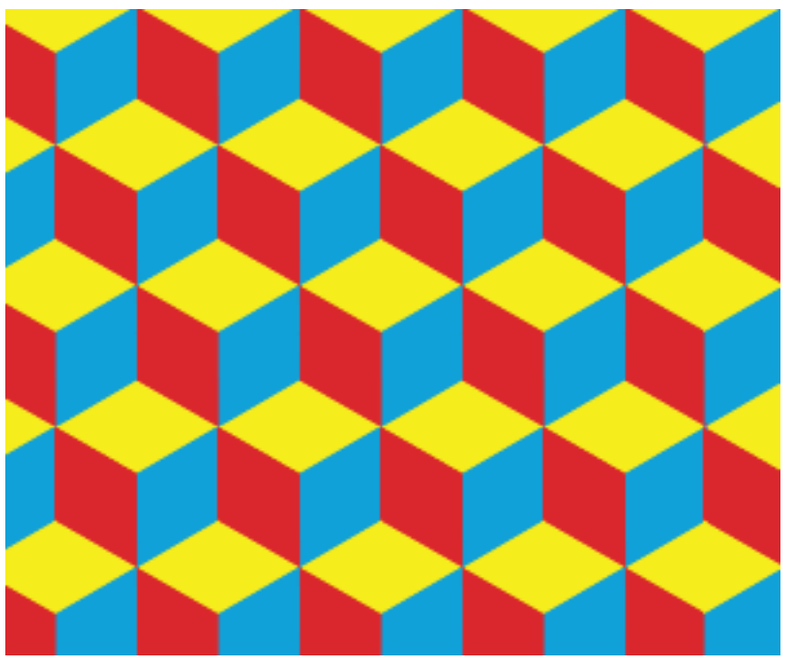
Managing Machine Learning Workflows with Scikit-learn Pipelines Part 3: Multiple Models, Pipelines, and Grid Searches

Previous post
Next post

Tags: Data Preprocessing, Hyperparameter, Optimization, Pipeline, Python, scikit-learn, Workflow

In this post, we will be using grid search to optimize models built from a number of different types estimators, which we will then compare and properly

evaluate the best hyperparameters that each model has to offer.



By Matthew Mayo, KDnuggets.

comments

First, I know that I promised we would be past the toy datasets last post, but for comparison purposes we will be sticking with iris for a bit longer. I think it's best we are able to still compare apples to apples throughout our entire process.

Thus far, in the previous 2 posts, we have:

- Introduced Scikit-learn piplines
- Demonstrated their basic usage by creating and comparing some pipelines
- Introduced grid search
- Demonstrated how pipelines and grid search work together by using grid search to find optimized hyperparameters, which was then apply to an embedded pipeline

Here's what we plan to do moving forward:

- In this post, we will be using grid search to optimize models built from a number of different types estimators, which we will then compare
- In the follow-up post, we will pivot toward using automated machine learning techniques to assist in the optimization of model hyperparameters, the end result of which will be an automatically generated, optimized Scikit-learn pipeline script file, courtesy of TPOT

There won't be much to re-explain this time; I recommend that you read the first post in this series to get a gentle introduction to pipelines in Scikit-learn, and the second post in this series for an overview of integrating grid search into your pipelines. What we will now do is build a series of pipelines of different estimators, using grid search for hyperparameter optimization, after which we will compare these various apples and oranges to determine the most accurate ("best") model.

The code below is well-commented, and if you have read the first 2 installments should be easy to follow.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.externals import joblib
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm

# Load and split the data
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=42)

# Construct some pipelines
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('clf', LogisticRegression(random_state=42))])

pipe_lr_pca = Pipeline([('scl', StandardScaler()),
                    ('pca', PCA(n_components=2)),
                    ('clf', LogisticRegression(random_state=42))])

pipe_rf = Pipeline([('scl', StandardScaler()),
                    ('clf', RandomForestClassifier(random_state=42))])

pipe_rf_pca = Pipeline([('scl', StandardScaler()),
                    ('pca', PCA(n_components=2)),
                    ('clf', RandomForestClassifier(random_state=42))])

pipe_svm = Pipeline([('scl', StandardScaler()),
                    ('clf', svm.SVC(random_state=42))])

pipe_svm_pca = Pipeline([('scl', StandardScaler()),
                    ('pca', PCA(n_components=2)),
                    ('clf', svm.SVC(random_state=42))])

# Set grid search params
param_range = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
param_range_fl = [1.0, 0.5, 0.1]

```

```
grid_params_lr = [{'clf__penalty': ['l1', 'l2'],
                   'clf__C': param_range_fl,
                   'clf__solver': ['liblinear']}]

grid_params_rf = [{'clf__criterion': ['gini', 'entropy'],
                   'clf__min_samples_leaf': param_range,
                   'clf__max_depth': param_range,
                   'clf__min_samples_split': param_range[1:]}]

grid_params_svm = [{'clf__kernel': ['linear', 'rbf'],
                    'clf__C': param_range}]

# Construct grid searches
jobs = -1

gs_lr = GridSearchCV(estimator=pipe_lr,
                     param_grid=grid_params_lr,
                     scoring='accuracy',
                     cv=10)

gs_lr_pca = GridSearchCV(estimator=pipe_lr_pca,
                         param_grid=grid_params_lr,
                         scoring='accuracy',
                         cv=10)

gs_rf = GridSearchCV(estimator=pipe_rf,
                     param_grid=grid_params_rf,
                     scoring='accuracy',
                     cv=10,
                     n_jobs=jobs)

gs_rf_pca = GridSearchCV(estimator=pipe_rf_pca,
                         param_grid=grid_params_rf,
                         scoring='accuracy',
                         cv=10,
                         n_jobs=jobs)

gs_svm = GridSearchCV(estimator=pipe_svm,
                      param_grid=grid_params_svm,
                      scoring='accuracy',
                      cv=10,
                      n_jobs=jobs)

gs_svm_pca = GridSearchCV(estimator=pipe_svm_pca,
                          param_grid=grid_params_svm,
                          scoring='accuracy',
                          cv=10,
                          n_jobs=jobs)

# List of pipelines for ease of iteration
grids = [gs_lr, gs_lr_pca, gs_rf, gs_rf_pca, gs_svm, gs_svm_pca]

# Dictionary of pipelines and classifier types for ease of reference
grid_dict = {0: 'Logistic Regression', 1: 'Logistic Regression w/PCA',
             2: 'Random Forest', 3: 'Random Forest w/PCA',
             4: 'Support Vector Machine', 5: 'Support Vector Machine w/PCA'}

```

```python
# Fit the grid search objects
print('Performing model optimizations...')
best_acc = 0.0
best_clf = 0
best_gs = ''
for idx, gs in enumerate(grids):
        print('\nEstimator: %s' % grid_dict[idx])
        # Fit grid search
        gs.fit(X_train, y_train)
        # Best params
        print('Best params: %s' % gs.best_params_)
        # Best training data accuracy
        print('Best training accuracy: %.3f' % gs.best_score_)
        # Predict on test data with best params
        y_pred = gs.predict(X_test)
        # Test data accuracy of model with best params
        print('Test set accuracy score for best params: %.3f ' % accuracy_score(y_test, y_pred))
        # Track best (highest test accuracy) model
        if accuracy_score(y_test, y_pred) > best_acc:
                best_acc = accuracy_score(y_test, y_pred)
                best_gs = gs
                best_clf = idx
print('\nClassifier with best test set accuracy: %s' % grid_dict[best_clf])

# Save best grid search pipeline to file
dump_file = 'best_gs_pipeline.pkl'
joblib.dump(best_gs, dump_file, compress=1)
print('\nSaved %s grid search pipeline to file: %s' % (grid_dict[best_clf], dump_file))
```



```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.externals import joblib
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm

# Load and split the data
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=42)

# Construct some pipelines
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('clf', LogisticRegression(random_state=42))])

pipe_lr_pca = Pipeline([('scl', StandardScaler()),
                    ('pca', PCA(n_components=2)),
                    ('clf', LogisticRegression(random_state=42))])

pipe_rf = Pipeline([('scl', StandardScaler()),
                    ('clf', RandomForestClassifier(random_state=42))])

```

```
pipe_rf_pca = Pipeline([('scl', StandardScaler()),
                        ('pca', PCA(n_components=2)),
                        ('clf', RandomForestClassifier(random_state=42))])

pipe_svm = Pipeline([('scl', StandardScaler()),
                     ('clf', svm.SVC(random_state=42))])

pipe_svm_pca = Pipeline([('scl', StandardScaler()),
                         ('pca', PCA(n_components=2)),
                         ('clf', svm.SVC(random_state=42))])

# Set grid search params
param_range = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
param_range_fl = [1.0, 0.5, 0.1]

grid_params_lr = [{'clf__penalty': ['l1', 'l2'],
                   'clf__C': param_range_fl,
                   'clf__solver': ['liblinear']}]

grid_params_rf = [{'clf__criterion': ['gini', 'entropy'],
                   'clf__min_samples_leaf': param_range,
                   'clf__max_depth': param_range,
                   'clf__min_samples_split': param_range[1:]}]

grid_params_svm = [{'clf__kernel': ['linear', 'rbf'],
                    'clf__C': param_range}]

# Construct grid searches
jobs = -1

gs_lr = GridSearchCV(estimator=pipe_lr,
                     param_grid=grid_params_lr,
                     scoring='accuracy',
                     cv=10)

gs_lr_pca = GridSearchCV(estimator=pipe_lr_pca,
                         param_grid=grid_params_lr,
                         scoring='accuracy',
                         cv=10)

gs_rf = GridSearchCV(estimator=pipe_rf,
                     param_grid=grid_params_rf,
                     scoring='accuracy',
                     cv=10,
                     n_jobs=jobs)

gs_rf_pca = GridSearchCV(estimator=pipe_rf_pca,
                         param_grid=grid_params_rf,
                         scoring='accuracy',
                         cv=10,
                         n_jobs=jobs)

gs_svm = GridSearchCV(estimator=pipe_svm,
                      param_grid=grid_params_svm,
                      scoring='accuracy',
                      cv=10,
                      n_jobs=jobs)
```

```
gs_svm_pca = GridSearchCV(estimator=pipe_svm_pca,
                          param_grid=grid_params_svm,
                          scoring='accuracy',
                          cv=10,
                          n_jobs=jobs)

# List of pipelines for ease of iteration
grids = [gs_lr, gs_lr_pca, gs_rf, gs_rf_pca, gs_svm, gs_svm_pca]

# Dictionary of pipelines and classifier types for ease of reference
grid_dict = {0: 'Logistic Regression', 1: 'Logistic Regression w/PCA',
                2: 'Random Forest', 3: 'Random Forest w/PCA',
                4: 'Support Vector Machine', 5: 'Support Vector Machine w/PCA'}

# Fit the grid search objects
print('Performing model optimizations...')
best_acc = 0.0
best_clf = 0
best_gs = ''
for idx, gs in enumerate(grids):
        print('\nEstimator: %s' % grid_dict[idx])
        # Fit grid search
        gs.fit(X_train, y_train)
        # Best params
        print('Best params: %s' % gs.best_params_)
        # Best training data accuracy
        print('Best training accuracy: %.3f' % gs.best_score_)
        # Predict on test data with best params
        y_pred = gs.predict(X_test)
        # Test data accuracy of model with best params
        print('Test set accuracy score for best params: %.3f ' % accuracy_score(y_test, y_pred))
        # Track best (highest test accuracy) model
        if accuracy_score(y_test, y_pred) > best_acc:
                best_acc = accuracy_score(y_test, y_pred)
                best_gs = gs
                best_clf = idx
print('\nClassifier with best test set accuracy: %s' % grid_dict[best_clf])

# Save best grid search pipeline to file
dump_file = 'best_gs_pipeline.pkl'
joblib.dump(best_gs, dump_file, compress=1)
print('\nSaved %s grid search pipeline to file: %s' % (grid_dict[best_clf], dump_file))
```


Note that there is a lot of opportunity for refactoring here. For example, each pipeline is defined explicitly, whereas a simple function could be used as a generator instead; the same goes for grid search objects. The longer form, again, hopefully allows for some better apples to apples comparisons in our next post.

Let's try it out.

```
$ python3 pipelines-3.py
```

And here's the output:

```
Performing model optimizations...

Estimator: Logistic Regression
Best params: {'clf__penalty': 'l1', 'clf__C': 1.0, 'clf__solver': 'liblinear'}
Best training accuracy: 0.917
Test set accuracy score for best params: 0.967

Estimator: Logistic Regression w/PCA
Best params: {'clf__penalty': 'l1', 'clf__C': 0.5, 'clf__solver': 'liblinear'}
Best training accuracy: 0.858
Test set accuracy score for best params: 0.933

Estimator: Random Forest
Best params: {'clf__criterion': 'gini', 'clf__min_samples_split': 2, 'clf__max_depth': 3, 'clf__min_samples_leaf': 2}
Best training accuracy: 0.942
Test set accuracy score for best params: 1.000

Estimator: Random Forest w/PCA
Best params: {'clf__criterion': 'entropy', 'clf__min_samples_split': 3, 'clf__max_depth': 5, 'clf__min_samples_leaf': 1}
Best training accuracy: 0.917
Test set accuracy score for best params: 0.900

Estimator: Support Vector Machine
Best params: {'clf__kernel': 'linear', 'clf__C': 3}
Best training accuracy: 0.967
Test set accuracy score for best params: 0.967

Estimator: Support Vector Machine w/PCA
Best params: {'clf__kernel': 'rbf', 'clf__C': 4}
Best training accuracy: 0.925
Test set accuracy score for best params: 0.900

Classifier with best test set accuracy: Random Forest

Saved Random Forest grid search pipeline to file: best_gs_pipeline.pkl
```

Note, importantly, that after we fit our estimators, we then tested each resulting model with best parameters of each of the 6 grid searches on our test dataset. This is not something we did last post, though we were comparing different models to one another, but given the introduction to other concepts the otherwise crucial step of comparing different models on previously unseen test data was overlooked until now. And our example proves why this step is necessary.

Shown above, the model which performed the "best" on our training data (highest training accuracy) was the support vector machine (without PCA), with the linear kernel and C value of 3 (controlling the amount of regularization), which learned how to accurately classify 96.7% of training instances. **However**, the model which performed best on the test data (the 20% of our dataset previously unseen to all models until after they were trained) was the random forest (without PCA), using the Gini criterion, minimum samples split of 2, max depth of 3, and minimum samples per leaf of 2, which managed to accurately classify 100% of the unseen data instances. Note that this model had a lower training accuracy of 94.2%.

So, beyond seeing how we can mix and match a variety of different estimator types, grid search parameter combinations, and data transformations, as well as how to accurately compare the trained models, it should be apparent that evaluating different models should always include testing them on previously unseen holdout data.

Sick of pipelines yet? Next time we'll look at an alternative approach to automating their construction.

Related:

- Managing Machine Learning Workflows with Scikit-learn Pipelines Part 1: A Gentle Introduction
- Managing Machine Learning Workflows with Scikit-learn Pipelines Part 2: Integrating Grid Search
- Using Genetic Algorithm for Optimizing Recurrent Neural Networks

---

Previous post
Next post

Top Stories Past 30 Days

Most Popular

1. 9 Must-have skills you need to become a Data Scientist, updated
2. The cold start problem: how to build your machine learning portfolio
3. Why You Shouldn't be a Data Science Generalist
4. How to go from Zero to Employment in Data Science
5. The Essence of Machine Learning
6. The Five Best Data Visualization Libraries
7. 10 More Must-See Free Courses for Machine Learning and Data Science

Most Shared

1. The Essence of Machine Learning
2. 10 More Must-See Free Courses for Machine Learning and Data Science
3. Introduction to Statistics for Data Science
4. Papers with Code: A Fantastic GitHub Resource for Machine Learning
5. A Guide to Decision Trees for Machine Learning and Data Science
6. Top Python Libraries in 2018 in Data Science, Deep Learning, Machine Learning
7. Top 10 Books on NLP and Text Analysis

Latest News

- Advance Your Career in Analytics
- 2019 CDO Virtual Event: Empowering Business Through Data
- The Data Science Gold Rush: Top Jobs in Data Science an...
- Data Science Project Flow for Startups
- Top tweets, Jan 16-22: How to build an API for a mach...
- How To Fine Tune Your Machine Learning Models To Improv...

Top Stories
Last Week

Most Popular

1. NEW **How to go from Zero to Employment in Data Science**

2. NEW **Ontology and Data Science**
3. ☁ **9 Must-have skills you need to become a Data Scientist, updated**
4. NEW **Data Scientists Dilemma: The Cold Start Problem - Ten Machine Learning Examples**
5. NEW **End To End Guide For Machine Learning Projects**
6. NEW **Math for Programmers**
7. NEW **The 6 Most Useful Machine Learning Projects of 2018**

Most Shared

1. **How to build an API for a machine learning model in 5 minutes using Flask**

2. **End To End Guide For Machine Learning Projects**
3. **How to solve 90% of NLP problems: a step-by-step guide**
4. **Data Scientists Dilemma: The Cold Start Problem Ten Machine Learning Examples**
5. **How to go from Zero to Employment in Data Science**
6. **Math for Programmers**
7. **The 6 Most Useful Machine Learning Projects of 2018**

Top Stories
Last Week

Most Popular

1. NEW **How to go from Zero to Employment in Data Science**

2. NEW **Ontology and Data Science**
3. **9 Must-have skills you need to become a Data Scientist, updated**
4. NEW **Data Scientists Dilemma: The Cold Start Problem - Ten Machine Learning Examples**
5. NEW **End To End Guide For Machine Learning Projects**
6. NEW **Math for Programmers**
7. NEW **The 6 Most Useful Machine Learning Projects of 2018**

Most Shared

1. **How to build an API for a machine learning model in 5 minutes using Flask**

2. **End To End Guide For Machine Learning Projects**
3. **How to solve 90% of NLP problems: a step-by-step guide**
4. **Data Scientists Dilemma: The Cold Start Problem Ten Machine Learning Examples**
5. **How to go from Zero to Employment in Data Science**
6. **Math for Programmers**
7. **The 6 Most Useful Machine Learning Projects of 2018**

X