Enhancing Classification Accuracy in the Iris Dataset: A Comparative Study of K-Nearest Neighbors and Decision Trees with Docker Integration

Mahnoor Yasin	Syeda Noor Fatima
Faculty of Computer Science & Engg.	Faculty of Computer Science & Engg.
AI	AI
GIK Institute of Engg. Sciences & Tech.	GIK Institute of Engg. Sciences & Tech.
Topi, Khyber Pakhtunkhwa, Pakistan.	Topi, Khyber Pakhtunkhwa, Pakistan.
mahnooryasin660@gmail.com	syedzadi.0703@gmail.com

Abstract Distinguishing the functionality of classification techniques in the ML field remains a significant factor for machine learning implementations. Nevertheless, the major gap that exists between many studies that exhaustively analyze individual algorithms is that a direct comparison of the algorithms is not inclusive using iris dataset. We dedicate this research to fill the void by a reality check, which is evaluation of the accuracy of the most popular classification algorithms K-nearest neighbors (KNN) and decision trees on the Iris dataset. The experiments pointed out that the exactism values of each algorithm are different. Decision trees showed the same accuracy, while KNN demonstrated its own strengths. However, our study may help increase the knowledge of the role of classification algorithms in comparison to other methods and may contribute to the informed decision of tasks involving machine learning.

The keywords: classifying, machine learning, k- nearest neighbors and decision trees

I: INTRODUCTION:

AI today creates a new field for the development of machine learning that serves as an elementary unit of data analytics and pattern recognition. Artificial intelligence algorithms informed with big amount of data can lead to developing models that use this well for forecasting and classifying the data big time with high precision. The major operation of machine learning is classification and this data set 'Iris' is the best example of this approach - here, variables of flower measurements are highly correlated across three species. The utilization of the one dataset as a customary benchmark favors something in that classifiers can work on it and be assessed. Classification is one of the important tasks that serve the manufacturing sectors including hospitals and banking industries; thus, they are highly dependent on accurate diagnoses and predictive outcomes that may lead to serious consequences. The Iris dataset belongs to groups of superficial data that is easy to gather and especially with labeled data used to design and even tune classification algorithms. Besides machine learning models as its application area, this effort puts these algorithms at the front situation of serving as reference on larger data sets and real-world phenomenon. Once machine-learning technology evolved to that point, the constant and simultaneous development of machine learning was the next critical concern from both the standpoint of researchers and practitioners. Currently, the data-driven world now proves that the accuracy of data is one's badge to successfully categorize and store information. Online and offline business companies being the collectors of enormous data sets, the desire for sophisticated artificial intelligence models, the ones capable of delivering human-friendly insights is on the increase. Trying to enhance a classifier's overall accuracy by paying attention to its application, the entire field will get involved, rather than just one or a few projects. As such it's more likely to obtain models that are more correct which lead to the use of technologies based on AI in areas such as personalized medicine, forecasting financial markets and this kind of decision making, where precision is of high value.

II: Problem Statement

The main research questions addressed in this study are:

Research Question 1: How does the Decision Tree perform in terms of accuracy when applied to the Iris dataset?

Research Question 2: Can K-Nearest Neighbors (KNN) algorithm improve classification accuracy on the Iris dataset when compared to KNN?

III: Our Solution:

In this report, the Iris dataset is classified, first, with the usage of Decision-tree and then, with the use of KNN, and then for both methods, a comparison of their performances is done. Our stress is on studying the algorithm being found the most accurate. We chose Python because it is the language we use for programming all the algorithms with proper debugging and ensuring the code's reliability, and for the project management we prefer using Git/GitHub. Furthermore, deployment become easier because of Docker containers that allow seamless smooth rehoming into the heterogeneous environments. We in the beginning revealed that although D-trees appeared to be promising regarding accuracy, our analysis showed that the accuracy is boosted even more by KNN.

IV: Methodology:

A: Dataset:

The Iris flower data set is a classic collection used in statistics and machine learning for exploring classification. Originally published by Ronald Fisher in 1936, the data set contains measurements of iris flowers from three species: Iris setosa, Iris versicolor, and Iris virginica.

The dataset includes four features for each flower:

* Sepal length in centimeters

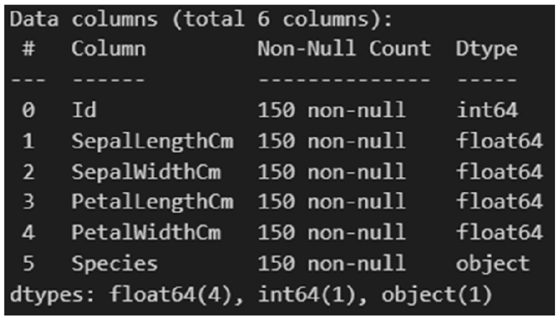
* Sepal width in centimeters

* Petal length in centimeters

* Petal width in centimeters

There are 50 samples from each of the three iris species, resulting in a total of 150 data points. One interesting aspect of this dataset is that while one species (Iris setosa) is distinct from the other two, Iris versicolor and Iris virginica have more overlap in their measurements, making them slightly more challenging to classify.

This combination of factors - manageable size, clear classification task, and some inherent complexity - makes the Iris flower data set a popular choice for introducing machine learning algorithms and data visualization techniques.



```
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
```

Figure 1

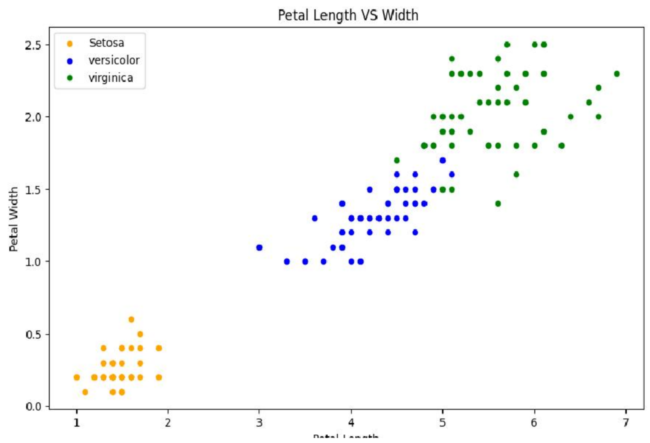


Figure 2

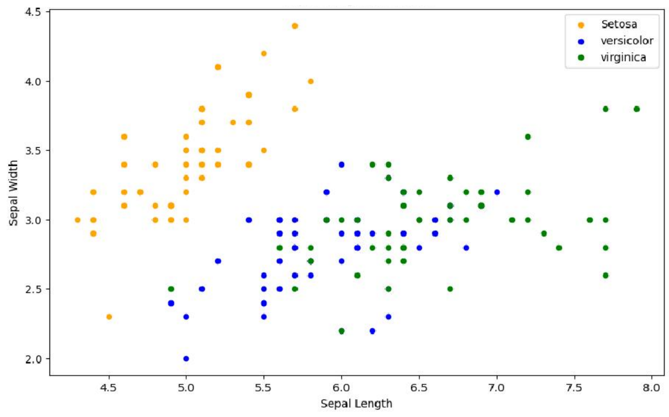


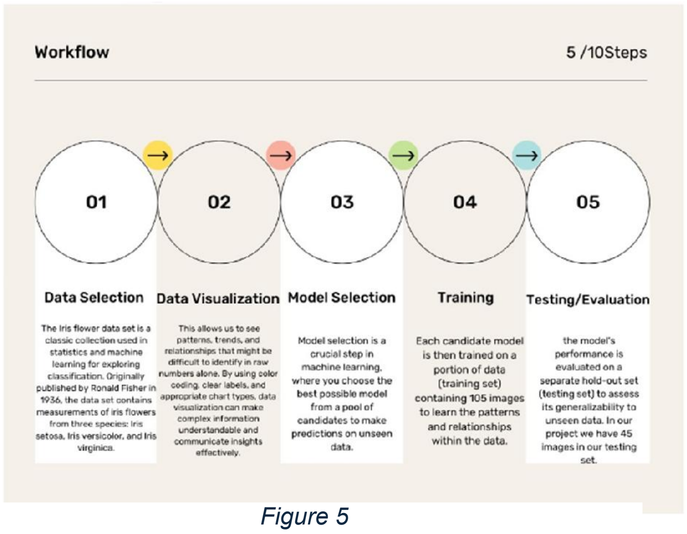
Figure 3

B: Overall Workflow:

Our methodology will be based on the straight-forward flow throughout we are going to implement and evaluate the classifier algorithms. In the first step, we load and process datasets and, then divide them into a training and a testing subset. Consequently, two machine learning algorithms—KNN and Decision Tree—are implemented to classify the Iris group for this task. We testify their performance and compared results in the basis of correctness and other parameters.

The process involves:

1. For the data set the following operations should be passed: loading the dataset and basic data cleaning.
2. Splitting the dataset into training and testing data subsets (training / testing).
3. Carrying through KNN for classification of the data that have high accuracy is our purpose.
4. The Decision Trees implementation for data classification was also completed while recording the accuracy rate.
5. Looking up the values to see if one of the algorithms performs better.
6. Working out the such code to allow for assurance and stability.
7. Leveraging the Git/GitHub for version control during the whole project development.
8. Docker containerizing is where we package our project such that the deployment and reproducibility processes can be simplified.


Workflow

5 /10Steps

01 Data Selection
The Iris flower data set is a classic collection used in statistics and machine learning for exploring classification. Originally published by Ronald Fisher in 1936, the data set contains measurements of iris flowers from three species: Iris setosa, Iris versicolor, and Iris virginica.

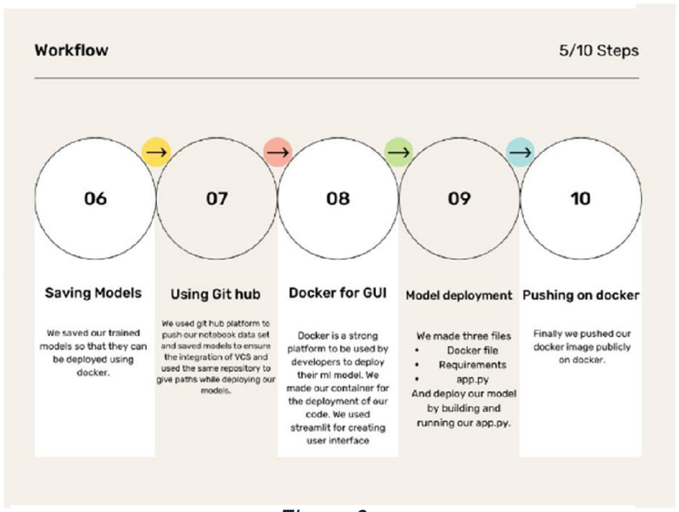
02 Data Visualization
This allows us to see patterns, trends, and relationships that might be difficult to identify in raw numbers alone. By using color coding, clear labels, and appropriate chart types, data visualization can make complex information understandable and communicate insights effectively.

03 Model Selection
Model selection is a crucial step in machine learning, where you choose the best possible model from a pool of candidates to make predictions on unseen data.

04 Training
Each candidate model is then trained on a portion of data (training set) containing 105 images to learn the patterns and relationships within the data.

05 Testing/Evaluation
the model's performance is evaluated on a separate hold-out set (testing set) to assess its generalizability to unseen data. In our project we have 45 images in our testing set.

Figure 5


Workflow

5/10 Steps

06 Saving Models
We saved our trained models so that they can be deployed using docker.

07 Using Git hub
We used git hub platform to push our notobook data set and saved models to ensure the integration of VCS and used the same repository to give paths while deploying our models.

08 Docker for GUI
Docker is a strong platform to be used by developers to deploy their ml model. We made our container for the deployment of our code. We used streamlit for creating user interface

09 Model deployment
We made three files
- Docker file
- Requirements
- app.py

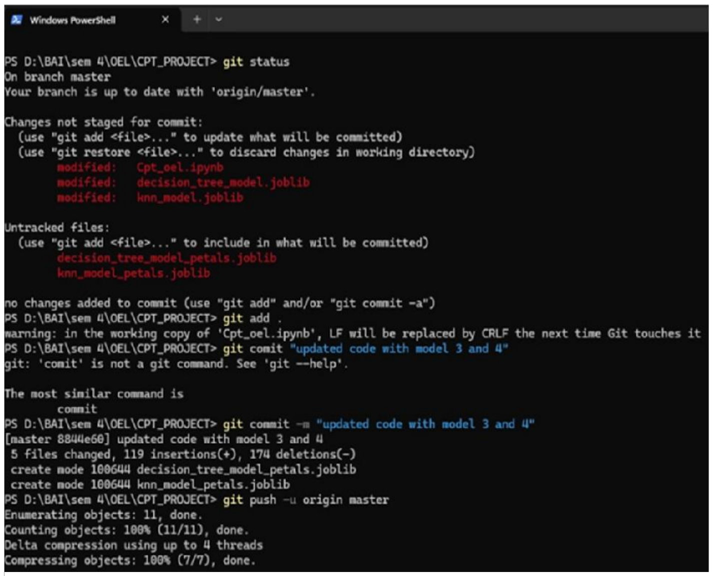
And deploy our model by building and running our app.py.

10 Pushing on docker
Finally we pushed our docker image publicly on docker.

Figure 6

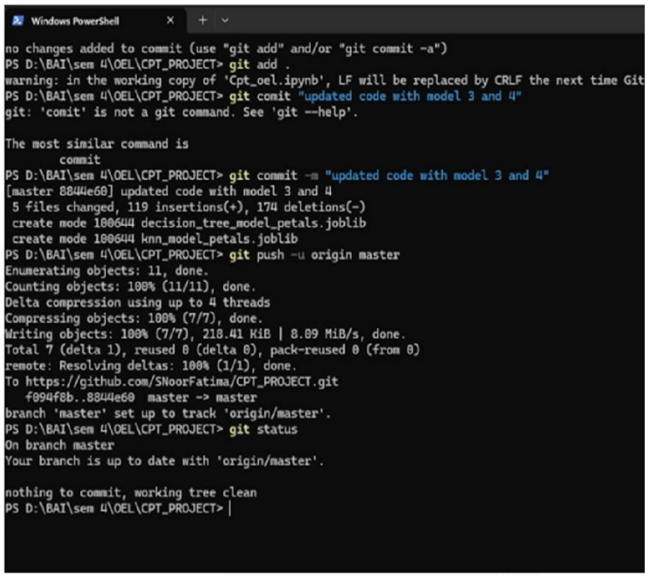
C: GitHub Integration and Version Control:

GitHub Integration and Version Control were core factors which helped develop the project as it was in progress. GitHub was the main tool we used to see changes, have a traceable history of commit and help teamwork. Our GitHub integration with the source repository was initially set up to ensure that we kept our codebase under version control. Each person had their own branches to work on the features and some fixes. The branches ensured that the main one is stable. The commit frequency was kept as reasonable as possible, commits with descriptive commit messages were made, representing changes in advancements. Besides the fact that this helped to maintain order within the entire project, I discovered that it also was a way of rolling back in the case of any suspected errors and unwanted changes. The git on GitHub made the code review process more efficient and as a result, everyone's work involved no delay. At last, after we had accomplished our task we have uploaded the code to the public GitHub repository where any user with a purpose to check out, to clone or fork the project, might have a chance to do so. Moreover, this was not only a recovery step but also was a backup just to make sure that we had saved the work for future use by others who are seeking to get an access or extension of the task.



```
PS D:\BAI\sem 4\OEL\CPT_PROJECT> git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Cpt_oel.ipynb
        modified:   decision_tree_model.joblib
        modified:   knn_model.joblib

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        decision_tree_model_petals.joblib
        knn_model_petals.joblib

no changes added to commit (use "git add" and/or "git commit -a")
PS D:\BAI\sem 4\OEL\CPT_PROJECT> git add .
warning: in the working copy of 'Cpt_oel.ipynb', LF will be replaced by CRLF the next time Git touches it
PS D:\BAI\sem 4\OEL\CPT_PROJECT> git comit "updated code with model 3 and 4"
git: 'comit' is not a git command. See 'git --help'.

The most similar command is
        commit
PS D:\BAI\sem 4\OEL\CPT_PROJECT> git commit -m "updated code with model 3 and 4"
[master 8844e60] updated code with model 3 and 4
 5 files changed, 119 insertions(+), 174 deletions(-)
 create mode 100644 decision_tree_model_petals.joblib
 create mode 100644 knn_model_petals.joblib
PS D:\BAI\sem 4\OEL\CPT_PROJECT> git push -u origin master
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 4 threads
Compressing objects: 100% (7/7), done.
```

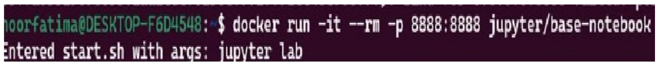
Figure 7



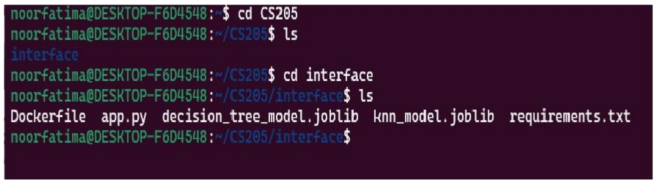
```
no changes added to commit (use "git add" and/or "git commit -a")
PS D:\BAI\sem 4\OEL\CPT_PROJECT> git add .
warning: in the working copy of 'Cpt_oel.ipynb', LF will be replaced by CRLF the next time Git
PS D:\BAI\sem 4\OEL\CPT_PROJECT> git comit "updated code with model 3 and 4"
git: 'comit' is not a git command. See 'git --help'.

The most similar command is
        commit
PS D:\BAI\sem 4\OEL\CPT_PROJECT> git commit -m "updated code with model 3 and 4"
[master 8844e60] updated code with model 3 and 4
 5 files changed, 119 insertions(+), 174 deletions(-)
 create mode 100644 decision_tree_model_petals.joblib
 create mode 100644 knn_model_petals.joblib
PS D:\BAI\sem 4\OEL\CPT_PROJECT> git push -u origin master
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 4 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 218.41 KiB | 8.09 MiB/s, done.
Total 7 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/SNoorFatima/CPT_PROJECT.git
   f094f8b..8844e60  master -> master
branch 'master' set up to track 'origin/master'.
PS D:\BAI\sem 4\OEL\CPT_PROJECT> git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
PS D:\BAI\sem 4\OEL\CPT_PROJECT> |
```

D: Docker Containerization:

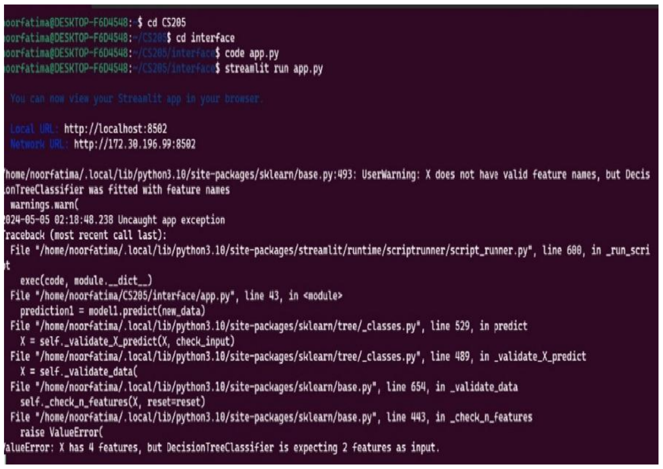
Docker Containerization was leveraged that enabled us to port the software components and detailed instructions on how to execute the project. For this purpose, we had setup a Docker in which the environment would be the same for the development, testing and in-production deployment of our AI algorithms. First, we wrote a Dockerfile that listed the base image, dependencies, and build processes to be used as a development environment. It will be explained below. Using this technique we were able to address all the needful components including Python libraries and other system dependencies such as they are in a single container. The next step was to prepare the Dockerfile, in which we defined the base image, the code, and the commands needed to start our application. After building the Docker image locally, we ensured that it could run the code without any errors. This phase enabled us to operate the container in a controlled test run which certified its conformance as intended. Finally, we assessed deployed local platforms then shifted the Docker image to Docker Hub, most used container registry, aiding easy location and deployment. In addition to securing the execution of the project, this containerization strategy enhanced our work both because of improving the consistency across all platforms and because of simplifying the collaboration through a simplified approach to share or reuse project execution setup. By opting for Docker, not only were our AI algorithms were run without any hiccups in any system that supported Docker, but also this approach gave us a big boost of flexibility and sustainability.



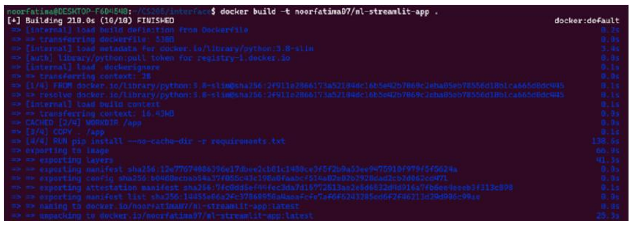
```
oorfatima@DESKTOP-F6D4548:~$ docker run -it --rm -p 8888:8888 jupyter/base-notebook
ntered start.sh with args: jupyter lab
```



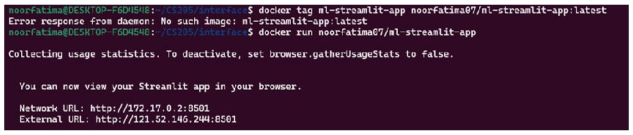
```
noorfatima@DESKTOP-F6D4548:~$ cd CS205
noorfatima@DESKTOP-F6D4548:~/CS205$ ls
interface
noorfatima@DESKTOP-F6D4548:~/CS205$ cd interface
noorfatima@DESKTOP-F6D4548:~/CS205/interface$ ls
Dockerfile  app.py  decision_tree_model.joblib  knn_model.joblib  requirements.txt
noorfatima@DESKTOP-F6D4548:~/CS205/interface$
```



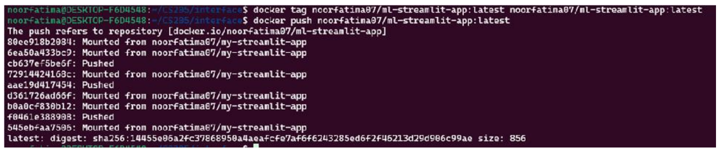
```
oorfatima@DESKTOP-F6D4548:~$ cd CS205
oorfatima@DESKTOP-F6D4548:~/CS205$ cd interface
oorfatima@DESKTOP-F6D4548:~/CS205/interface$ code app.py
oorfatima@DESKTOP-F6D4548:~/CS205/interface$ streamlit run app.py

 You can now view your Streamlit app in your browser.

 Local URL: http://localhost:8502
 Network URL: http://172.30.196.99:8502

home/noorfatima/.local/lib/python3.10/site-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but Decis
onTreeClassifier was fitted with feature names
 warnings.warn(
024-05-05 02:18:48.238 Uncaught app exception
raceback (most recent call last):
 File "/home/noorfatima/.local/lib/python3.10/site-packages/streamlit/runtime/scriptrunner/script_runner.py", line 600, in _run_scri
t
   exec(code, module.__dict__)
 File "/home/noorfatima/CS205/interface/app.py", line 43, in <module>
   prediction1 = model1.predict(new_data)
 File "/home/noorfatima/.local/lib/python3.10/site-packages/sklearn/tree/_classes.py", line 529, in predict
   X = self._validate_X_predict(X, check_input)
 File "/home/noorfatima/.local/lib/python3.10/site-packages/sklearn/tree/_classes.py", line 489, in _validate_X_predict
   X = self._validate_data(
 File "/home/noorfatima/.local/lib/python3.10/site-packages/sklearn/base.py", line 654, in _validate_data
   self._check_n_features(X, reset=reset)
 File "/home/noorfatima/.local/lib/python3.10/site-packages/sklearn/base.py", line 443, in _check_n_features
   raise ValueError(
alueError: X has 4 features, but DecisionTreeClassifier is expecting 2 features as input.
```



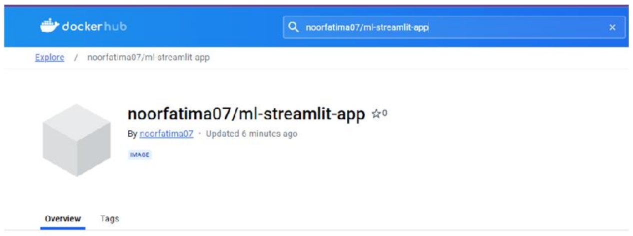
```
noorfatima@DESKTOP-F6D4548:~/CS205/interface$ docker build -t noorfatima07/ml-streamlit-app .
[+] Building 210.0s (10/10) FINISHED                                                             docker:default
 => [internal] load build definition from Dockerfile                                                       0.2s
 => => transferring dockerfile: 338B                                                                       0.0s
 => [internal] load metadata for docker.io/library/python:3.8-slim                                         3.4s
 => [auth] library/python:pull token for registry-1.docker.io                                              0.0s
 => [internal] load .dockerignore                                                                          0.1s
 => => transferring context: 2B                                                                            0.0s
 => [1/4] FROM docker.io/library/python:3.8-slim@sha256:2f911e2866173a52184dc16b5e42b7869c2eba85eb78556d18b1ca665d8dc445  0.1s
 => => resolve docker.io/library/python:3.8-slim@sha256:2f911e2866173a52184dc16b5e42b7869c2eba85eb78556d18b1ca665d8dc445  0.1s
 => [internal] load build context                                                                          0.1s
 => => transferring context: 16.43kB                                                                       0.3s
 => CACHED [2/4] WORKDIR /app                                                                              0.0s
 => [3/4] COPY . /app                                                                                      0.1s
 => [4/4] RUN pip install --no-cache-dir -r requirements.txt                                             138.6s
 => exporting to image                                                                                    66.9s
 => => exporting layers                                                                                   41.3s
 => => exporting manifest sha256:12e77674886396e17dbee2cb81c1480ce3f5f2b9a53ee9475910f979f5f5624a          0.0s
 => => exporting config sha256:b0468ecbab54a37f055c43c198a0faabcf514a02a02b2928dad2cb2d062cd471            0.0s
 => => exporting attestation manifest sha256:7fc0dd5ef49fec3da7d15772513aa2e6d6532d4d916a7fb6ee4eeeb3f313c898  0.1s
 => => exporting manifest list sha256:14455e06a2fc37868950a4aeafcfe7af6f6243285ed6f2f46213d29d906c99ae     0.0s
 => => naming to docker.io/noorfatima07/ml-streamlit-app:latest                                            0.0s
 => => unpacking to docker.io/noorfatima07/ml-streamlit-app:latest                                        25.3s
```



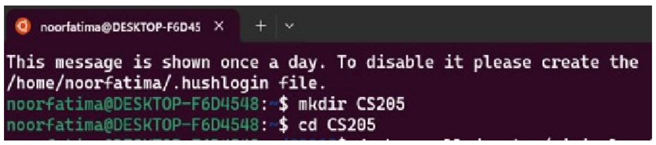
```
noorfatima@DESKTOP-F6D4548:~/CS205/interface$ docker tag ml-streamlit-app noorfatima07/ml-streamlit-app:latest
Error response from daemon: No such image: ml-streamlit-app:latest
noorfatima@DESKTOP-F6D4548:~/CS205/interface$ docker run noorfatima07/ml-streamlit-app

Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.


  You can now view your Streamlit app in your browser.

  Network URL: http://172.17.0.2:8501
  External URL: http://121.52.146.244:8501
```



```
noorfatima@DESKTOP-F6D4548:~/CS205/interface$ docker tag noorfatima07/ml-streamlit-app:latest noorfatima07/ml-streamlit-app:latest
noorfatima@DESKTOP-F6D4548:~/CS205/interface$ docker push noorfatima07/ml-streamlit-app:latest
The push refers to repository [docker.io/noorfatima07/ml-streamlit-app]
80ee918b2084: Mounted from noorfatima07/my-streamlit-app
6ea50a433bc9: Mounted from noorfatima07/my-streamlit-app
cb637ef5be6f: Pushed
72914424168c: Mounted from noorfatima07/my-streamlit-app
aae19d417454: Pushed
d361726ad66f: Mounted from noorfatima07/my-streamlit-app
b0a0cf830b12: Mounted from noorfatima07/my-streamlit-app
f0461e388908: Pushed
545ebfaa7505: Mounted from noorfatima07/my-streamlit-app
latest: digest: sha256:14455e06a2fc37868950a4aeafcfe7af6f6243285ed6f2f46213d29d906c99ae size: 856
```



docker hub
noorfatima07/ml-streamlit-app
Explore / noorfatima07/ml streamlit app
noorfatima07/ml-streamlit-app ☆0
By noorfatima07 · Updated 6 minutes ago
IMAGE
Overview
Tags



```
This message is shown once a day. To disable it please create the
/home/noorfatima/.hushlogin file.
noorfatima@DESKTOP-F6D4548:~$ mkdir CS205
noorfatima@DESKTOP-F6D4548:~$ cd CS205
```

E. De bugging in python

In Python, the debugger is a built-in module called pdb, which stands for "Python Debugger." It provides a simple yet powerful way to debug Python code interactively. Here's how you can use the debugger in Python:

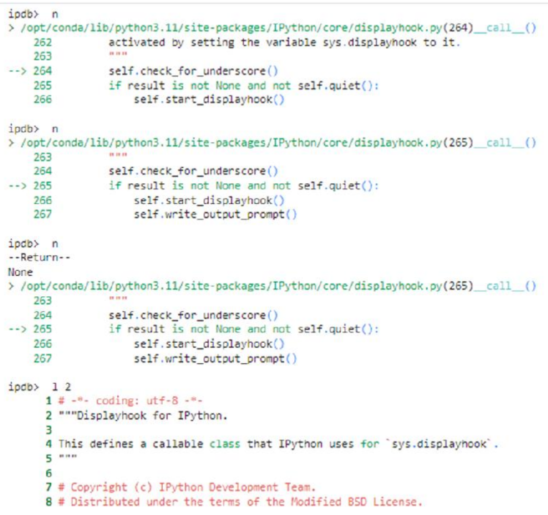
```
import pdb
```

You can set breakpoints in your code using the pdb.set_trace() function. Place this function call at the point in your code where you want the debugger to stop execution and enter interactive mode.

Execute your Python script as usual. When the interpreter encounters the pdb.set_trace() statement, it pauses execution and enters the debugger's interactive mode.

Once the debugger is active, you can use various commands to inspect the code's state, variables, and control the execution flow. Some common debugger commands include:

- n (next): Execute the current line and move to the next line in the code.
- s (step into): Step into the function call if the current line contains a function call.
- c (continue): Continue execution until the next breakpoint is encountered.
- l (list): Display the current line of code and a few surrounding lines.
- p (print): Print the value of a variable or expression.
- q (quit): Quit the debugger and terminate program execution.



```
ipdb>  n
> /opt/conda/lib/python3.11/site-packages/IPython/core/displayhook.py(264)__call__()
    262         activated by setting the variable sys.displayhook to it.
    263         """
--> 264         self.check_for_underscore()
    265         if result is not None and not self.quiet():
    266             self.start_displayhook()

ipdb>  n
> /opt/conda/lib/python3.11/site-packages/IPython/core/displayhook.py(265)__call__()
    263         """
    264         self.check_for_underscore()
--> 265         if result is not None and not self.quiet():
    266             self.start_displayhook()
    267             self.write_output_prompt()

ipdb>  n
--Return--
None
> /opt/conda/lib/python3.11/site-packages/IPython/core/displayhook.py(265)__call__()
    263         """
    264         self.check_for_underscore()
--> 265         if result is not None and not self.quiet():
    266             self.start_displayhook()
    267             self.write_output_prompt()

ipdb>  l 2
      1 # -*- coding: utf-8 -*-
      2 """Displayhook for IPython.
      3
      4 This defines a callable class that IPython uses for `sys.displayhook`.
      5 """
      6
      7 # Copyright (c) IPython Development Team.
      8 # Distributed under the terms of the Modified BSD License.
```

V: Result:

Research Question 1 was specifically developed to determine the Decision Tree performance in terms of accuracy using the Iris dataset for an input. By producing the Decision Tree, we found that indeed, its accuracy was 0. 9555555555555556. Figure 8 is given as a visualization of this fact with the modeled values and the actual ones of the three species of the Iris plants represented. The chart does also prove that Decision Tree algorithm is fairly successful with the correct classifications in the majority of the Iris species but just feel into a few misguided classifications.

The research question effecting in determining that K-NN (K-Nearest Neighbor) algorithm can furnish better classification to Iris dataset than decision tree algorithms. Our result shows that the case of KNN is 0 to be true. 9777777777.77777777, indicating that it is performing slightly better than that of the Decision Tree. In the light of the fact that KNN was more accurate than the Decision Tree in dealing with the given dataset, it demonstrates that KNN is more effective. Besides, in regard to the precision that is relevant to the algorithm itself, KNN using Petals is the same and stays fixed at 0. Using the flow character of broad-scale regions yielded a higher accuracy rate of 0.77, while via Sepals a lower accuracy score of 0.27 was returned. 7333333333333333. Shown in figure 9.

In general, our findings highlight the fact that both Decision Tree and KNN models are robust classifiers capable of differentiating between Iris species. While KNN gives an even higher accuracy, we can see that the Decision Tree provides a stronger algorithm thanks to its accuracy that is not sensitive to the way we feature some of the features. These results clearly state the crucial point of the selection of the specific algorithm being targeted upon the possessed characteristics of the dataset, and being aware of the goal. More research could run on top of the methods of improvement to get better outcomes of both algorithms and : if they can do well to the given datasets and domains.

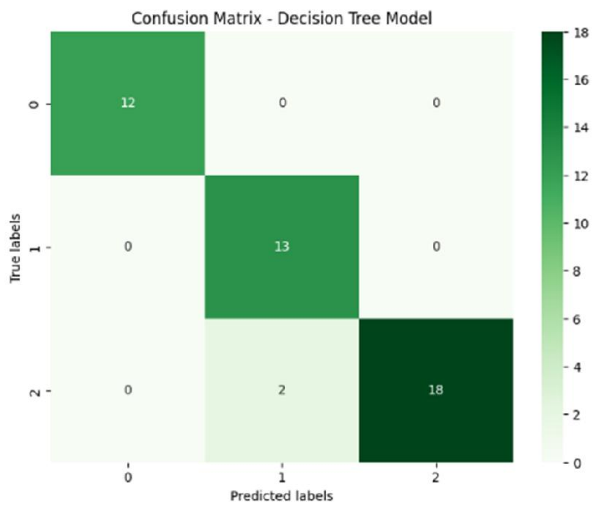


Figure 8

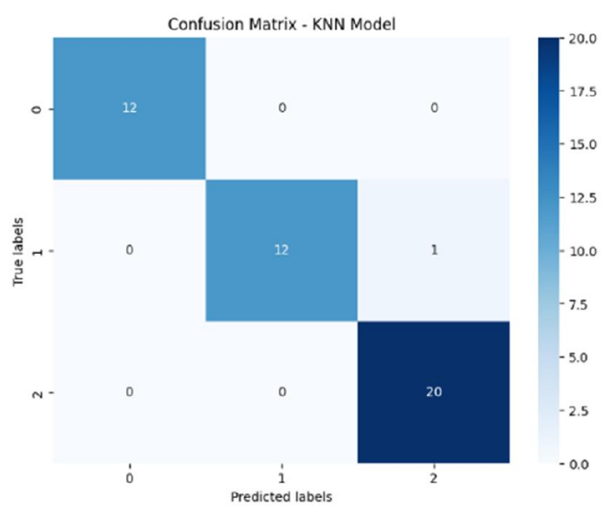


Figure 9

Model	Overall Accuracy	Accuracy with Petals	Accuracy with Sepals
KNN	97.78%	97.78%	73.33%
Decision Tree	93.33%	95.56%	64.44%

VI: Discussion;

From the results, it's evident that both KNN and Decision Tree classifiers perform better when using petal features compared to sepal features. The KNN classifier achieves the highest accuracy of 97.78% when using petal features, suggesting that petal measurements are more effective in distinguishing between different plant species. Similarly, the Decision Tree classifier also shows higher accuracy when using petal features compared to sepal features.

However, when using sepal features alone, both classifiers experience a decrease in accuracy. The KNN classifier's accuracy drops to 73.33% when using sepal features, indicating that sepal measurements alone are not as effective in classification. Similarly, the Decision Tree classifier's accuracy decreases to 64.44% when using sepal features, further highlighting the importance of using petal measurements for better classification performance.

These results underscore the significance of feature selection in achieving accurate classification results. Petal measurements emerge as crucial features for both KNN and Decision Tree classifiers, emphasizing the importance of selecting appropriate features for optimal classifier performance.

VII: Conclusion:

Through the experiments, with the K-Nearest Neighbors and Decision Tree algorithms on the Iris dataset, the significance of the performance as well as the capability of these algorithms intended for the classification of task have been observed. The conclusions shown the way that KNN, amongst them the ones using petal size, proved to be very high accurate, surpassing Decision Trees. The obtain result from this typifies that there are certain metrics, for example, petal size that bring clear differentiation between types of iris flower and thus successful classification results. Both Decision Trees and HR-Vs assisted good models overall. But it was observed that in the case of Decision Trees, feature selection had a far more dramatic impact on accuracy and their models just did not perform well with sepal measurements only. In this process of voting classifier of the model was used, but with that additional accuracy was not increased, which shows that there is no guarantee of receiving better quality materials integrating additional complexity. The result of these tests brought the significance of feature selection to light as well as proved that the accurate model for which the right features are selected provides high performance. Moreover, the integration of Docker and GitHub made the development process stronger, as it helped to keep the working environment dependable and ensured the availability of versioning. Through this work, the importance of such a feature engineering is illustrated as well as the given KNN results' reliability in classification tasks when configured properly through the step-by-step presented process.