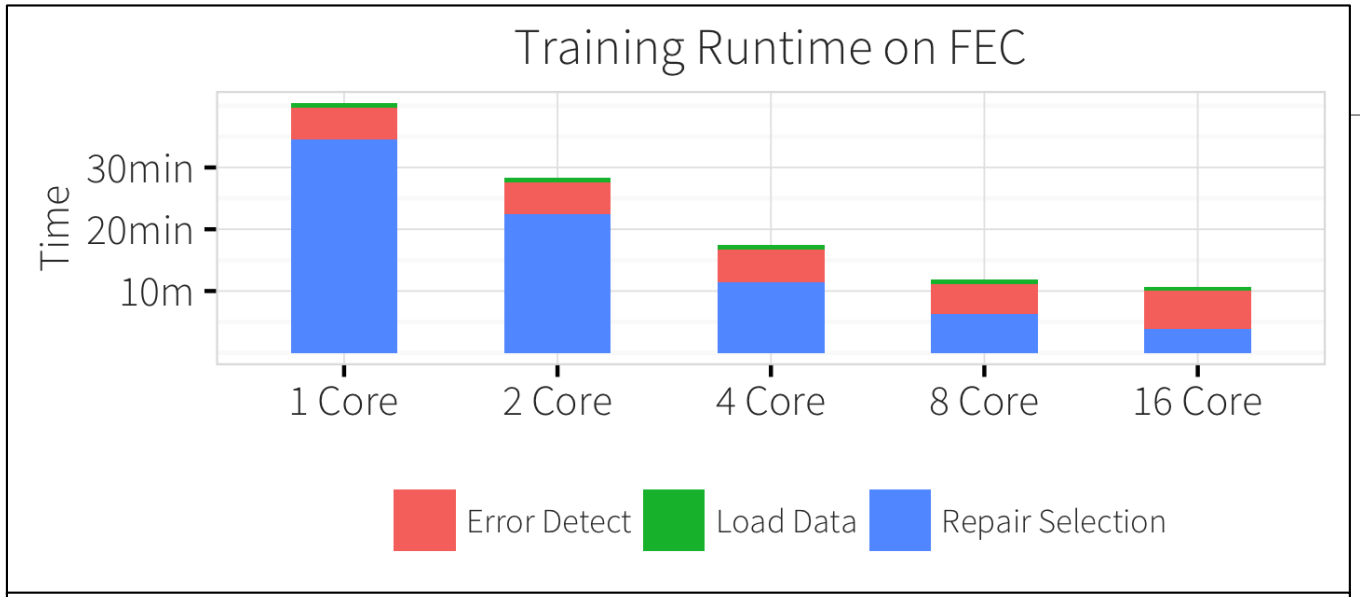
- Proprietary data (so can’t say much about it)
- Significant class imbalance
    - Accuracy measured by AUC

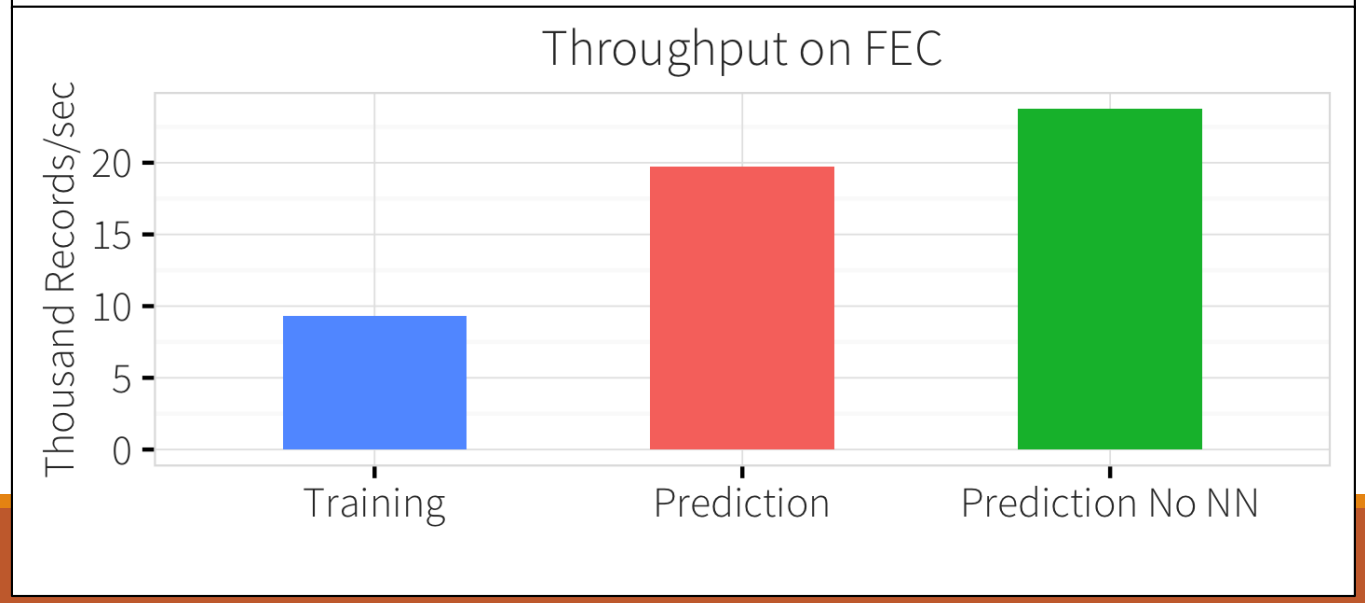
Experiments > Accuracy

ML Competition	#rows	#cols	NC	Q	IC	Q+IC	Best-1	Worst-1	BC-3	BC5	Rel. Improvement
USCensus	32561	15	0.85	0.82	0.86	0.84	0.87	0.79	0.88	***0.91***	+4.5%
Emergency	11176	9	0.67	0.72	0.67	0.72	0.72	0.66	0.72	***0.75***	+4.7%
Sensor	928991	5	0.92	0.93	0.92	0.89	0.92	0.8	***0.94***	0.94	+1.3%
NFL	46129	65	0.74	0.74	0.76	0.75	0.76	0.74	0.79	***0.82***	+5.1%
EEG	2406	32	0.79	0.82	0.79	0.83	0.83	0.7	0.85	***0.89***	+6.8%
Titanic	891	12	0.83	0.72	0.83	0.76	0.83	0.69	0.83	***0.84***	+1.1%
Housing	1460	81	0.73	0.76	0.73	0.77	0.77	0.65	***0.81***	0.76	+5.1%
Retail	541909	8	0.88	0.88	0.91	0.91	0.91	0.87	0.94	***0.95***	+4.3%
Data Analytics	#rows	#cols	NC	Q	IC	Q+IC	Best	Worst	BC-3	BC5	Rel. Improvement
FEC	6410678	18	0.62	0.53	0.61	0.57	0.71	0.51	0.74	***0.77***	+8.4%
Restaurant (Multiclass)	758	4	0.42	0.42	0.58	0.68	***0.62***	0.36	0.61	0.60	(1.61)%
Company X	#rows	#cols	NC	Q	IC	Q+IC	Best	Worst	BC-3	BC5	Rel. Improvement
Dataset 1 (AUC)	76684	6	0.60	0.60	0.60	0.60	0.61	0.59	0.66	***0.69***	+13.3%
Dataset 2 (AUC)	83986	6	0.55	0.55	0.52	0.55	0.55	0.52	0.61	***0.63***	+14.5%

Table 1: End-to-end accuracy results for each dataset and experimental method. We report standard classification accuracy. The right column summarizes the absolute accuracy improvement over the best non BC-3/5 approach. The `Company X` datasets have high class imbalances cause artificially high accuracy statistics, so we report AUC statistics for those datasets instead.

Experiments > End-to-End Run Time





Parallelize inner-loop of boosting algorithm

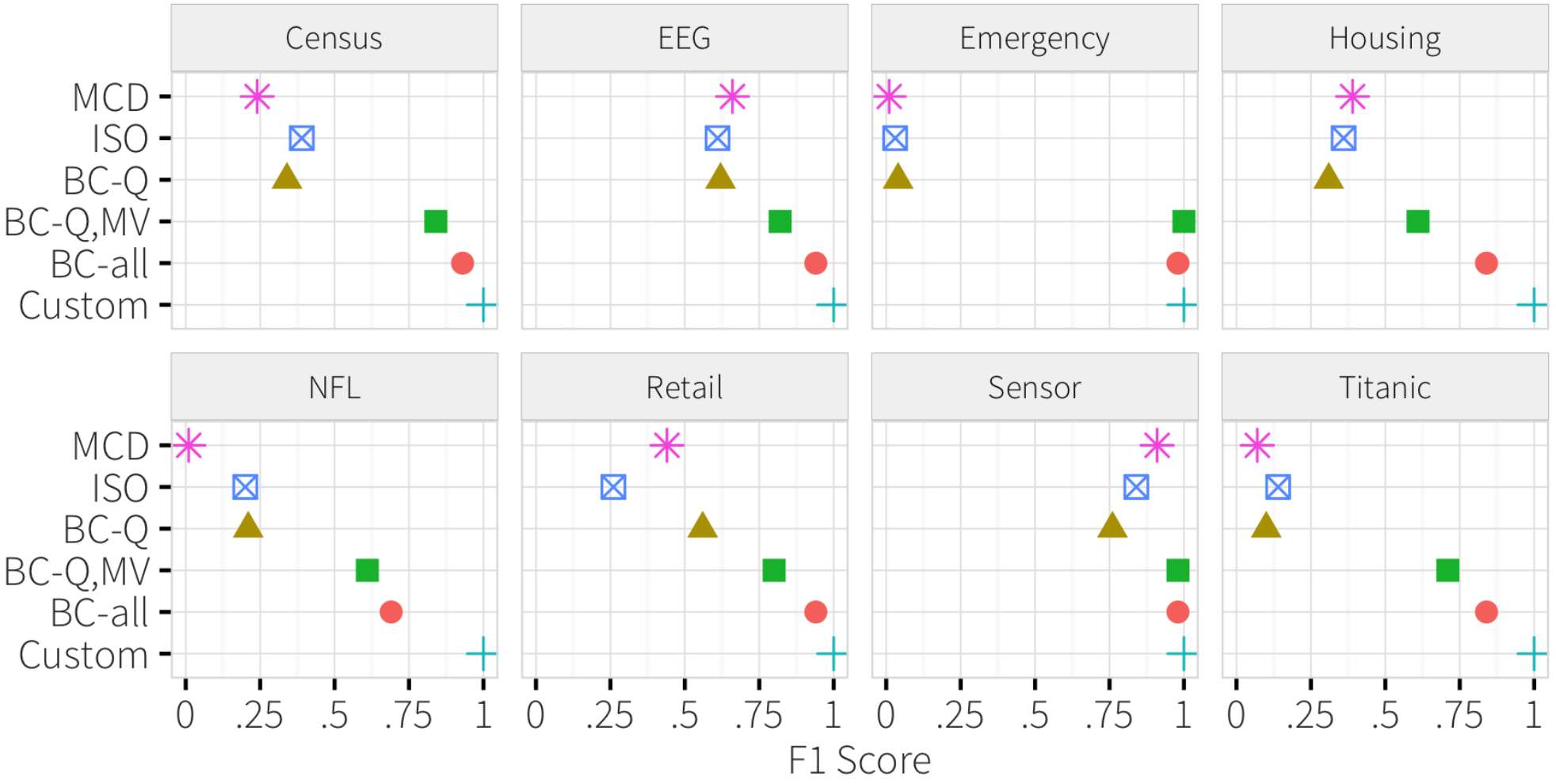
Test2Vec is a little expensive

FEC Dataset
- 1.5GB
- ~6 million records

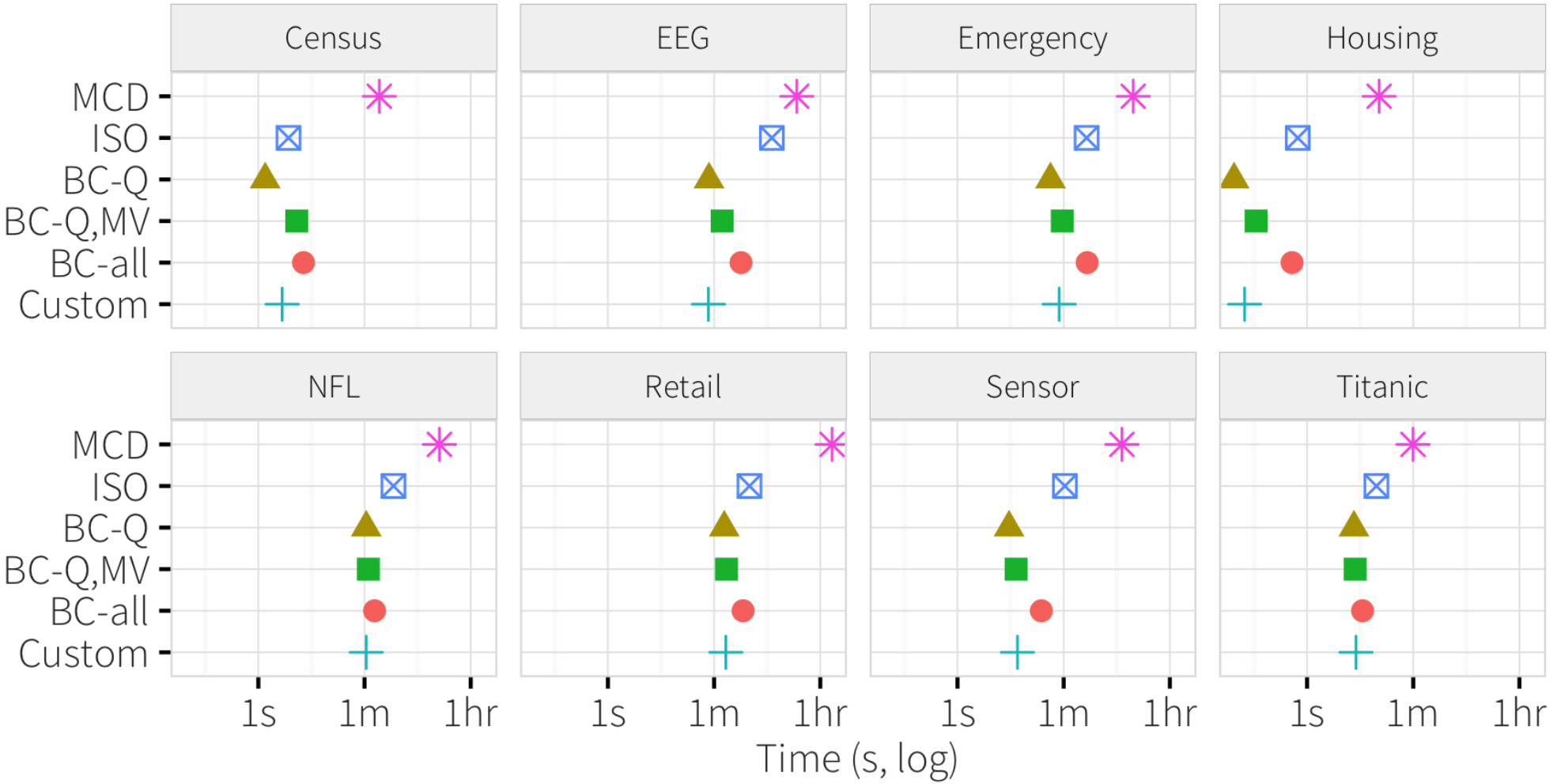
Experiments > Detector Micro-Benchmarks

Method	Description
Hand-crafted rules (Custom)	Manually written rules
Minimum Covariance Determinant (MCD)	Featureized as one-hot encoding for categories and BoW for strings
Isolation Forests (ISO)	One-hot encoding for categories
BoostClean – Q (BC-Q)	Quantitative only
BoostClean – Q,MV (BC-Q,MV)	Quantitative and missing value featurizers
BoostClean – All (BC-all)	Quantitative, missing value featurizers, and word embeddings

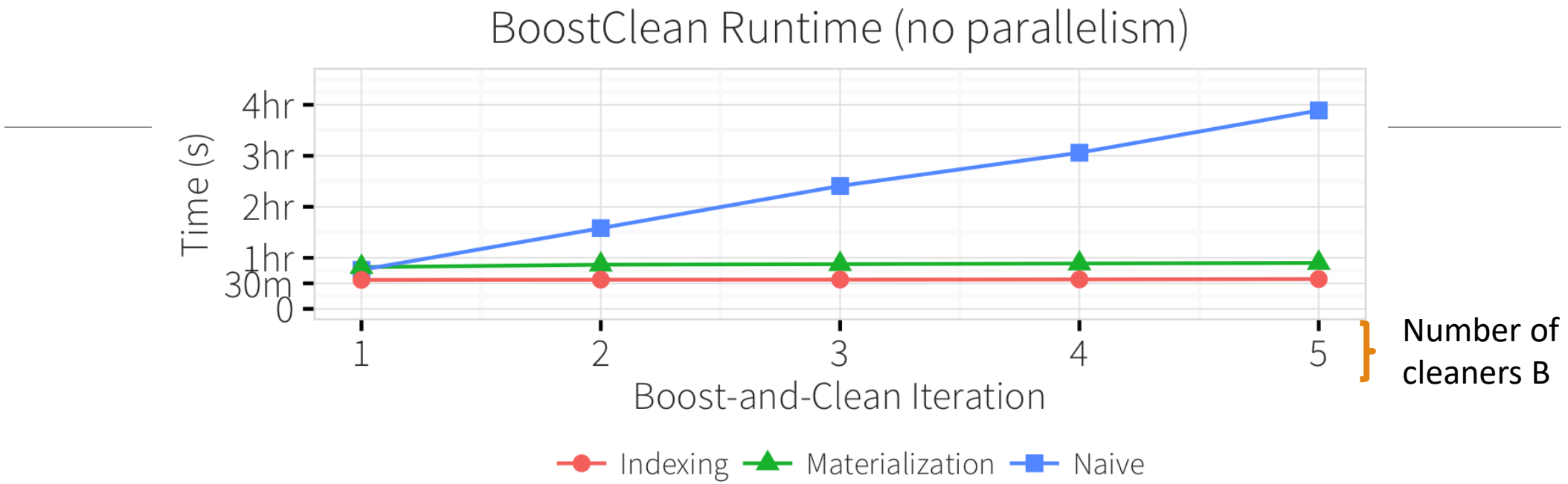
Experiments > Detector Micro-Benchmarks > F1 Score



Experiments > Detector Micro-Benchmarks > Runtime



Experiments > Repair Micro-Benchmarks

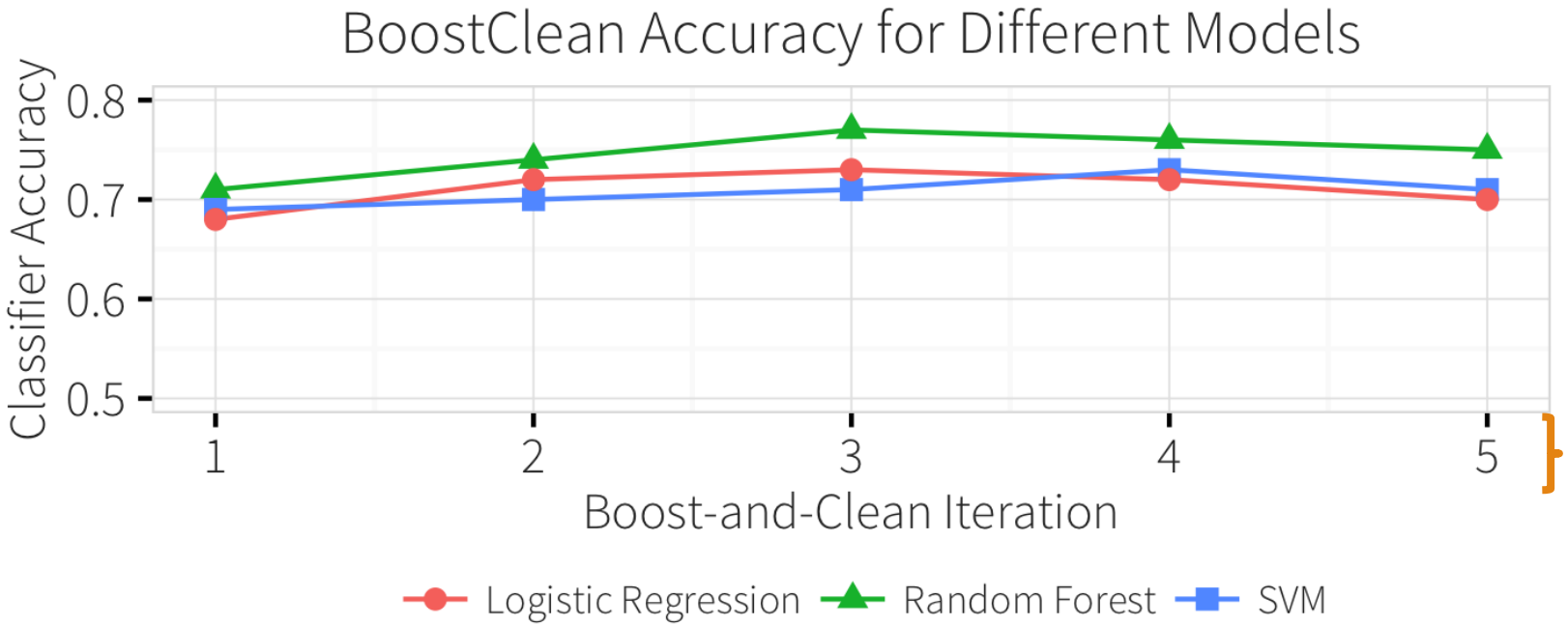


- Prediction Indexing
  - Cache <test record, prediction> pairs for C_i
  - Use cached pairs when computing test accuracy

- Prediction Materialization
  - Pre-train classifier C_i with cleaning operation l_i
  - When searching, just run pre-trained C_i on reweighted test data

FEC Dataset
- 1.5GB
- ~6 million records

Experiments > Repair Micro-Benchmarks



Number of cleaners B

BoostClean begins to overfit with B > 3

FEC Dataset
- 1.5GB
- ~6 million records

Summary > Contributions

What we saw in this paper:

1. **Cleaning as Boosting**
   - Using **statistical boosting (ensembles)**, automatically select the best cleaning operations from a library in order to maximize the predictive performance of a downstream model
2. **Automatic Model Improvements**
   - Evaluate BoostClean on 12 different datasets
   - Improved prediction accuracy of up to 9% compared to non-ensembled approaches
3. **Error Detection Library**
   - Built-in error detection library
   - Includes Word2Vec featurization
   - Achieves 81% accuracy of all errors found by hand-written rules
4. **Optimizations**
   - Parallelism, materialization, and indexing
   - 22.2x end-to-end speedup on a 16-core machine