

[Abrir en la aplicación](#) ↗**Medium**

Buscar



Escribir



Validación cruzada

sobre pramod · [Seguir](#)

16 minutos de lectura · 21 de enero de 2023



825



La validación cruzada es una técnica que se utiliza para evaluar un modelo de aprendizaje automático y probar su rendimiento. La validación cruzada es una técnica que se utiliza para evaluar el rendimiento de un modelo de aprendizaje automático entrenándolo en diferentes subconjuntos de datos y probándolo en el subconjunto restante. Esto es algo diferente de la división general de entrenamiento y prueba.

Introducción:

La validación cruzada también se conoce como estimación de rotación o prueba fuera de la muestra. La estimación de rotación se refiere al proceso de rotar o dividir los datos en diferentes subconjuntos. En pocas palabras, en el proceso de validación cruzada, la muestra de datos original se divide aleatoriamente en varios subconjuntos. El modelo de aprendizaje automático se entrena en todos los subconjuntos, excepto uno. Después del entrenamiento, el modelo se prueba haciendo predicciones en el

subconjunto restante. El objetivo es estimar el rendimiento del modelo en datos no vistos, de manera similar a cómo funciona la validación cruzada.

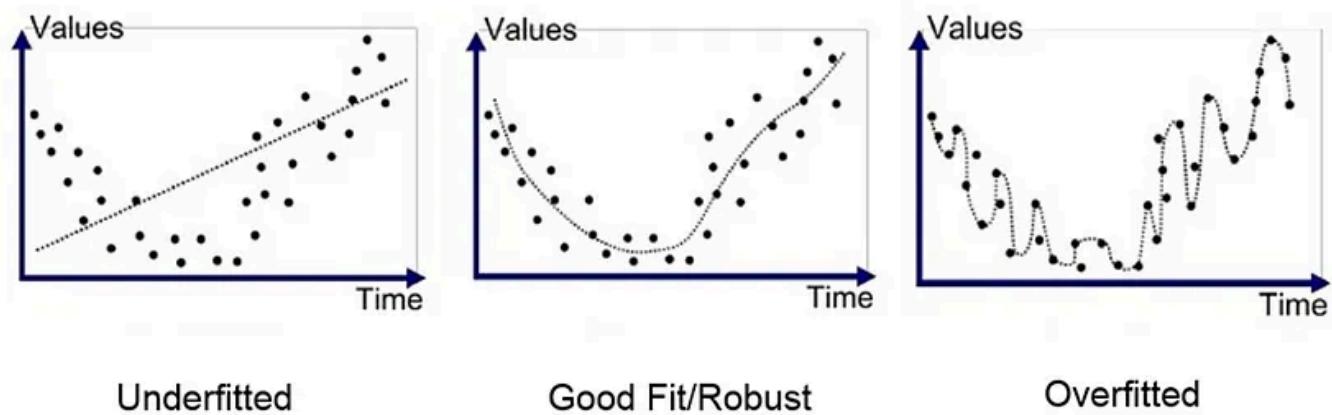
La prueba fuera de muestra es otro término que se utiliza para la validación cruzada, que se refiere al proceso de probar el rendimiento de un modelo con datos que no se utilizaron para entrenarlo. Esto es importante para determinar qué tan bien es probable que funcione el modelo con datos nuevos que no se han visto.

¿Por qué utilizar la validación cruzada?

Supongamos que crea un modelo de aprendizaje automático para resolver un problema y lo ha entrenado con un conjunto de datos determinado. Cuando comprueba la precisión del modelo con los datos de entrenamiento, se encuentra cerca del 95 %. ¿Significa esto que su modelo se ha entrenado muy bien y es el mejor modelo debido a su alta precisión?

No, no lo es. Debido a que su modelo se entrena con los datos proporcionados, los conoce bien, captó incluso las variaciones más pequeñas (ruido) y ha generalizado muy bien sobre los datos proporcionados. Si expone el modelo a datos completamente nuevos e inéditos, es posible que no realice predicciones con la misma precisión y que no pueda generalizar sobre los nuevos datos. Este problema se denomina sobreajuste.

A veces, el modelo no se entrena bien en el conjunto de entrenamiento porque no puede encontrar patrones. En este caso, tampoco funcionaría bien en el conjunto de prueba. Este problema se denomina "ajuste insuficiente".



[Referencia](#)

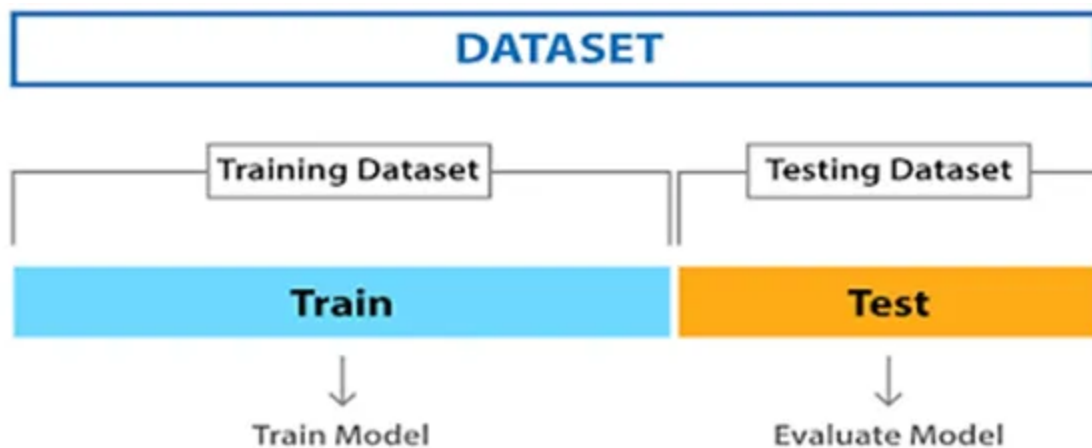
La validación cruzada es una forma de verificar si un modelo de aprendizaje automático está sobreajustado o subajustado a un conjunto de datos determinado.

Métodos utilizados para la validación cruzada:

Existen varios métodos que se utilizan para la validación cruzada, cada uno con sus propias ventajas y desventajas. Algunos de los métodos más comunes son:

Validación de retención:

El método de retención, también conocido como división de entrenamiento y prueba, es el tipo más simple de validación cruzada. Implica dividir aleatoriamente el conjunto de datos en dos subconjuntos: un conjunto de entrenamiento y un conjunto de retención (o conjunto de prueba). El modelo se entrena en el conjunto de entrenamiento y su rendimiento se evalúa en el conjunto de retención.



Una regla general común es utilizar aproximadamente el 70 % del conjunto de datos como conjunto de entrenamiento y el 30 % como conjunto de validación. A continuación, se muestra cómo puede implementar la validación cruzada de retención utilizando el conjunto de datos iris:

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = datasets.load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
```

This method is straightforward and easy to implement but can be less accurate than other cross-validation techniques primarily because it does not account for the variability of the data split.

1. The performance of the model evaluated through holdout cross-validation can vary significantly based on the specific split of the data. This variability arises because the data used for training and testing is not guaranteed to be representative of the entire dataset. This can lead to an overly optimistic or pessimistic evaluation of the model's performance.
2. Holdout cross-validation does not allow for a comprehensive assessment of model performance across different hyperparameters. Since the model is trained and tested only once, it's challenging to determine the optimal hyperparameters for the model.
3. Holdout cross-validation does not leverage the entire dataset for training and testing. This can be particularly problematic when dealing with small datasets or when the data is imbalanced. In such cases, a single split might not adequately represent the diversity of the data, leading to inaccurate performance evaluations.

Thus, Holdout cross-validation is a good starting point for model evaluation, but for more reliable performance estimates, other cross-validation techniques like k-fold cross-validation are recommended. However, it's worth noting that for datasets with a large number of observations, holdout cross-validation can be computationally efficient and may still provide a reasonable estimate of model performance.

1. K-fold cross validation:

K-fold cross validation is a method of evaluating the performance of a machine learning model by splitting the data into k subsets or "folds" of roughly equal size. The model is trained on k-1 of the folds and tested on the remaining fold. This process is repeated k times, with each fold being used

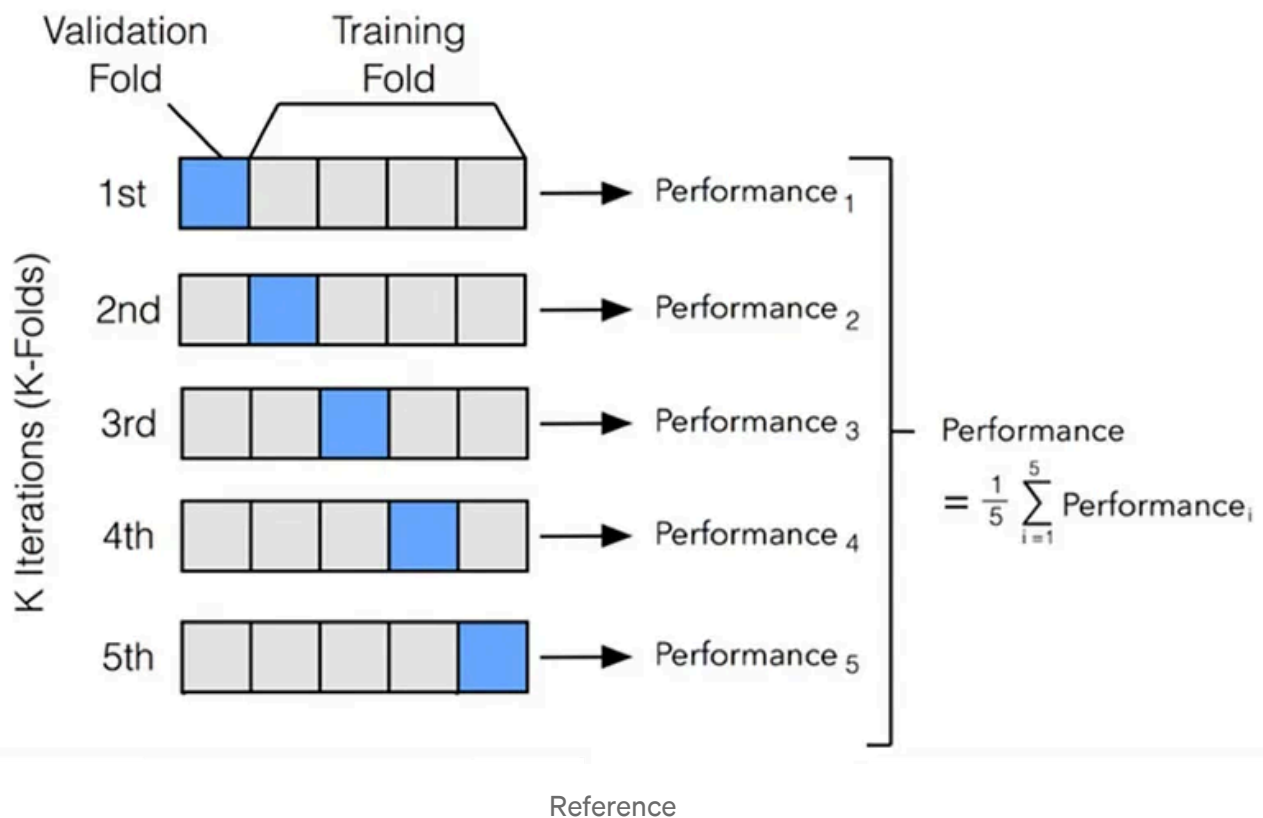
as the test set once. The performance of the model is then averaged across all k iterations. This gives an estimate of how well the model is likely to perform on new, unseen data.

Here's an example to explain it:

Suppose we have a dataset of 100 samples, and we want to train a model to predict if a person has diabetes based on their age and BMI. We can use k -fold cross validation to evaluate the performance of the model. We can split the data into 10 folds, meaning that each fold will have 10 samples.

In the first iteration, we will use the first 9 folds (90 samples) to train the model and the remaining fold (10 samples) to test the model. We will record the performance of the model (e.g. accuracy, precision, recall, etc.) In the second iteration, we will use the second fold as the test set, and the rest of the 9 folds as the training set. We will again record the performance of the model.

This process will repeat until all the 10 folds have been used as the test set once. After all the iterations, we will average the performance of the model across all 10 iterations. This will give us an estimate of how well the model is likely to perform on new, unseen data.



```

from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# create an instance of the model
model = LogisticRegression()

# create an instance of the KFold cross-validator
kf = KFold(n_splits=10)

# create lists to store the accuracy scores for each fold
accuracies = []

# iterate over the splits
for train_index, test_index in kf.split(X):
    # split the data into training and test sets
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # fit the model on the training data
    model.fit(X_train, y_train)

    # make predictions on the test data

```

```
y_pred = model.predict(X_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# append the accuracy score to the list
accuracies.append(accuracy)

# calculate the average accuracy across all folds
average_accuracy = sum(accuracies) / len(accuracies)

# print the average accuracy
print("Average accuracy:", average_accuracy)
```

In this example, we create an instance of the KFold cross-validator and specify the number of folds to be used for cross-validation by passing the number of splits to the `n_splits` parameter. In this example, we are using 10 folds. We use a for loop to iterate over the splits generated by the KFold cross-validator. The `kf.split(X)` returns the indices of the training and test sets for each fold. The `train_index` variable contains the indices of the training samples and the `test_index` variable contains the indices of the test samples. In each iteration of the loop, we use the `train_index` and `test_index` to split the data into training and test sets.

`sklearn.model_selection` has a method `cross_val_score` which simplifies the process of cross-validation. Instead of iterating through the complete data using the 'split' function, we can use `cross_val_score` and check the accuracy score for the chosen cross-validation method

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold, cross_val_score

X, y = datasets.load_iris(return_X_y=True)
```



```
clf = DecisionTreeClassifier(random_state=42)

k_folds = KFold(n_splits = 5)

scores = cross_val_score(clf, X, y, cv = k_folds)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

output may look like this:

```
Cross Validation Scores: [1.          1.          0.83333333 0.93333333 0.8
Average CV Score: 0.9133333333333333
Number of CV Scores used in Average: 5
```

In this case, an average score of approximately 0.91 suggests a strong performance. We used 5 folds for cross-validation, so we have 5 individual scores.

2. *Stratified k-fold cross validation*

Stratified k-fold cross validation is a method of cross-validation that ensures that the proportion of samples for each class is roughly the same in each fold. This is useful when the class distribution is imbalanced, meaning that there are different number of samples for each class. let's take an example to illustrate how Stratified k-fold cross validation works.

Suppose we have a dataset of 1000 samples, and we want to train a model to classify the samples as either "A" or "B" class. The dataset has 600 samples of

class “A” and 400 samples of class “B”. We want to use k-fold cross validation to evaluate the performance of the model, where $k = 5$.

In regular k-fold cross validation, the data is randomly split into 5 subsets or “folds” of roughly equal size (200 samples per fold). However, this could lead to a situation where all samples of class “A” end up in one fold and all samples of class “B” end up in another fold. This would not be an accurate representation of the model’s performance on unseen data.

With stratified k-fold cross validation, we first divide the data into 5 folds based on the class distribution. This means that each fold will have roughly 120 samples of class “A” and 80 samples of class “B”. This way, the proportion of samples for each class is roughly the same in each fold.

The model is then trained on 4 folds and tested on the remaining fold, this process is repeated 5 times with each fold being used as the test set once. The performance of the model is then averaged across all 5 iterations. This gives an estimate of how well the model is likely to perform on new, unseen data.

Note: The main difference between Stratified k-fold cross validation and regular k-fold cross-validation is how the data is split into folds.

In regular k-fold cross-validation, the data is randomly split into k subsets or “folds” of roughly equal size. This means that each fold will have roughly the same number of samples, regardless of their class labels.

In contrast, in stratified k-fold cross-validation, the data is first divided into k folds based on the class distribution. This means that each fold will have roughly the same proportion of samples for each class. This is useful when

the class distribution is imbalanced, meaning that there are different number of samples for each class.

The main advantage of stratified k-fold cross-validation is that it helps to prevent bias in the evaluation process.

Stratified K-fold Cross Validation (K = 5)



Reference

Here is an implementation of Stratified K-Fold using sklearn-

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_score

X, y = datasets.load_iris(return_X_y=True)
```

```
clf = DecisionTreeClassifier(random_state=42)

sk_folds = StratifiedKFold(n_splits = 5)

scores = cross_val_score(clf, X, y, cv = sk_folds)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

the output may look like-

```
Cross Validation Scores: [0.96666667 0.96666667 0.9          0.93333333 1.
Average CV Score: 0.9533333333333334
Number of CV Scores used in Average: 5
```

In this case, we have 5 individual scores. This corresponds to the 5 folds created by the StratifiedKFold cross-validation iterator.

3. *Leave-One-Out (LOO):*

Leave-One-Out (LOO) cross-validation is a method of cross-validation where we use $n-1$ samples of the dataset as the training set and 1 sample as the test set, where n is the total number of samples in the dataset. This process is repeated n times, each time leaving out a different sample as the test set. This method is an exhaustive technique.

For example, let's say we have a dataset of 100 samples and we want to train a model to predict if a person has diabetes based on their age and BMI.

In Leave-One-Out cross-validation, we would:

- Select 1 sample as the test set and use the remaining 99 samples as the training set. Train the model on the training set and evaluate its performance on the test set.
- Select another sample as the test set and use the remaining 99 samples as the training set. Train the model again on the new training set and evaluate its performance on the new test set.
- Repeat this process 99 times, each time using a different sample as the test set.

The array of scores represents the accuracy scores for each iteration of cross-validation. Each value represents the accuracy achieved in a specific subset of the data. Each score indicates how well the model performed in each iteration. In each iteration, since there's only one test case, and a score of 1 indicates that the model correctly classified that single instance. Whereas, a score of 0 would indicate misclassification. We can observe that the number of cross validation scores performed is equal to the number of observations in the dataset. In this case there are 150 observations in the iris dataset. The average CV score is 94%. Each observation contributes one score, and the average is taken over all these scores.

4. *Leave-P-Out (LPO)*:

Leave-P-Out (LPO) cross-validation is a method of cross-validation similar to Leave-One-Out (LOO) but instead of leaving out one sample at a time, it leaves out p samples at a time. In other words, it uses $n-p$ samples of the dataset as the training set and p samples as the test set, where n is the total number of samples in the dataset. This process is repeated in all possible ways of choosing p samples from the dataset.

For example, let's say we have a dataset of 100 samples and we want to train a model to predict if a person has diabetes based on their age and BMI.

In Leave-2-Out cross-validation (LPO), we would:

- Select 2 samples as the test set and use the remaining 98 samples as the training set. Train the model on the training set and evaluate its performance on the test set.
- Select another 2 samples as the test set and use the remaining 98 samples as the training set. Train the model again on the new training set and evaluate its performance on the new test set.
- Repeat this process in all possible ways of choosing 2 samples from 100 samples.

The advantage of Leave-P-Out cross-validation is that it can be less computationally expensive than Leave-One-Out cross-validation when the dataset is large, as it leaves out p samples at a time.

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import LeavePOut, cross_val_score

X, y = datasets.load_iris(return_X_y=True)
clf = DecisionTreeClassifier(random_state=42)
lpo = LeavePOut(p=2)
scores = cross_val_score(clf, X, y, cv = lpo)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

Here each score of 1 indicates that the model correctly classified both instances in the respective test sets. during each iteration of Leave-One-Out cross-validation. A score of 0.5 suggests that only one instance is correctly classified, while the other is misclassified. A score of 0 would mean misclassification of both instances. The average of these cross-validation scores is approximately 93.83%. This average score provides an overall measure of how well the model generalizes to new data across all iterations. With a total of 150 instances in the dataset, there are 11175 unique combinations of two instances that can be selected for testing in Leave-Two-Out cross-validation. These combinations contribute to the 11175 cross-validation scores used in computing the average performance of the model.

5. Repeated k -Fold cross-validation:

Repeated k-Fold cross-validation or Repeated random sub-sampling CV is probably the most robust of all CV techniques.

Let's say you are building a machine learning model to predict if a person has diabetes or not based on their age, BMI, and blood pressure. You have a dataset of 100 samples, each sample has these three features and a label indicating whether the person has diabetes or not.

You want to check how well your model performs on this data. To do this, you decide to use Repeated k-fold cross-validation with $k=5$ (5-fold cross-validation) and repeat the process 10 times. If the data is split into 5 groups and the process is repeated 10 times, then in total 50 folds are formed.

The number of splits (k) represents the number of folds or groups that the data is divided into. In this case, the data is split into 5 groups, with 20 samples in each group. The number of repeats represents the number of times the k-fold process is repeated with different random splits of the data. In this case, the process is repeated 10 times. In each repetition of the k-fold process, one of the k-folds is used as the test set and the remaining $k-1$ folds are used as the training set. We repeat this process k -times, and each time a different fold is used as the test set.

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RepeatedKFold, cross_val_score

X, y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)

rkf = RepeatedKFold(n_splits=5, n_repeats=10, random_state=42)

scores = cross_val_score(clf, X, y, cv=rkf)
```

```
print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

The output of this code will look like this:

```
Cross Validation Scores: [0.96666667 1.          0.96666667 0.96666667 1.
 1.          0.96666667 1.          1.          0.96666667 0.96666667
0.9        1.          0.96666667 0.96666667 1.          0.96666667
1.          1.          0.96666667 1.          0.96666667 1.
1.          0.96666667 0.96666667 0.96666667 1.          0.96666667
0.96666667 0.96666667 1.          0.96666667 0.96666667 1.
0.96666667 0.96666667 1.          1.          0.96666667 0.96666667]
Average CV Score:  0.9733333333333334
Number of CV Scores used in Average:  50
```

Higher scores indicate better classification accuracy, with 1.0 representing perfect classification. There are 50 cross-validation scores used in calculating the average. We have set up the RepeatedKFold cross-validation with 5 splits and 10 repeats, resulting in a total of 50 folds.

These are just a few of the CV methods that can be applied to models. There are many more cross validation classes, with most models having their own class.

how to choose K in K fold cross validation:

Choosing the value of k in k-fold cross-validation is a trade-off between bias and variance. In general, a larger value of k will result in a lower bias and a

higher variance, while a smaller value of k will result in a higher bias and a lower variance.

Here are a few examples to illustrate how the choice of k can affect the performance of k -fold cross-validation:

- If k is set to be very small, say $k = 2$ or 3 , the model will have a higher bias and a lower variance. In this case, the model is trained on a small portion of the data and the test set is also small, so the model may not generalize well to new, unseen data.
- If k is set to be very large, say $k = n-1$ where n is the number of samples, the model will have a lower bias and a higher variance. In this case, the model is trained on almost all the data and the test set is very small, so the model may fit the training data very well but may not generalize well to new, unseen data.
- A common choice for k is 10 , which is widely used in practice. This value tends to work well in most cases and provides a good balance between bias and variance.

It is important to experiment with different values of k and evaluate the performance of the model using different evaluation metrics. Also, the choice of k is highly dependent on the dataset and the model, so it's good to try multiple values of k and choose the one that gives the best performance.

Reference-

Final Note: Thanks for reading! I hope you find this article informative.

Wanna connect with me? Hit me up on [LinkedIn](#)

Machine Learning

Deep Learning

K Fold Cross Validation




Written by om pramod

Follow

206 Followers

AIML enthusiast

More from om pramod


 om pramod

Decision Trees

Part 2: Information Gain

Jan 29, 2023  904  3




 om pramod

Decision Trees

Part 4: Gini Index

Jan 29, 2023  768  2




 om pramod

Decision Trees

Part 3: Gain Ratio

Jan 29, 2023  828  1



 om pramod

Autoencoders Explained

Part 5: Denoising Autoencoders


Jun 16  379



See all from om pramod

Recommended from Medium

 Juan C Olamendy

 Punyakeerthi BL

A Comprehensive Guide to Regularization in Machine Learning

Have you ever trained a machine learning model that performed exceptionally on your...

Apr 23



Understanding Feature Scaling in Machine Learning

Welcome back to another insightful exploration. Today, we explore the critical...

Jun 21



6



1



Lists

Predictive Modeling w/ Python

20 stories · 1595 saves



Practical Guides to Machine Learning

10 stories · 1938 saves

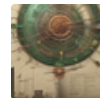
Natural Language Processing

1754 stories · 1351 saves



data science and AI

40 stories · 265 saves



Ibtissam Makdoun

Exploring Data Sampling Techniques in PySpark...

Sampling data is a fundamental aspect of data analysis, especially when dealing with...

Apr 28



Maxim Gusarov in CodeX

Do I need to tune logistic regression hyperparameters?

¿No estamos demasiado comprometidos con la optimización del trabajo de ciencia de...

9 de abril de 2022




140



3



 Prasanna de Chanaka

Explicación del informe de clasificación: precisión,...

¿Por qué necesitamos un informe de clasificación?

★ 21 de abril 🖱️ 111 💬 3



 Satria Suria

Evaluación del modelo: validación cruzada

La validación cruzada es un método estadístico que se utiliza para evaluar el...

18 de junio 🖱️ 1



Ver más recomendaciones