Data Leakage in Notebooks: Static Detection and Better Processes

Chenyang Yang, Rachel Brower-Sinning, Grace A. Lewis, Christian Kästner

Carnegie Mellon University
Why ML Models Fail in Production?

ML models

High test accuracy

Software systems

Low production accuracy
When is Test Accuracy not Reliable?

Non-representative test data

African Bush Elephant  →  North America Wild Horse

Low production accuracy
When is Test Accuracy not Reliable?

Data leakage: leak test data into model development through repeated evaluation, pre-processing, and dependency

We use static analysis to detect data leakage in ~281k notebooks ~81k GitHub repositories created in Sep. 2021 2 top Kaggle competitions
Principle of Independent Evaluation

Model development

Model training

Validation

(validation holdout sample)

Testing

(testing holdout sample)

Independent evaluation

Model selection
Data Leakage #1: through Repeated Evaluation

Models overfit to test data after repeated evaluation

- Inflated test accuracy!
- Found by our tool in ~18% notebooks
Data Leakage #2: through Preprocessing

Peeking at test data in competitions is common

<table>
<thead>
<tr>
<th>Training data</th>
<th>the</th>
<th>red</th>
<th>dog</th>
<th>cat</th>
<th>eats</th>
<th>food</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. the red dog</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2. cat eats dog</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3. dog eats food</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4. red cat eats</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Unknown words
Different distribution

Inflated test accuracy!

Found by our tool in ~12% notebooks
Data Leakage #3: through Dependency

Data augmentation could introduce dependency

Inflated test accuracy!

Train/test dependency

Found by our tool in
~6% notebooks
Data Leakage is Prevalent in Practice

~281k notebooks from GitHub and Kaggle

~30% GitHub notebooks have data leakage issues
- 33% assignments (keyword: ‘assignment’, ‘homework’)
- 20% popular notebooks (>=10 stars)
- 16% tutorials (keyword: ‘this tutorial’)

55% competition solutions leak through preprocessing
Leakage Exhibits Non-local Patterns

Leakage and training are often far apart
span >20% of the whole notebook in >50% cases

Hard for manual detection!

Training

```
Leakage happens here
A, y = B(1, 1) + e, y_true

Lots of code in between

import pandas as pd
from sklearn.model_selection import train_test_split
import SelectFromModel, ch2

def get_training_samples(model, X_train, y_train)
    X_train = SelectFromModel(model, threshold=0.1)

    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y)

    from sklearn.linear_model import Ridge, Lasso, ElasticNet

    ridge = Ridge()
    ridge.fit(X_train, y_train)
    y_pred = ridge.predict(X_test)
```

```
results = []
for clf, name in [(DecisionTreeClassifier(‘entropy’), ‘dt’),
                 (SVM(), ‘svm’), (Perceptron(), ‘perceptron’),
                 (LogisticRegression(), ‘logreg’),
                 (GaussianNB(), ‘gaussian’),
                 (KNeighborsClassifier(3), ‘knn’),
                 (RandomForestClassifier(10), ‘rf’),
                 (AdaBoostClassifier(), ‘ada’),
                 (GradientBoostingClassifier(), ‘gbc’),
                 (XGBClassifier(), ‘xgb’),
                 (LGBMClassifier(), ‘lgb’),
                 (LightGBM(), ‘lightgbm’),
                 (CatBoostClassifier(), ‘catboost’)]:
    score = cross_val_score(clf, X, y, cv=CV, scoring=metric)
    results.append((name, score.mean()))
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

X_train = StandardScaler().fit_transform(X_train)
X_test = StandardScaler().fit_transform(X_test)
```

```
from sklearn.linear_model import Ridge, Lasso, ElasticNet
ridge = Ridge()
ridge.fit(X_train, y_train)
y_pred = ridge.predict(X_test)
```

```
```
```
Could we statically detect data leakage?
import pandas as pd
from sklearn.feature_selection import SelectPercentile, chi2
from sklearn.model_selection import LinearRegression, train_test_split
data = pd.read_csv('data.csv')
X_raw = data.drop('label', axis=1)
y = data['label']

select = SelectPercentile(chi2, percentile=50)
select.fit(X_raw)
X = select.transform(X_raw)

X_train, y_train, X_test, y_test = train_test_split(X, y)
lr = LinearRegression()
lr.fit(X_train, y_train)
lr_score = lr.score(X_test, y_test)
Test Data is Used for Feature Selection

```python
import pandas as pd
from sklearn.feature_selection import SelectPercentile, chi2
from sklearn.model_selection import LinearRegression, train_test_split

data = pd.read_csv('data.csv')
X_raw = data.drop('label', axis=1)
y = data['label']

select = SelectPercentile(chi2, percentile=50)
select.fit(X_raw)
X = select.transform(X_raw)

X_train, y_train, X_test, y_test = train_test_split(X, y)
lr = LinearRegression()
lr.fit(X_train, y_train)
lr_score = lr.score(X_test, y_test)
```
When is an Operation Leakage-inducing?

```python
import pandas as pd
from sklearn.feature_selection import SelectPercentile, chi2
from sklearn.model_selection import LinearRegression, train_test_split

data = pd.read_csv('data.csv')
X_raw = data.drop('label', axis=1)
y = data['label']

select = SelectPercentile(chi2, percentile=50)
select.fit(X_raw)
X = select.transform(X_raw)

X_train, y_train, X_test, y_test = train_test_split(X, y)
lr = LinearRegression()
lr.fit(X_train, y_train)
lr_score = lr.score(X_test, y_test)
```

Computing across rows could lead to leakage
When is an Operation Leakage-inducing?

Computing each row independently is safe
When is an Operation Leakage-inducing?

Computing each row independently is safe
Reduce-like Operations could Lead to Leakage

<table>
<thead>
<tr>
<th>col1</th>
<th>col2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>-3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

reduce

map

filter
Detecting Data Leakage with Data-flow

```python
import pandas as pd
from sklearn.feature_selection import SelectPercentile, chi2
from sklearn.model_selection import LinearRegression, train_test_split
data = pd.read_csv('data.csv')
X_raw = data.drop('label', axis=1)
y = data['label']

select = SelectPercentile(chi2, percentile=50)
select.fit(X_raw)
X = select.transform(X_raw)

X_train, y_train, X_test, y_test = train_test_split(X, y)
lr = LinearRegression()
lr.fit(X_train, y_train)
rr_score = lr.score(X_test, y_test)
```

*There are more subtleties in tracking data-flow and determining whether two datasets are related: see our paper for details.*
Implementation

Front-end
- Raw Python
- Python (SSA)
- Type Inference
- Datalog Facts

Back-end
- Datalog Facts
- API Specs
- Pointer Analysis
- Data-flow Analysis
- Related Data Analysis
- Data-Model Mappings
- Dataset Transformations
- Leakage Detection

2-call-site-sensitive
Evaluation: Accuracy & Efficiency

**93% accuracy** from comparing results with 100 manually labeled sample notebooks

**3 seconds** (avg.) of analysis on a standard desktop with Intel Xeon CPU and 32GB memory
Recall: Data Leakage is Prevalent in Practice

~30% GitHub notebooks have data leakage issues
  33% assignments
  20% popular notebooks
  16% tutorials

55% competition solutions leaks through preprocessing
Could we avoid data leakage in practice?
Data Leakage: Better Processes

Static analysis as **warnings** in notebooks

```python
import pandas as pd
from sklearn.feature_selection import SelectPercentile, chi2
from sklearn.model_selection import LinearRegression, train_test_split

data = pd.read_csv('data.csv')
X_raw = data.drop('label', axis=1)
y = data['label']

select = SelectPercentile(chi2, percentile=50)
select.fit(X_raw)  # data leakage (preprocessing)
X = select.transform(X_raw)

X_train, y_train, X_test, y_test = train_test_split(X, y)
lr = LinearRegression()
lr.fit(X_train, y_train)  # train
lr_score = lr.score(X_test, y_test)  # test
```
Data Leakage: Better Processes

Limited access to test label/data

Do not share test data
Data Leakage: Better Processes

API Design to prevent leakage

```python
X_selected = SelectKBest(k=25).fit_transform(X, y)
X_train, X_test, y_train, y_test = train_test_split(
    X_selected, y, random_state=42)
gbc = GradientBoostingClassifier(random_state=1)
gbc.fit(X_train, y_train)

y_pred = gbc.predict(X_test)
accuracy_score(y_test, y_pred)
```

```python
from sklearn.pipeline import make_pipeline
pipeline = make_pipeline(SelectKBest(k=25),
GradientBoostingClassifier(random_state=1))
pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)
accuracy_score(y_test, y_pred)
```
Takeaways

Data Leakage is **prevalent** in practice (in ~30% GitHub notebooks)

Static analysis and better process designs could help

```python
import pandas as pdrom sklearn.feature_selection import SelectPercentile, chi2from sklearn.model_selection import LinearRegression, train_test_splitdata = pd.read_csv('data.csv')X_raw = data.drop('label', axis=1)y = data['label']select = SelectPercentile(chi2, percentile=50)select.fit(X_raw)data = data[reprocess()]
X = select.transform(X_raw)

X_train, y_train, X_test, y_test = train_test_split(X, y)lr = LinearRegression()lr.fit(X_train, y_train)train
lr_score = lr.score(X_test, y_test) test
```
Bonus: Practical Impact of Data Leakage

Often marginal accuracy differences

Data leakage makes models “learn” from random data

Data leakage leads to flawed experiments and wasted time

```python
1 import numpy as np
2 # generate random data
3 n_samples, n_features, n_classes = 200, 10000, 2
4 rng = np.random.RandomState(42)
5 X = rng.standard_normal((n_samples, n_features))
6 y = rng.choice(n_classes, n_samples)
7 # leak test data through feature selection
8 X_selected = SelectKBest(k=25).fit_transform(X, y)
10 X_train, X_test, y_train, y_test = train_test_split(
12     X_selected, y, random_state=42)
13 gbc = GradientBoostingClassifier(random_state=1)
14 gbc.fit(X_train, y_train)
15 y_pred = gbc.predict(X_test)
17 accuracy_score(y_test, y_pred)
18 # expected accuracy ~0.5; reported accuracy 0.76
```