Ballet: A lightweight framework for open-source, collaborative feature engineering

Micah J. Smith, Kelvin Lu, and Kalyan Veeramachaneni
Massachusetts Institute of Technology

Introduction and Motivation

Open data science projects consist of community-driven, open-source analysis of data and development of predictive models to address societal problems.

Motivation In the Fragile Families Challenge [1], which tasked participants with predicting `GPA`, `Grit`, `Material hardship`, `Eviction`, `Layoff`, and `Job training`, feature engineering was the main competition bottleneck.

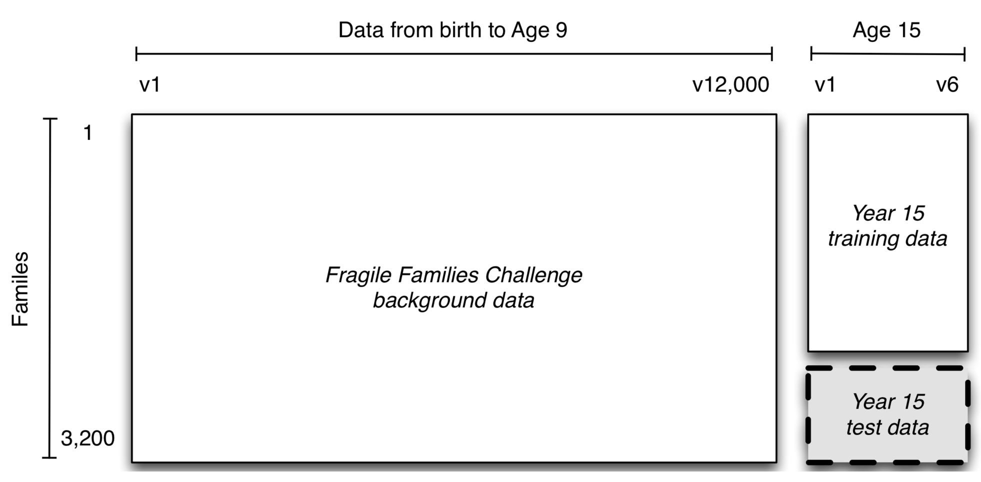

Figure 1: The Fragile Families Challenge dataset

- Development of such models must support synchronous collaboration by hundreds of interested contributors
- Need a framework to support splitting data science tasks and combining individual components of source code
- Feature engineering, an important task in practical machine learning, is susceptible this strategy.
- **Goals**: Accelerate development, improve quality of features, facilitate collaborative contributions, maintain pipeline integrity, avoid "heavy" infrastructure

Ballet is a lightweight software framework for collaborative, open-source data science through a focus on feature engineering.

⇒ https://github.com/HDI-Project/ballet

The Ballet framework

- **Challenge 1**: Facilitate incremental patches → maintain *feature engineering pipeline invariant* through extensive software validation and streaming logical feature selection
- **Challenge 2**: Can't rely on any custom infrastructure (open-source world) → design for lightweight architecture

Initialize Ballet generates a new GitHub repository from a template which contains an empty *feature engineering pipeline*.

Develop How do contributors grow the pipeline?

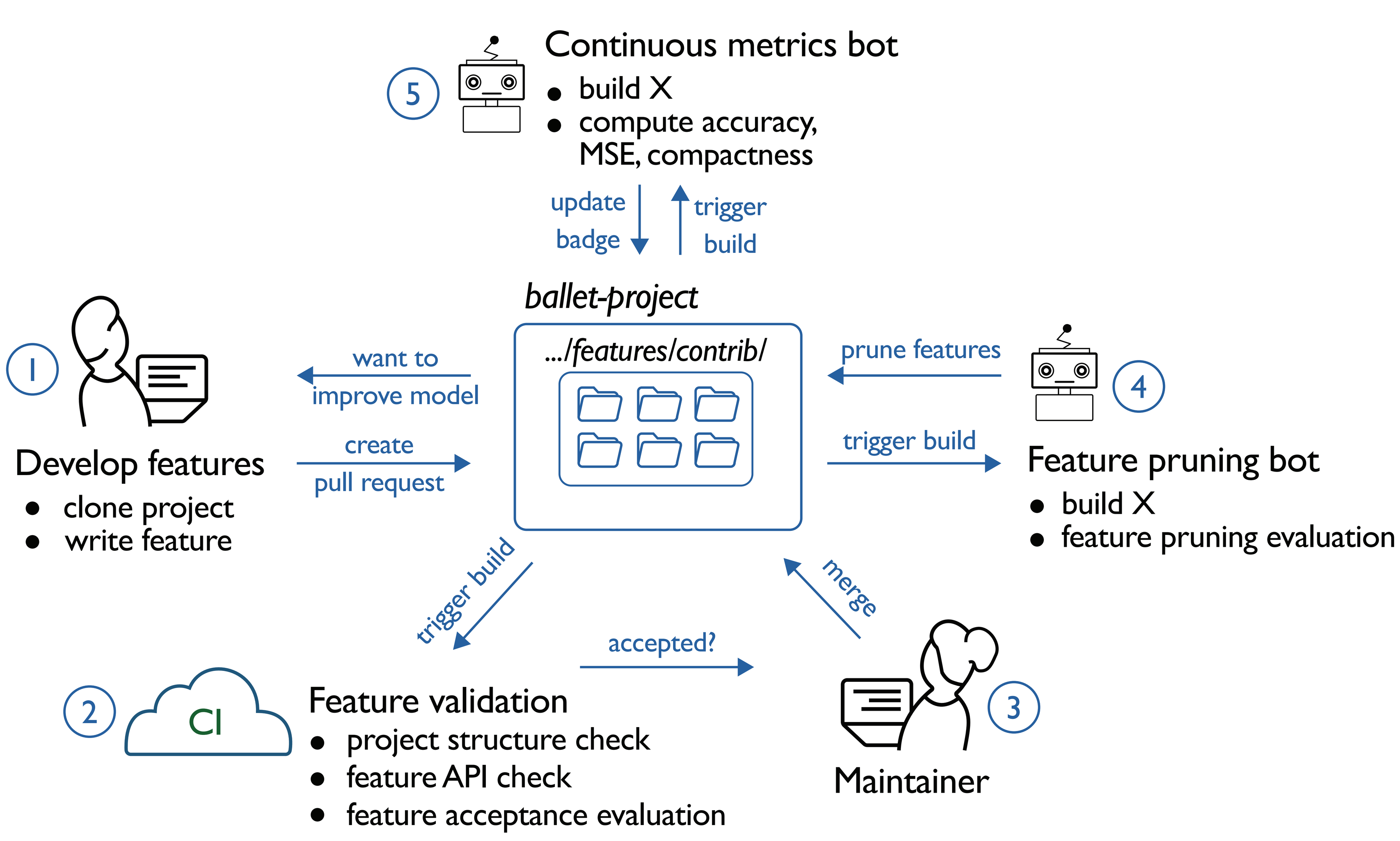

Figure 2: The feature development lifecycle

```
import ballet.eng
input = ['Full Bath', 'Half Bath',
         'Bsmt Full Bath', 'Bsmt Half Bath']
def count_baths(df):
    return df['Full Bath'] + 0.5 * df['Half Bath'] + \
           df['Bsmt Full Bath'] + 0.5 * df['Bsmt Half Bath']
transformer = ballet.eng.SimpleFunctionTransformer(func=count_baths)
feature = Feature(input=input,
                  transformer=transformer,
                  name='Bathroom Count')
```

Figure 3: A Ballet feature from the Ames problem

Model The resulting feature engineering pipeline is used as a dependency to an AutoML solution or a custom ML model (not the focus of this work).

Logical feature selection

A *logical feature* is a function that maps raw variables in one data instance to a vector of feature values,

$$f_j^{\mathcal{P}} : \mathcal{V}^p \to \mathbb{R}^{q_j},$$

where q_j is the dimensionality of the feature vector.

⇒ Can have $q_j > 0$, such as with one-hot encodings, embeddings, etc.

The *logical feature selection problem* is to select a subset of feature functions,

$$\mathcal{F}^* = \underset{\mathcal{F}' \in \mathcal{P}(\mathcal{F})}{\arg\min}\, \mathbb{E}[\mathcal{L}(\mathcal{A}_{\mathcal{F}'})]$$

Contrast this with the traditional feature selection problem to select a subset of feature values, $X^* \subseteq X$.

Streaming logical feature selection (SLFS): Let $\mathcal{F}_t$ be the set of features accepted as of time t, and let f_{t+1} be proposed at time $t+1$.

- *Acceptance problem*: accept f_{t+1}, setting $\mathcal{F}_{t+1} = \mathcal{F}_t \cup f_{t+1}$, or *reject*, setting $\mathcal{F}_{t+1} = \mathcal{F}_t$.
- *Pruning problem*: remove a subset $S \subseteq \mathcal{F}_{t+1}$ of low-quality features, setting $\mathcal{F}_{t+1} = \mathcal{F}_{t+1} \setminus S$.

Example Adapt α-investing [2] to the logical feature selection setting:

$$T = -2(logL(\mathcal{F}_t) - logL(\mathcal{F}_t^{\dagger}))$$
$$T \sim \chi^2(q)$$

We can compute a p-value and accept if $p_t < \alpha_t$.

Evaluation

Case study: Ames housing price prediction.

- Extract 249 logical features from 9 public notebooks on Kaggle.
- Simulate a scenario in which Kagglers submitted their features to a Ballet project instead.
- Iteratively select random notebook, simulate its submission, and validate using SLFS.

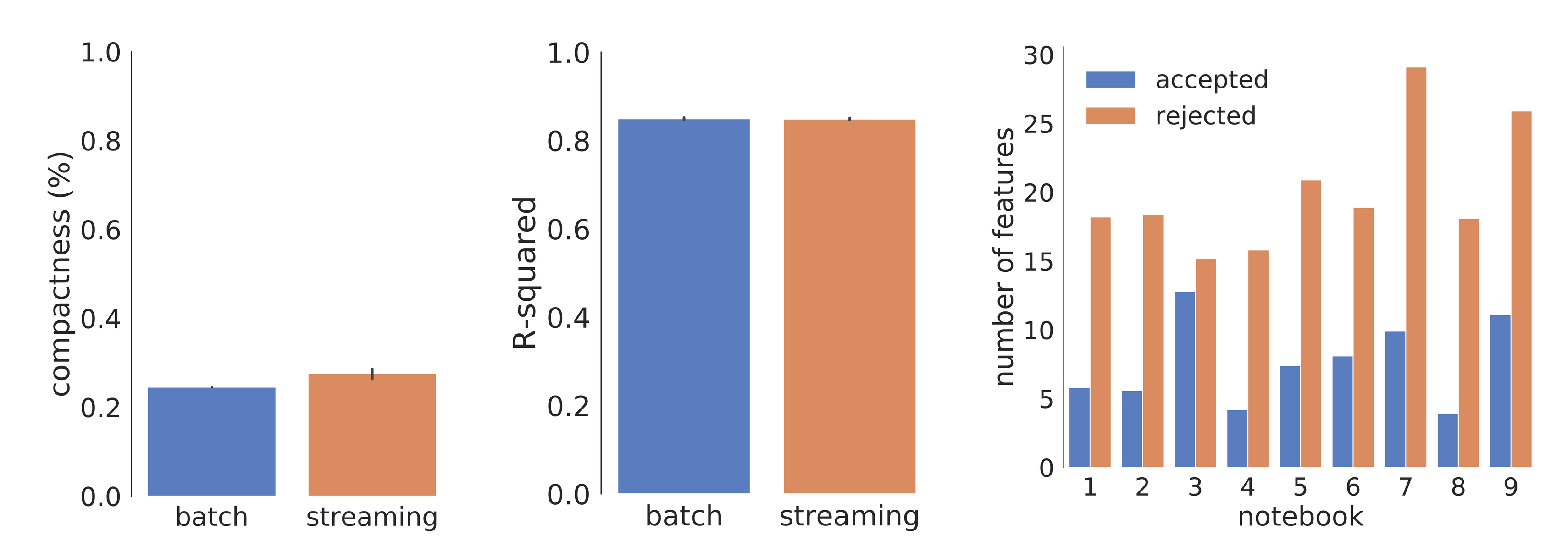

Figure 4: Performance of streaming and batch feature selection on Ames notebook features in terms of compactness (percentage of logical features selected by algorithm, left) and R^2 (center); mean accepted and rejected features per data scientist using SLFS (right).

Takeaways: 72.4% of all features are rejected by the feature validation and SLFS algorithm, suggesting substantial work was redundant across notebooks. Every notebook had both accepted and rejected features, suggesting both that everyone had something to contribute to final pipeline but that everyone did redundant work.

Conclusion and future work

We describe the design and implementation of the Ballet framework and demonstrate its use through a case study on real-world data.

As we continue to develop Ballet, in further research, we hope to assess impacts on developer efficiency through user studies, deploy it in large-scale collaborations (are you interested?), and investigate more sophisticated SLFS algorithms.

References

[1] A. Kindel, V. Bansal, K. Catena, T. Hartshorne, K. Jaeger, D. Koffman, S. McLanahan, M. Phillips, S. Rouhani, R. Vinh, and et al. Improving metadata infrastructure for complex surveys: Insights from the fragile families challenge, Sep 2018.

[2] J. Zhou, D. Foster, R. Stine, and L. Ungar. Streaming feature selection using alpha-investing. *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining - KDD '05*, page 384, 2005.

{micahs,kelvinlu,kalyanv}@mit.edu