Lightweight Inspection of Data Preprocessing in Native Machine Learning Pipelines

Conference on Innovative Data Systems Research (CIDR) 2021

Stefan Grafberger
TUM

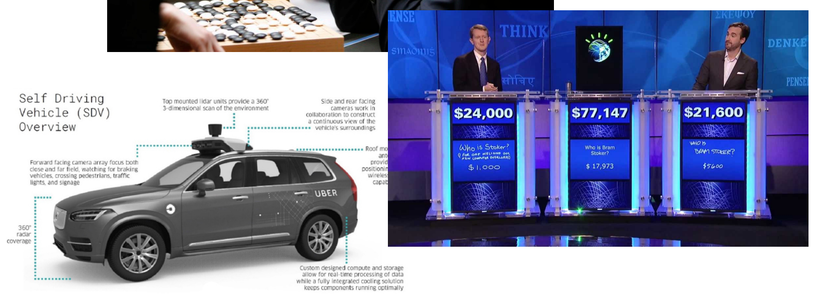
Julia Stoyanovich
NYU

Sebastian Schelter
UvA

Astonishing progress in ML in the last years

University of Amsterdam

- Machines beat us humans in more and more games like Go or Jeopardy thought to be our domain
- Advances in computer vision enable wide spread use of facial recognition and wake expectations for autonomous driving
- More and more decisions are being automated with ML techniques
- “Gold rush” mentality and ML arms race in industry

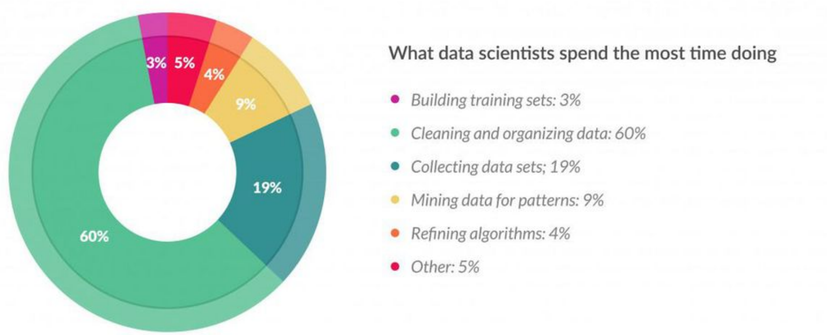


https://www.theatlantic.com/technology/archive/2016/03/the-invisible-opponent/475611/
https://www.youtube.com/watch?v=P18EdAKuC1U
https://spectrum.ieee.org/cars-that-think/transportation/self-driving/ntsb-investigation-into-deadly-uber-selfdriving-car-crash-reveals-lax-attitude-toward-safety
https://thepointsguy.com/news/amsterdam-schiphol-test-trial-facial-recognition-boarding/

Beyond the Hype: Serious technical and societal problems

University of Amsterdam

- **Data preparation accounts for 80% of the work of data scientists**



- Automated decision making can **reproduce and amplify existing biases and discrimination**

“Forbes: Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says”
https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/

“MIT Researcher Exposing Bias in Facial Recognition Tech”
https://www.insurancejournal.com/news/national/2019/04/08/523153.htm

“Machine Bias - ProPublica”
https://www.propublica.org/articlemachine-bias-risk-assessments-in-criminal-sentencing

Technical Bias

UNIVERSITY OF AMSTERDAM

- Fairness & accountability:
  - **socio-technological problem**, requires collaboration with law experts and social scientists
  - responsibility cannot be automated, but we can assist data scientists with some issues

- Three types of bias in automated decision making systems (ADS)
  - **Pre-existing bias** (origin in society, dependent on belief system)
  - **Technical bias** (introduced by technical systems)
  - **Emergent bias** (arising from feedback loops of deployed ADS)

- **Our focus: technical bias introduced by data preprocessing, e.g.,**
  - Changes in the proportion of protected groups by joins and selections (e.g., filtering demographic data by zip codes)
  - Missing value imputation for sensitive attributes (e.g., gender)

Stoyanovich et al.: Responsible data management, VLDB Keynote 2020

Why is this difficult?

UNIVERSITY OF AMSTERDAM

- **Data Science in Production**
  - ML models often designed under “lab conditions” on small, clean data samples
  - Data preprocessing not in the focus
  - Training code often handed over to non-ML experts

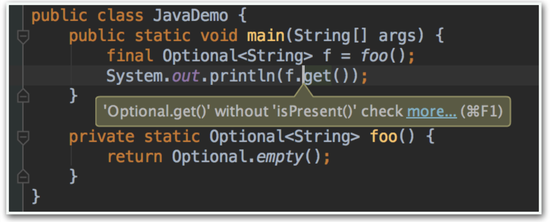
- **No algebraic foundation for data preprocessing in ML**
  - Pipelines often “glue together” different technologies (Spark + pandas + sklearn) using different data representations and operations

- **Even the experts can’t do it right**
  - See our work on “FairPrep” from EDBT’20

Research question:
Can we hint data scientists at potentially problematic operations in the preprocessing code of their ML pipelines?

Can we connect this inspection to the actual data that is being processed?

Inspiration from software engineering, e.g. **code inspection in modern IDE’s**



mlinspect

University of Amsterdam

- Library to **instrument ML preprocessing code** with custom inspections

  - Automatically applies **inspections** (user-defined functions allowing for annotation propagation) **to the inputs and outputs of certain operations**

- **Core ideas:**
  - Work with **“native” preprocessing pipelines** in pandas / scikit-klearn written declaratively (e.g., with relational ops + estimator/transformer pipelines)
  - Represent preprocessing operations as a dataflow graph

- **Execution:**
  - Instrumentation of function calls in the Python AST
  - Delegation of relevant function calls to library-specific backends for inspection
  - Returns extracted dataflow graph with inspection results
  - Runtime overhead linear in the number of records

- **Use cases:** lineage tracking, sampling of intermediate outputs, tests for distribution changes

```
PipelineInspector
 .on_pipeline_from_py_file('healthcare.py')
  .expect_no_bias_introduced_for(['age_group', 'race'])
  .expect_no_use_of_illegal_features()
  .expect_no_missing_embeddings()
 .verify()
```

```
data = data[data.county = “CountyA”]
```

age_group	county
60	CountyA
60	CountyA
20	CountyA
60	CountyB
20	CountyB
20	CountyB

50% vs 50%

age_group	county
60	CountyA
60	CountyA
20	CountyA

66% vs 33%

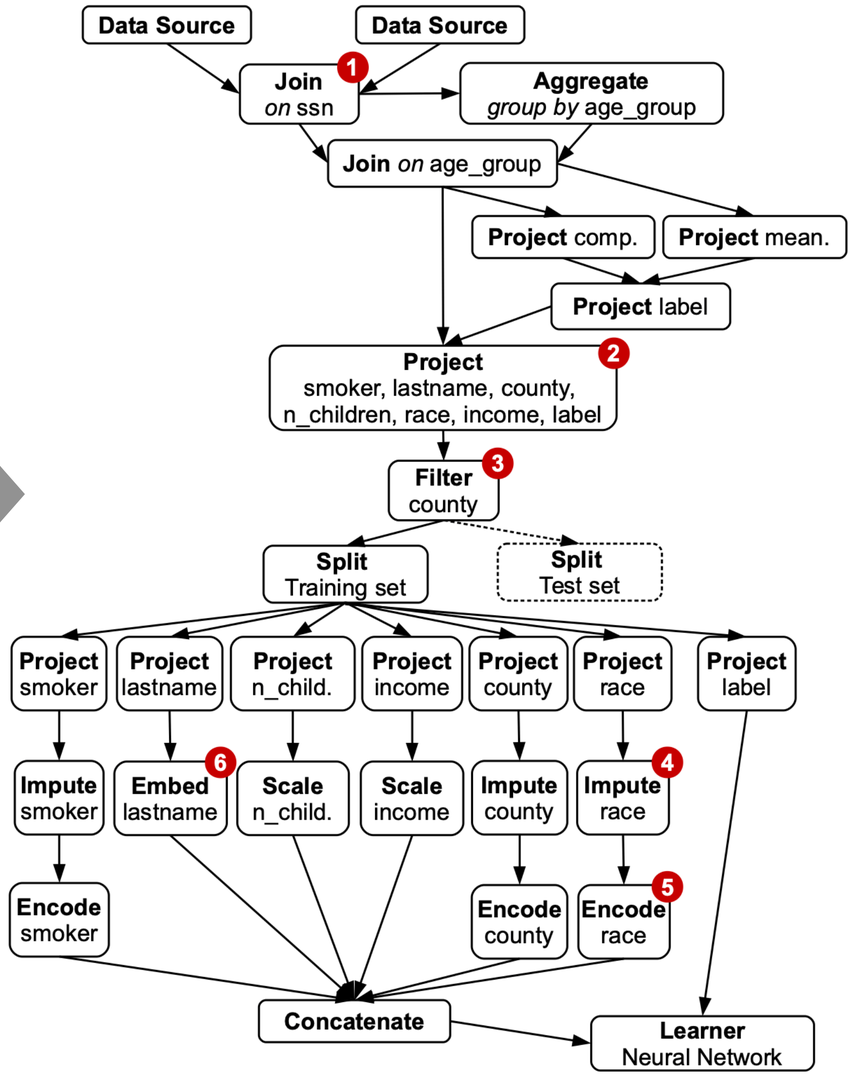
Potential issues in preprocessing pipeline:

1. Join might change proportions of groups in data
2. Column ‘age_group’ projected out, but required for fairness
3. Selection might change proportions of groups in data
4. Imputation might change proportions of groups in data
5. ‘race’ as a feature might be illegal!
6. Embedding vectors may not be available for rare names!

Python script for preprocessing, written exclusively with native pandas and sklearn constructs

```
# load input data sources, join to single table
patients = pandas.read_csv(…)
histories = pandas.read_csv(…)
data = pandas.merge([patients, histories], on=['ssn'])

# compute mean complications per age group, append as column
complications = data.groupby('age_group')
  .agg(mean_complications=('complications','mean'))
data = data.merge(complications, on=['age_group'])

# Target variable: people with frequent complications
data['label'] = data['complications'] >
  1.2 * data['mean_complications']

# Project data to subset of attributes, filter by counties
data = data[['smoker', 'last_name', 'county',
             'num_children', 'race', 'income', 'label']]
data = data[data['county'].isin(counties_of_interest)]

# Define a nested feature encoding pipeline for the data
impute_and_encode = sklearn.Pipeline([
  (sklearn.SimpleImputer(strategy='most_frequent')),
  (sklearn.OneHotEncoder())])
featurisation = sklearn.ColumnTransformer(transformers=[
  (impute_and_encode, ['smoker', 'county', 'race']),
  (Word2VecTransformer(), 'last_name')
  (sklearn.StandardScaler(), ['num_children', 'income']])

# Define the training pipeline for the model
neural_net = sklearn.KerasClassifier(build_fn=create_model())
pipeline = sklearn.Pipeline([
  ('features', featurisation),
  ('learning_algorithm', neural_net)])

# Train-test split, model training and evaluation
train_data, test_data = train_test_split(data)
model = pipeline.fit(train_data, train_data.label)
print(model.score(test_data, test_data.label))
```

Corresponding dataflow DAG



Thank you!

University of Amsterdam

- Check out our paper for technical details
- **Implementation & demo notebook** for example available at

  https://github.com/stefan-grafberger/mlinspect

- **Next steps**
  - Backends for more libraries (e.g. Tensorflow Transform)
  - Extend approach to distributed execution for SparkML
  - Study how well approach works on “code in the wild”