**K-Nearest Neighbors (KNN)** **Classifier:** This is a simple and intuitive supervised machine learning algorithm used for both classification and regression tasks. K Stands for number of datapoints closest to a given sample. Here’s a detailed explanation of how it works, its key components, advantages, and disadvantages:

**How KNN Works**

KNN makes predictions based on the similarity between a new data point and the data points in the training set. Here’s a step-by-step explanation of the KNN algorithm:

1. **Select the Number of Neighbors (K)**: Choose the number of nearest neighbors, K, to consider for making the prediction.
2. **Compute Distances:** Calculate the distance between the new data point and all the data points in the training set. Common distance metrics include Euclidean distance, Manhattan distance, and Minkowski distance.
3. **Find Nearest Neighbors:** Identify the K data points in the training set that are closest to the new data point based on the calculated distances.
4. **Make Predictions:**
   * **Classification:** The predicted class is determined by the majority vote among the K nearest neighbors. The class with the most occurrences among the neighbors is assigned to the new data point.
   * **Regression:** The predicted value is the average (or weighted average) of the values of the K nearest neighbors.

**Key Components**

* **Distance Metric:** The choice of distance metric affects the performance of KNN. Euclidean distance is the most used metric, but others can be used depending on the problem.
* **Number of Neighbors (K):** The choice of K significantly impacts the algorithm’s performance. A small K can lead to noise sensitivity, while a large K can cause over-smoothing.

**Example in Python using scikit-learn:**

Here’s an example of how to implement KNN for classification using the scikit-learn library:

A computer screen with white text

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Advantages of KNN**

* **Simplicity:** KNN is easy to understand and implement.
* **No Training Phase:** KNN is a lazy learner, meaning it doesn’t involve an explicit training phase. The training data is used directly for predictions.
* **Versatility:** KNN can be used for both classification and regression tasks.

**Disadvantages of KNN**

* **Computationally Expensive:** KNN requires computing the distance between the new data point and all training data points, which can be slow for large datasets.
* **Memory Intensive:** KNN stores all the training data, which can require significant memory.
* **Sensitivity to Irrelevant Features and the Curse of Dimensionality:** KNN performance can degrade if the feature space is high-dimensional or if irrelevant features are included.
* **Choice of K and Distance Metric:** The performance of KNN is highly dependent on the choice of K and the distance metric.

2D Array

No of Rows = #No. of Samples

#No. of Features

**Decision Regions and boundaries**:

K = 1: Rarely good choice. Sensitive to changes in training data. Likely to Overfit with poor performance on new datasets. Low bias with high variance.

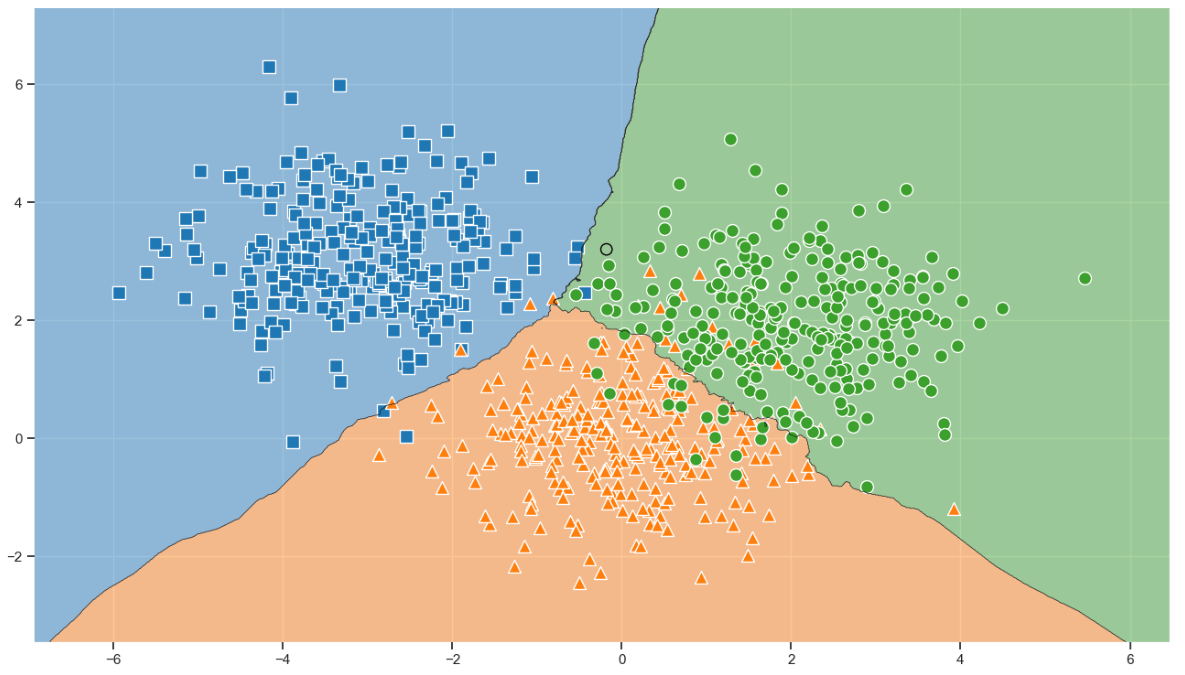
Decision Regions

Decision Boundaries

A map of different colored squares

Description automatically generated

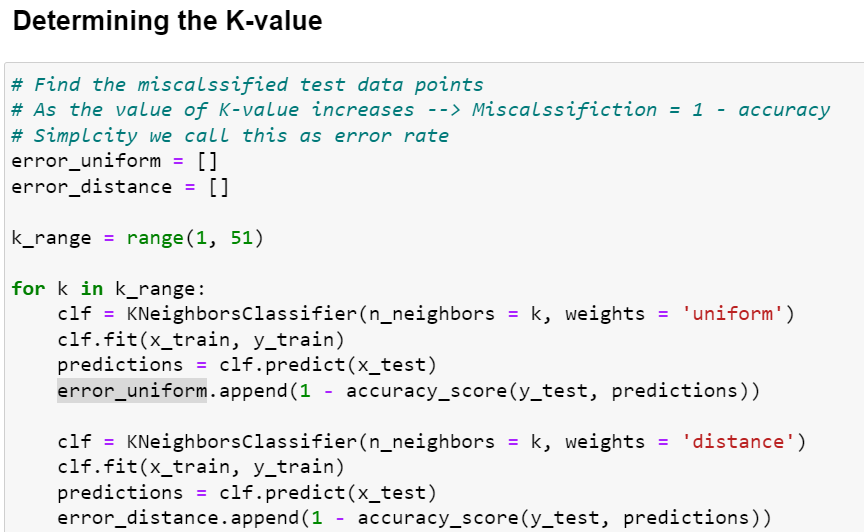
K = 30: Smoother decision boundaries. Model not affected background noise and new data points. Tendency to misclassify points. Low variance with high bias.



**Choosing the best K-Value**:

Lower value of K (K = 1) gives best results in training data but shows poor results in test data. Higher chances of Overfitting.

Higher value of K (K = 30) gives smoother decision boundaries and tendency to misclassify the points. best results in training data but shows poor results in test data. Higher chances of underfitting.

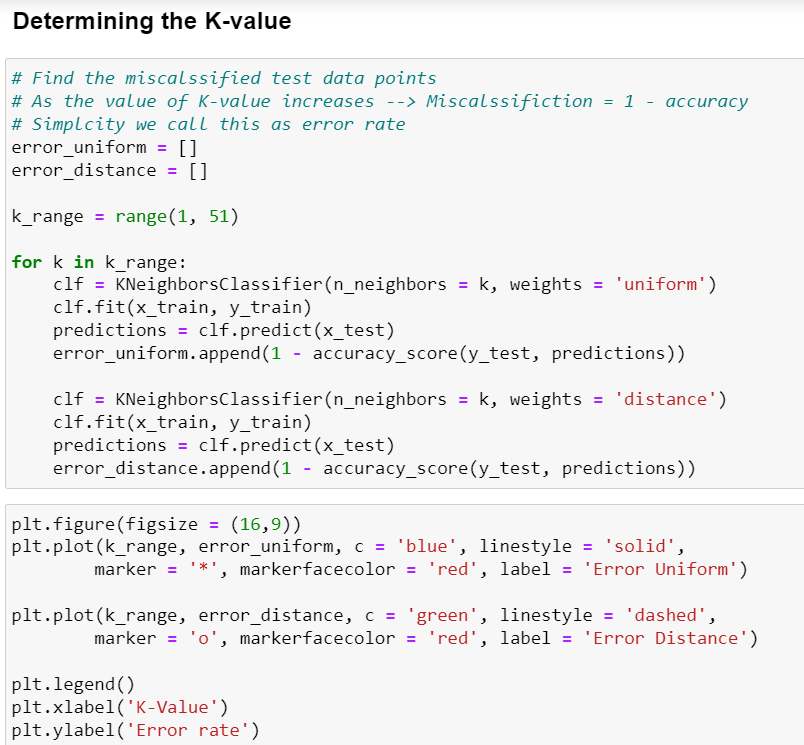


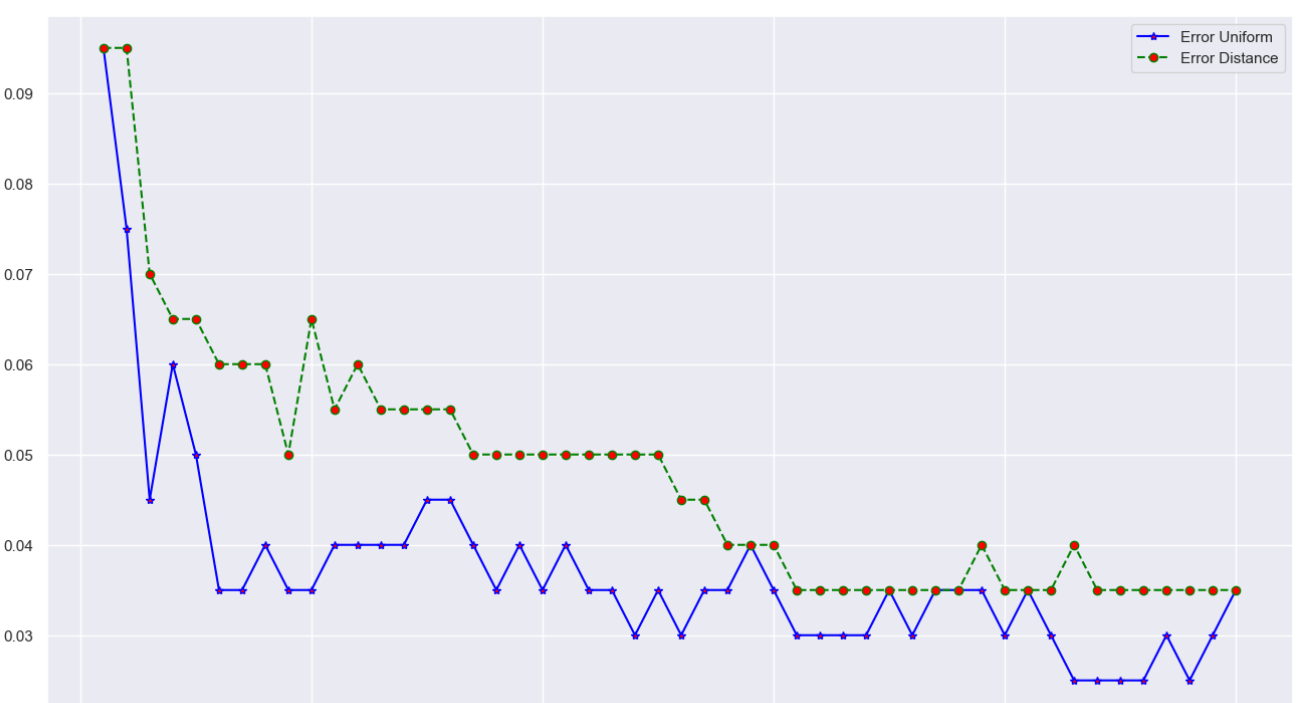
Created a range from (1, 51). Fit the model, made predictions on the test data an calculated the miscalculated rate.

K-Value must be selected neither small value to avoid high variance nor high value to avoid High bias. K- value must be selected the smallest error rate.

To identify the best K-value import the GridSearchCV from sklearn as below







Cross-Validation of the data is performed, here training data is split further into training and test data. Fine tuned

Training Data

Test Data

Data

Training Set

Validation Set

Fit the model.

Fine tuning the model

A screenshot of a computer program

Description automatically generated

This syntax creates a model of each possible combinations to fit this model as below.

|  |  |  |
| --- | --- | --- |
| # of Neighbors | Weights | Model |
| 1 | Uniform | Model1 |
| 1 | Distance | Model1 |
| 2 | Uniform | Model1 |
| 2 | Distance | Model1 |
| … | … | … |

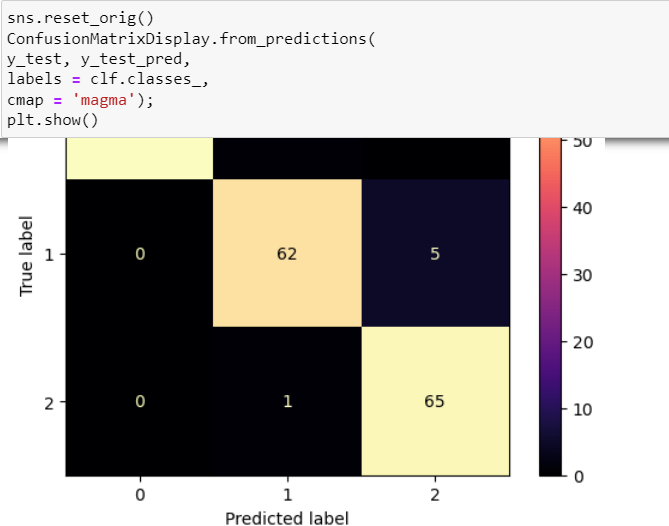
Best K parameter is 10 with “uniform” weights with score of 0.95125 (This is mean value of all accuracies obtained during cross-validation)

A screenshot of a computer

Description automatically generated

A graph showing a line graph

Description automatically generated

A chart with numbers and labels

Description automatically generated with medium confidence

0 🡪 Squares

1 🡪 Triangles (5 have been misclassified)

2 🡪 Round

A screenshot of a computer

Description automatically generated

Here is how precision, recall and F1 Score is calculated for 2

Precision = TP / (TP + FP) = 65 / (65 + 5 + 0) ~ 0.93

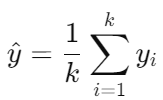
Recall = TP / (TP + FN) = 65 / (65 + 1 + 0) ~ 0.98

F1 Score = 2 \* Precision \* Recall / (Precision + Recall) = 2 \* 0.93 \* 0.98 / (0.93 + 0.98) ~ 0.96

**K-Nearest Neighbors (KNN) Regressor**: This is a type of machine learning model used for regression tasks. It is a non-parametric method, meaning it makes predictions based on the data itself without assuming a predefined form for the underlying function.

Here’s a brief overview of how the KNN Regressor works:

1. **Training Phase**:
   * The KNN regressor doesn't involve a training phase in the traditional sense. Instead, it simply stores the training data.
2. **Prediction Phase**:
   * To predict the value for a new data point, the algorithm follows these steps:
     1. **Calculate Distances**: Compute the distance between the new data point and all points in the training dataset. The most common distance metric used is Euclidean distance, but other metrics like Manhattan or Minkowski can also be used.
     2. **Find Nearest Neighbors**: Identify the K nearest neighbors to the new data point. The value of K is a hyperparameter and is usually chosen based on cross-validation.
     3. **Compute Prediction**: The predicted value is typically the average of the values of the K nearest neighbors. This can be expressed as:



where yi​ are the values of the K nearest neighbors.

**Choosing the Right Value for K**

Selecting the right value for K is crucial for the performance of the KNN regressor. Here are some tips:

* **Cross-Validation**: Use cross-validation to determine the best K. This involves dividing the data into subsets and evaluating the performance for different values of K.
* **Odd Values**: When the dataset size is small, using an odd number for K can help avoid ties.
* **Bias-Variance Tradeoff**: Smaller values of K can lead to a model with high variance (overfitting), while larger values of K can result in high bias (underfitting).

**Advantages and Disadvantages**

**Advantages**:

* Simple and easy to understand.
* No training phase required.
* Works well with smaller datasets.

**Disadvantages**:

* Computationally expensive during prediction, especially with large datasets.
* Sensitive to irrelevant or redundant features.
* Performance can degrade with high-dimensional data (curse of dimensionality).

By using the KNN regressor, you can effectively predict continuous values based on the similarity of data points. However, it's essential to carefully choose the value of K and preprocess the data to ensure the best performance.

**Key Differences between KNN Regressor vs KNN Classifier**:

**KNN Regressor**

1. **Purpose**: Used for regression tasks, where the goal is to predict a continuous output value.
2. **Output**: Produces a continuous value as the prediction.
3. **Prediction Mechanism**: The prediction is typically the mean (or sometimes the median) of the target values of the k-nearest neighbors.
4. **Evaluation Metrics**: Common metrics include Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R²).

**KNN Classifier**

1. **Purpose**: Used for classification tasks, where the goal is to predict a categorical class label.
2. **Output**: Produces a discrete class label as the prediction.
3. **Prediction Mechanism**: The prediction is the most frequent class label among the k-nearest neighbors (majority vote).
4. **Evaluation Metrics**: Common metrics include Accuracy, Precision, Recall, F1-Score, and Confusion Matrix.

**Key Differences**

1. **Nature of the Problem**:
   * KNN Regressor deals with continuous data (e.g., predicting house prices).
   * KNN Classifier deals with categorical data (e.g., classifying emails as spam or not spam).
2. **Prediction Output**:
   * KNN Regressor outputs a continuous value.
   * KNN Classifier outputs a categorical label.
3. **Aggregation Method**:
   * KNN Regressor uses statistical measures like mean or median of the neighbors' target values.
   * KNN Classifier uses majority voting among the neighbors' class labels.
4. **Evaluation Metrics**:
   * KNN Regressor is evaluated using regression-specific metrics such as MSE and MAE.
   * KNN Classifier is evaluated using classification-specific metrics such as accuracy and F1-score.

**Similarities**

1. **Data Preparation**: Both require normalization or standardization of features to ensure that the distance metric (usually Euclidean) is meaningful.
2. **Distance Metric**: Both use a distance metric (e.g., Euclidean, Manhattan) to find the k-nearest neighbors.
3. **Hyperparameters**: Both have similar hyperparameters, mainly the number of neighbors K.

In summary, while the core mechanism of identifying the k-nearest neighbors remains the same, the way predictions are made and evaluated differs significantly between KNN Regressor and KNN Classifier, tailored to suit the nature of the output (continuous vs. categorical).

**Linear Regression (Parametric Approach) vs Non-Linear Regression (Non-Parametric Approach):**

**Parametric Approach:** Assumed dependent and independent variables vary linearly. Ex: 1 unit of independent variable movement makes 1 unit of movement in dependent variable.

**Non – Parametric Approach**: Above assumption is neglected. KNN is the best example of non-parametric approach.

**Linear Regression (Parametric Approach):**

If non-parametric approach like KNN is already available, what is need of Parametric approach? Come I will take you there 😊

Below graph depicts that

|  |  |  |
| --- | --- | --- |
|  | KNN (non-parametric) | Linear (Parametric) |
| K | 10 | 1 to 40 |
| MSE | 0.03 | 0.025 (Constant across all K values) |

In KNN method, Lowest MSE is found at K = 10 and doesn’t go below 0.03 value.

However, Lowest MSE is found at K = 1 to 40 and is equal to 0.025 value. That’s why, parametric approach is better in some cases.



**Non-Linear Regression (Non-Parametric Approach):**

Below function is used as non – linear method,

**y = x2 +sin(5x)**

A screenshot of a computer program

Description automatically generated

A close-up of a math problem

Description automatically generated

A graph of a function

Description automatically generated with medium confidence



A graph of a graph of a line and a line

Description automatically generated with medium confidence

**Pros and Cons of KNN Algorithm:**

Pros:

* + 1. Easily Implemented
    2. Only 2 Parameters (No. of Neighbors and Distance Metric)
    3. Straight Forward and competes with more complicated algorithms.
    4. Perfect choice for non-linear problems (Classification or Regression)
    5. Memory efficient in fitting the process

Cons:

1. If you are solving non-parametric problem, extrapolation data cannot be trusted. KNN cannot predict the true form of the data, and it solely dependent on neighbors.
2. More data is required for good predictions.
3. Big dataset can pose the problem and takes too much memory.
4. Predictions could be computationally intensive.

**StratifiedKFold**: is a cross-validation technique in machine learning, particularly useful when dealing with imbalanced datasets. It is part of the scikit-learn library in Python. The key feature of StratifiedKFold is that it ensures that each fold of the dataset maintains the same proportion of classes as the entire dataset.

**How It Works:**

* **Cross-Validation:** 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, with each fold being used as the test set once.
* **Stratification:** StratifiedKFold ensures that each fold has a representative distribution of the target classes, which is crucial when the dataset is imbalanced (i.e., when one class significantly outnumbers the other(s)).

**When to Use:**

* When your dataset has an uneven distribution of classes (e.g., rare events in binary classification tasks).
* When you want to ensure that your cross-validation performance metrics are reliable and not skewed by the imbalance in your dataset.

**Why It's Useful:**

* In a typical cross-validation scenario, especially with an imbalanced dataset, some folds might have too few samples of a minority class. This could lead to misleading performance metrics, as the model might not have enough data to learn from. StratifiedKFold mitigates this issue by ensuring that each fold has a representative distribution of classes, leading to more reliable and generalizable model evaluation.