Step Forward Feature Selection: A Practical Example in Python

<= Previous post
Next post =>

Tags: Feature Selection, Machine Learning, Python

When it comes to disciplined approaches to feature selection, wrapper methods are those which marry the feature selection process to the type of model being built, evaluating feature subsets in order to detect the model performance between features, and subsequently select the best performing subset.

By Matthew Mayo, KDnuggets.

comments

Many methods for feature selection exist, some of which treat the process strictly as an artform, others as a science, while, in reality, some form of domain knowledge along with a disciplined approach are likely your best bet.

When it comes to disciplined approaches to feature selection, wrapper methods are those which marry the feature selection process to the type of model being built, evaluating feature subsets in order to detect the model performance between features, and subsequently select the best performing subset. In other words, instead of existing as an independent process taking place *prior* to model building, wrapper methods attempt to optimize feature selection process for a given machine learning algorithm *in tandem* with this algorithm.

2 prominent wrapper methods for feature selection are step forward feature selection and step backward features selection.

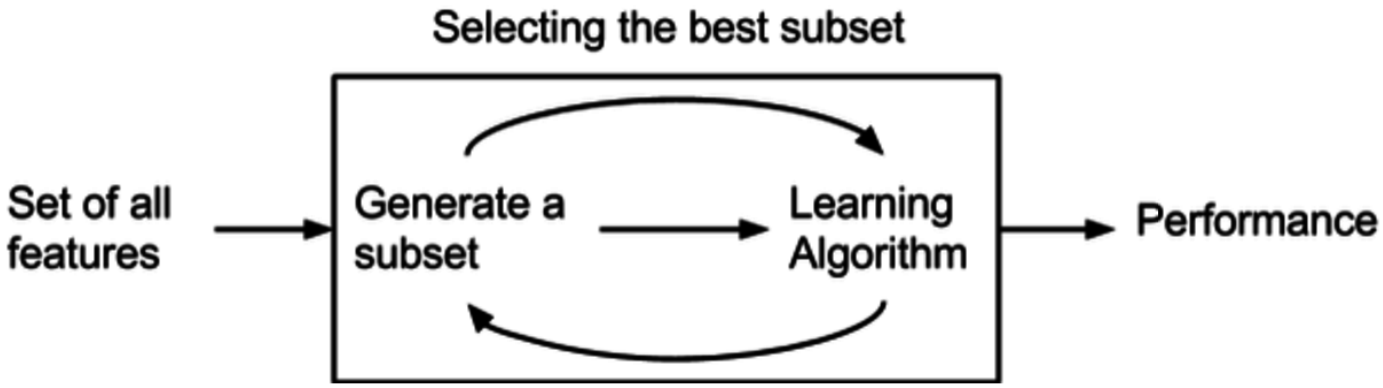

Image source

Step forward feature selection starts with the evaluation of each individual feature, and selects that which results in the best performing selected algorithm model. What's the "best?" That depends entirely on the defined evaluation criteria (AUC, prediction accuracy, RMSE, etc.). Next, all possible combinations of the that selected feature and a subsequent feature are evaluated, and a second feature is selected, and so on, until the required predefined number of features is selected.

Step backward feature selection is closely related, and as you may have guessed starts with the entire set of features and works backward from there, removing features to find the optimal subset of a predefined size.

These are both potentially very computationally expensive. Do you have a large, multidimensional dataset? These methods may take too long to be at all useful, or may be totally infeasible. That said, with a dataset of accommodating size and dimensionality, such an approach may well be your best possible approach.

To see how they work, let's take a look at step forward feature selection, specifically. Note that, as discussed, a machine learning algorithm must be defined prior to beginning our symbiotic feature selection process.

Keep in mind that an optimized set of selected features using a given algorithm may or may not perform equally well with a different algorithm. If we select features using logistic regression, for example, there is no guarantee that these same features will perform optimally if we then tried them out using K-nearest neighbors, or an SVM.

Implementing Feature Selection and Building a Model

So, how do we perform step forward feature selection in Python? Sebastian Raschka's mlxtend library includes an implementation (Sequential Feature Selector), and so we will use it to demonstrate. It goes without saying that you should have mlxtend installed before moving forward (check the Github repo).

We will use a Random Forest classifier for feature selection and model building (which, again, are intimately related in the case of step forward feature selection).

We need data to use for demonstration, so let's use the wine quality dataset. Specifically, I have used the untouched `winequality-white.csv` file as input in the code below.

Arbitrarily, we will set the desired number of features to 5 (there are 12 in the dataset). What we are able to do is compare the evaluation scores for each iteration of the feature selection process, and so keep in mind that if we find that a lower number of features has a better score we can alternatively choose that best-performing subset to run with in our "live" model moving forward. Also keep in mind that setting our desired number of features too low could lead to a sub-optimal number and combination of features being decided upon (say, if some combination of 11 features in our case is better than the best combination of <= 10 features we find during the selection process).

Since we are more interested in demonstrating how to implement step forward feature selection than we are with the actual results on this particular dataset, we won't be overly concerned with the actual performance of our models, but we will compare the performances anyhow, as to show how it would be done in a meaningful project.

First, we will make our imports, load the dataset, and split it into training and testing sets.

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score as acc
from mlxtend.feature_selection import SequentialFeatureSelector as sfs

# Read data
df = pd.read_csv('winequality-white.csv', sep=';')

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    df.values[:,:-1],
    df.values[:,-1:],
    test_size=0.25,
    random_state=42)

y_train = y_train.ravel()
y_test = y_test.ravel()

print('Training dataset shape:', X_train.shape, y_train.shape)
print('Testing dataset shape:', X_test.shape, y_test.shape)
```

```
Training dataset shape: (3673, 11) (3673,)
Testing dataset shape: (1225, 11) (1225,)
```

Next, we will define a classifier, as well as a step forward feature selector, and then perform our feature selection. The feature feature selector in mlxtend has some parameters we can define, so here's how we will proceed:

- First, we pass our classifier, the Random Forest classifier defined above the feature selector
- Next, we define the subset of features we are looking to select (k_features=5)
- We then set floating to False; see the documentation for more info on floating:

  The floating algorithms have an additional exclusion or inclusion step to remove features once they were included (or excluded), so that a larger number of feature subset combinations can be sampled.

- We set he desired level of verbosity for mlxtend to report
- Importantly, we set our scoring to accuracy; this is but one metric which could be used to score our resulting models built on the selected features
- mlxtend feature selector uses cross validation internally, and we set our desired folds to 5 for our demonstration

The dataset we chose isn't very large, and so the following code should not take long to execute.

```
# Build RF classifier to use in feature selection
clf = RandomForestClassifier(n_estimators=100, n_jobs=-1)

# Build step forward feature selection
sfs1 = sfs(clf,
           k_features=5,
           forward=True,
           floating=False,
           verbose=2,
           scoring='accuracy',
           cv=5)

# Perform SFFS
sfs1 = sfs1.fit(X_train, y_train)
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:   24.3s finished

[2018-06-12 14:47:47] Features: 1/5 -- score: 0.49768148939247264[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.2s remaining
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:   22.7s finished

[2018-06-12 14:48:09] Features: 2/5 -- score: 0.5442629071398873[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.7s remaining:
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:   21.2s finished

[2018-06-12 14:48:31] Features: 3/5 -- score: 0.6052194438136681[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.5s remaining:
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:   20.3s finished
```

```
[2018-06-12 14:48:51] Features: 4/5 -- score: 0.6261526236769334[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.4s remaining:
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:   17.3s finished

[2018-06-12 14:49:08] Features: 5/5 -- score: 0.6444222989869156
```

Our best performing model, given our scoring metric, is some subset of 5 features, with a score of 0.644 (remember that this is using cross validation, and so will be different than that which is reported on our full models below, using train and test sets). But which subset of 5 features were selected?

```
# Which features?
feat_cols = list(sfs1.k_feature_idx_)
print(feat_cols)
```

```
[1, 2, 3, 7, 10]
```

The columns at these indexes are those which were selected. Great! So what now...?

We can now use those features to build a full model using our training and test sets. If we had a much larger set (i.e. many more instances as opposed to many more features), this would be especially beneficial as we could have used the feature selector above on a smaller subset of instances, determined our best performing subset of features, and then applied them to the full dataset for classification.

The code below builds a classifier on **only** the subset of selected features.

```
# Build full model with selected features
clf = RandomForestClassifier(n_estimators=1000, random_state=42, max_depth=4)
clf.fit(X_train[:, feat_cols], y_train)

y_train_pred = clf.predict(X_train[:, feat_cols])
print('Training accuracy on selected features: %.3f' % acc(y_train, y_train_pred))

y_test_pred = clf.predict(X_test[:, feat_cols])
print('Testing accuracy on selected features: %.3f' % acc(y_test, y_test_pred))
```

```
Training accuracy on selected features: 0.558
Testing accuracy on selected features: 0.512
```

Don't worry about the actual accuracies; we're concerned here with the process, not the end result.

But what if we *were* concerned with the end result, and wanted to know if our feature selection troubles had been worth it? Well, we could compare the resultant accuracies of the full model built using the selected features (immediately above) with the resultant accuracies of another full model using **all** of the features, just as we do below:

```
# Build full model on ALL features, for comparison
clf = RandomForestClassifier(n_estimators=1000, random_state=42, max_depth=4)
clf.fit(X_train, y_train)

y_train_pred = clf.predict(X_train)
print('Training accuracy on all features: %.3f' % acc(y_train, y_train_pred))

y_test_pred = clf.predict(X_test)
print('Testing accuracy on all features: %.3f' % acc(y_test, y_test_pred))
```

```
Training accuracy on all features: 0.566
Testing accuracy on all features: 0.509
```

And there you have them for comparison. They are both poor and comparable to our model built with the selected features (though I promise this is not always the case!). With very little work, you could see how these selected features perform with a different algorithm, to help scratch that itch as to wondering whether these features selected with one algorithm are equally well performing with another.

Such a feature selection method can be an effective part of a disciplined machine learning pipeline. Keep in mind that step forward (or step backward) methods, specifically, can provide problems when dealing with especially large or highly-dimensional datasets. There are ways of getting around (or **trying** to get around) these sticking points, such as sampling from the data to find the feature subset which works best, and then using these features for the modeling process on the full dataset. Of course, these are not the only disciplined approaches to feature selection either, and so checking out alternatives may be warranted when dealing with these larger datasets.

Related:

- Quick Feature Engineering with Dates Using fast.ai
- Generating Text with RNNs in 4 Lines of Code
- Multi-objective Optimization for Feature Selection

<= Previous post
Next post =>

Top Stories Past 30 Days

Most Popular

1. **Five Cool Python Libraries for Data Science**
2. **The Super Duper NLP Repo: 100 Ready-to-Run Colab Notebooks**
3. **Natural Language Processing Recipes: Best Practices and Examples**
4. **24 Best (and Free) Books To Understand Machine Learning**
5. **Free High-Quality Machine Learning & Data Science Books & Courses: Quarantine Edition**
6. **Mathematics for Machine Learning: The Free eBook**
7. **How to select rows and columns in Pandas using [ ], .loc, iloc, .at and .iat**

Most Shared

1. **Free High-Quality Machine Learning & Data Science Books & Courses: Quarantine Edition**
2. **Beginners Learning Path for Machine Learning**
3. **Should Data Scientists Model COVID19 and other Biological Events**
4. **The Super Duper NLP Repo: 100 Ready-to-Run Colab Notebooks**
5. **Natural Language Processing Recipes: Best Practices and Examples**
6. **AI and Machine Learning for Healthcare**
7. **Deep Learning: The Free eBook**

Latest News

- The Best NLP with Deep Learning Course is Free
- Appropriately Handling Missing Values for Statistical M...
- A Holistic Framework for Managing Data Analytics Projects
- Build and deploy your first machine learning web app
- Top tweets, May 13-19: Linear algebra and optimizatio...
- Caserta Announces Pro-Bono Data and Analytics Workshop ...

Top Stories
Last Week

Most Popular

1. **Start Your Machine Learning Career in Quarantine**

2. **Deep Learning: The Free eBook**
3. **24 Best (and Free) Books To Understand Machine Learning**
4. **I Designed My Own Machine Learning and AI Degree**
5. **How to select rows and columns in Pandas using [ ], .loc, iloc, .at and .iat**
6. **Five Cool Python Libraries for Data Science**
7. **The Elements of Statistical Learning: The Free eBook**

Most Shared

1. **AI and Machine Learning for Healthcare**
2. **Start Your Machine Learning Career in Quarantine**
3. **Satellite Image Analysis with fast.ai for Disaster Recovery**

4. **Machine Learning in Power BI using PyCaret**
5. **The Elements of Statistical Learning: The Free eBook**
6. **What You Need to Know About Deep Reinforcement Learning**
7. **AI Channels to Follow**

More Recent Stories

- Caserta Announces Pro-Bono Data and Analytics Workshop for Sen...
- Dimensionality Reduction with Principal Component Analysis (PCA)
- An easy guide to choose the right Machine Learning algorithm
- Spotting Controversy with NLP
- Pandas in action!
- Complex logic at breakneck speed: Try Julia for data science
- 13 must-read papers from AI experts
- Looking Normal(ly Distributed)
- KDnuggets 20:n20, May 20: I Designed My Own ML and AI Degre...
- Google Unveils TAPAS, a BERT-Based Neural Network for Querying...
- What they do not tell you about machine learning
- Sparse Matrix Representation in Python
- Linear algebra and optimization and machine learning: A textbook
- Easy Text-to-Speech with Python
- Top Stories, May 11-17: Start Your Machine Learning Career in ...
- Evidence Counterfactuals for explaining predictive models on B...
- Automated Machine Learning: The Free eBook
- Cartoon: The Worst Telemedicine?
- 5 Great New Features in Scikit-learn 0.23
- AI Channels to Follow

KDnuggets Home » News » 2018 » Jun » Tutorials, Overviews » Step Forward Feature Selection: A Practical Example in Python (18:n24)