Foreshadow Documentation

Release 1.0.1

Adithya Balaji, Alexander Allen

Apr 20, 2020

Contents

Foreshadow is an automatic pipeline generation tool that makes creating, iterating, and evaluating machine learning pipelines a fast and intuitive experience allowing data scientists to spend more time on data science and less time on code.

CHAPTER 1

Key Features

- Scikit-Learn compatible
- **Automatic column intent inference**
  - Numerical
  - Categorical
  - Text
  - Droppable (All values in a column are either the same or different)
- Allow user override on column intent and transformation functions
- **Automatic feature preprocessing depending on the column intent type**
  - Numerical: imputation followed by scaling
  - Categorical: a variety of categorical encoding
  - Text: TFIDF followed by SVD
- Automatic model selection
- Rapid pipeline development / iteration

CHAPTER 2

Features in the road map

- Automatic feature engineering
- Automatic parameter optimization

Foreshadow supports python 3.6+

CHAPTER 3

Installing Foreshadow

```
$ pip install foreshadow
```

Read the documentation to set up the project from source.

CHAPTER 4

Getting Started

To get started with foreshadow, install the package using pip install. This will also install the dependencies. Now create a simple python script that uses all the defaults with Foreshadow.

First import foreshadow

```
from foreshadow.foreshadow import Foreshadow
from foreshadow.estimators import AutoEstimator
from foreshadow.utils import ProblemType
```

Also import sklearn, pandas, and numpy for the demo

```
import pandas as pd

from sklearn.datasets import boston_housing
from sklearn.model_selection import train_test_split
```

Now load in the boston housing dataset from sklearn into pandas dataframes. This is a common dataset for testing machine learning models and comes built in to scikit-learn.

```
boston = load_boston()
bostonX_df = pd.DataFrame(boston.data, columns=boston.feature_names)
bostony_df = pd.DataFrame(boston.target, columns=['target'])
```

Next, exactly as if working with an sklearn estimator, perform a train test split on the data and pass the train data into the fit function of a new Foreshadow object

```
X_train, X_test, y_train, y_test = train_test_split(bostonX_df,
   bostony_df, test_size=0.2)

problem_type = ProblemType.REGRESSION

estimator = AutoEstimator(
    problem_type=problem_type,
    auto="tpot",
```

(continues on next page)

(continued from previous page)

```
    estimator_kwargs={"max_time_mins": 1},
)
shadow = Foreshadow(estimator=estimator, problem_type=problem_type)
shadow.fit(X_train, y_train)
```

Now *fs* is a fit Foreshadow object for which all feature engineering has been performed and the estimator has been trained and optimized. It is now possible to utilize this exactly as a fit sklearn estimator to make predictions.

```
shadow.score(X_test, y_test)
```

Great, you now have a working Foreshaow installation! Keep reading to learn how to export, modify and construct pipelines of your own.

CHAPTER 5

Tutorial

We also have a jupyter notebook tutorial to go through more details under the *examples* folder.

CHAPTER 6

Documentation

Read the docs!

CHAPTER 7

The Developer Guide

7.1 Developers Guide

Thank you for taking the time to contribute and reading this page, any and all help is appreciated!

7.1.1 Setting up the Project From Source

General Setup

1. Clone the project down to your computer

```
$ git clone https://github.com/georgianpartners/foreshadow.git
$ cd foreshadow
$ git checkout development
```

2. Install and setup pyenv and pyenv-virtualenv

   Follow the instructions on their pages or use homebrew if you have a Mac

```
$ brew install pyenv
$ brew install pyenv-virtualenv
```

   Make sure to add the following lines to your `.bash_profile`

```
export PYENV_ROOT="$HOME/.pyenv"
export PATH="$PYENV_ROOT/bin:$PATH"
if command -v pyenv 1>/dev/null 2>&1; then
  eval "$(pyenv init -)"
fi
eval "$(pyenv virtualenv-init -)"
```

   Restart your shell session for the changes to take effect and perform the following setup **in the root directory of the project**. This sets up a convenient virtualenv that automatically activates in the root of your project. (Note: there is a known error with pyenv. Also, you may need to change the file path depending on your version or you may not even need to do that step.

```
$ open /Library/Developer/CommandLineTools/Packages/macOS_SDK_headers_for_
↪macOS_10.14.pkg
$ pyenv install 3.6.8
$ pyenv global 3.6.8
$ pyenv virtualenv -p python3.6 3.6.8 venv
$ pyenv local venv 3.6.8
```

3. Install poetry package manager

```
(venv) $ pyenv shell system
$ curl -sSL https://raw.githubusercontent.com/sdispater/poetry/master/get-
↪poetry.py | python
$ pyenv shell --unset
```

Prepare for Autosklearn install Autosklearn was setup as an optional dependency as it can be sometimes difficult to install because of its requirement of xgboost. In order to have a development environment that passes all tests, autosklearn is required.

1. Install swig

   Use your package manager to install swig

```
(venv) $ brew install swig # (or apt-get)
```

2. Install gcc (MacOS only)

   Use your package manager to install gcc (necessary for xgboost)

```
(venv) $ brew install gcc@5 # (or apt-get)
```

Install all the packages and commit hooks When the project is installed through poetry both project requirements and development requirements are installed. Install commit-hooks using the pre-commit utility.

```
(venv) $ poetry install -v
(venv) $ export CC=gcc-5; export CXX=g++-5;
(venv) $ poetry install -E dev
(venv) $ poetry run pre-commit install
```

Configure PlantUML

(venv) $ brew install plantuml # MacOS (requires brew cask install adoptopenjdk) (venv) $ sudo apt install plantuml # Linux

Making sure everything works

1. Run pytest to make sure you're good to go

```
(venv) $ poetry run pytest
```

2. Run tox to run in supported python versions (optional)

```
(venv)$ poetry run tox -r # supply the -r flag if you changed the dependencies
```

3. Run make html in foreshadow/doc to build the documentation (optional)

```
(venv) $ poetry run make html
```

If all the tests pass you're all set up!

Note: Our platform also includes integration tests that asses the overall performance of our framework using the default settings on a few standard ML datasets. By default these tests are not executed, to run them, set an environmental variable called *FORESHADOW_TESTS* to *ALL*

Suggested development work flow

1. Create a branch off of development to contain your change

   ```
   (venv) $ git checkout development
   (venv) $ git checkout -b {your_feature}
   ```

2. Run pytest and pre-commit while developing This will help ensure something hasn't broken while adding a feature. Pre-commit will lint the code before each commit.

   ```
   $ poetry run pytest
   $ poetry run pre-commit run --all-files
   ```

3. Run tox to test your changes across versions Make sure to add test cases for your change in the appropriate folder in foreshadow/tests and run tox to test your project across python 3.5 and 3.6

   ```
   $ poetry run tox
   ```

4. Submit a pull request This can be tricky if you have cloned the project instead of forking it but no worries the fix is simple. First go to the project page and **fork it there**. Then do the following.

   ```
   (venv) $ git remote add upstream https://github.com/georgianpartners/
   ↪foreshadow.git
   (venv) $ git remote set-url origin https://github.com/{YOUR_USERNAME}/
   ↪foreshadow.git
   (venv) $ git push origin {your_feature}
   ```

   Now you can go to the project on your github page and submit a pull request to the main project.

   Note: Make sure to submit the pull request against the development branch.

7.1.2 Adding Transformers

Adding transformers is quite simple. Simply write a class with the *fit transform* and *inverse_transform* methods that extends `scikit_learn.base.BaseEstimator` and `sklearn.base.TransformerMixin`. Take a look at the structure below and modify it to suit your needs. We would recommend taking a look at the sklearn.preprocessing.RobustScaler source code for a good example.

```
from foreshadow.base import TransformerMixin, BaseEstimator
from sklearn.utils import check_array

class CustomTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        X = check_array(X)
```

(continues on next page)

(continued from previous page)

```
        return self

    def transform(self, X, y=None):
        X = check_array(X, copy=True)
        # modify input based on fit here
        return X

    def inverse_transform(self, X):
        X = check_array(X, copy=True)
        # if applicable, write inverse transform here
        return X
```

After writing your transformer make sure place it in the internals folder in its own file with the associated tests for the transformer in the mirrored test directory and you are all set. If you want to add an external transformer that is not already supported by foreshadow submit a pull request with the appropriate modification to the *externals.py* file in transformers.

7.1.3 Future Architecture Roadmap

In progress

CHAPTER 8

Changelog

8.1 Foreshadow 1.0.0 (2020-03-20)

8.1.1 Features

- **AutoIntent Resolving Dependency upgrade** The automl intent resolving model has now been retrained with pandas 0.25 and sklearn 0.22 (autointent-resolving-dependency-upgrade)
- **Adding support for Text Intent and Text transformation pipeline** Foreshadow now supports Text Intent and has a pipeline processing Text using TFIDF and TruncatedSVD. (text-transformation-pipeline)

8.2 Foreshadow 0.4.5 (2020-03-09)

8.2.1 Features

- **Dependency upgrade and related code changes** Scikit-Learn 0.19 -> 0.22.1 Pandas 0.23 -> 0.25

  Removed ParallelProcessor and DynamicPipeline in favor of native ColumnTransformer. (dependency-upgrade)

8.3 Foreshadow 0.4.4 (2020-01-29)

8.3.1 Features

- **Bug fixes and updates:**
  1. AutoIntent Resolving code update.
  2. fixing intent override from Categorical to Numeric issue in DataExportor. (bug-fixes)

8.4 Foreshadow 0.4.3 (2020-01-08)

8.4.1 Features

- **Bug fixes and updates:**

    1. Fix an issue in exported pickle file missing label encoder for target

  variabes.

    2. [Experimental] Allow user to supply customized cleaning functions. (bug-fixes-and-updates)

8.5 Foreshadow 0.4.2 (2019-12-23)

8.5.1 Features

- **Bug fixes and updates:**

    1. Add drop functionality to the transform method in the cleaner mapper. (bug-fix-4)

8.6 Foreshadow 0.4.1 (2019-12-23)

8.6.1 Features

- **Bug fixes and updates:**

    1. Abort the training process is the whole dataframe is empty due to

  missing data after data cleaning step. (bug-fix-3)

8.7 Foreshadow 0.4.0 (2019-12-20)

8.7.1 Features

- **Bug fixes and updates:**

    1. Allow user to pickle fitted_pipeline.
    2. Treat NaN as a category.
    3. Runtime performance improving with data sampling on cleaning and intent

  resolving steps.

    4. Export processed dataset before fitting the estimator.
    5. Disable dummy encoding in categorical encoding process temporarily. (bug-fix-2)

8.8 Foreshadow 0.3.2 (2019-12-03)

8.8.1 Features

- **Feature: Bug fix for intent override** Add back the missing intent override (bug-fix-for-intent-override)

8.9 Foreshadow 0.3.0 (2019-11-21)

8.9.1 Features

- **Feature: Auto Intent Resolving** Automatically resolve the intent of a column with a machine learning model. (auto-intent-resolving)
- Bug fix of pick_transformer may transform dataframe in place, causing inconsistency between the data and intended downstream logic. (bug-fix)
- **Feature: Enable logging on Foreshadow** This feature allows Foreshadow to display the progress of the training. (enable-logging)
- **Feature: Adding more default models in Foreshadow** This allows user to select estimators from the following categories: - Linear - SVM - RandomForest - NeuralNetwork (more-default-model)
- **Feature: Allow user to override intent resolving decisions** This feature allows users to override the intent resolving decisions through API calls. It can be done both before and after fitting the foreshadow object. (user-override)

8.10 Foreshadow 0.2.1 (2019-09-26)

8.10.1 Features

- Bug fix of pick_transformer may transform dataframe in place, causing inconsistency between the data and intended downstream logic. (bug-fix)

8.11 Foreshadow 0.2.0 (2019-09-24)

8.11.1 Features

- Add feature_summarizer to produce statistics about the data after intent resolving to show the users why such decisions are made. (data-summarization)
- Foreshadow is able to run end-to-end with level 1 optimization with the tpot auto-estimator. (level1-optimization)
- Add Feature Reducer as a passthrough transformation step. (pass-through-feature-reducer)
- Multiprocessing: 1. Enable multiprocessing on the dataset. 2. Collect changes from each process and update the original columnsharer. (process-safe-columnsharer)
- Serialization and deserialization: 1. Serialization of the foreshadow object in a non-verbose format. 2. Deserialization of the foreshadow object. (serialization)

- Adding two major components: 1. usage of metrics for any statistic computation 2. changing functionality of wrapping sklearn transformers to give them DataFrame capabilities. This now uses classes and metaclasses, which should be easier to maintain (#74)
- Adding ColumnSharer, a lightweight wrapper for a dictionary that functions as a cache system, to be used to pass information in the foreshadow pipeline. (#79)
- Creating DataPreparer to handle data preprocessing. Data Cleaning is the first step in this process. (#93)
- Adds skip resolve functionality to SmartTransformer, restructure utils, and add is_wrapped to utils (#95)
- Add serializer mixin and resture package import locations. (#96)
- Add configuration file parser. (#99)
- Add Feature Engineerer as a passthrough transformation step. (#112)
- Add Intent Mapper and Metric wrapper features. (#113)
- Add Preprocessor step to DataPreparer (#118)
- Create V2 architecture shift. (#162)

8.12 Foreshadow 0.1.0 (2019-06-28)

8.12.1 Features

- Initial release. (#71)

CHAPTER 9

Indices and tables

- genindex
- modindex
- search