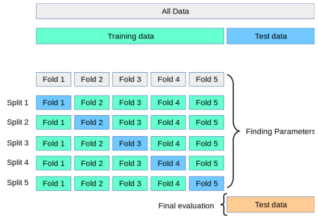
K-Fold Cross Validation – Python Example

August 15, 2020 by Ajitesh Kumar · Leave a comment



In this post, you will learn about **K-fold Cross Validation** concepts with Python code example. It is important to learn the concepts **cross validation** concepts in order to perform **model tuning** with an end goal to choose model which has the **high generalization performance**. As a data scientist / machine learning Engineer, you must have a good understanding of the cross validation concepts in general.

The following topics get covered in this post:

- What and why of K-fold cross validation
- When to select what values of K?
- K-fold cross validation with python (using cross-validation generators)
- K-fold cross validation with python (using cross_val_score)

Table of Contents

What and Why of K-fold Cross Validation

K-fold cross validation is a technique used for **hyperparameters tuning** such that the model with most optimal value of hyperparameters can be trained. It is a **resampling technique without replacement**. The **advantage** of this approach is that each example is used for training and validation (as part of a test fold) exactly once. This yields a **lower-variance estimate of the model performance** than the holdout method. The following is done in this technique:

1. The dataset is split into training and test dataset.
2. The training dataset is then split into K-folds.
3. Out of the K-folds, (K-1) fold is used for training
4. 1 fold is used for validation
5. The model with specific hyperparameters is trained with training data (K-1 folds) and validation data as 1 fold. The performance of the model is recorded.
6. The above steps (step 3, step 4 and step 5) is repeated until each of the k-fold got used for validation purpose. This is why it is called k-fold cross validation.
7. Finally, the mean and standard deviation of the model performance is computed by taking all of the model scores calculated in step 5 for each of the K models.
8. Step 3 to Step 7 is repeated for different values of hyperparameters.
9. Finally, the hyperparameters which result in most optimal mean and standard value of model scores get selected.

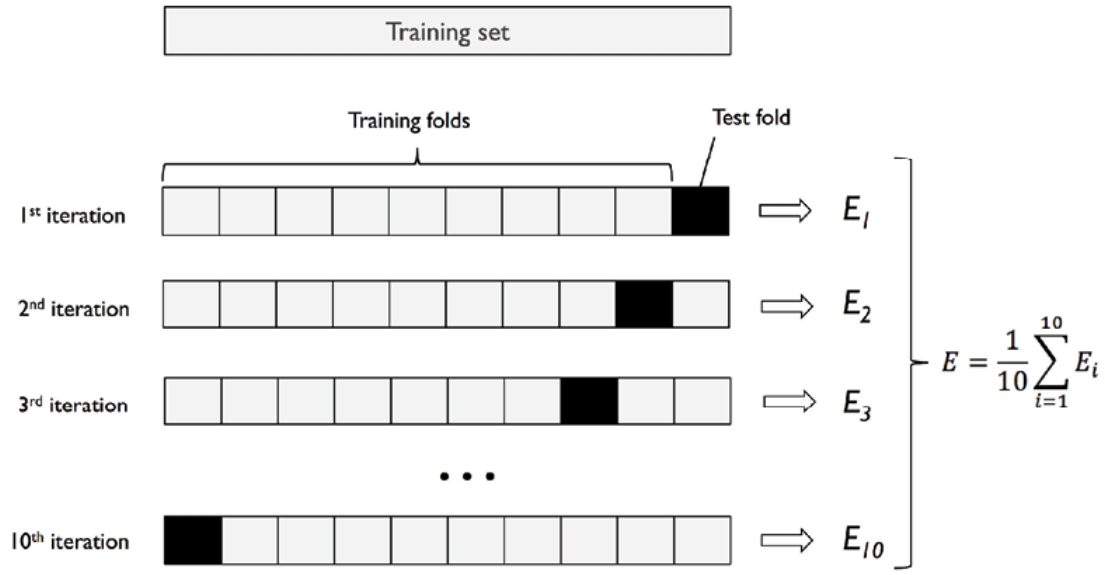


Fig 1. Compute the mean score of model performance of model trained using K-folds

Why use Cross-validation technique?

The conventional technique for training and testing the model is to split the data in two different splits which are termed as training and test split. For a decent size of data, the training and test split is taken as 70:30. Here are few challenges due to which cross-validation technique is used:

- In order to train the model of optimal performance, the hyperparameters are tweaked appropriately to achieve good model performance with the test data. However, this technique results in the risk of overfitting *on the test set*. This is because the parameters can be tweaked until the estimator performs optimally. This way, knowledge about the test set can "leak" into the model and evaluation metrics no longer report on generalization performance.
- In order to take care of above issue, there are three splits which get created. They are training, validation and test split. The model hyperparameters get tuned using training and validation set. And, finally, the model generalization performance is determined using test data split. However, this technique also has the shortcomings. By partitioning the data into three sets, the number of samples which can be used for learning the model gets reduced. The results depend on a particular random choice for the pair of (train, validation) sets.

To overcome above challenges, the cross-validation technique is used. As described earlier in this section, two different splits such as training and test split get created. However, cross-validation is applied on the training data by creating K-folds of training data in which (K-1) fold is used for training remaining fold is used for testing. This process is repeated for K times and the model performance is calculated for a particular set of hyperparameters by taking mean and standard deviation of all the K models created. The hyperparameters giving the most optimal model is calculated. Finally, the model is trained again on the training data set using the most optimal hyperparameter and the generalization performance is computed by calculating model performance on the test dataset. The diagram given below represents the same.

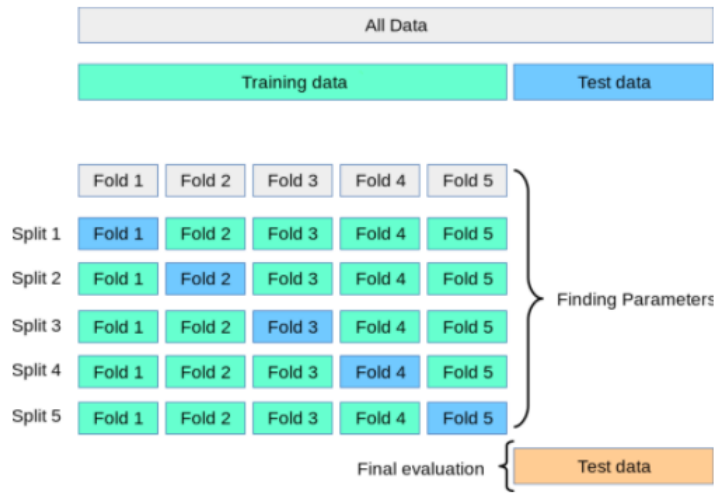


Fig 2. Why K-Fold Cross Validation

- For a very large data set, one can use the value of K as 5 (K-5). One can obtain an accurate estimate of the average performance of the model while reducing the computational cost of refitting and evaluating the model on the different folds.
- The number of folds increases if the data is relatively small. However, larger values of *k* results in the increase of the runtime of the cross-validation algorithm. This yield model performance estimates with higher variance, since the training folds become smaller.
- For very small data sets, **leave-one-out cross-validation** (**LOOCV**) technique is used. In this technique, the validation data consists of just one record.

It is recommended to use stratified k-fold cross-validation in order to achieve better bias and variance estimates, especially in cases of unequal class proportions.

K-fold Cross-Validation with Python (using Cross-Validation Generators)

In this section, you will learn about how to use cross-validation generators such as some of the following to compute the cross validation scores. The cross-validator generators given below returns the indices of training and test splits. These indices can be used to create training and test splits and train different models. Later, the mean and standard deviation of model performance of different models is computed to assess the effectiveness of hyperparameter values and further tune them appropriately.

- KFold
- StratifiedKFold
- GroupKFold
- ShuffleSplit
- StratifiedShuffleSplit
- GroupShuffleSplit
- LeaveOneGroupOut
- LeavePGroupsOut
- LeaveOneOut
- LeavePOut
- PredefinedSplit

Here is the Python code which illustrates usage of the class StratifiedKFold (sklearn.model_selection) for creating training and test splits. The code can be found on this Kaggle page, K-fold cross-validation example. Pay attention to some of the following in the Python code given below:

- Instance of StratifiedKFold is created by passing number of folds (n_splits=10)
- Split method is invoked on the instance of StratifiedKFold to gather the indices of training and test splits for those many folds
- Training and test data is passed to the instance of pipeline.
- Scores of different models get calculated.
- Finally, mean and standard deviation of model scores is computed.

```
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
#
# Create an instance of Pipeline
#
pipeline = make_pipeline(StandardScaler(), RandomForestClassifier(n_estimators=100, max_depth=4))
#
# Create an instance of StratifiedKFold which can be used to get indices of different training and test fo
#
strtfdKFold = StratifiedKFold(n_splits=10)
kfold = strtfdKFold.split(X_train, y_train)
scores = []
#
#
#
for k, (train, test) in enumerate(kfold):
    pipeline.fit(X_train.iloc[train, :], y_train.iloc[train])
    score = pipeline.score(X_train.iloc[test, :], y_train.iloc[test])
    scores.append(score)
    print('Fold: %2d, Training/Test Split Distribution: %s, Accuracy: %.3f' % (k+1, np.bincount(y_train.il

print('\n\nCross-Validation accuracy: %.3f +/- %.3f' %(np.mean(scores), np.std(scores)))
```

Here is how the output from above code execution would look like:

[94]:
```
for k, (train, test) in enumerate(kfold):
    pipeline.fit(X_train.iloc[train, :], y_train.iloc[train])
    score = pipeline.score(X_train.iloc[test, :], y_train.iloc[test])
    scores.append(score)
    print('Fold: %2d, Training/Test Split Distribution: %s, Accuracy: %.3f'
          % (k+1, np.bincount(y_train.iloc[train]), score))

print('\n\nCross-Validation accuracy: %.3f +/- %.3f' %(np.mean(scores), np.std(scores)))
```

```
Fold:  1, Training/Test Split Distribution: [615  96], Accuracy: 0.863
Fold:  2, Training/Test Split Distribution: [615  97], Accuracy: 0.861
Fold:  3, Training/Test Split Distribution: [615  97], Accuracy: 0.873
Fold:  4, Training/Test Split Distribution: [615  97], Accuracy: 0.873
Fold:  5, Training/Test Split Distribution: [616  96], Accuracy: 0.861
Fold:  6, Training/Test Split Distribution: [616  96], Accuracy: 0.861
Fold:  7, Training/Test Split Distribution: [616  96], Accuracy: 0.861
Fold:  8, Training/Test Split Distribution: [616  96], Accuracy: 0.861
Fold:  9, Training/Test Split Distribution: [616  96], Accuracy: 0.861
Fold: 10, Training/Test Split Distribution: [616  96], Accuracy: 0.861


Cross-Validation accuracy: 0.863 +/- 0.005
```

Fig 3. Cross-validation Scores using StratifiedKFold Cross-validator generator

K-fold Cross-Validation with Python (using Sklearn.cross_val_score)

Here is the Python code which can be used to apply cross validation technique for model tuning (hyperparameter tuning). The code can be found on this Kaggle page, K-fold cross-validation example. Pay attention to some of the following in the code given below:

- **cross_val_score** class of sklearn.model_selection module is used for computing the cross validation scores. This is one of the simplest way to It computes the scores by splitting the data repeatedly into a training and a testing set, trains the estimator using the training set and computes t scores based on the testing set for each iteration of cross-validation. The input to the cross_val_score includes an estimator (having fit and pre method), the cross-validation object and the input dataset.
- The **input estimator** to the **cross_val_score** can be either an estimator or a pipeline (sklearn.pipeline).
- One other input to the cross_val_score is **cross validation object** which is assigned to the parameter, **cv.** The parameter, cv, can take one of following values:
  - An integer that represents the number of folds in a StratifiedKFold cross validator.
  - If cv is not specified, 5-fold cross-validation is applied.
  - An instance of **cross-validation splitter** which can be one of the following:
    - Cross-validation generators such as some of the following:
    - Cross-validation estimators which represent An estimator that has built-in cross-validation capabilities to automatically select the best hyper-parameters. The following are some of the examples:
      - LogisticRegressionCV
      - ElasticNetCV

```
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
#
# Create an instance of Pipeline
#
pipeline = make_pipeline(StandardScaler(), RandomForestClassifier(n_estimators=100, max_depth=4))
#
# Pass instance of pipeline and training and test data set
# cv=10 represents the StratifiedKFold with 10 folds
#
scores = cross_val_score(pipeline, X=X_train, y=y_train, cv=10, n_jobs=1)

print('Cross Validation accuracy scores: %s' % scores)

print('Cross Validation accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

Here is how the output would look like as a result of execution of above code:

[55]:
```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(pipeline, X=X_train, y=y_train, cv=10, n_jobs=1)
```

```
print('CV accuracy scores: %s' % scores)
```

```
CV accuracy scores: [0.8625     0.86075949 0.87341772 0.87341772 0.86075949 0.86075949
 0.86075949 0.86075949 0.86075949 0.86075949]
```

[57]:
```
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

```
CV accuracy: 0.863 +/- 0.005
```

Fig 4. Sklearn.model_selection method cross_val_score used for K-fold cross validation

Conclusions

Here is the summary of what you learned in this post about **k-fold cross validation**:

- K-fold cross validation is used for **model tuning / hyperparameters tuning**.
- K-fold cross validation involves split the data into training and test data sets, applying K-fold cross-validation on training data set and selecting the model with most optimal performance
- There are several **cross validation generators** such as KFold, StratifiedKFold which can be used for this technique.
- Sklearn.model_selection module's **cross_val_score** helper class can be used for applying K-fold cross validation in simple manner.
- Use LOOCV method for very small data sets.
- For very large data sets, one can use the value of K as 5.
- The value of K = 10 is standard value of K.
- **It is recommended to use stratified k-fold cross-validation** in order to achieve better bias and variance estimates, especially in cases o unequal class proportions.

Author Recent Posts

Ajitesh Kumar

I have been recently working in the area of Data Science and Machine Learning / Deep Learning. In addition, I am also passionate about various different technologies including programming languages such as Java/JEE, Javascript, Python, R, Julia etc and technologies such as Blockchain, mobile computing, cloud-native technologies, application security, cloud computing platforms, big data etc. I would love to connect with you on Linkedin.

Follow me

Posted in Data Science, Machine Learning, Python. Tagged with Data Science, machine learning, python, sklearn.

← Sklearn Machine Learning Pipeline – Python Example

Logistic Regression Quiz Questions & Answers →

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Time limit is exhausted. Please reload the CAPTCHA.

IX + =

Post Comment

- Search

- Recent Posts
  - What is Machine Learning? Concepts & Examples
  - NLP Pre-trained Models Explained with Examples
  - Data-Driven Decision Making: What, Why & How
  - Covid-19 Machine Learning Use Cases
  - Federated Analytics & Learning Explained with Examples

(26) freshers (14) google (14) google glass (11) hyperledger (18) Interview questions (71) Java (92) javascript (103) Kubernetes (19) machine learning (294) mongodb (16) news (13) nlp (12) nosql (17) python (88) QA (12) quantum computing (12) reactjs (15) r programming (11) sklearn (30) Software Quality (11) spring framework (16) statistics (15) testing (16) tools (11) tutorials (14) UI (13) Unit Testing (18) web (16)

About Us

Vitalflux.com is dedicated to help software engineers & data scientists get technology news, practice tests, tutorials in order to reskill / acquire newer skills from time-to-time.

Thank you for visiting our site today. We welcome all your suggestions in order to make our website better. Please feel free to share your thoughts.

Latest Technologies

Machine Learning / Data Science
Quantum Computing
Robotics
Blockchain
Cloud Computing
Web