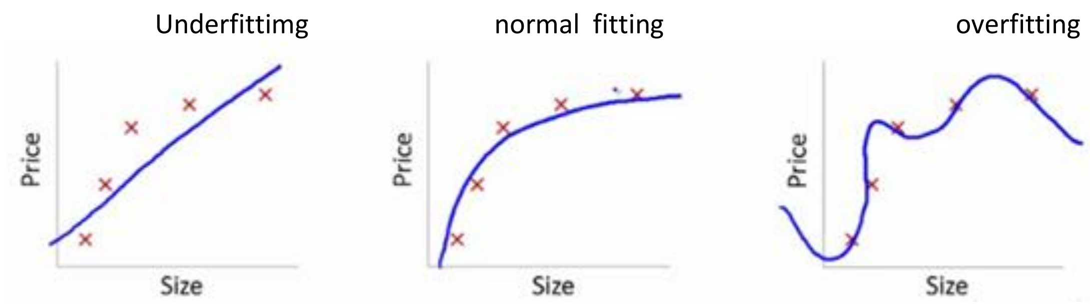
LESSON 17 CrossValidation Last update May 15, 2021

CROSS VALIDATION

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The limited sample is used in order to estimate how the model is expected to perform in general, when used to make predictions on data that is not used during the training of the model.

Cross-validation is mainly used as a way to check for under or over-fit.
In the following plots we have underfitting, fitting and overfitting.



The first plot represents a linear relationship and has a high error from training data point and is an example of underfitting. The model fails to capture the underlining trend of data.

The second plots represents normal fitting has a low training error and a generalization of the relationship between price and size. The curve follows the points very tightly.

In the third plot we have almost zero training error, but is too sensitive and captures random patterns that are only in the present training set but not present in other datasets. This is an example overfitting and there could be a high deviation between training sets and actual data.

A common practice in data science competitions is to iterate over various models to find a better performing model. However, it becomes difficult to distinguish whether this improvement in score is coming because we are capturing the relationship better, or we are just over-fitting the data.

To find the right answer for this question, we use **validation** techniques. This method helps us in achieving more generalized relationships.

To **avoid over-fitting**, we have to define two different sets:

training set (X_train, y_train)

which is used for learning the parameters of a predictive model.

testing set **(** X_test, y_test)

which is used for evaluating the fitted predictive model.

We can now quickly sample a training set while holding out a percentage of the data for testing (evaluating) our classifier:

However, by defining these two sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, test) sets.

A solution is to **split the whole data several consecutive times in different train set and test set**, and to return the averaged value of the prediction scores obtained with the different sets. Such a procedure is called **cross-validation**. This approach can be **computationally expensive, but does not waste too much data** (as it is the case when fixing an arbitrary test set), which is a major advantage in problem such as inverse inference where the number of samples is very small.

k-fold cross-validation

The procedure has a single parameter called **k** that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups

3. For each unique group:
   3.1Take the group as a hold out or test data set
   3.2Take the remaining groups as a training data set
   3.3Fit a model on the training set and evaluate it on the test set|
   3.4Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining k − 1 folds.

The results of a k-fold cross-validation run are often summarized with the mean of the model skill scores. It is also good practice to include a measure of the variance of the skill scores, such as the standard deviation or standard error.

Configuration of k

The k value must be chosen carefully for your data sample. A poorly chosen value for k may result in a mis-representative idea of the skill of the model, such as a score with a high variance (that may change a lot based on the data used to fit the model), or a high bias, (such as an overestimate of the skill of the model).

Three common tactics for choosing a value for k are as follows:

- **Representative**: The value for k is chosen such that each train/test group of data samples is large enough to be statistically representative of the broader dataset

- **k=10**: The value for k is fixed to 10, a value that has been found through experimentation to generally result in a model skill estimate with low bias a modest variance.

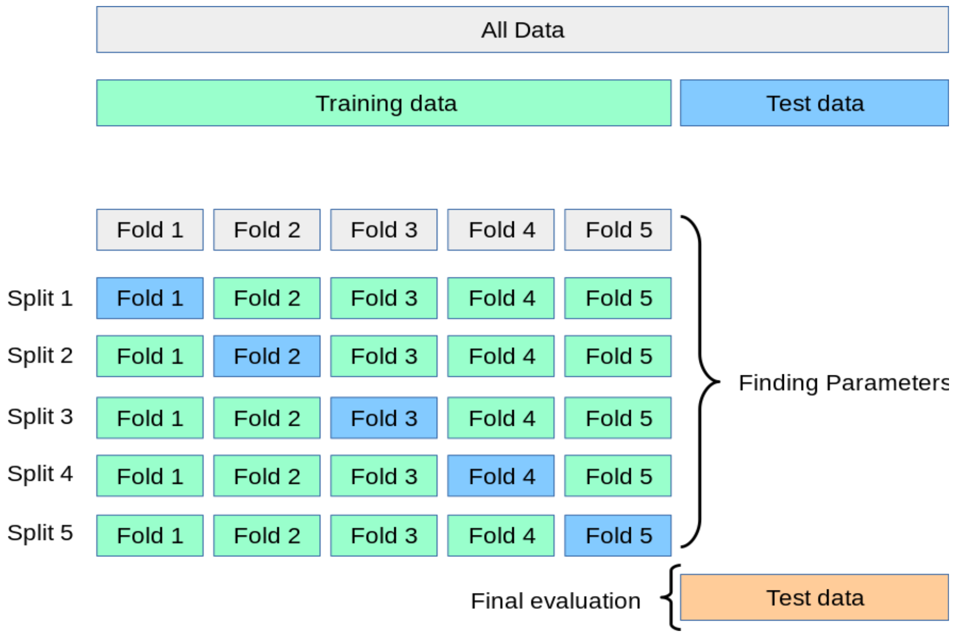
- **k=n**: The value for k is fixed to n, where n is the size of the dataset to give each test sample an opportunity to be used in the hold out dataset. This approach is called leave-one-out cross-validation.

The choice of k is usually 5 or 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller

A value of k=10 is very common in the field of applied machine learning, and is recommend if you are struggling to choose a value for your dataset.

To summarize, there is a bias-variance trade-off associated with the choice of k in k-fold cross-validation. Typically, given these considerations, one performs k-fold cross-validation using k = 5 or k = 10, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance.

If a value for k is chosen that does not evenly split the data sample, then one group will contain a remainder of the examples. It is preferable to split the data sample into k groups with the same number of samples, such that the sample of model skill scores are all equivalent



Evaluating a Machine Learning model can be quite tricky. Usually, we split the data set into training and testing sets and use the training set to train the model and testing set to test the model. We then evaluate the model performance based on an error metric to determine the accuracy of the model. This method however, is not very reliable as the accuracy obtained for one test set can be very different to the accuracy obtained for a different test set.

The **K-fold Cross Validation(CV)** provides a solution to this problem by dividing the data into folds and ensuring that each fold is used as a testing set at some point. K-Fold CV is where a given data set is split into a ***K*** number of sections/folds where each fold is used as a testing set at some point. Lets take the scenario of 5-Fold cross validation(K=5). Here, the data set is split into 5 folds. In the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 5 folds have been used as the testing set.

kFold using sklearn

Sklearn has the K-Folds cross-validator **model** located in **sklearn.model_selection.** It provides train/test indices to split data in train/test sets, splits dataset into k consecutive folds (without shuffling by default). Each fold is then used once as a validation while the k - 1 remaining folds form the training set. Sklearn also has the **cross_val_score** and **cross_val_predict** modules located in sklearn.model_selection. **cross_val_score** return the scores of the kFold training sets where as **cross_val_predict** return the prediction of the kFold training sets. Both receive a classifier model and X and Y data points.

KFold test program using sklearn

We first make the necessary imports:

```
from sklearn.model_selection import KFold
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
```

We will use a logistic regression model because it is very good in classifying binary data . We use the sklearn **make_classification function** to make our X and y dataset**.** We make 100 samples with 5 features and 1 binary target .We then use the KFold function to calculate the different K Fold training sets.

```
# make 100 training samples with 5 features
X, y = make_classification(n_samples=100, n_features=5)

# store scores
scores = []
predictions = []

# make LinearRegression model
model = LogisticRegression()

# make KFold with 10 splits
cv = KFold(n_splits=10, random_state=42, shuffle=True)
```

For each folded training set, we print out the indexes, get the training data from the indexes then use the classifier to calculate the score and predictions. We store the score and predictions in a list.

```
# for each training set produced
for train_index, test_index in cv.split(X):

    # print train indexes
    print("Train Index:\n", train_index, "\n")

    # print test indexes
    print("Test Index:\n", test_index)

    # get the data from the indexes
    X_train, X_test, y_train, y_test = X[train_index], X[test_index], y[train_index], y[test_index]

    # use the model to classify the data
    model.fit(X_train, y_train)

    # calculate score accuracy using the model
    score = model.score(X_test, y_test)
    # print score
    print("score: ",score)
```

# store score
scores.append(score)

 # calculate predictions using model
prediction = model.predict(X_test)

# print predictions
print("predistion: ",prediction)

# store predictions
predictions.append(prediction)

# print scores
print("KFold scores:\n",scores)

# print mean and std of scores
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))

# print predictions
print("kFold predictions:\n",predictions)

```
KFold scores:

 [0.7, 1.0, 1.0, 0.9, 0.9, 1.0, 0.9, 1.0, 0.7, 0.9]

Accuracy: 0.900 (0.110)
```

# print mean and std of predictions
print('Accuracy: %.3f (%.3f)' % (np.mean(predictions), np.std(predictions)))

```
KFold predictions:

 [array([0, 0, 0, 1, 0, 1, 0, 0, 1, 1]),
  array([0, 0, 0, 1, 1, 1, 1, 0, 1, 1]),
  array([0, 1, 0, 1, 0, 1, 1, 1, 0, 1]),
  array([1, 0, 0, 0, 1, 1, 0, 1, 1, 1]),
  array([0, 0, 1, 0, 0, 1, 1, 1, 1, 0]),
  array([1, 1, 1, 1, 0, 1, 1, 0, 1, 0]),
  array([1, 0, 1, 0, 0, 1, 0, 1, 0, 1]),
  array([0, 1, 0, 1, 0, 1, 1, 0, 1, 0]),
  array([1, 1, 0, 0, 0, 0, 0, 0, 0, 0]),
  array([0, 0, 0, 0, 0, 1, 0, 0, 0, 1])]

Accuracy: 0.480 (0.500)
```

Using sklearn cross_val_score module

We now use the sklearn score module to calculate and print the scores.

```
from sklearn.model_selection import cross_val_score

# using cross val score
print("using cross val score")

# use cross validation to calculate score using model and dataset
scores = cross_val_score(model, X, y, cv=10)

# print out scores
print("Cross Val scores:\n",scores)

# report performance
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
using cross val score
Cross Val scores:
 [1.  1.  0.8 1.  0.7 0.9 0.9 0.8 0.9 1. ]
Accuracy: 0.900 (0.100)
```

The results are comparable to the classifier kfold results.

Using sklearn cross_val_predict module

We now use the sklearn predict module to calculate and print the predictions

```
from sklearn.model_selection import cross_val_predict

# using cross val predict
print("using cross val score")

# using cross val predict
predictions = cross_val_predict(model, X, y, cv=10)

# print predictions
print("Cross Val predictions:\n",predictions)
```

# report performance
print('Accuracy: %.3f (%.3f)' % (np.mean(predictions), np.std(predictions)))

```
using cross val predictions
Cross Val predictions:
 [0 1 1 1 0 1 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0
  0 0 0 1 1 0 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 0 1 1 0 0 0
  0 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 1
  1 0 1 0 1 0 0 0 0 1 0 1 1 1 0 1 0 0 1]
Accuracy: 0.500 (0.500)
```

The results are comparable to the classifier kfold results.

Todo:

Try different values of k, and try to get better accuracy.

CROSS VALIDATION HOMEWORK Question 1

Run the kfold program with out cross validation and compare the results to the cross validation results. Name your python file crossvalidation_homework.py

You should get something like this:

score: 0.76
prediction: [1 0 0 0 1 0 1 1 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 0 1]
Accuracy: 0.760 (0.000)
Accuracy: 0.520 (0.500)

End