Reference guide: Validation and cross-validation

Earlier in this course, you learned that using your model to predict on data that wasn't used to train the model is an important part of the model development process known as validation. You learned about validation using a separate holdout dataset; you also learned about cross-validation. Model validation is one of the most important parts of predictive modeling, a process that any data professional must understand. This reading is meant to serve as a reference guide—a collection of useful tools and processes and tips on how to use them to perform model validation.

A note about validation and hyperparameter tuning

It's important to remember that even though validation and hyperparameter tuning are closely related, they are two separate things. It's possible to perform model validation without tuning hyperparameters, and it's also possible to tune hyperparameters without performing validation. Most often, however, both steps are undertaken during the model development process.

Import statements

The following are some of the most commonly used tools related to validation and cross-validation using scikit-learn.

```
from sklearn.model_selection import train_test_split
```

- This function is used to split data. It can be used as many times as needed to achieve the desired sets. For example, you could split the dataset 80/20 (train/test), then use the function again on the train set, splitting it 75/25 (train/validate). This would result in a final ratio of 60/20/20 (train, validate, test).

```
from sklearn.model_selection import GridSearchCV
```

- `GridSearchCV` is a class. You use it to create a GridSearch object. When you use the `fit()` method on the GridSearch object, it partitions the data into a user-specified

number of folds, fits a model to the non-holdout data (all folds except one), and evaluates it against the holdout fold. Scores on each fold and a mean final score are captured for inspection.

- This is a very useful tool used during cross-validation and can also be used to tune hyperparameters with a single holdout validation set.

```
from sklearn.model_selection import PredefinedSplit
```

- `PredefinedSplit` is a class that allows you to specify which rows of a dataset to hold out as validation data. Among other things, it's useful for tuning hyperparameters using a single holdout validation set.

Cross-validate/tune hyperparameters with `GridSearchCV`

Strengths:

- Provides a rigorous estimation of model performance
- More thorough than tuning hyperparameters with a separate holdout dataset
- Good for maximizing utility of limited amounts of data

Weaknesses:

- More time consuming and computationally expensive than tuning against a holdout validation set

Here are the steps to cross-validate using GridSearchCV. Note that you can cross-validate without tuning hyperparameters. In that case, instead of indicating multiple values of each hyperparameter to search over (in step 2 below), just enter the single value that you want to use for each hyperparameter.

1. Instantiate the model (set the `random_state` parameter if you want reproducible results).
2. Create a dictionary of hyperparameters to search over.
3. Create a set of scoring metrics to capture.
4. Instantiate the `GridSearchCV` object. Pass as arguments:

- `estimator =` the model from step 1
- `param_grid =` the dictionary of hyperparameters to search over from step 2
- `scoring =` the set of scoring metrics you want to capture
- `cv =` the number of cross-validation folds you want to use
- `refit =` The scoring metric that you want `GridSearchCV` to use when it selects the "best" model (i.e., the model that performs best on average over all validation folds). When it's done, `GridSearchCV` will refit the best-scoring model to all of the data you give it in the step below.

5. Fit the `GridSearchCV` object to the data `(X, y)`

Example:

```
rf = RandomForestClassifier(random_state=0)
cv_params = {'max_depth': [2,3,4,5, None],
             'min_samples_leaf': [1,2,3],
             'min_samples_split': [2,3,4],
             'max_features': [2,3,4],
             'n_estimators': [75, 100, 125, 150]
             }
scoring = {'accuracy', 'precision', 'recall', 'f1'}

rf_cv = GridSearchCV(estimator=rf, param_grid=cv_params, scoring=scoring, cv=5,
refit='f1')

rf_cv.fit(X_train, y_train)
```

Use `GridSearchCV` and `PredefinedSplit` to tune hyperparameters on a separate validation set

Strengths:

- Faster and less computationally expensive than a multi-fold (k-fold) cross-validation

- Allows you to choose exactly which samples to include in the validation set (for example, suppose one of your features is “year,” and you want to ensure that an equal number of samples from each year are represented in the validation set.

Weaknesses:

- Less rigorous than a k-fold cross-validation
- Not as efficient with data usage (works best with very large datasets)

If you want to tune a model’s hyperparameters using a separate validation set, one way to do so is by designating which rows of your training data you want to use as your validation set. Here is one way of doing it:

1. Use `train_test_split` to separate your data into training and testing data.
   Example:
   ```
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                       stratify=y, random_state=42)
   ```

2. Use `train_test_split` again to separate your training data into training and validation data.
   Example:
   ```
   X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size=0.2,
                                               stratify=y_train, random_state=42)
   ```

3. Use a list comprehension to make a list of length `len(X_train)` where each element is either a 0 or -1. A zero in index *i* will indicate to `GridSearchCV` that index *i* of `X_train` is to be held out for validation. A -1 at a given index will indicate that that index of `X_train` is to be used as training data.

   The list comprehension looks at the index number of each row in `X_train`. If that index number is in the validation set’s indices, then the list comprehension appends a 0. If it’s not, then it appends a -1. If the training data is:
   [A, B, C, D],

and your list is:

[-1, 0, 0, -1],

then your training set will contain [A, D] and your validation set will contain [B, C].

Example:

```
split_index = [0 if x in X_val.index else -1 for x in X_train.index]
```

4. Pass this list as a parameter to `PredefinedSplit` and assign the result to a variable.

   Example:

   ```
   custom_split = PredefinedSplit(split_index)
   ```

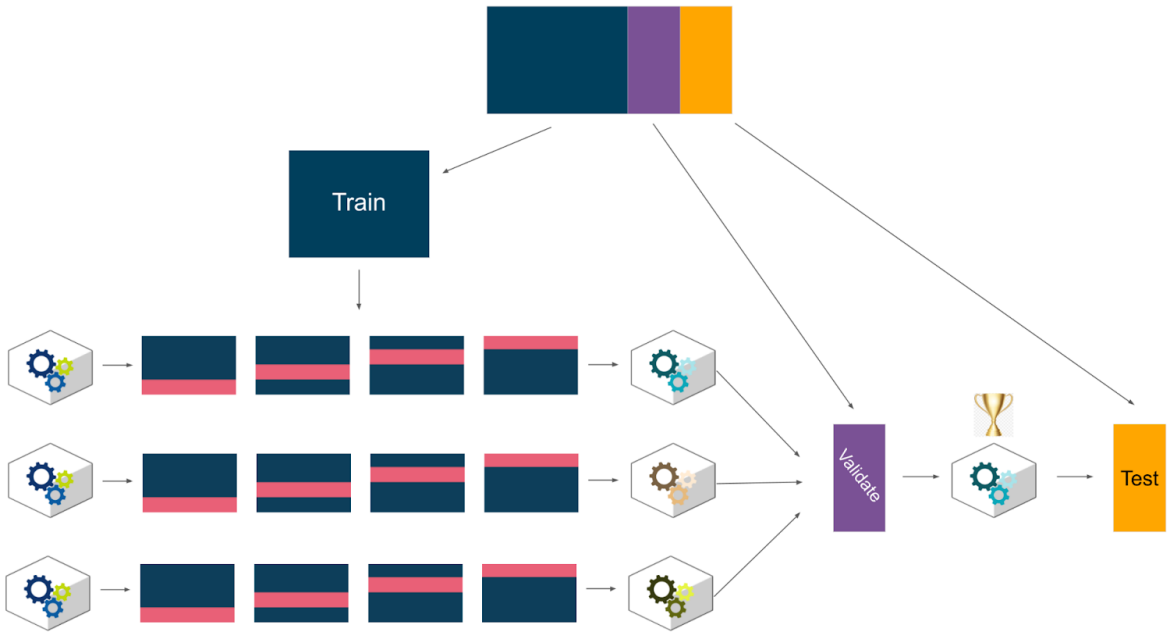
5. Designate this variable as the `cv` parameter when you instantiate your `GridSearchCV` object.

   Example:

   ```
   grid_search = GridSearchCV(estimator=rf, param_grid=cv_params,
                              scoring=scoring, cv=custom_split, refit='f1')
   ```

Selection of a champion model with a separate validation set

"Validation" can also refer to the process of choosing a champion model. Note that this is a related but (usually) distinct concept from hyperparameter tuning. In most cases, if you're tuning hyperparameters of different model architectures (e.g., logistic regression, decision tree, and random forest) and then selecting one of these architectures as a champion, it's worthwhile to perform cross-validation first to tune, *and* validation later with a separate validation set to select the champion model. The cross-validation is performed using the training data, and it's done to tune the hyperparameters of a particular model architecture . The data held out for validation is then used to compare the tuned model of each different architecture to get an objective comparison of their performance. It ensures that the model you choose as the champion model indeed generalizes well and does not simply overfit the training data.



After validating

When performing hyperparameter tuning, GridSearchCV will automatically refit the model with the best hyperparameters on all of the training data. However, if you have a holdout validation set to compare different model architectures, once you've selected a champion model, go back and train it on the training data + validation data together. Then use that model (and no other) to predict on the test data to get a measure of future performance. If you then deploy the model, you might want to finally retrain it using the full dataset (train + validate + test) so it can learn from as much data as possible before being deployed.

Key takeaways

There are different ways of validating machine learning models, and each way can be executed using different workflows, techniques, functions, and coding approaches. The methods demonstrated in this course are just some of them. What's important is that you understand that validation is performed to help prevent overfitting models to the training data and to provide a meaningful way of comparing different models to one another. It's also

important to understand the strengths and weaknesses of different approaches so you're better equipped to make these decisions yourself. Be inquisitive and try different approaches on your own!

Resources for more information

More detailed information about random forest tuning can be found here:

- scikit-learn documentation:
  - scikit-learn cross-validation documentation
  - developers.google.com - Validation Sets