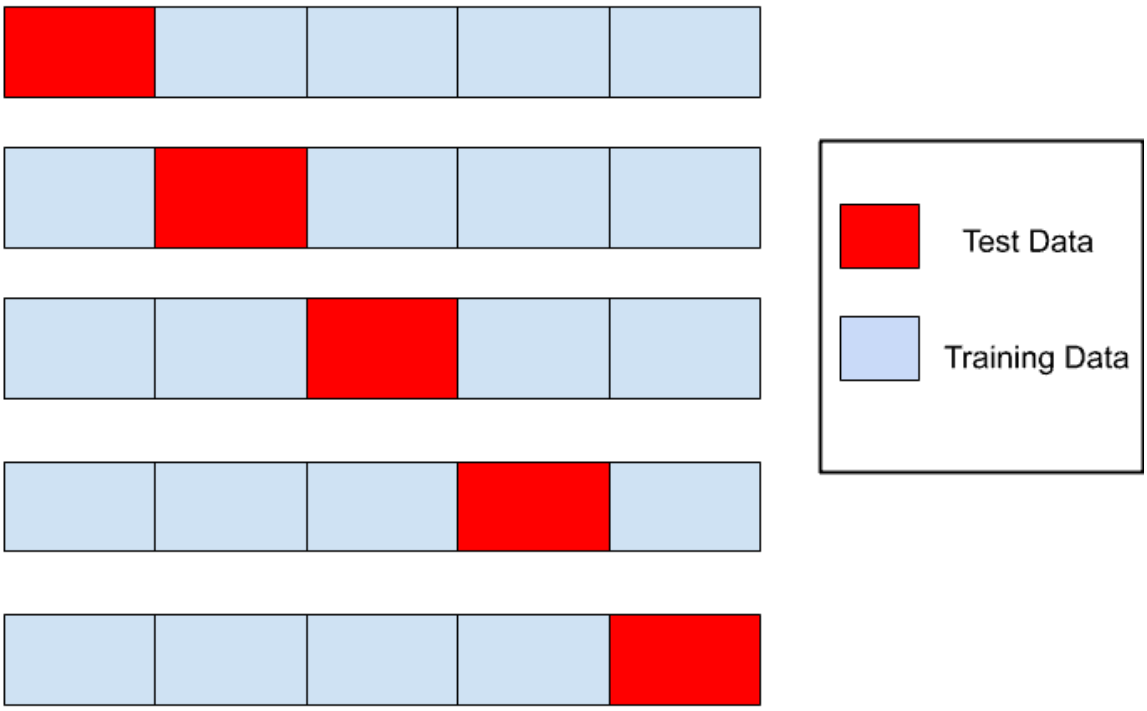
K Fold Cross-Validation in Machine

The **purpose of cross–validation** is to test the ability of a machine learning model to predict new data. It is also used to flag problems like overfitting or selection bias and gives insights on how the model will generalize to an independent dataset.

We get more matrix to take the decision, hekpful in parameter tunning.

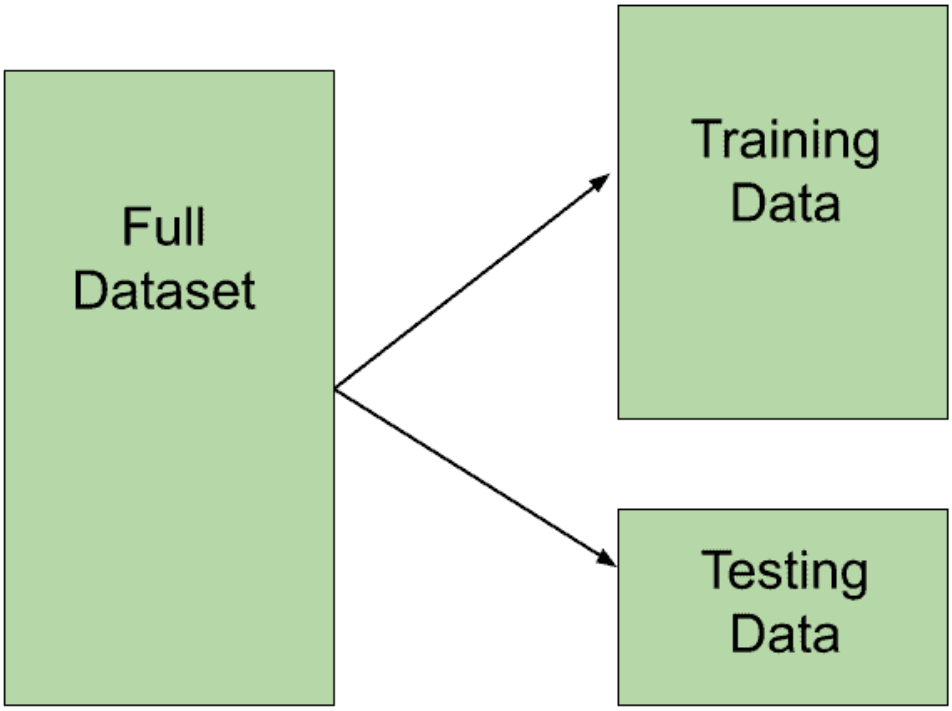


K Fold Cross-Validation

K Fold Cross Validation is used to solve the problem of **Train-Test Split.**

What is Train-Test Split Problem?

Train-Test split is nothing but splitting your data into two parts. Traning Data and Test Data.



Training Data is data that is used to train the model.

Test Data is used to test the performance of the model after the training phase.

Suppose we have 1000 records in our data. So, we can split data into 70% and 30% or 75% or 25% data. That means we require 70% data for training and 30% data for testing. It depends upon the number of data records.

So, we train our model with 70% data. And after Training, we test our model performance with 30% data. And check the accuracy.

Accuracy maybe 80%, 85%, and anything else.

The Data we have selected for training and testing is randomly selected.

Due to random selection, some kind of data that is present in Test data may not present in Training data. **So The accuracy will decrease.**

So, whenever we perform Train-Test split, we use a random_state variable. And then we define the value of random_state.

And based on this random_state variable, data is selected randomly.

So the problem is, Suppose, first we have chosen random_state =5. And Split the data in 70% and 30%. So the accuracy we got is around 80%.

But, again when we choose random_state=10. And Train-Test split at 70% and 30%. So the Training data and Testing Data gets shuffled. That means the data which we had in the previous round, is now shuffled. Now we have different data in Training Data and in Testing Data.

And now we got an accuracy of 85%.

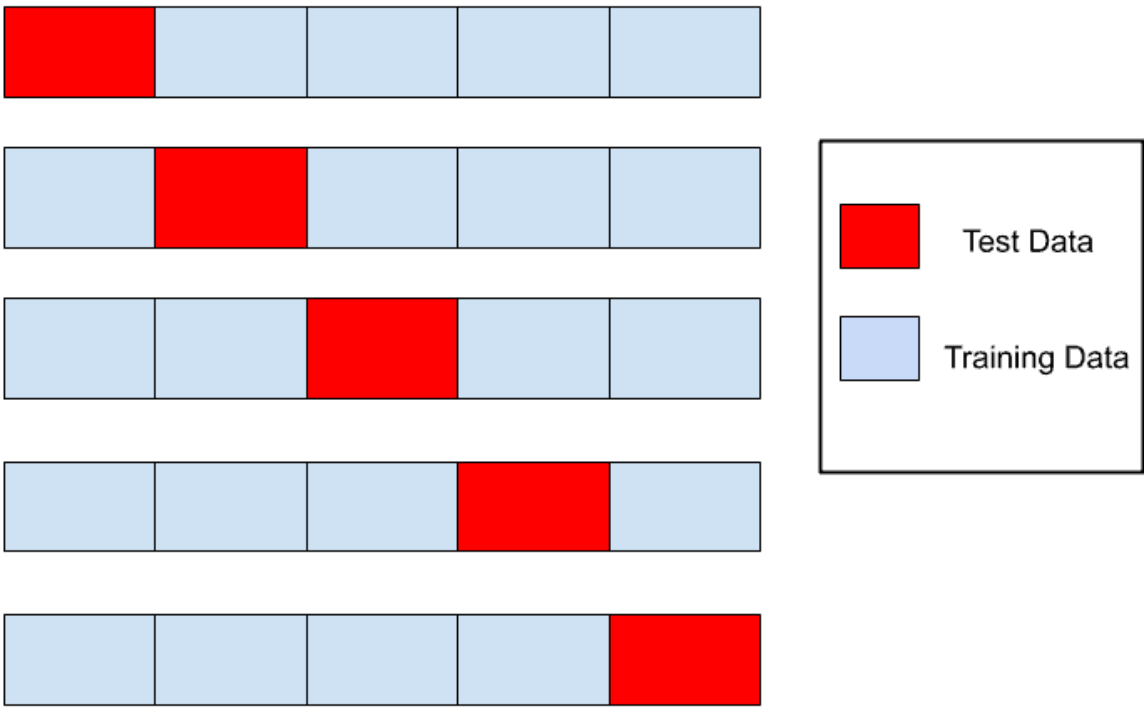
So, the problem is our accuracy fluctuates with different random_state. And we can't say that what accuracy our model has.

Therefore to prevent this problem, **Cross-Validation is used.**

What is Cross-Validation?

To evaluate the performance of a model, we use **Cross-Validation.**
Cross-Validation split the dataset into different segments. And use each segment for training as well as testing one by one.

You can understand with the help of this image-



Types of Cross-Validation

Cross-Validation has following types-

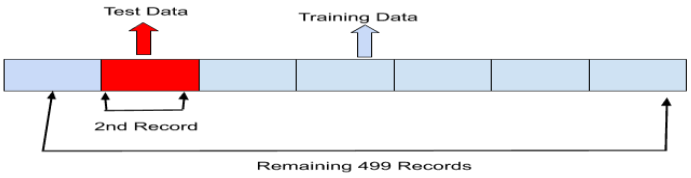
1. Leave One Out Cross-Validation
2. K-Fold Cross-Validation.
3. Stratified Cross-Validation.
4. Time-Series Cross-Validation

1. Leave One Out Cross-Validation (LOOCV)

Suppose, we have 500 records in our dataset. It basically takes one record for the Test dataset and the remaining record for the Training dataset.

So, here Leave One Out method have chosen one record for **test data**, and the remaining 499 records for **Training data**. Here Leave One Out method train the model with 499 records. And test the model performance with a single record. This is round 1.

In the second round, it will choose the **second record for test data** and the remaining 499 records for **training data.**



Similarly, it will take the next one record for testing, and the remaining 499 records for Training.

As its name suggests, “Leave One Out”, therefore it leave only one record for testing.

So, the problem with Leave One Out method is that it is a time-consuming process. Suppose we have 10000 records in our dataset. Then we have to perform 10000 iterations for every single record. And that is a very time-consuming process.

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from numpy import mean
from numpy import absolute
from numpy import sqrt
import pandas as pd
df = pd.DataFrame({'y': [6, 8, 12, 14, 14, 15, 17, 22, 24, 23],
                   'x1': [2, 5, 4, 3, 4, 6, 7, 5, 8, 9],
                   'x2': [14, 12, 12, 13, 7, 8, 7, 4, 6, 5]})

#define predictor and response variables
X = df[['x1', 'x2']]
y = df['y']

#define cross-validation method to use
cv = LeaveOneOut()

#build multiple linear regression model
model = LinearRegression()

#use LOOCV to evaluate model
scores = cross_val_score(model, X, y,
scoring='neg_mean_absolute_error',
                         cv=cv, n_jobs=-1)
```

#view mean absolute error
mean(absolute(scores))

3.1461548083469726

The disadvantage of Leave One Out Cross-Validation
Leave One Out method has the following disadvantages,

1. We need to perform many iterations depending upon the number of records.
2. Leave One Out lead to low bias.

2. K Fold CV

K Fold Cross-Validation is very simple. And easy to understand.

K fold cross-validation solves the problem of the **Train-Test split.**

K Fold selects the k value, and based on this k value, we split the data.

Suppose we have 1000 records in our dataset. And we select the value of K as 5. So, K value means, the number of rounds we perform Training and Testing.

And here, our k value is 5, that means we have to perform 5 round of Training and Testing.

For each experiment, based on the K value, it will decide the number of Test Data.

Here, our k value is 5. And we have 1000 records.

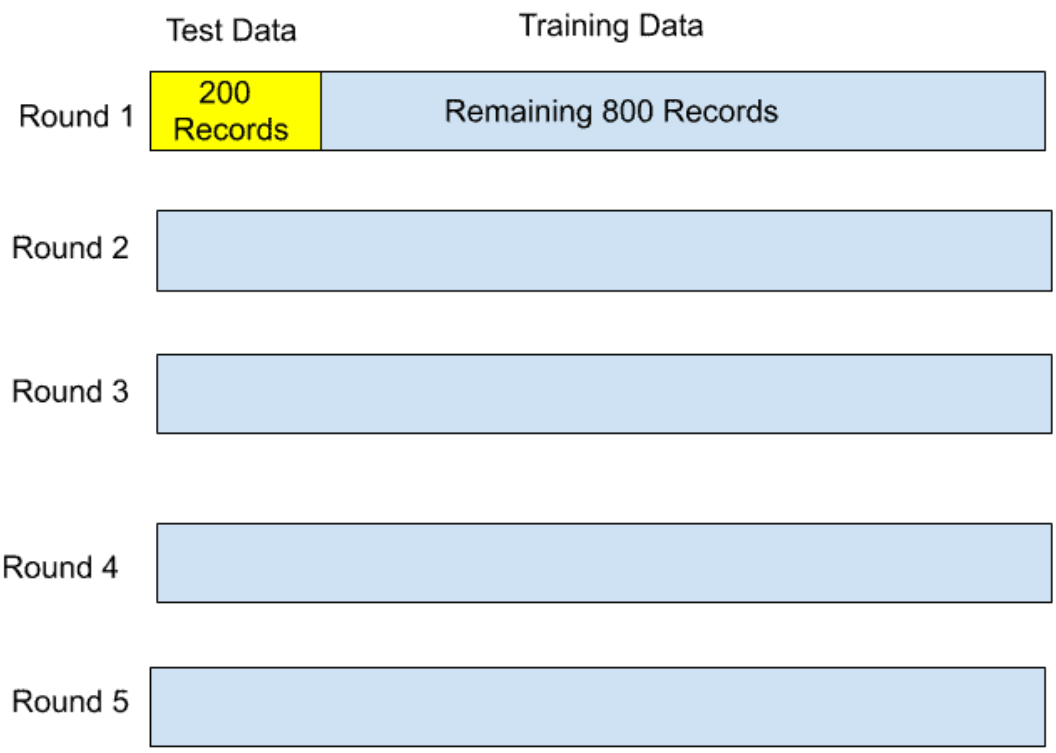
Right?.

So when we divide 1000 with 5.

1000/5=200

So, 200 will be our Test Data. And the remaining 800 records will be our Training Data.

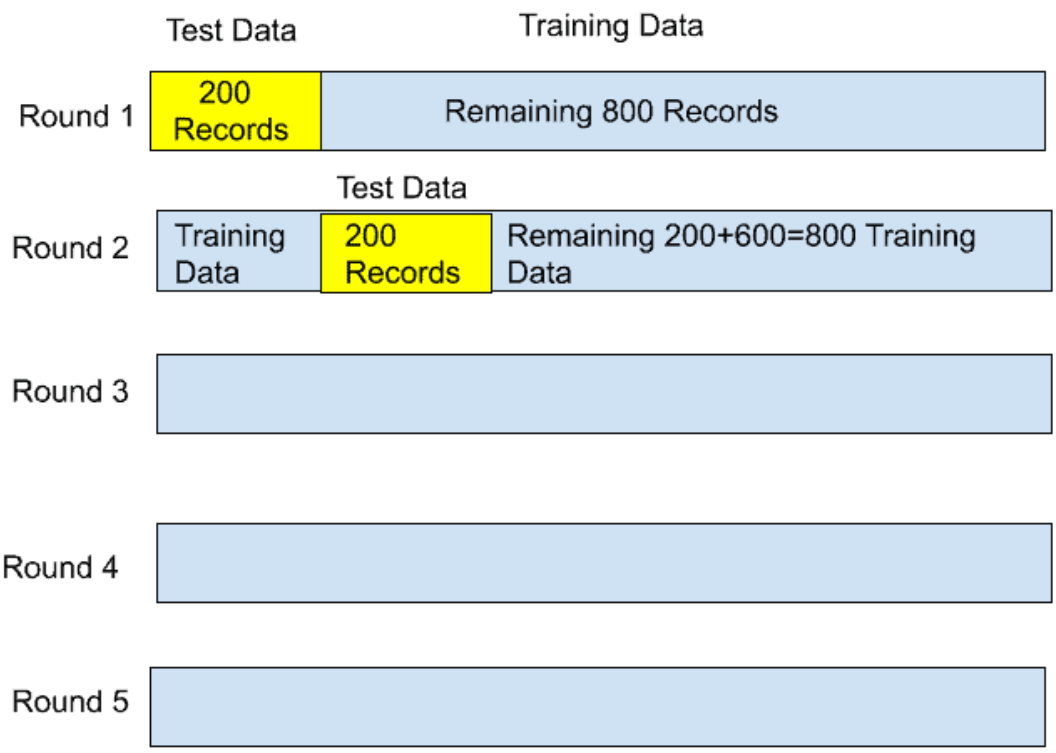
So for the first round, the first 200 records are used for Test Data and the remaining 800 for Training data.



This is the first round. So, this model is trained with 800 datasets. And tested on 200 datasets. After that, we got our first Accuracy.

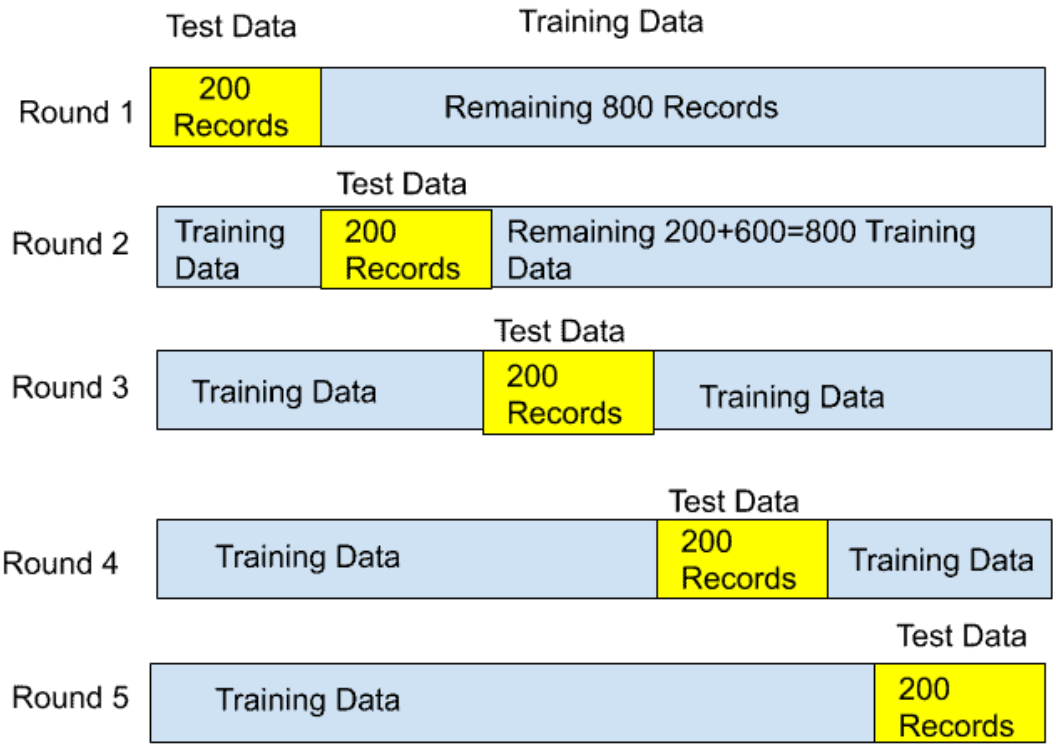
Now, What will happen in the second round?

In second round, the next 200 records will be our Test data, and the remaining 800 data is our Training data.



Here, our model is trained on 800 records and tested on the next 200 records. And Again, we got some accuracy. Let's say Accuracy 2.

In the same way, all rounds happened., and we got accuracies with each round.



So after completing all 5 rounds, we got 5 accuracies. The Accuracy 1, Accuracy 2, Accuracy 3, Accuracy 4, and accuracy 5.

We can take all Accuracies, and find out the **Mean**. And that **Mean** of all 5 accuracies is the **actual accuracy of your model.**

By doing so, your accuracy will not fluctuate as in Train-Test Split.
Moreover, you will get your model **minimum accuracy** and **maximum accuracy.**

How to Choose the K value?
The value of k should be chosen carefully. If you choose poor k value, it will result in high variance or high bias.

The value of K depends upon the size of the data. Along with that, How much your system is capable to afford the computational cost. The high K value, the more rounds or folds you need to perform.

So, before selecting the K value, look at your data size and your system computation power.

There are some common strategies for choosing the k value-

k=10-> This is the value found by after various experiments. k=10 will result in a model with low bias and moderate variance. So if you are struggling to choose the value of k for your dataset, you can choose k=10. The value of k as 10 is very common in the field of machine learning.
K=n-> The value of k is n, where n is the size of the dataset. That means using each record in a dataset to test the model. That is nothing but **Leave One Out Approach.**
There is no formal rule but the value of k should be 5 or 10.

```
# prepare the cross-validation procedure
cv = KFold(n_splits=10, random_state=1, shuffle=True)
# create model
model = LogisticRegression()
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Disadvantages of K Fold Cross-Validation

Suppose in Round 1, we have 200 Test Dataset. But what if all these test datasets are of the same type?.

That means, suppose we are working on classification problem, and we have dataset only in binary form, like 0 and 1.

And we have only 0's in the test dataset. So, this is an Imbalanced dataset. And that can be a problem. In that case, you will not get accurate results.

In order to solve this problem, Stratified Cross-Validation is used.

3. Stratified Cross-Validation

In Stratified Cross-validation, everything will be the same as in K fold Cross-Validation. But in Stratified Cross-Validation, whenever the Test Data is selected, make sure that the number of instances of each class for each round in train and test data, is taken in a proper way.

Suppose we have 500 records, in which we have 400 Yes, and 100 Now. And that is Imbalanced Dataset.

So in that case, **Stratified Cross-Validation will** make sure that in Training Dataset, there should be a proper proportion of Yes and No. And also In the Test dataset, there should be a proper proportion of Yes and No.

By doing so, our model will give an accurate result.

```
CV = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
```

```
# create model
model = LogisticRegression()

scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

4. Time-Series Cross-Validation

Time Series Cross-Validation works for a completely different problem.

So, Where Time Series Cross-Validation will work?

When you have a dataset related to time series. Like Stock Price Prediction. In Stock Price Prediction, we have to predict future data.

So, in Stock Price Prediction data, we can't perform Train-Test Split. Because this data is time-series data.

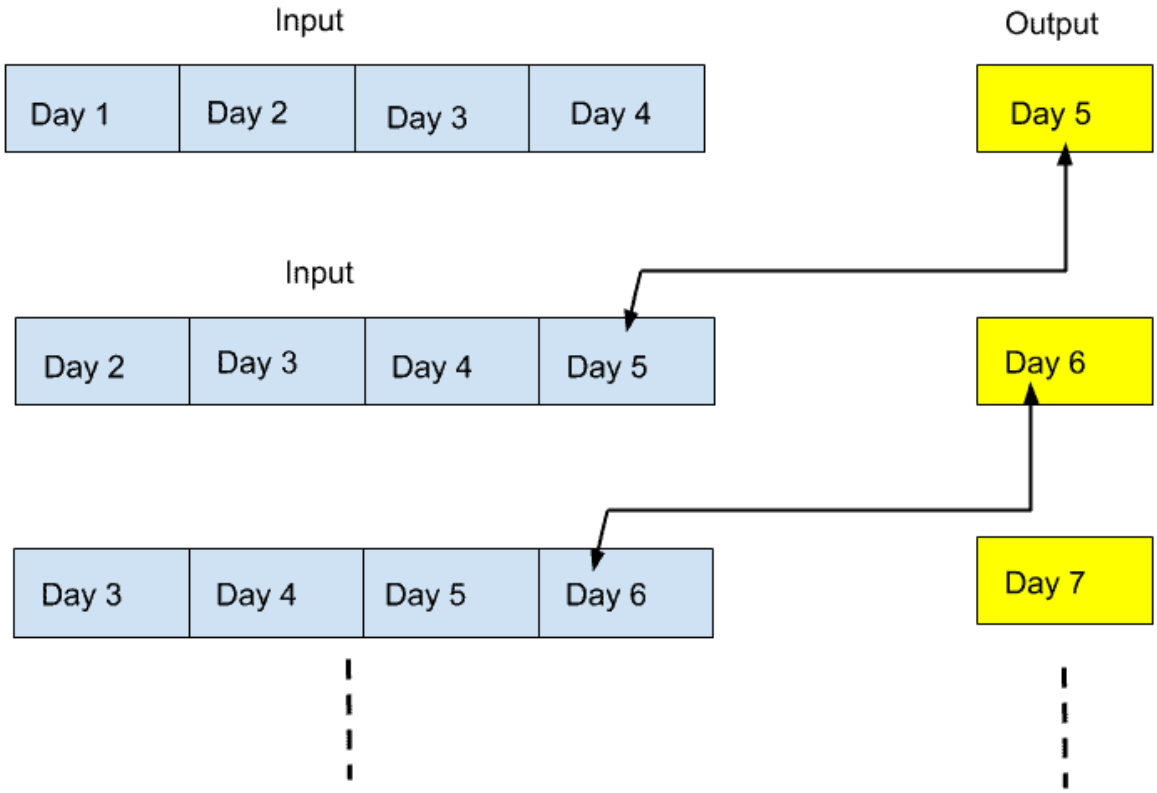
Suppose we have 4 days Data. And we have to predict the stock price for Day 5 and Day 6.

For that, we are dependent upon data of 4 days.

Based on Day1, Day2, Day3, and Day4 data, we can predict the stock price of Day 5 Data.

Similarly, for Day 6, we use Day 2, Day 3, Day 4, and Day 5 data for predicting stock price for Day 6.

Understand with the help of this image.



So, that's how Time Series Cross-Validation works. It will use the previous day data to predict the stock price for next day data

Thumb Rules Associated with K Fold

Now, we will discuss a few thumb rules while playing with K – fold

- K should be always >= 2 and = to number of records, (LOOCV)
    - If 2 then just 2 iterations
    - If K=No of records in the dataset, then 1 for testing and n- for training
- The optimized value for the K is 10 and used with the data of good size. (Commonly used)

- If the K value is too large, then this will lead to less variance across the training set and limit the model currency difference across the iterations.
- The number of folds is indirectly proportional to the size of the data set, which means, if the dataset size is too small, the number of folds can increase.
- Larger values of K eventually increase the running time of the cross-validation process.

Python Code:

You can see the Train and Test array and how the array got split in every iteration.

Model Selection using K-Fold

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
digits = load_digits()
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(digits.data,digits.target,test_size=0.3)
```

imported required libraries and loaded digits (**hand-written digits – open source), let's apply a different algorithm.**

Logistic Regression

I am using liblinear. This is the "**Large Linear Classification"** category. It uses a Coordinate-Descent Algorithm. This would minimize a multivariate function by resolving the univariate and its optimization problems during the loop.

```
lr = LogisticRegression(solver='liblinear',multi_class='ovr')
```

```
lr.fit(X_train, y_train)
lr.score(X_test, y_test)
```

Output

Score : 0.972222

SVC

Just using gamma is a parameter for non-linear perspective for hyperplanes. The value of the gamma tries to fit the training data set and uses 1/n_features.

```
svm = SVC(gamma='auto')
svm.fit(X_train, y_train)
svm.score(X_test, y_test)
```

Output

Score : 0.62037

Random Forest

For RFC, I am assigning estimators as 40.

```
rf = RandomForestClassifier(n_estimators=40)
rf.fit(X_train, y_train)
rf.score(X_test, y_test)
```

Output

Score: 0.96666

Scores from the above list of algorithms **Logistic Regression and Random Forest** are doing comparatively better than SVM.

Now will use cross_val_score function and get the scores, passing different algorithms with dataset and cv.

```
from sklearn.model_selection import cross_val_score
```

Set LogisticRegression, CV =3

```
score_lr=cross_val_score(LogisticRegression(solver='liblinear',multi
_class='ovr'), digits.data, digits.target,cv=3)
print(score_lr)
print("Avg :",np.average(score_lr))
```

Output: for 3 fold we have 3 scores

```
[0.89482471 0.95325543 0.90984975]
Avg : 0.9193099610461881
```

Set SVM and CV=3

score_svm =cross_val_score(SVC(gamma='auto'), digits.data, digits.target,cv=3)
print(score_svm)
print("Avg :",np.average(score_svm))

Output: Scores

```
[0.38063439 0.41068447 0.51252087]
Avg : 0.4346132442960489
```

Set Random Forest and CV=3

score_rf=cross_val_score(RandomForestClassifier(n_estimators=40),digits.data, digits.target,cv=3)
print(score_rf)
print("Avg :",np.average(score_rf))

Output: Scores

```
[0.92821369 0.95325543 0.92320534]
Avg : 0.9348914858096827
```

	Before K Fold apply	**After K Fold applied (Avg)**
Logistic Regression	97%	91%
SVM	62%	43%
Random Forest	96%	93%

Based on the above table, we will go with **Random Forest** for this dataset for production. But we have to monitor the model performance based on the data drift and as the business case changes, we have to revisit the model and redeploy.

Parameter Tuning **Using K-Fold**

consider the **RandomForestClassifier** for this analysis, and n_estimators is our parameter for this case and CV as 10 (commonly used)

```
scores1 =
cross_val_score(RandomForestClassifier(n_estimators=5),digits.data,
digits.target, cv=10)
print("Avg Score for Estimators=5 and CV=10 :",np.average(scores1))
```

Output

```
Avg Score for Estimators=5 and CV=10 : 0.87369
```

```
scores2 =
cross_val_score(RandomForestClassifier(n_estimators=20),digits.data,
digits.target, cv=10)
print("Avg Score for Estimators=20 and CV=10 :",np.average(scores2))
```

Output

```
Avg Score for Estimators=20 and CV=10 : 0.93377
```

```
scores3 =
cross_val_score(RandomForestClassifier(n_estimators=30),digits.data,
digits.target, cv=10)
print("Avg Score for Estimators=30 and CV=10 :",np.average(scores3))
```

Output

```
Avg Score for Estimators=30 and CV=10 : 0.94879
```

```
scores4 =
cross_val_score(RandomForestClassifier(n_estimators=40),digits.data,
digits.target, cv=10)
print("Avg Score for Estimators=40 and CV=10 :",np.average(scores4))
```

Output

```
Avg Score for Estimators=40 and CV=10 : 0.94824
```

scores1	87.36%
scores2	93.33%
scores3	94.87%
scores4	94.82%

Based on the above observation, we will go with Estimators=30.

K-Fold in Visual form

Visual representation is always the best evidence for any data which is located across the axes.

from sklearn.model_selection import cross_val_score

```
knn = KNeighborsClassifier(n_neighbors=5)
scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
print(scores.mean())
```

Output

```
0.9666666666666668
k_range = list(range(1, 25))
k_scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())
print(k_scores)
```

Output

```
[0.96, 0.95333, 0.96666, 0.96666, 0.966668, 0.96666, 0.966666,
0.966666, 0.97333, 0.96666, 0.96666, 0.97333, 0.9800, 0.97333,
0.97333, 0.97333, 0.97333, 0.98000, 0.9733333, 0.980000, 0.966666,
0.96666, 0.973333, 0.96, 0.96666, 0.96, 0.96666, 0.953333, 0.95333,
0.95333]
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated-Accuracy')
```

Output: With a simple plot, X=> value of K and Y=> Accuracy for respective CV

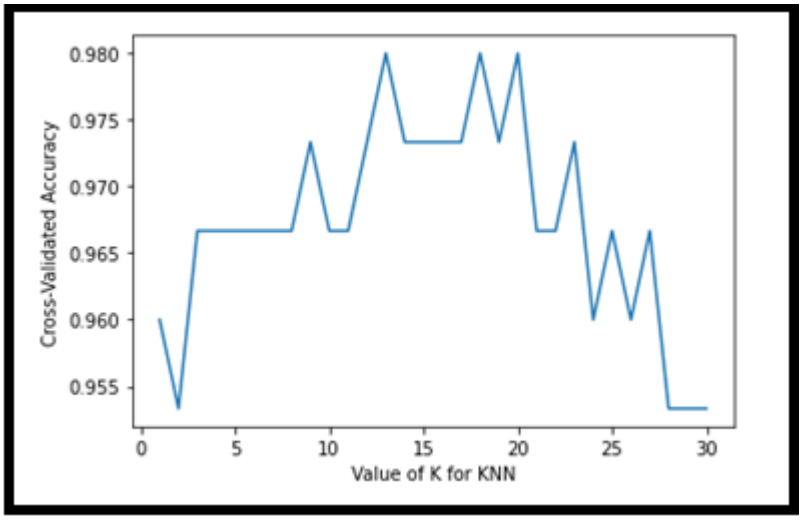


Image designed by the author

The above visual representation helps us to understand the accuracy is ~98%for K=12,18 and 19 for KNN.