{AMS}: Generating AutoML search spaces from weak specifications

José Pablo Cambronero (MIT)
Jürgen Cito (TU Wien/MIT)
Martin Rinard (MIT)
Team

José

Jürgen

Martin
Machine Learning Pipelines
How can developers write ML pipelines?
How can developers write ML pipelines?

Manually implement and validate your own pipelines
How can developers write ML pipelines?

Manually implement and validate your own pipelines

Use AutoML

ML Expertise Spectrum

Expert knowledge

Non-ML expert
Pros/Cons of Manual/AutoML

- Manual
  - High degree of control
  - Requires expert knowledge
  - Developer-time consuming
Pros/Cons of Manual/AutoML

- **Manual**
  - ✔️ High degree of control
  - ✗ Requires expert knowledge
  - ✗ Developer-time consuming

- **AutoML**
  - ✗ Low degree of control
  - ✔️ Does not require expert knowledge
  - ✔️ Reduces developer-time
How can developers write ML pipelines?

Manually implement and validate your own pipelines

Use AutoML

ML Expertise Spectrum

Expert knowledge

Some knowledge

Non-ML expert
How can developers write ML pipelines?

- Manually implement and validate your own pipelines
- Use AutoML
How can developers write ML pipelines?

ML Expertise Spectrum

Manually implement and validate your own pipelines

[AMS]

Use AutoML

Expert knowledge

Some knowledge

ML Expertise Spectrum

Non-ML expert
ML Pipeline Search Space
ML Pipeline Search Space
ML Pipeline Search Space
ML Pipeline Search Space

Partial knowledge

Desirable
ML Pipeline Search Space

Partial knowledge → Desirable
1. User provides a set of API components

2. {AMS} adds alternative API components

3. {AMS} adds complementary API components

4. {AMS} populates the set of hyperparameters and values

5. {AMS} pairs with a user-chosen search procedure, fully defining search space
1. User provides a set of API components
2. {AMS} adds alternative API components
3. {AMS} adds complementary API components
4. {AMS} populates the set of hyperparameters and values
5. {AMS} pairs with a user-chosen search procedure, fully defining search space
1. User provides a set of API components

2. {AMS} adds alternative API components

3. {AMS} adds complementary API components

4. {AMS} populates the set of hyperparameters and values

5. {AMS} pairs with a user-chosen search procedure, fully defining search space
1. User provides a set of API components
2. {AMS} adds alternative API components
3. {AMS} adds complementary API components
4. {AMS} populates the set of hyperparameters and values
5. {AMS} pairs with a user-chosen search procedure, fully defining search space
{AMS} Workflow

1. User provides a set of API components
2. {AMS} adds alternative API components
3. {AMS} adds complementary API components
4. {AMS} populates the set of hyperparameters and values
5. {AMS} pairs with a user-chosen search procedure, fully defining search space
Let’s zoom in...
1. User provides a set of API components

2. {AMS} adds alternative API components

3. {AMS} adds complementary API components

4. {AMS} populates the set of hyperparameters and values

5. {AMS} pairs with a user-chosen search procedure, fully defining search space
1. User provides set of API components

    Wants linear classifier

    Knows LogisticRegression is a linear classifier

    

    \{ 
        LogisticRegression
    \}
1. User provides a set of API components

2. {AMS} adds alternative API components

3. {AMS} adds complementary API components

4. {AMS} populates the set of hyperparameters and values

5. {AMS} pairs with a user-chosen search procedure, fully defining search space
2. {AMS} adds alternative API components

Functionally-related components

Insight: Use component descriptions to identify related components
2. **AMS** uses API documentation

Natural language descriptions

```python
In [3]: help(sklearn.linear_model.LogisticRegression)
Help on class LogisticRegression in module sklearn.linear_model._logistic:

    LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)

    Logistic Regression (aka logit, MaxEnt) classifier.
    In the multiclass case, the training algorithm uses the one-vs-rest (OvR)
    scheme if the 'multi_class' option is set to 'ovr', and uses the
```
2. {AMS} adds alternative API components

Use initial specification documentation as query

Retrieved components are relevant and related
2. {AMS} adds alternative API components

D: document $\rightarrow$ potential components’ documentation

Q: query $\rightarrow$ existing components’ documentation

C: corpus $\rightarrow$ complete API documentation

\[
BM25(D, Q) = \sum_{i=1}^{n} \text{IDF}(C, q_i) \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{\text{Len}(D)}{\text{AvgLen}(C)})}
\]
{ }

LogisticRegression,
LinearSVC
}

1. User provides a set of API components
2. {AMS} adds alternative API components
3. {AMS} adds complementary API components
4. {AMS} populates the set of hyperparameters and values
5. {AMS} pairs with a user-chosen search procedure, fully defining search space
3. {AMS} adds complementary API components

Complementary Components

Insight: useful components appear together in existing code
3. {AMS} uses existing source code

```
xgb = XGBClassifier(learning_rate=0.02, n_estimators=500, objective='binary:logistic',
                    silent=True, nthread=1)
```

```
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler(feature_range=(0,1))
X_train = mms.fit_transform(X_train)
X_test = mms.fit_transform(X_test)
```

```
svc_classifier.fit(X_train,y_train)
y_pred = svc_classifier.predict(X_test)
```
3. {AMS} uses existing source code

```python
xgb = XGBClassifier(learning_rate=0.02, n_estimators=500, objective='binary:logistic',
                    silent=True, nthread=1)

In [14]:
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler(feature_range=(0, 1))
X_train = mms.fit_transform(X_train)
X_test = mms.fit_transform(X_test)

In [15]:
svc_classifier.fit(X_train, y_train)
y_pred = svc_classifier.predict(X_test)
```
3. {AMS} adds complementary API components

Pointwise Mutual Information: appear more than expected if independent?

\[ \frac{p(x, y)}{p(x)p(y)} \]
3. {AMS} adds complementary API components

*Normalized* Pointwise Mutual Information (-1, 1)

Build NPMI-based association rules table

\[
\frac{p(x, y)}{p(x)p(y)} \quad \text{NPMI}(x,y) = \frac{\log_2 \left( \frac{p(x,y)}{p(x)p(y)} \right)}{-\log_2(p(x,y))}
\]
{ 
   PolyFeatures, 
   MinMaxScaler, 
   VarianceThreshold, 
   LogisticRegression, 
   LinearSVC 
}
{AMS} Workflow

1. User provides a set of API components
2. {AMS} adds alternative API components
3. {AMS} adds complementary API components
4. {AMS} populates the set of hyperparameters and values
5. {AMS} pairs with a user-chosen search procedure, fully defining search space
4. {AMS} populates hyperparameters

Different algorithms have different hyperparameters to choose/tune

Insight: users’ code sets/tunes useful hyperparameters
4. {AMS} uses existing source code

```python
In [4]:
xgb = XGBClassifier(
    learning_rate=0.02, n_estimators=500, objective='binary:logistic',
    silent=True, nthread=1)
```

```python
In [14]:
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler(feature_range=(0,1))
X_train = mms.fit_transform(X_train)
X_test = mms.fit_transform(X_test)
```

```python
In [15]:
svc_classifier.fit(X_train, y_train)
y_pred = svc_classifier.predict(X_test)
```
4. {AMS} populates hyperparameters

Just count!

Top-k frequency distribution
{  
  PolyFeatures: {"degree": [2, 3, 4]},
  MinMaxScaler: {},
  VarianceThreshold: {"threshold": [0.2]},
  LogisticRegression: {
    "penalty": ["l1", "elastic"], "C": [0.1, 100.0],
  },
  LinearSVC: {
    "penalty": ["l1"], "C": [0.1],
  }
}
1. User provides a set of API components
2. {AMS} adds alternative API components
3. {AMS} adds complementary API components
4. {AMS} populates the set of hyperparameters and values
5. {AMS} pairs with a user-chosen search procedure, fully defining search space
5. **AMS pairs with different sampling approaches**

   - Genetic Programming (TPOT)
   - Random search
Performance Evaluation
**Concept of Pipeline Win**

- Start with N systems
- Pick best pipeline from each one (F1 score on held-out test set)
- Compare all best pipelines
- System K wins if
  - It has best pipeline and
  - Its pipeline has F1 score at least 0.01 larger than next closest pipeline
Comparison with AL

- **AL**: AutoML tool that learns from existing code (OOPSLA 2019)
- AL gives *limited control* over pipelines produced
  - Like existing AutoML tools
- **AMS exposes control** through weak specifications and their augmentation
Comparison with AL

- 9 datasets, 5-fold cross validation
- Total 45 pipelines generated by each system
- Specification:

\{LogisticRegression, LinearSVC, StandardScaler\}
AMS wins (in spec): 48.9%
AL wins: 46.7%
Ties: 4.4%
...but AL doesn't use spec

AMS wins (in spec) 48.9%
AL wins (tree) 6.7%
AL wins (ensembles) 35.6%
Ties 4.4%
... AMS does

- All pipelines produced adhere to spec
- 42 wins after removing non-spec adherent AL pipelines
More Performance Evaluation

- Comparisons
  - Weak spec as pipeline
  - Weak spec + Search
  - Expert hyperparameters/values for weak spec + Search
  - AMS + Search

- Generate 15 weak specifications by composing popular components
  - 3 classifiers (logistic regression, random forest, decision tree)
  - 4 preprocessors (feature scaling, polynomial features, PCA, variance-based feature selection)

- 5 minutes search budget, 9 datasets
- 5-fold cross-validation
And pipelines generated reflect spec...
PolynomialFeatures,
MinMaxScaler,
VarianceThreshold,
RandomForestClassifier,
Additional results in paper

- Precision for functionally related component retrieval
- Precision for complementary component rules
- Characterize hyperparameter use in corpus
- Impact of varying corpus size
- And more!
AMS: A new model for interacting with AutoML

- Automatically generate search space
- Reflect influence of original specification

Partial user information (Weak Specification)  Automated Augmentation
Additional Information

- [Paper](#)
- [Zenodo Artifact](#)
- [Github](#)
Image Courtesy

- Binary Icon: Creative Stall (Noun Project)
- User Icon: Luis Prado (Noun Project)