Cross Validation: Takeaways

Syntax

- Implementing holdout validation:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
train_one = split_one
test_one = split_two
train_two = split_two
test_two = split_one
model = KNeighborsRegressor()
model.fit(train_one[["accommodates"]], train_one["price"])
test_one["predicted_price"] = model.predict(test_one[["accommodates"]])
iteration_one_rmse = mean_squared_error(test_one["price"], test_one["predicted_price"])**(1/2)
model.fit(train_two[["accommodates"]], train_two["price"])
test_two["predicted_price"] = model.predict(test_two[["accommodates"]])
iteration_two_rmse = mean_squared_error(test_two["price"], test_two["predicted_price"])**(1/2)
avg_rmse = np.mean([iteration_two_rmse, iteration_one_rmse])
```

- Implementing k-fold cross validation:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
model = KNeighborsRegressor()
train_iteration_one = dc_listings[dc_listings["fold"] != 1]
test_iteration_one = dc_listings[dc_listings["fold"] == 1].copy()
model.fit(train_iteration_one[["accommodates"]], train_iteration_one["price"])
labels = model.predict(test_iteration_one[["accommodates"]])
test_iteration_one["predicted_price"] = labels
iteration_one_mse = mean_squared_error(test_iteration_one["price"],
test_iteration_one["predicted_price"])
iteration_one_rmse = iteration_one_mse ** (1/2)
```

- Instantiating an instance of the KFold class from sklearn.model_selection:

```
from sklearn.model_selection import cross_val_score, KFold
kf = KFold(5, shuffle=True, random_state=1)
```

- Implementing cross_val_score along with the KFold class:

```
from sklearn.model_selection import cross_val_score
model = KNeighborsRegressor()
mses = cross_val_score(model, dc_listings[["accommodates"]], dc_listings["price"],
scoring="neg_mean_squared_error", cv=kf)
```

Concepts

- Holdout validation is a more robust technique for testing a machine learning model's accuracy on new data the model wasn't trained on. Holdout validation involves:
  - Splitting the full data set into two partitions:
    - A training set.
    - A test set.
  - Training the model on the training set.
  - Using the trained model to predict labels on the test set.
  - Computing an error to understand the model's effectiveness.
  - Switching the training and test sets and repeat.
  - Averaging the errors.

- In holdout validation, we use a 50/50 split instead of the 75/25 split from train/test validation to eliminate any sort of bias towards a specific subset of data.
- Holdout validation is a specific example of k-fold cross-validation, which takes advantage of a larger proportion of the data during training while still rotating through different subsets of the data, when `k` is set to two.
- K-fold cross-validation includes:
    - Splitting the full data set into `k` equal length partitions:
        - Selecting `k-1` partitions as the training set.
        - Selecting the remaining partition as the test set.
    - Training the model on the training set.
    - Using the trained model to predict labels on the test fold.
    - Computing the test fold's error metric.
    - Repeating all of the above steps `k-1` times, until each partition has been used as the test set for an iteration.
    - Calculating the mean of the `k` error values.
- The parameters for the KFold class are:
    - `n_splits` : The number of folds you want to use.
    - `shuffle` : Toggle shuffling of the ordering of the observations in the data set.
    - `random_state` : Specify the random seed value if `shuffle` is set to `True` .
- The parameters for using `cross_val_score` are:
    - `estimator` : Scikit-learn model that implements the `fit` method (e.g. instance of KNeighborsRegressor).
    - `X` : The list or 2D array containing the features you want to train on.
    - `y` : A list containing the values you want to predict (target column).
    - `scoring` : A string describing the scoring criteria.
    - `cv` : The number of folds. Here are some examples of accepted values:
        - An instance of the `KFold` class.
        - An integer representing the number of folds.
- The workflow for k-fold cross-validation with scikit-learn includes:
    - Instantiating the scikit-learn model class you want to fit.
    - Instantiating the KFold class and using the parameters to specify the k-fold cross-validation attributes you want.
    - Using the cross_val_score() function to return the scoring metric you're interested in.
- Bias describes error that results in bad assumptions about the learning algorithm. Variance describes error that occurs because of the variability of a model's predicted

Variance describes error that occurs because of the variability of a model's predicted value. In an ideal world, we want low bias and low variance when creating machine learning models.

Resources

- Accepted values for scoring criteria
- Bias-variance Trade-off
- K-Fold cross-validation documentation