Theoritical & Practical Concepts of K-Nearest Neighbors (KNN) Algorithm

K-Nearest Neighbors (KNN) Algorithm

K-Nearest Neighbors (KNN) stands as a versatile algorithm, finding its utility in both classification and regression tasks.

The Essence of K Nearest Neighbors

KNN operates on a simple principle: identifying the 'k' nearest data points to make predictions. But what does 'k' signify?

Understanding 'k': The Key to Proximity

The 'k' in KNN represents the count of nearest neighbors we consider while predicting. For instance, if 'k' is 3, we look at the three closest data points from our training set to predict the outcome for a new data point.

KNN for Classification

In classification tasks, we categorize a data point based on the classes of its nearest neighbors.

The Decision Rule: Majority Wins

For instance, when 'k' is 5 and three out of these five nearest neighbors belong to Class A while two belong to Class B, we assign the new data point to Class A. It's all about the majority!

KNN for Regression

In regression, the output for a new data point is a continuous value, not a class. 'k' still plays a vital role.

The Regression Formula: Averaging Insights

For example, if we're predicting house prices, we calculate the average price of the three closest houses to determine our prediction.

Distance Metrics

To find these 'k' neighbors, we measure distances. Two common distance metrics are:

- **Euclidean Distance**: The straight-line distance between two points.
- **Manhattan Distance**: The sum of absolute differences of their coordinates.

Deciding the Right 'k'

The choice of 'k' matters! Data scientists often choose odd 'k' if the classes are even. Techniques like cross-validation and the elbow method guide us in selecting the optimal 'k' that ensures accurate predictions.

Practical Tips: Navigating Real-World Scenarios

- **Choosing 'k' Value**: Experiment with various 'k' values to strike the right balance of accuracy and performance.
- **Handling Outliers and Imbalanced Data**: KNN is sensitive to outliers; handle them wisely. For imbalanced data, resampling techniques come to the rescue.
- **Best Use Cases**: KNN excels in classifying well-defined clusters and capturing irregular decision boundaries. For regression, it thrives when the data shows a smooth, continuous relationship.

With KNN, experiment with different 'k' values, and let the neighbors guide you towards precise predictions!

🎯 🔍 #MachineLearning #KNN #Classification #Regression #DataScience

Applying KNN Machine Learning

Import python Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Load Dataset

https://www.kaggle.com/datasets/mysarahmadbhat/lung-cancer/data

Context of the Dataset : The effectiveness of cancer prediction system helps the people to know their cancer risk with low cost and it also helps the people to take the appropriate decision based on their cancer risk status.

Goal: Using KNN algorithm with multiple K-value to predict two discrete classes - Lung Cancer: YES , NO. based on a dataset of independent variables.

```
df = pd.read_csv('/content/survey lung cancer.csv')
df.head()
```

MOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEI
1	2	2	1	1	2	1	
2	1	1	1	2	2	2	
1	1	1	2	1	2	1	
2	2	2	1	1	1	1	
1	2	1	1	1	1	1	

Explotarory Data Analysis

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 309 entries, 0 to 308
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   GENDER                 309 non-null    object
 1   AGE                    309 non-null    int64 
 2   SMOKING                309 non-null    int64 
```

```
 3   YELLOW_FINGERS         309 non-null    int64
 4   ANXIETY                309 non-null    int64
 5   PEER_PRESSURE          309 non-null    int64
 6   CHRONIC DISEASE        309 non-null    int64
 7   FATIGUE                309 non-null    int64
 8   ALLERGY                309 non-null    int64
 9   WHEEZING               309 non-null    int64
 10  ALCOHOL CONSUMING      309 non-null    int64
 11  COUGHING               309 non-null    int64
 12  SHORTNESS OF BREATH    309 non-null    int64
 13  SWALLOWING DIFFICULTY  309 non-null    int64
 14  CHEST PAIN             309 non-null    int64
 15  LUNG_CANCER            309 non-null    object
dtypes: int64(14), object(2)
memory usage: 38.8+ KB
```

```
cat_feature = df.select_dtypes(include='object').columns
num_feature = df.select_dtypes(exclude='object').columns
```

```
cat_feature.isnull().sum()
```

```
0
```

```
num_feature.isnull().sum()
```

```
0
```

```
df['GENDER'].value_counts()
```

```
M    162
F    147
Name: GENDER, dtype: int64
```

```
df['LUNG_CANCER'].value_counts()
```

```
YES    270
NO      39
Name: LUNG_CANCER, dtype: int64
```

```
df[num_feature].describe().T
```

	count	mean	std	min	25%	50%	75%	max
AGE	309.0	62.673139	8.210301	21.0	57.0	62.0	69.0	87.0
SMOKING	309.0	1.563107	0.496806	1.0	1.0	2.0	2.0	2.0
YELLOW_FINGERS	309.0	1.569579	0.495938	1.0	1.0	2.0	2.0	2.0
ANXIETY	309.0	1.498382	0.500808	1.0	1.0	1.0	2.0	2.0
PEER_PRESSURE	309.0	1.501618	0.500808	1.0	1.0	2.0	2.0	2.0
CHRONIC DISEASE	309.0	1.504854	0.500787	1.0	1.0	2.0	2.0	2.0
FATIGUE	309.0	1.673139	0.469827	1.0	1.0	2.0	2.0	2.0
ALLERGY	309.0	1.556634	0.497588	1.0	1.0	2.0	2.0	2.0
WHEEZING	309.0	1.556634	0.497588	1.0	1.0	2.0	2.0	2.0
ALCOHOL CONSUMING	309.0	1.556634	0.497588	1.0	1.0	2.0	2.0	2.0
COUGHING	309.0	1.579288	0.494474	1.0	1.0	2.0	2.0	2.0
SHORTNESS OF BREATH	309.0	1.640777	0.480551	1.0	1.0	2.0	2.0	2.0
SWALLOWING DIFFICULTY	309.0	1.469256	0.499863	1.0	1.0	1.0	2.0	2.0
CHEST PAIN	309.0	1.556634	0.497588	1.0	1.0	2.0	2.0	2.0

Summary of variables

- There are 16 columns, out of this Gender columns has replace into numerical columns
- The target variable lung cencer
- There are not null values into the dataset.
- The numerical value does not have any outlier.

Estimating correlation coefficients

we can compute the standard correlation coefficient (also called Pearson's r) between every pair of attributes.

- The correlation coefficient ranges from -1 to +1.
- When it is close to +1, this signifies that there is a strong positive correlation. So, we can see that there is a strong positive correlation.
- When it is clsoe to -1, it means that there is a strong negative correlation.

- When it is close to 0, it means that there is no correlation.

Multicollinearity

multicollinearity, that is, features in our feature matrix that are highly correlated with each other. A good way to detect this is to use a heatmap.
If few feature has corelation, we can only keep one feature

```
df.corr()
```

```
<ipython-input-173-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will
  df.corr()
```

	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SHORT OF BR
AGE	1.000000	-0.084475	0.005205	0.053170	0.018685	-0.012642	0.012614	0.027990	0.055011	0.058985	0.169950	-0.01
SMOKING	-0.084475	1.000000	-0.014585	0.160267	-0.042822	-0.141522	-0.029575	0.001913	-0.129426	-0.050623	-0.129471	0.06
YELLOW_FINGERS	0.005205	-0.014585	1.000000	0.565829	0.323083	0.041122	-0.118058	-0.144300	-0.078515	-0.289025	-0.012640	-0.10
ANXIETY	0.053170	0.160267	0.565829	1.000000	0.216841	-0.009678	-0.188538	-0.165750	-0.191807	-0.165750	-0.225644	-0.14
PEER_PRESSURE	0.018685	-0.042822	0.323083	0.216841	1.000000	0.048515	0.078148	-0.081800	-0.068771	-0.159973	-0.089019	-0.22
CHRONIC DISEASE	-0.012642	-0.141522	0.041122	-0.009678	0.048515	1.000000	-0.110529	0.106386	-0.049967	0.002150	-0.175287	-0.02
FATIGUE	0.012614	-0.029575	-0.118058	-0.188538	0.078148	-0.110529	1.000000	0.003056	0.141937	-0.191377	0.146856	0.44
ALLERGY	0.027990	0.001913	-0.144300	-0.165750	-0.081800	0.106386	0.003056	1.000000	0.173867	0.344339	0.189524	-0.03
WHEEZING	0.055011	-0.129426	-0.078515	-0.191807	-0.068771	-0.049967	0.141937	0.173867	1.000000	0.265659	0.374265	0.03
ALCOHOL CONSUMING	0.058985	-0.050623	-0.289025	-0.165750	-0.159973	0.002150	-0.191377	0.344339	0.265659	1.000000	0.202720	-0.17
COUGHING	0.169950	-0.129471	-0.012640	-0.225644	-0.089019	-0.175287	0.146856	0.189524	0.374265	0.202720	1.000000	0.27
SHORTNESS OF BREATH	-0.017513	0.061264	-0.105944	-0.144077	-0.220175	-0.026459	0.441745	-0.030056	0.037834	-0.179416	0.277385	1.00
SWALLOWING DIFFICULTY	-0.001270	0.030718	0.345904	0.489403	0.366590	0.075176	-0.132790	-0.061508	0.069027	-0.009294	-0.157586	-0.16
CHEST PAIN	-0.018104	0.120117	-0.104829	-0.113634	-0.094828	-0.036938	-0.010832	0.239433	0.147640	0.331226	0.083958	0.02

Prepocessing

Only the variable GENDER needs to prepocess

```
df['GENDER'].value_counts()
```

```
M    162
F    147
Name: GENDER, dtype: int64
```

```
df['GENDER'] = df['GENDER'].replace({'M':1, 'F':2})
```

```
df.head()
```

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SHORTNESS OF BREATH	SWALLOWING DIFFICULTY	CH P
0	1	69	1	2	2	1	1	2	1	2	2	2	2	2	
1	1	74	2	1	1	1	2	2	2	1	1	1	2	2	
2	2	59	1	1	1	2	1	2	1	2	1	2	2	1	
3	1	63	2	2	2	1	1	1	1	1	2	1	1	2	
4	2	63	1	2	1	1	1	1	1	2	1	2	2	1	

Split

```
X = df.drop(columns='LUNG_CANCER')
y = df['LUNG_CANCER']

# split X and y into training and testing sets
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)


# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

```
((247, 15), (62, 15))
```

Feature Scaling

Feature scaling is the process of setting the variables on a similar scale. This is usually done using normalization, standardization.

Standardization

Standardization is the process of centering the variable at 0 (zero mean) and standardizing the variance to 1 (unit variance), and it is suitable for variables with a Gaussian distribution.

```
cols = X_train.columns
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])


X_train.head()
```

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SHORTNESS OF BREATH	S
0	1.028753	-1.487580	0.881464	0.859905	1.045572	0.995960	-1.037126	0.698535	0.925820	-1.172604	-1.125191	-1.202308	0.71787	
1	-0.972050	1.140512	0.881464	0.859905	1.045572	0.995960	0.964203	0.698535	-1.080123	0.852803	0.888738	0.831734	0.71787	
2	-0.972050	-0.486402	0.881464	-1.162919	-0.956414	0.995960	-1.037126	-1.431567	-1.080123	-1.172604	0.888738	0.831734	0.71787	
3	-0.972050	-1.362433	0.881464	-1.162919	-0.956414	-1.004057	0.964203	0.698535	0.925820	0.852803	0.888738	-1.202308	-1.39301	

The X_train dataset is ready to be fed into the KNN classifier.

Model Training

The steps to building and using a model are:

Define: What type of model will it be? A decision tree, or KNN? Some other type of model? Some other parameters of the model type are specified too.

Fit: Capture patterns from provided data. This is the heart of modeling.

Predict: Just what it sounds like

Evaluate: Determine how accurate the model's predictions are.

Define the model

```
# import KNeighbors ClaSSifier from sklearn
from sklearn.neighbors import KNeighborsClassifier
# instantiate the model
knn = KNeighborsClassifier(n_neighbors=3)
```

Fit The Model

```
# fit the model to the training set
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

Predict The Model

```
y_pred = knn.predict(X_test)
y_pred
```

```
array(['YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'YES',
       'YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'NO',
       'YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'NO', 'NO', 'YES', 'YES',
       'YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'YES',
       'YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'YES', 'NO',
       'YES', 'YES', 'YES', 'NO', 'YES', 'YES', 'YES', 'YES', 'YES',
       'YES', 'YES', 'NO', 'YES', 'YES', 'YES', 'YES'], dtype=object)
```

Predicting test set result

At this point, the model is trained and ready to predict the output of new observations. Remember, we split our dataset into train and test sets. We will provide test sets to the model and check its performance.

```
# y_test and y_pred are your actual and predicted labels
prediction_df = pd.DataFrame({
    'Actual Value': y_test,
    'Predicted Value': y_pred,
    'Prediction Correct': y_test == y_pred  # True if prediction is correct, False otherwise
})

# Display the prediction_df DataFrame
print(prediction_df)
```

```
    Actual Value Predicted Value  Prediction Correct
63           YES             YES                True
231          YES             YES                True
167          YES             YES                True
```

```
159          NO            YES           False
189         YES            YES            True
..          ...            ...             ...
34           NO             NO            True
250         YES            YES            True
33          YES            YES            True
21          YES            YES            True
103         YES            YES            True

[62 rows x 3 columns]
```

Evaluate the Model

```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)

print('Model accuracy score: {0:0.4f}'. format(score))
```

```
Model accuracy score: 0.9032
```

We can see that our model accuracy score is 0.9032 but null or baseline accuracy score is 0.8387.

So, we can conclude that our K Nearest Neighbors model is doing a very good job in predicting the class labels.

Check for overfitting and underfitting:

Overfitting usually manifests as a significant gap between training and test accuracies.

Underfitting is marked by low accuracies on both sets due to insufficient model complexity.

Generalized Model is demonstrates consistent performance on both training and test sets, suggesting it is well-generalized and not overfit.

```
print('Training set score: {:.4f}'.format(knn.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(knn.score(X_test, y_test)))
```

```
Training set score: 0.9514
Test set score: 0.9032
```

A slight difference in scores is normal and doesn't necessarily indicate overfitting.

To further assess the model's performance and ensure it's not overfitting, you can also look at other evaluation metrics like precision, recall, and F1-score, and visualize the model's performance using a confusion matrix.

Additionally, cross-validation can be a helpful technique to assess the model's stability and performance across different subsets of the data.

Compare model accuracy with Null/Baseline accuracy

Null or Baseline accuracy

Comapring model accuracy with a null or baseline accuracy is a good practice to evaluate the model's performance. The null accuracy is the accuracy achieved by a model that always predicts the most frequent class in the dataset. It provides a baseline for comparison, helping to gauge whether the model's performance is meaningful and better than a simple baseline prediction strategy.

I will do so, & compare

```
y_test.value_counts()
```

```
YES    52
NO     10
Name: LUNG_CANCER, dtype: int64
```

```
baseline_accuracy = (52/(52+10))
baseline_accuracy
```

```
0.8387096774193549
```

We can see that our **model accuracy score is 0.9032** but **null/baseline accuracy score is 0.83870**. So, we can conclude that our K Nearest Neighbors model is doing a very good job in predicting the class labels.

we could also find baseline accuracy is the target variable convert into int by excuting this code:

y_mean = y_train.mean()

print("Mean score:", y_mean)

Rebuild kNN Classification model using different values of k

Above kNN classification model has build using k=3. Lets play around with k value and see, if increasing the value of k, does the accuracy increase?

kNN Classification model using k=4

```
# instantiate the model with k=4
knn_4 = KNeighborsClassifier(n_neighbors=4)

# fit the model to the training set
knn_4.fit(X_train, y_train)

# predict on the test-set
y_pred_4 = knn_4.predict(X_test)

print('Model accuracy score with k=4 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_4)))
```

```
Model accuracy score with k=4 : 0.9032
```

kNN Classification model using k=5

```
# instantiate the model with k=5
knn_5 = KNeighborsClassifier(n_neighbors=5)

# fit the model to the training set
knn_5.fit(X_train, y_train)

# predict on the test-set
y_pred_5 = knn_5.predict(X_test)

print('Model accuracy score with k=5 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_5)))
```

```
Model accuracy score with k=5 : 0.8710
```

kNN Classification model using k=6

```
# instantiate the model with k=6
knn_6 = KNeighborsClassifier(n_neighbors=6)

# fit the model to the training set
knn_6.fit(X_train, y_train)

# predict on the test-set
y_pred_6 = knn_6.predict(X_test)

print('Model accuracy score with k=6 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_6)))
```

```
Model accuracy score with k=6 : 0.8871
```

kNN Classification model using k=7

```
# instantiate the model with k=7
knn_7 = KNeighborsClassifier(n_neighbors=7)

# fit the model to the training set
knn_7.fit(X_train, y_train)

# predict on the test-set
y_pred_7 = knn_7.predict(X_test)

print('Model accuracy score with k=7 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_7)))
```

```
Model accuracy score with k=7 : 0.8871
```

kNN Classification model using k=8

```
# instantiate the model with k=8
knn_8 = KNeighborsClassifier(n_neighbors=8)

# fit the model to the training set
knn_8.fit(X_train, y_train)

# predict on the test-set
y_pred_8 = knn_8.predict(X_test)

print('Model accuracy score with k=8 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_8)))
```

```
Model accuracy score with k=8 : 0.8871
```

kNN Classification model using k=9

```
# instantiate the model with k=9
knn_9 = KNeighborsClassifier(n_neighbors=9)

# fit the model to the training set
knn_9.fit(X_train, y_train)

# predict on the test-set
y_pred_9 = knn_9.predict(X_test)

print('Model accuracy score with k=9 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_9)))
```

```
Model accuracy score with k=9 : 0.8710
```

Interpretation

model accuracy score with k=3 is 0.9032

Model accuracy score with k=4 : 0.9032

Model accuracy score with k=5 : 0.8710

Model accuracy score with k=6 : 0.8871

Model accuracy score with k=7 : 0.8871

Model accuracy score with k=8 : 0.8871

Model accuracy score with k=9 : 0.8710

k=3 and k=4:

The model might be finding an optimal balance between overfitting and underfitting with these 'k' values, resulting in higher accuracy. A smaller 'k' value allows the model to capture finer patterns in the data.

k=5, k=7, and k=9:

The accuracy slightly drops. This could be due to the increasing influence of noise and outliers as 'k' gets larger. Larger 'k' values smooth out the decision boundaries, potentially making the model less sensitive to local variations.

k=6 and k=8:

The accuracy is consistent with k=5, k=7, and k=9. This suggests that once 'k' is large enough to capture the underlying patterns in the data, further increasing it does not significantly impact the accuracy.

Since your dataset is small, having a larger 'k' might not necessarily improve accuracy as it might introduce more noise from the dataset. Additionally, KNN can be sensitive to the choice of 'k', especially in smaller datasets. It's important to experiment with different 'k' values, possibly using techniques like cross-validation, to find the optimal 'k' that balances bias and variance for your specific dataset

Evaluation metrics

Confusion Matrix

A confusion matrix is a tool for summarizing the performance of a classification algorithm. A confusion matrix will give us a clear picture of classification model performance and the types of errors produced by the model. It gives us a summary of correct and incorrect predictions broken down by each category. The summary is represented in a tabular form.

Four types of outcomes are possible while evaluating a classification model performance. These four outcomes are described below:-

True Positives (TP) – True Positives occur when we predict an observation belongs to a certain class and the observation actually belongs to that class.

True Negatives (TN) – True Negatives occur when we predict an observation does not belong to a certain class and the observation actually does not belong to that class.

False Positives (FP) – False Positives occur when we predict an observation belongs to a certain class but the observation actually does not belong to that class. This type of error is called Type I error.

False Negatives (FN) – False Negatives occur when we predict an observation does not belong to a certain class but the observation actually belongs to that class. This is a very serious error and it is called Type II error.

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[ 5  5]
 [ 1 51]]

True Positives(TP) =  5

True Negatives(TN) =  51

False Positives(FP) =  5

False Negatives(FN) =  1
```

Interpret

The confusion matrix shows (TP+TN) 5 + 51 = 56 correct predictions

and (FP+FN) 5 + 1 = 6 incorrect predictions.

In this case, we have

True Positives - 5

True Negatives - 51

False Positives - 5 (Type I error)

False Negatives - 1 (Type II error)

Classification Report

Classification report is another way to evaluate the classification model performance. It displays the precision, recall, f1 and support scores for the model.

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
                precision    recall  f1-score   support

            NO       0.83      0.50      0.62        10
           YES       0.91      0.98      0.94        52

      accuracy                           0.90        62
     macro avg       0.87      0.74      0.78        62
  weighted avg       0.90      0.90      0.89        62
```

Precision

Precision can be defined as the percentage of correctly predicted positive outcomes out of all the predicted positive outcomes. It can be given as the ratio of true positives (TP) to the sum of true and false positives (TP + FP).

precision = TP / float(TP + FP)

Recall

Recall can be defined as the percentage of correctly predicted positive outcomes out of all the actual positive outcomes. It can be given as the ratio of true positives (TP) to the sum of true positives and false negatives (TP + FN). Recall is also called Sensitivity.

recall = TP / float(TP + FN)

f1-score

f1-score is the weighted harmonic mean of precision and recall. The best possible f1-score would be 1.0 and the worst would be 0.0.

f1-score is the harmonic mean of precision and recall. So, f1-score is always lower than accuracy measures as they embed precision and recall into their computation. The weighted average of f1-score should be used to compare classifier models, not global accuracy.

Support

Support is the actual number of occurrences of the class in our dataset.

Cross-validation

Cross-validation is a statistical method which can be a helpful technique to assess and evaluating the model's stability and generalization performance. It is more stable and thorough than using a train-test split to evaluate model performance.

k-fold Cross Validation

I will apply k-fold Cross Validation technique to improve the model performance.

```
 # Applying 10-Fold Cross Validation

from sklearn.model_selection import cross_val_score

scores = cross_val_score(knn, X_train, y_train, cv = 10, scoring='accuracy')

print('Cross-validation scores:{}'.format(scores))
```

```
Cross-validation scores:[0.84       0.88       0.96       0.88       0.88       0.92
 0.92       0.83333333 0.91666667 0.875     ]
```

We can summarize the cross-validation accuracy by calculating its mean.

```
# compute Average cross-validation score

print('Average cross-validation score: {:.4f}'.format(scores.mean()))
```

```
Average cross-validation score: 0.8905
```

The average cross-validation score of 0.8905 indicates a strong performance of the model, which is consistent and likely to generalize well to unseen data.

In conclusion, having an actual accuracy (k=3 = 0.9032) close to the cross-validation score (score= 0.8905) is a positive indicator of the model's generalization ability, stability, and reliability. It suggests that the model is likely to perform well in real-world scenarios. Always aim for a good alignment between cross-validation results and actual performance to build robust and dependable machine learning models.