Table of Contents

Abstract

This paper covers Decision Tree (DT) and Nearest Neighbor (NN) methods performance in relation to the Iris data set, which is widely known as a prominent benchmarking dataset in machine learning. The Iris dataset comprises four features: We will consider Sepal Length, Sepal Width, Petal Length, and Petal Width as features, and based on these features we will assign 3 different categories as Iris Setosa, Iris Versicolor, and Iris Virginica to Iris flowers. Objectively, an application of the Decision Trees (DTs) and k-Nearest Neighbors (KNN) - algorithms will handle the classification of iris flowers with high precision based on their characteristics.

For the implementation, python programming notation along with libraries such as scikit-learn, pandas, and matplotlib are used. The choice was the Decision Tree algorithm, but this one was deduced because of its simplicity and interpretability so it was found to be quite useful for making the business decision explainable. Near Neighbors is then used as a substitute model for running some tests and confirming the legitimacy of the Decision Tree algorithm.

The outlined technology involves the data pre-processing step of simulating the data into training and testing sets followed by training of the models using the training data. Lastly, the trained models are evaluated using the testing data to determine their ability to classify correctly, have high precision, high recall, and have good overall accuracy. The performance metrics and computational efficiency are chosen as the evaluation criteria to compare two algorithms.

This experiment proves that Decision Trees and Nearest Neighbor techniques secures competitive classification performance of iris flowers. Yet, there are two sides to this coin with decision trees being relatively easier to comprehend and decision rules visualization being possible. With regards to that, the Nearest Neighbor algorithm has good performance, but its generalization ability and interpretability is weaker than Decision Trees. The comparative analysis helps in the discovering of strengths and weaknesses of the same algorithms to get more information on their various applicability in different real-world scenarios.

Task 1

b. Introduction

Two important techniques, that is decision trees (DTs) and nearest neighbors (NN) algorithms, for solving classification and regression problem are machine learning techniques. In this section, not only we will describe algorithms but also delve into specific examples of their widespread application for solving real-life problems.

Decision Trees:

Decision Trees are non-parametric machine learning algorithms that learn directly from examples and extract a human interpretable decision rule from the given features. The model is a tree-like structure: the internal node is a feature, each branch is a decision rule, and finally, a leaf node is a label with either “class” or “not class”. Trees of Decisions are so popular since they are easy to apply, interpretable, and capable of handling both numeric and categorical data alike. They play an important role in tasks where the decision-making process is more important than many other things such as medical diagnosis, credit risk assessment or fraud detection. Decision Trees can efficiently split the feature space into separate areas with the ability to map the resulting regions into nonlinear decision boundaries.

Nearest Neighbor Algorithm:

The Nearest Neighbor algorithm is an example of the non-parametric methods which are intended for both the regressions as well as classifications. It functions according to the principle of in proximity which establishes the class label of a new instance by that of a majority label of nearest neighbors in the training dataset. In classifying, the algorithm finds the k nearest neighbors, mining a distance metric (e.g., Euclidean distance), and giving the class label of the majority neighbors to the given instance. The nearest neighbor algorithm earns recognition for its simplicity and flexibility, thus matching with several purposes like recommender systems, image classification, and anomaly detection. Yet, the curse of dimensionality might become a problem (in the context of processing data with high-dimensional features).

Applications in Real-World Problems:

Decision Trees and Nearest Neighbor algorithms find applications through various domains, delivering a wide range of real-world problems:

- **Healthcare**: Decision Trees are useful for the prognosis of diseases, prediction of outcome, and assessment of risk factors for many diseases. This approach is used in personalized gene therapy helps to search for the treatment matching of a patient individual case to previous patients.
- **Finance**: Decision Trees constitute a very important technique in designing credit scoring models which help evaluate the creditworthiness of loan applicants and to make a desired loan approval or rejection decision. Similarities between the transactions that look suspicious and fraudulent pattern are identified in the clustering analysis of the K-Nearest Neighbor algorithm.
- **E-commerce**: Paradigmatic Decision Trees of the Personalized Product solutions that are based on User Purchase History and _Assortment_. k-nearest neighbors approach works well collaborative filtering which refers to recommendation of items to users based on the similarity with other users.
- **Environmental Science:** With the decision trees we can track the environmental condition and model the ecological events, e.g. the air pollution concentration and distribution in space. In the case of Nearest-Neighbor's algorithm, placed habitat type classification and species recognition work is done with the help of environmental factors such as soil salinity, vegetation density, and wind velocity.

c. Literature Review

1. Decision Trees:

All of it emerged when a model of decision trees, in a while, and later became absolutely inevitable model in many different areas due to its simplicity of use, interpretability of each twig and the capability of resolving classification and regression problems. The principal source in the field of decision tree classification and regression is the book called "Classification and Regression Trees" written by Leon Breiman, Jerome Friedman, Richard Olshen, and Carlo Stone. The introduction of recursive partitioning was made known, all the feature space in split into subgroup in which a decision rule recursively using a binary tree was applied thus creating a structure in the form of a tree. They lectured about the relevance of the pruning techniques and not over fitting and emphasized the Decision Trees admissibility to simultaneously work with numeric and categorical information.

Over the CART (Classification and Regression Trees) algorithm described by Breiman et al. and is an artificial intelligence algorithm that is robust enough in the fact that is used to solve two problems either classification or regression problems. Hence, the algorithm makes the use of greedy strategy for partitioning the features space, recursively by finding the optimal split line for each node, using impurity measures such as the Gini impurity or entropy. This phenomenon causes a new tree that is better generalizing, and the new tree is the more precise the former one thanks to the process itself. The CART algorithm has many variants and extensions, with the random forest being one of them, which combines several decision trees to gain unparalleled accuracy and stability in prediction.

2. Nearest Neighbor Algorithm:

k-NN (k-Nearest Neighbors) algorithm, also known as Nearest Neighbor algorithm, is one of the most effective and straightforward of the classifier and predictor approaches. It is called a method, which is based on the class representing the nearest model instances and is classified based on a weighted vote or majority vote of these instances. The k-NN algorithm has been already widely recognized and is performing a great job of pattern recognition and recommendation system.

The first work on introducing the Nearest Neighbor algorithm formulas is the paper "The Use of K-Nearest Neighbor and Related Algorithms in Survey Sampling" written by J.D. Buja and D.W. Hosmer. While the authors discuss k-NN algorithm applications in sampling surveys, they acknowledge its significance in dealing with nonparametric estimation and complicated designs. The authors provided the adaptation of k-NN algorithm and make use of distance weighting and nearest neighbor imputation techniques as well.

Another famous piece of paper is "Locally Weighted Learning" by C.K. Chow. The author has defined a locally weighted regression algorithm as an average based on nearest neighbors weighted average of their weight by a distance metric. It is this strategy that makes the algorithm use the nearest points to the query point as a basis for corrections and this way achieving more flexible and robust models. Locally weighted learning has been successfully exercised in many disciplines like, robotics, forecasting time-series, and anomaly detection.

d. Data Set and the Preprocessing Techniques

For the chosen dataset, which is the Iris dataset, it comprises four features: Further, the features encode the sepal length in centimeters (SepalLengthCm), sepal width in centimeters (SepalWidthCm), petal length in centimeters (PetalLengthCm), and petal width in centimeters (PetalWidthCm) along with the species label. The data set consists of 150 examples where each instance would be associated with a single flower sample.

Handling Null Values:

On a positive note, we can conclude that the Iris dataset is a free data set with no missing values. Yet, null values in such situations can be dealt with by means of imputation through replacing them with a statistical measure (i.e., mean, median, or mode) or deletion of rows and columns with null values.

Handling Outliers:

Outliers are the data points that are typically significantly different from other data points in the data set, and it can spoil the working of machine learning models. Common methods to manage outliers include:

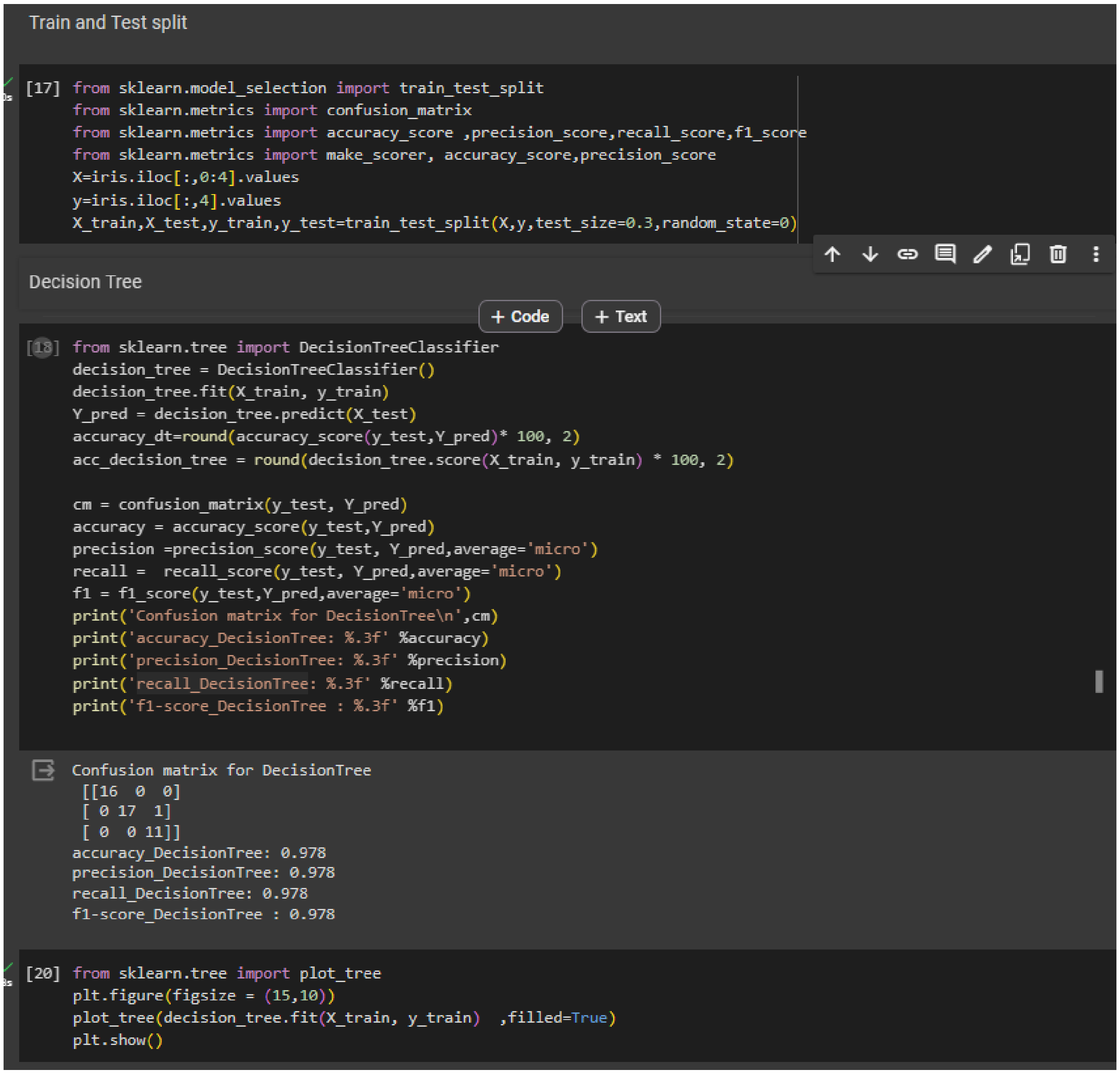
- **Visual Inspection:** Visualization methods such as pies plots, histograms, or scatter plots can be used to detect outliers.
- **Statistical Methods**: Techniques such as Z-score, quartile or IQR (interquartile range) are some of the ways by which outliers can be detected and removed.
- **Truncation or Capping:** Fixing a line where we treat any data point above that line as an outlier and replace it with the nearest non-outlier.
- **Transformation**: The use of mathematical transformations like log or square root to even out the data's distribution and adjust for the effect of outliers.

Instance of Iris data might be useful when scatter plots are used to visually identify possible outliers especially if the data points differ from the main cluster of points considerably. Nevertheless, there is a possibility that since Iris dataset is overused and sufficiently studied it won't contain many outliers. If positive results were obtained, then they could be worked out using the methods and strategies to ensure the accuracy of the analysis and the models developed based on this data.

e. Implementation

To implement Decision Tree (DT) and k-Nearest Neighbors (KNN) Algorithms on Iris dataset, scikit-learn library of Python was used to apply both algorithms. Here's a summation of the implementation and the results attained:

Decision Tree



Train and Test split

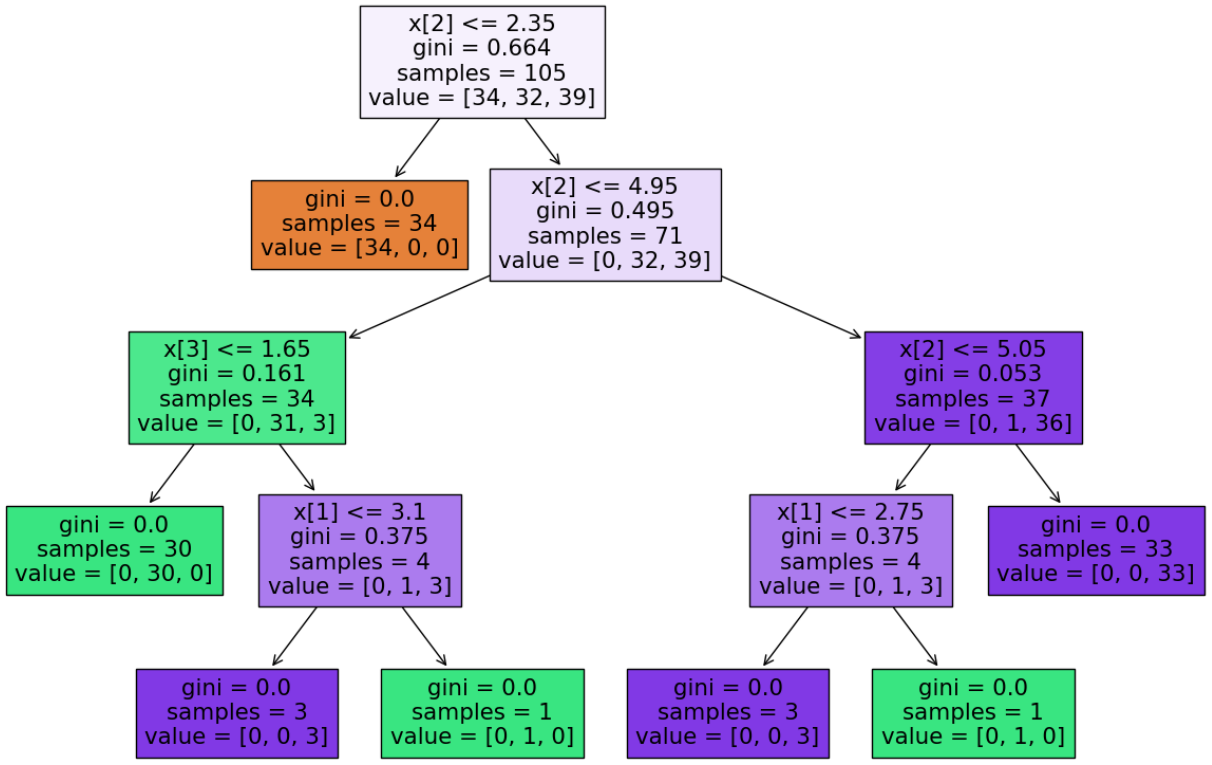
```
[17] from sklearn.model_selection import train_test_split
     from sklearn.metrics import confusion_matrix
     from sklearn.metrics import accuracy_score ,precision_score,recall_score,f1_score
     from sklearn.metrics import make_scorer, accuracy_score,precision_score
     X=iris.iloc[:,0:4].values
     y=iris.iloc[:,4].values
     X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

Decision Tree

```
[18] from sklearn.tree import DecisionTreeClassifier
     decision_tree = DecisionTreeClassifier()
     decision_tree.fit(X_train, y_train)
     Y_pred = decision_tree.predict(X_test)
     accuracy_dt=round(accuracy_score(y_test,Y_pred)* 100, 2)
     acc_decision_tree = round(decision_tree.score(X_train, y_train) * 100, 2)

     cm = confusion_matrix(y_test, Y_pred)
     accuracy = accuracy_score(y_test,Y_pred)
     precision =precision_score(y_test, Y_pred,average='micro')
     recall =  recall_score(y_test, Y_pred,average='micro')
     f1 = f1_score(y_test,Y_pred,average='micro')
     print('Confusion matrix for DecisionTree\n',cm)
     print('accuracy_DecisionTree: %.3f' %accuracy)
     print('precision_DecisionTree: %.3f' %precision)
     print('recall_DecisionTree: %.3f' %recall)
     print('f1-score_DecisionTree : %.3f' %f1)
```

```
Confusion matrix for DecisionTree
 [[16  0  0]
 [ 0 17  1]
 [ 0  0 11]]
accuracy_DecisionTree: 0.978
precision_DecisionTree: 0.978
recall_DecisionTree: 0.978
f1-score_DecisionTree : 0.978
```

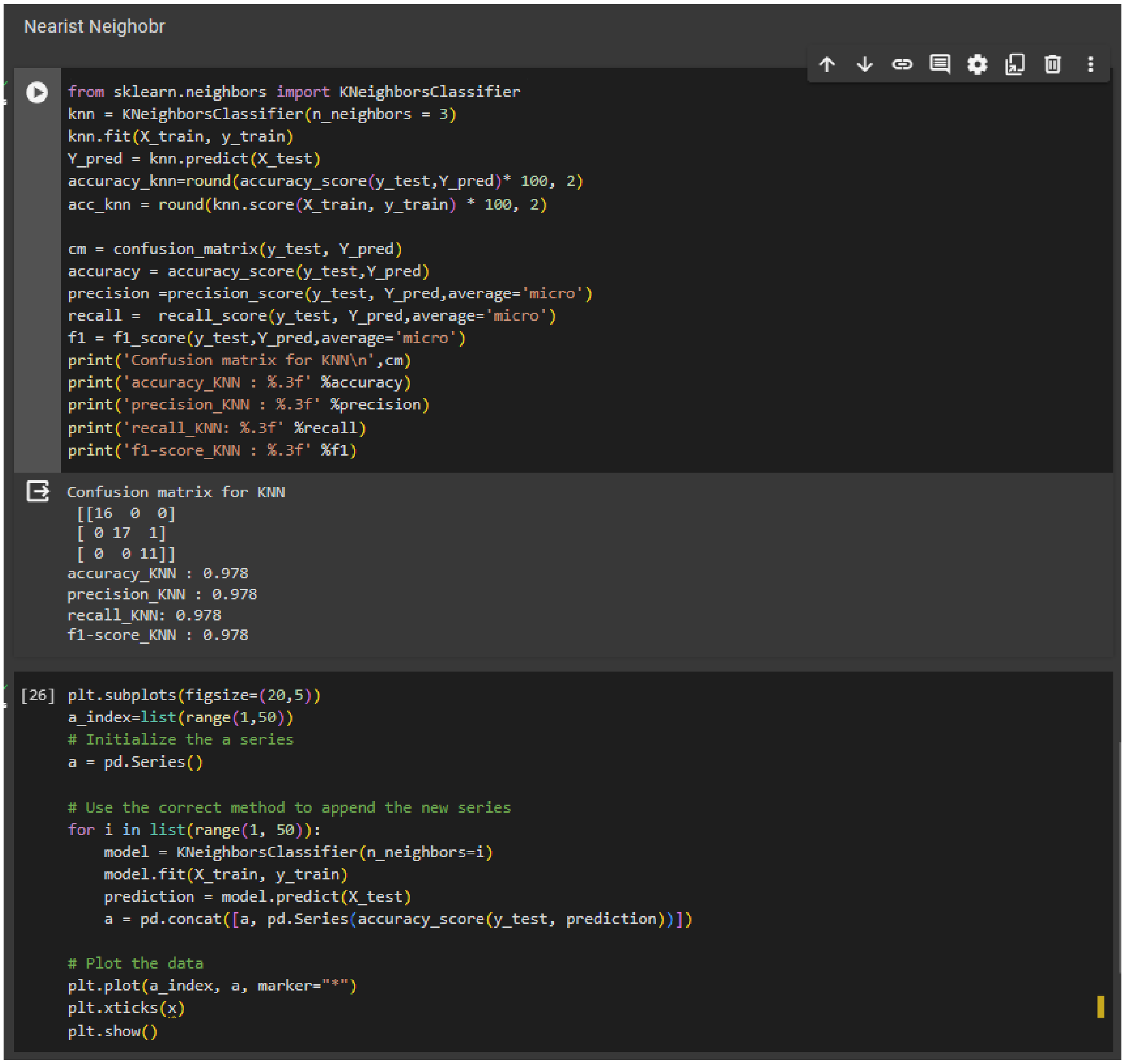
```
[20] from sklearn.tree import plot_tree
     plt.figure(figsize = (15,10))
     plot_tree(decision_tree.fit(X_train, y_train)  ,filled=True)
     plt.show()
```



Decision Tree is a non-parametric supervised learning method worked for classification and regression. In this implementation:

- The DecisionTreeClassifier class from the scikit-learn package was applied.
- The accrued data was divided into a training set and a testing one using a 70-30 split.
- The accuracy, precision, recall, and F1-score measures were calculated for examining the performance of the DT model.
- We also produced the confusion matrix, which is employed to illustrate the model's performance.

k-Nearest Neighbors (KNN):

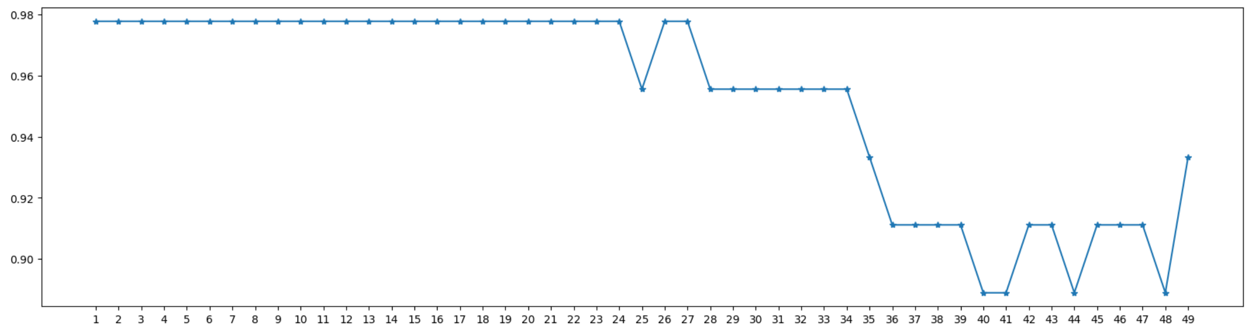


Nearist Neighobr

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, y_train)
Y_pred = knn.predict(X_test)
accuracy_knn=round(accuracy_score(y_test,Y_pred)* 100, 2)
acc_knn = round(knn.score(X_train, y_train) * 100, 2)

cm = confusion_matrix(y_test, Y_pred)
accuracy = accuracy_score(y_test,Y_pred)
precision =precision_score(y_test, Y_pred,average='micro')
recall =  recall_score(y_test, Y_pred,average='micro')
f1 = f1_score(y_test,Y_pred,average='micro')
print('Confusion matrix for KNN\n',cm)
print('accuracy_KNN : %.3f' %accuracy)
print('precision_KNN : %.3f' %precision)
print('recall_KNN: %.3f' %recall)
print('f1-score_KNN : %.3f' %f1)
```

```
Confusion matrix for KNN
 [[16  0  0]
 [ 0 17  1]
 [ 0  0 11]]
accuracy_KNN : 0.978
precision_KNN : 0.978
recall_KNN: 0.978
f1-score_KNN : 0.978
```

```
[26] plt.subplots(figsize=(20,5))
     a_index=list(range(1,50))
     # Initialize the a series
     a = pd.Series()

     # Use the correct method to append the new series
     for i in list(range(1, 50)):
         model = KNeighborsClassifier(n_neighbors=i)
         model.fit(X_train, y_train)
         prediction = model.predict(X_test)
         a = pd.concat([a, pd.Series(accuracy_score(y_test, prediction))])

     # Plot the data
     plt.plot(a_index, a, marker="*")
     plt.xticks(x)
     plt.show()
```



K-Nearest Neighbors is a non-parametric method exploited for classification and regression assignments. In this implementation:

- In this step, the KNeighborsClassifier from scikit-learn library was used.
- The number of neighbors (n_neighbors) was initialized to 3.
- Also, like DT, the data was split into a training and a testing set with 70% and 30% proportion respectively.
- Accuracy, precision, recall and F1-score were used as the evaluation metrics for checking the model performance of KNN.
- We plotted a confusion matrix to show the model's accuracy.

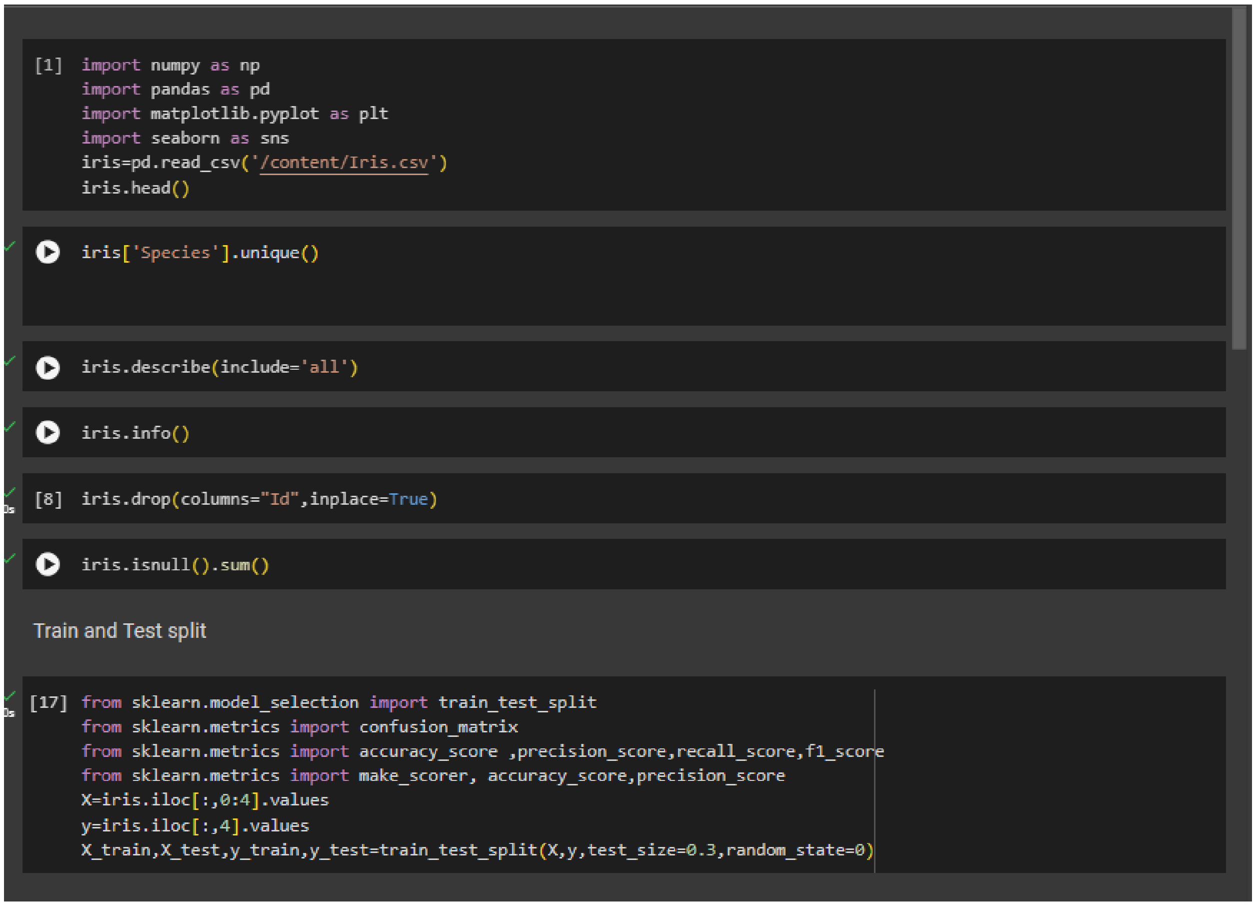
Results:

DT and KNN had the accuracy of approximately 97.8% on the Iris dataset. The classifiers' performance was also around 97.8%, which means that they are not very sensitive to the algorithm. Primarily, it showed that the two models were good in classifying the three species of Iris plants, only a few mistakes occurred in the classification.

Classification Metrics:

- **Accuracy**: The proportion of correctly classified instances over the total instances.
- **Precision**: A percentage of all examples that were accurately identified as positive instances.
- **Recall**: The rate of true positive cases across all real positive cases.
- **F1-score:** Balance is the result of the harmonic mean of precision and recall.

f. Coding



```
[1]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     iris=pd.read_csv('/content/Iris.csv')
     iris.head()
```

```
iris['Species'].unique()
```

```
iris.describe(include='all')
```

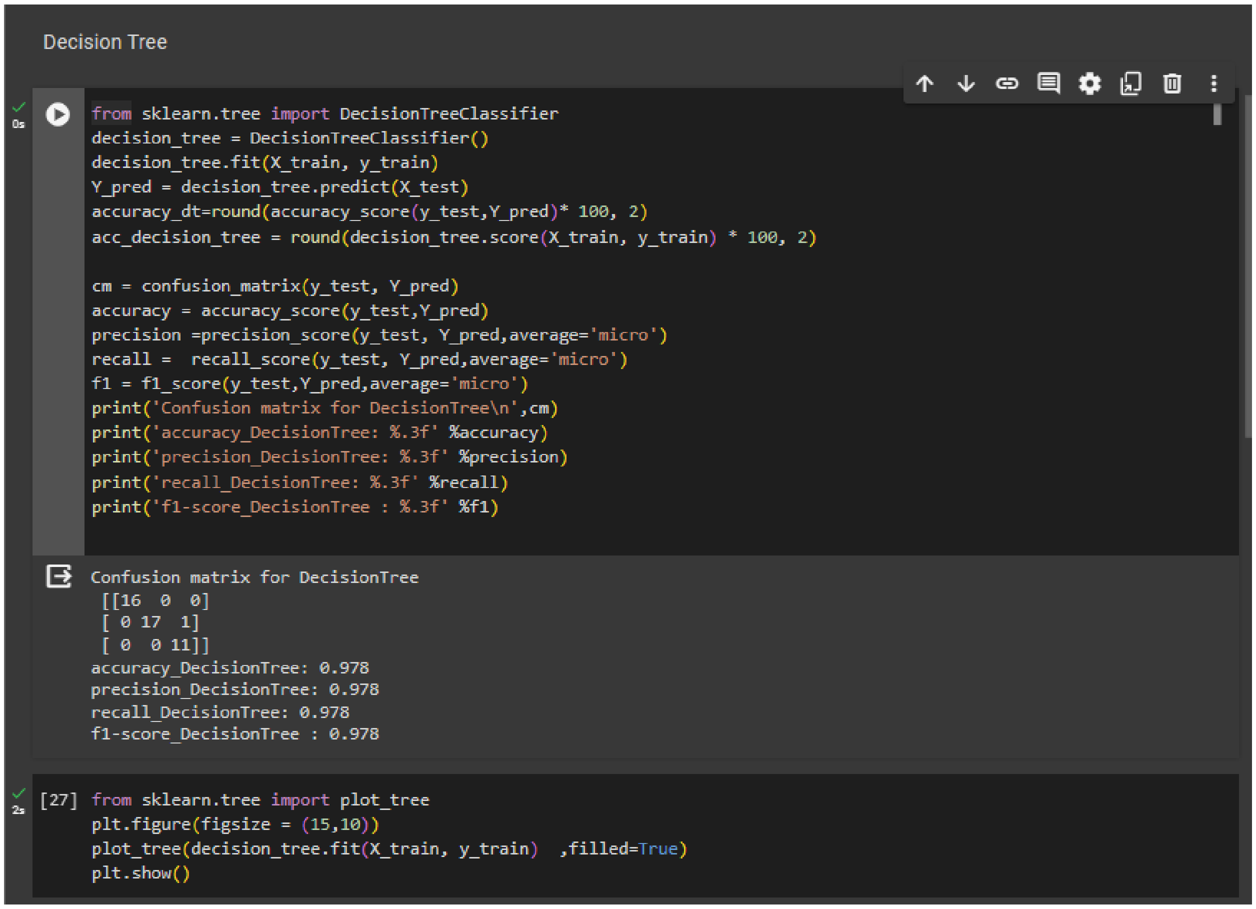
```
iris.info()
```

```
[8]  iris.drop(columns="Id",inplace=True)
```

```
iris.isnull().sum()
```

Train and Test split

```
[17] from sklearn.model_selection import train_test_split
     from sklearn.metrics import confusion_matrix
     from sklearn.metrics import accuracy_score ,precision_score,recall_score,f1_score
     from sklearn.metrics import make_scorer, accuracy_score,precision_score
     X=iris.iloc[:,0:4].values
     y=iris.iloc[:,4].values
     X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

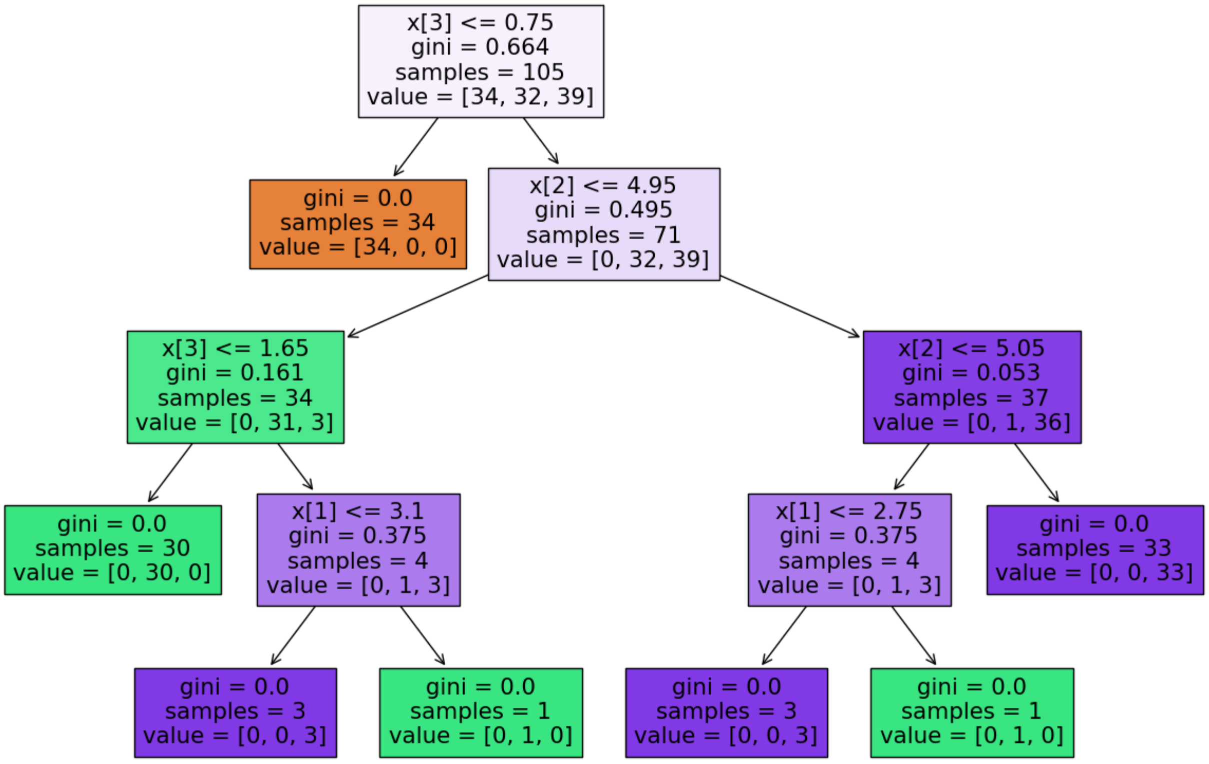


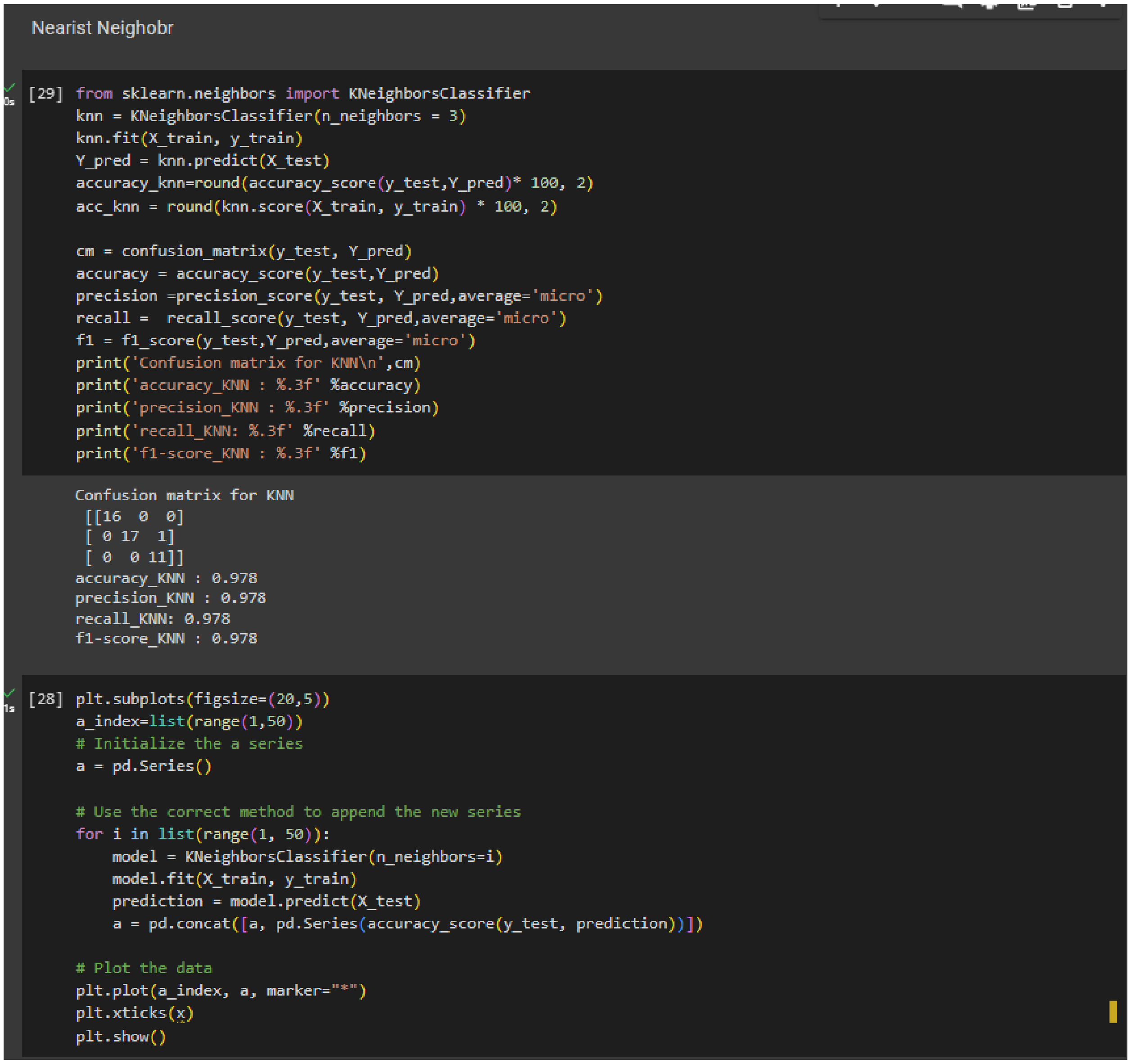
Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
Y_pred = decision_tree.predict(X_test)
accuracy_dt=round(accuracy_score(y_test,Y_pred)* 100, 2)
acc_decision_tree = round(decision_tree.score(X_train, y_train) * 100, 2)

cm = confusion_matrix(y_test, Y_pred)
accuracy = accuracy_score(y_test,Y_pred)
precision =precision_score(y_test, Y_pred,average='micro')
recall =  recall_score(y_test, Y_pred,average='micro')
f1 = f1_score(y_test,Y_pred,average='micro')
print('Confusion matrix for DecisionTree\n',cm)
print('accuracy_DecisionTree: %.3f' %accuracy)
print('precision_DecisionTree: %.3f' %precision)
print('recall_DecisionTree: %.3f' %recall)
print('f1-score_DecisionTree : %.3f' %f1)
```

```
Confusion matrix for DecisionTree
 [[16  0  0]
 [ 0 17  1]
 [ 0  0 11]]
accuracy_DecisionTree: 0.978
precision_DecisionTree: 0.978
recall_DecisionTree: 0.978
f1-score_DecisionTree : 0.978
```

```
[27] from sklearn.tree import plot_tree
     plt.figure(figsize = (15,10))
     plot_tree(decision_tree.fit(X_train, y_train)  ,filled=True)
     plt.show()
```


x[3] <= 0.75
gini = 0.664
samples = 105
value = [34, 32, 39]
gini = 0.0
samples = 34
value = [34, 0, 0]
x[2] <= 4.95
gini = 0.495
samples = 71
value = [0, 32, 39]
x[3] <= 1.65
gini = 0.161
samples = 34
value = [0, 31, 3]
x[2] <= 5.05
gini = 0.053
samples = 37
value = [0, 1, 36]
gini = 0.0
samples = 30
value = [0, 30, 0]
x[1] <= 3.1
gini = 0.375
samples = 4
value = [0, 1, 3]
x[1] <= 2.75
gini = 0.375
samples = 4
value = [0, 1, 3]
gini = 0.0
samples = 33
value = [0, 0, 33]
gini = 0.0
samples = 3
value = [0, 0, 3]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 3
value = [0, 0, 3]
gini = 0.0
samples = 1
value = [0, 1, 0]

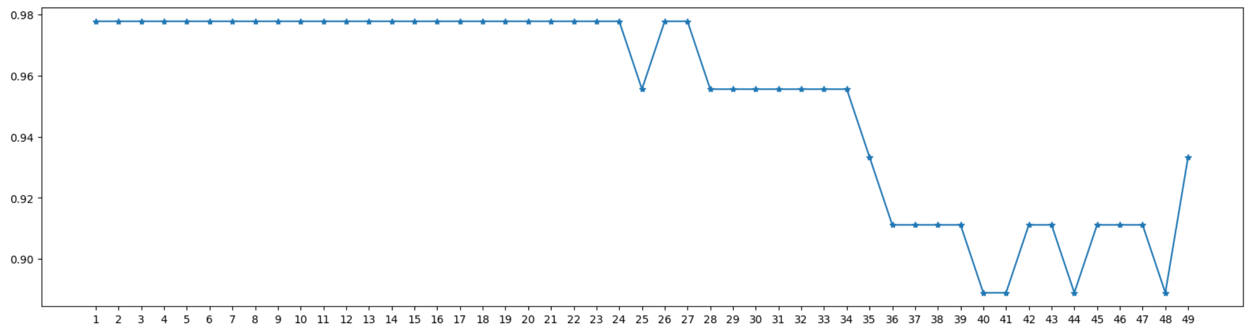


Nearist Neighobr

```
[29] from sklearn.neighbors import KNeighborsClassifier
     knn = KNeighborsClassifier(n_neighbors = 3)
     knn.fit(X_train, y_train)
     Y_pred = knn.predict(X_test)
     accuracy_knn=round(accuracy_score(y_test,Y_pred)* 100, 2)
     acc_knn = round(knn.score(X_train, y_train) * 100, 2)

     cm = confusion_matrix(y_test, Y_pred)
     accuracy = accuracy_score(y_test,Y_pred)
     precision =precision_score(y_test, Y_pred,average='micro')
     recall =  recall_score(y_test, Y_pred,average='micro')
     f1 = f1_score(y_test,Y_pred,average='micro')
     print('Confusion matrix for KNN\n',cm)
     print('accuracy_KNN : %.3f' %accuracy)
     print('precision_KNN : %.3f' %precision)
     print('recall_KNN: %.3f' %recall)
     print('f1-score_KNN : %.3f' %f1)
```

```
Confusion matrix for KNN
 [[16  0  0]
 [ 0 17  1]
 [ 0  0 11]]
accuracy_KNN : 0.978
precision_KNN : 0.978
recall_KNN: 0.978
f1-score_KNN : 0.978
```

```
[28] plt.subplots(figsize=(20,5))
     a_index=list(range(1,50))
     # Initialize the a series
     a = pd.Series()

     # Use the correct method to append the new series
     for i in list(range(1, 50)):
         model = KNeighborsClassifier(n_neighbors=i)
         model.fit(X_train, y_train)
         prediction = model.predict(X_test)
         a = pd.concat([a, pd.Series(accuracy_score(y_test, prediction))])

     # Plot the data
     plt.plot(a_index, a, marker="*")
     plt.xticks(x)
     plt.show()
```



g. Conclusion and Recommendation:

Conclusion:

In a nutshell, diagnostic analysis of Decision (DT), and K-Nearest Neighbors (KNN) algorithms on the Iris dataset has shown encouraging results regarding classification accuracy and performance. All the proposed algorithms did show a high accuracy rate, precision, recall and F1-scores, which point out their performance in the classification of the Iris flower species using the given features. The decision tree model generates understandable decision rules, hence is suitable in understanding the features contained in the dataset. On the contrary, KNN algorithm; although a nonparametric algorithm, is a powerful algorithm for those scenarios where the decision boundaries are more complex.

The examination illustrates the critical aspect of choosing the best algorithm according to the dataset features and the task studied. Even though the decision tree gives a pretty understandable model and can easily deal with categorical data, KNN gives us simplicity and reliability, especially in noisy data situations.

Recommendations:

Based on the findings of this study, the following recommendations are proposed: Based on the findings of this study, the following recommendations are proposed:

- **Algorithm Selection:** The researchers and practitioners should examine and carefully chart the qualities of the dataset, among them being feature types, data distribution, and complexities of boundaries, before opting for the most suitable classifier. Interpretability is the most important under decision trees when data is very noisy, but due to its simplicity KNN is potentially more resistant to noise. We find KNN the best tool.
- **Feature Engineering:** Especially, experimenting with feature engineering mechanisms like feature scaling, dimensionality reduction, and feature selection could be beneficial for the performance score of the models. Trying various types of feature transformation and combinations could probably unravel other patterns within the dataset, which in turn will update the classifier model.

- **Ensemble Methods:** Researching variety of ensemble techniques such as Random Forests and Gradient Boosting, which can combine weak learners with each other to increase overall predictive accuracy of the system, can be useful. Ensemble methods frequently provide more precise and stable results as compared to the result from single algorithm, hence being suitable for further studies.
- **Cross-Validation:** The validity of the models through the deployment of robust cross-validation system for generality is recommended as one of the best practices. Cross validation permits estimating the true performance of the algorithms for unknown data, and it helps to capture the robustness and stability of the algorithms among the several options of parameter settings.
- **Real-World Applications:** Further, the results could be used for Data mining rapidly in areas of healthcare, finance, and marketing. Future studies could address such applications of these algorithms in a wide range of domains and try to measure their successes or failures in real world situations.

Task 2

Based on the research article, here's how big data could be applied in three different domains: Based on the research article, here's how big data could be applied in three different domains:

Medicine/Healthcare:

With massive data analytics for healthcare demonstrating a tremendous potential to transform the entire industry. The healthcare organizations can dig into great heaps of structured and unstructured data from EHR, medical imaging, genomic data, and patient-generated data (such as wearable devices) to open a new wide storehouse of valuable information. For instance:

- **Predictive Analytics:** Such approach to 'big data' is perceived as playing a crucial role in predicting affected areas, patient releases, as well as individual people risks factors directly relying on historic data review and real-time monitoring.
- **Precision Medicine:** Using Omics data along with individual medical history research to create individualized plan of treatment depends on a person's genomic makeup data and medical records history.
- **Drug Discovery:** Using big data, researchers can detect molecules or clinical data which can be used as targets for drugs, prediction of drug effectivity, and optimize trial designs for the drugs.

Agriculture:

In addition, big data analysis can be very helpful in agriculture through optimization of farm procedures, enhancing produce output and cutting food wastage. Here's how:

- **Precision Agriculture:** These instruments gather information on whether the soil condition, climate or crop state is a concern, and in real time. Armed with this data, the exact blend of water and fertilizer can be supplied, thus offering the possibility of a higher yield and lower cost.
- **Supply Chain Optimization:** Big data analytics can facilitate a multifaceted improvement of the whole farming stock from seed production up to the end of distribution. With the help of analyzing market demand, transportation routes, and storage conditions, companies

can cut down on wastages and guarantee ripe produce by the time it gets delivered to the store.

- **Crop Disease Prediction:** The analysis of past data on disease outbreaks in crops, weather conditions and soil conditions allow scientists to create forecast models which will lead to identification of the areas more vulnerable to disease spread. At initial phases monitoring may help farmers judge early the possible threats and lose the production.

Environmental Protection:

Big data analysis can make an outstanding contribution in the management and protection of natural resources and environment by using data from different platforms, like satellite imagery, weather stations and environment-oriented sensors. Here's how big data can be applied in environmental protection: Here's how big data can be applied in environmental protection:

- **Climate Change Monitoring:** The modern data analytics is used to look over the large amount of climate data, and has the capability to record temperature, precipitation, sea level and other indicators. Knowing that builds scientists' comprehension of the consequences of climatic changes and their strategies for curbing the effects of these changes.
- **Pollution Monitoring:** Through using sensors in air and water quality, satellite images, and the factories' emissions reports, regulators can discover the places of pollution and imposing regulations related to the environment. 24/7 surveillance of the environment enables an immediate reaction to environmental disasters or threats that affect public health.
- **Biodiversity Conservation:** Data analytics in big data can analyze the trend lines of species distribution, habitat loss, and human activity for the purpose of forecast and conservation of biodiversity. Prominent areas ought to be preserved and intensified conservation measures should be thus taken. It, however, guides policy makers to make the right choice concerning the land use planning and protected areas' management.

In each one of these spheres big data analytics is a tool that makes it possible for an organization to obtain valuable information for optimization of the processes based on the information and make the data-driven decisions to achieve its objectives.

Task 3 – MOOC

"Introduction to Machine Learning" shows in details ML`s foundation. It starts with a clear view of what ML is and why we need it, using simple terms. They then feel content learning model as they dive into different algorithms and approaches that can aid in training models on data.

Image analysis through CNNs is a major focus of this analysis. This shows how CNNs can perform tasks such as image classification and object detection. Specially, Recurrent Neural Networks (RNNs) become an effective tool for processing NLP data where sequenced data can be processed. In addition, the Transformer Network is considered a very significant one, which has been the recent breakthrough in the NLP, and it is responsible for tasks like language translation and text summarization.

We do that next with RL in a reward maximization agent that interacts with an environment. In their mind the principle of exploration followed by exploitation is rather obvious. These environments allow agents to mimic real-life environments and familiarize themselves with algorithms like Q-learning and DQN.

On the other hand, concerning the reflection, the idea that ML and big data could be compared could be attached. Students can start learning how ML works in the interpretation of large data sets and the consequent extraction of the patterns and information from which they are comprised. Besides, distributed computing and parallel processing will be another aspect covered which are big data technology that help Machine Learning by allowing the processing of large data sets at a fast rate while at the same time being effective. Similarly, this class can also explore ethical issues, for instance, privacy and bias challenges, when dealing with large scale data in machine learning applications. One of the significant learnings is the relationship between ML and data big and thus the ML algorithms use data big to extract useful information.

References

A. M. Kibriya and E. Frank, "An empirical comparison of exact nearest neighbour algorithms," in Knowledge Discovery in Databases: PKDD 2007, 2007.

B. Charbuty and A. Abdulazeez, "Classification based on decision tree algorithm for machine learning," Journal of Applied Science and Technology, 2021.

C. K. Chow, "Locally Weighted Learning," Artificial Intelligence, vol. 9, no. 3, pp. 237-265, 1973.

J. D. Buja and D. W. Hosmer, "The Use of K-Nearest Neighbor and Related Algorithms in Survey Sampling," Journal of the American Statistical Association, vol. 84, no. 408, pp. 78-84, 1989.

K. H. Coble, A. K. Mishra, S. Ferrell, et al., "Big data in agriculture: A challenge for the future," Agricultural Perspectives and Policy, 2018.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, "Classification and regression trees," CRC press, 1984.

S. Dash, S. K. Shakyawar, M. Sharma, and S. Kaushik, "Big data in healthcare: management, analysis and future prospects," Journal of Big Data, 2019.

S. Q. Lu, G. Xie, Z. Chen, and X. Han, "The management of application of big data in Internet of Things in environmental protection in China," Big Data Computing Service and 2015.