14-Cross_Validation

October 20, 2024

1 Cross Validation

Cross-Validation is a technique used in machine learning to assess the performance of a model by testing it on different subsets of the data. Instead of using just a single training and testing split, cross-validation helps evaluate the model's ability to generalize to unseen data by dividing the dataset into multiple parts (or folds) and training/testing the model on each part.

The most commonly used form of cross-validation is k-fold cross-validation, where the dataset is divided into k subsets or "folds." The model is trained on k−1 folds and tested on the remaining fold. This process is repeated k times, each time using a different fold as the test set and the rest as the training set. The final performance is averaged over all folds.

Cross-validation is important because it gives a better estimate of the model's performance by reducing the risk of overfitting or underfitting the data.

When to Use Cross-Validation? Cross-Validation should be used when:

- **You want to evaluate model performance**: It's used to estimate how well a model generalizes to an independent dataset.
- **You're working with limited data**: Cross-validation helps make the most out of the available data by using all of it for both training and testing across different iterations.
- **You're tuning hyperparameters**: It's often employed when trying to find the optimal hyperparameters of a model (e.g., in grid search).
- **To prevent overfitting**: When you want to avoid overfitting on a particular training set by providing a more robust evaluation of model performance.
- **In model selection**: It is used to compare different models and choose the best-performing one based on a more reliable performance metric.

Use cases include:

- **Machine learning competitions**: Such as Kaggle competitions, where high generalization performance is crucial.
- **Scientific research**: Where precise model evaluation is necessary to make reliable predictions.
- **Production-ready systems**: To validate that the model will work well in the real world, where unseen data is used.

1.0.1 Common Types of Cross-Validation:

1. **k-Fold Cross-Validation:**
   - The most commonly used form, as described above. The number of folds, k, is usually 5 or 10.
   - Advantages:
     - Provides a better estimation of model performance.
     - Each instance is used for both training and validation.
   - Disadvantages:
     - More computationally expensive than simple train-test split, as the model is trained and evaluated k times.

2. **Leave-One-Out Cross-Validation (LOOCV):**
   - A special case of k-fold where k equals the number of data points. That means for each iteration, only one instance is left out for testing, and the rest is used for training.
   - Advantages:
     - No data point is wasted.
     - Provides an unbiased estimate of the generalization error.
   - Disadvantages:
     - Computationally expensive for large datasets.
     - High variance, as training on almost the entire dataset may lead to overfitting.

3. **Stratified k-Fold Cross-Validation:**
   - A variation of k-fold cross-validation where the data is split such that the proportion of classes in each fold is the same as the original dataset. This is especially useful in cases of imbalanced datasets.
   - Advantages:
     - Better representation of the minority and majority classes in each fold.
     - Reduces bias in datasets with imbalanced class distributions.
   - Disadvantages:
     - Still computationally intensive, similar to k-fold cross-validation.

4. **Time Series Cross-Validation (Rolling or Expanding Window):**
   - In time series data, data points are not independent, so traditional k-fold cross-validation may not be suitable. Instead, the training and test splits are done based on time order.
   - Advantages:
     - Maintains the temporal order of data, which is important for time series problems.

- Disadvantages:
  - More complex to implement and evaluate.

1.0.2 Who Should Use Cross-Validation?

Cross-validation is suitable for:

- **Data scientists and machine learning practitioners** looking for a robust model evaluation technique.
- **Machine learning beginners** who want to prevent overfitting or underfitting and need to compare different models and tune hyperparameters.
- **Researchers and engineers** working with limited data, where splitting the data into a simple train-test set might not be enough.
- **Kaggle competitors** or those involved in machine learning competitions where cross-validation provides a better estimate of performance on unseen data.

Advantages of Cross-Validation:

- **More reliable evaluation**: By using all the data for training and testing in different iterations, cross-validation gives a better estimate of model performance.
- **Reduces overfitting risk**: Prevents the model from overfitting to a particular train-test split.
- **Maximizes data usage**: Every data point is used for both training and validation, ensuring the model is evaluated on the entire dataset.
- **Model comparison**: Helps compare the performance of different models using a fair and unbiased technique.
- **Hyperparameter tuning**: Aids in finding the best hyperparameters for a model by testing performance across multiple folds.

Disadvantages of Cross-Validation:

- **Computationally expensive**: Training and testing the model multiple times (depending on k) can be time-consuming, especially for large datasets or complex models.
- **Bias-variance tradeoff**: While k-fold cross-validation provides a more reliable estimate, it can still have some variance if k is small. LOOCV reduces bias but may increase variance.

Real-World Applications of Cross-Validation:

- **Model Selection**: Comparing several machine learning algorithms (e.g., decision trees, logistic regression, SVM) and selecting the one with the best cross-validated performance.
- **Hyperparameter Tuning**: Tuning parameters like the depth of a decision tree, the learning rate of a neural network, or the C parameter in an SVM using cross-validation as part of a grid search or randomized search.
- **Time Series Prediction**: Using time-based cross-validation (rolling windows) to validate models for stock prices, weather forecasting, and other time-dependent data.

- **Medical Research**: Evaluating predictive models that classify diseases using limited patient dat

1.1 Example

[26]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import KFold, cross_val_score, StratifiedKFold
from sklearn.linear_model import LogisticRegression

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Initialize the model
model = LogisticRegression(max_iter=200)

# Define k-fold cross-validation with 5 folds
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Perform cross-validation and get scores
cv_scores = cross_val_score(model, X, y, cv=kf)

# Print cross-validated accuracy scores
print(f"Cross-validated scores: {cv_scores}")
print(f"Mean score: {cv_scores.mean()}")
```

```
Cross-validated scores: [1.         1.         0.93333333 0.96666667 0.96666667]
Mean score: 0.9733333333333334
```

[39]:
```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

np.random.seed(42)
```

[28]:
```
fastball_speed = np.random.randint(90,106, size=500)
tommy_john = np.where(fastball_speed > 96, np.random.choice([0,1], size=500,␣
 ↪p=[0.3, 0.7]), 0)

d = {
    'fastball_speed': fastball_speed,
    'tommy_john': tommy_john
}
```

```
df = pd.DataFrame(d)
df
```

```
[28]:      fastball_speed  tommy_john
      0                96           0
      1                93           0
      2               102           1
      3               104           1
      4               100           0
      ..              ...         ...
      495             104           1
      496              92           0
      497             101           1
      498              90           0
      499              93           0

      [500 rows x 2 columns]
```

```
[29]: X = df[['fastball_speed']]
      y = df[['tommy_john']]

      X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,␣
       ↪random_state=11)
```

```
[30]: lr = LogisticRegression()

      lr.fit(X_train, y_train.values.ravel())
```

```
[30]: LogisticRegression()
```

```
[31]: lr.score(X_test, y_test)
```

```
[31]: 0.71
```

1.2 Test again

```
[32]: X2_train, X2_test, y2_train, y2_test = train_test_split(X,y, test_size=0.2,␣
       ↪random_state=25)

      lr = LogisticRegression()
      lr.fit(X2_train, y2_train.values.ravel())
```

```
[32]: LogisticRegression()
```

```
[33]: lr.score(X2_test, y2_test)
```

```
[33]: 0.8
```

[34]:
```
cvscore = cross_val_score(lr, X, y.values.ravel(), cv=10)
cvscore
```

[34]:
```
array([0.68, 0.78, 0.72, 0.72, 0.78, 0.82, 0.74, 0.74, 0.76, 0.82])
```

[35]:
```
print(np.average(cvscore))
print(np.std(cvscore))
```

```
0.756
0.04270831300812523
```

[36]:
```
kf = KFold(n_splits=15, shuffle=True, random_state=42)

kfscore = cross_val_score(lr, X, y.values.ravel(), cv=kf, scoring='f1')
kfscore2 = cross_val_score(lr, X, y.values.ravel(), cv=kf, scoring='accuracy')

print(kfscore)
print(np.average(kfscore))
print(np.std(kfscore))

print(kfscore2)
print(np.average(kfscore2))
print(np.std(kfscore2))
```

```
[0.66666667 0.82758621 0.72727273 0.7027027  0.6875     0.72727273
 0.74074074 0.72       0.66666667 0.60869565 0.66666667 0.63157895
 0.64516129 0.66666667 0.6       ]
0.6856785107611354
0.05620337068328653
[0.76470588 0.85294118 0.82352941 0.67647059 0.70588235 0.81818182
 0.78787879 0.78787879 0.78787879 0.72727273 0.72727273 0.78787879
 0.66666667 0.78787879 0.63636364]
0.7559120617944146
0.060970963505883366
```

[37]:
```
kf3 = StratifiedKFold(n_splits=10, shuffle=True, random_state=11)

kfscore3 = cross_val_score(lr, X, y.values.ravel(), cv=kf3)

print(kfscore3)
print(np.average(kfscore3))
print(np.std(kfscore3))
```

```
[0.66 0.76 0.76 0.8  0.8  0.74 0.76 0.74 0.78 0.76]
0.756
0.03773592452822642
```

1.3 Pipeline Sample

[40]:
```
scales = StandardScaler()
pipe1 = make_pipeline(scales, lr)
pipe1.fit(X_train, y_train.values.ravel())
```

[40]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])
```

[42]:
```
scorespipe = cross_val_score(pipe1, X, y.values.ravel(), cv=10)

print(scorespipe)
print(np.average(scorespipe))
print(np.std(scorespipe))
```

```
[0.68 0.78 0.72 0.72 0.78 0.82 0.74 0.74 0.76 0.82]
0.756
0.04270831300812523
```