Northeastern

College of Professional Studies
Northeastern University San Jose

MPS Analytics

Course: ALY6020

Assignment:

Module 1 – Midweek Project

Submitted on:

September 25, 2023

Submitted to:

Professor: Ahmadi Behzad

Submitted by:

Heejae Roh

Introduction

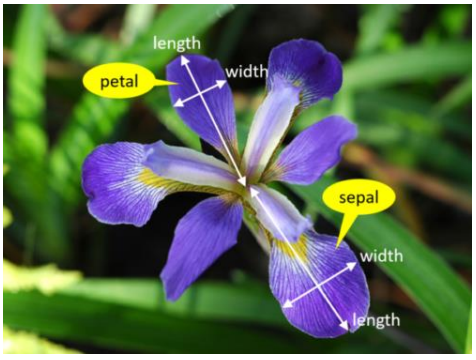
The iris data includes the Length and Width of the petal and sepal of the iris flower, respectively, and includes the type of iris. Nearest Neighbor Modeling is lazy machine learning and is one of the simplest machine learning algorithms. Nearest Neighbor is a supervised model and is a method of predicting the category of new data based on the points located closest to it. Nearest Neighbor use k number of points, then we could say it as K Nearest Neighbor model, often referred to as KNN. Since there is no learning process, the process can be applied directly without splitting train and test datasets, but sometimes train data can be used for verification. The code provided in this module uses the train and test datasets separately. I will run three more KNN modeling based on this code and think about how to measure the KNN model and performance.

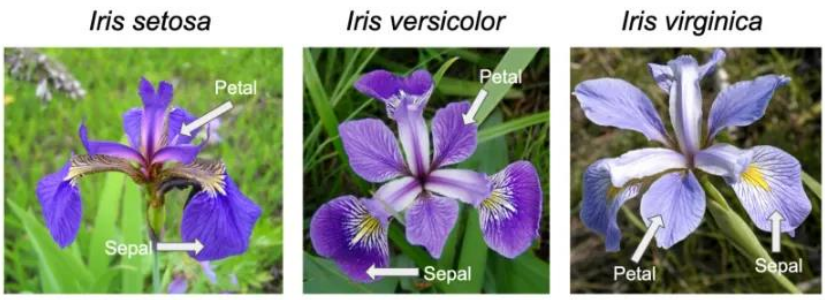
Dataset Understanding & Data Splitting

The dataset on iris is inherent in Python. There are 5 columns and 150 rows. Iris dataset is the Hello World for the Data Science, so if you have started your career in Data Science and Machine Learning you will be practicing basic ML algorithms on this famous dataset. Iris dataset contains five columns such as Petal Length, Petal Width, Sepal Length, Sepal Width and Species Type. Iris is a flowering plant, the researchers have measured various features of the different iris flowers and recorded digitally (kashishlohiya5, 2023).

Description of the variables/features in the dataset.

No.	Feature	Dictionary
1.	Sepal length (cm)	Sepal: each of the parts of the calyx of a flower, enclosing the petals and typically green and leaflike. Length: the length from the center of the flower to the end of the sepal Width: Perpendicular to Length, wide length
2.	Sepal width (cm)	
3.	Petal length (cm)	Length: the length from the center of the flower to the end of the sepal Width: Perpendicular to Length, wide length
4.	Petal width (cm)	
5.	Species	Three types of Iris: Sentosa, Versicolor, and Virginica





Data Splitting

1. Rather than using a library to share the data, I will share it directly.
2. The code provided by the module shows 12 test samples, but I will set it to 30, which is 20% of the total (150).
3. Ultimately, 150 rows are divided into 30 test data and 120 train datasets.

Exploratory Data Analysis

Headtail of Dataset

	sepal length	sepal width	petal length	petal width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
…			…		…
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Descriptive Analysis of Dataset

	sepal length	sepal width	petal length	petal width
count	150	150	150	150
mean	5.84	3.06	3.76	1.20
std	0.83	0.44	1.77	0.76
min	4.3	2.0	1.0	0.1
25%	5.1	2.8	1.6	0.3
50%	5.8	3.0	4.35	1.3
75%	6.4	3.3	5.1	1.8
max	7.9	4.4	6.9	2.5

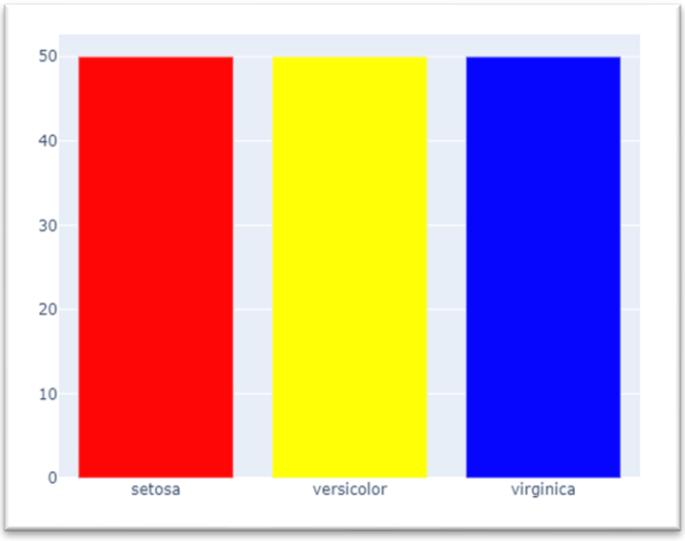
information of Dataset

#	Column	Non-Null Count	Dtype
0	sepal length (cm)	150	float64
1	sepal width (cm)	150	float64
2	petal length (cm)	150	float64
3	petal width (cm)	150	float64
4	species	150	category

1. A total of 150 pieces of data from 0 to 149 are composed of 4 columns of numeric data, each containing 2 lengths and 2 widths. The species column is the category that contains the species.
2. In descriptive analysis, you can see the meaning and distribution of each column. Sepal length is the longest on average at 5.84, followed by petal length, sepal width, and petal width.
3. Since the minimum value of sepal length is greater than the maximum value of petal width, the two can be said to be clearly distinguished.
4. We can see that the sepal width and petal length are within a relatively similar range.
5. The data type of the four columns for length is ‘float64’, a number including decimal points, and the species is displayed as a category.

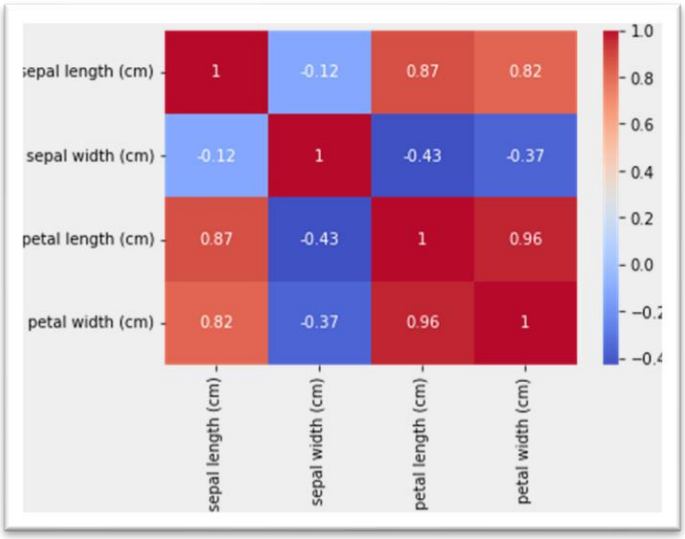
Data Visualizations

Each Number of Category:



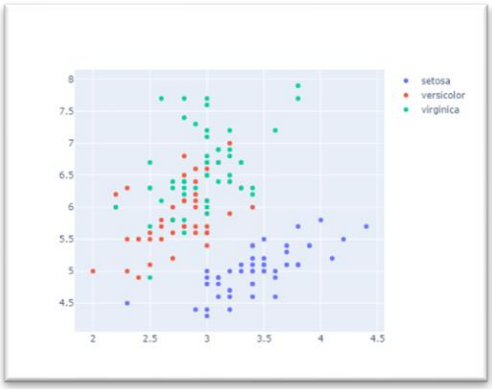
The number of three categories is 50 each, which is exactly balanced data.

Heatmap of Dataset:

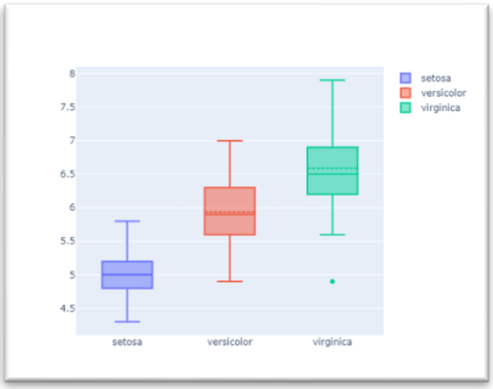


This heatmap shows that red color indicates more relevance and darker blue indicates less relevance. Petal length and petal width have the largest positive correlation, 0.96. Next, petal length and sepal length show a positive correlation of 0.87. On the other hand, petal length and sepal width show the smallest negative correlation at -0.43, and petal width and sepal width show the next smallest correlation at -0.37. The smallest absolute values of correlation are sepal width and sepal length, which can be said to have the lowest multicollinearity.

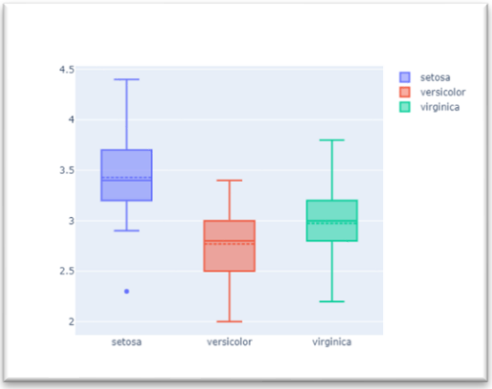
Sepal Scatter plot:



Sepal Length box plot:

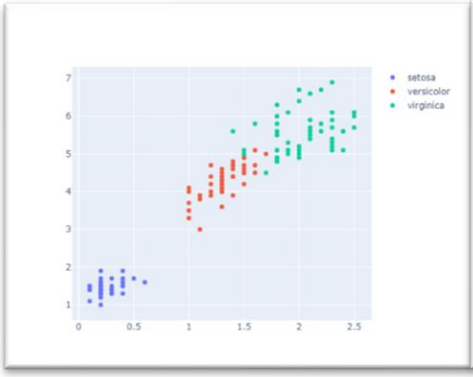


Sepal Width box plot:

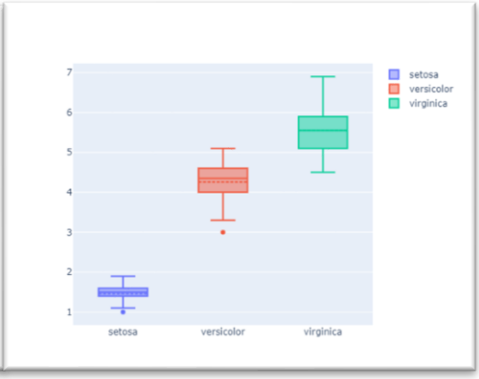


These graphs represent information about Sepal. In a scatter plot, the y-axis represents length, and the x-axis represents width. In the 'Sepal Scatter plot', you can see that setosa is relatively far away from other points, but versicolor and virginica are located close together. In the 'Sepal Length' box plot on the right, the three categories show differences, but in the 'Sepal Width box plot' at the bottom, you can see that versicolor and virginica are in a similar range.

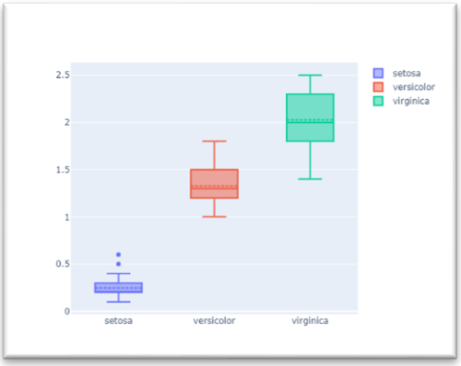
Petal Scatter plot:



Petal Length box plot:



Petal Width box plot:

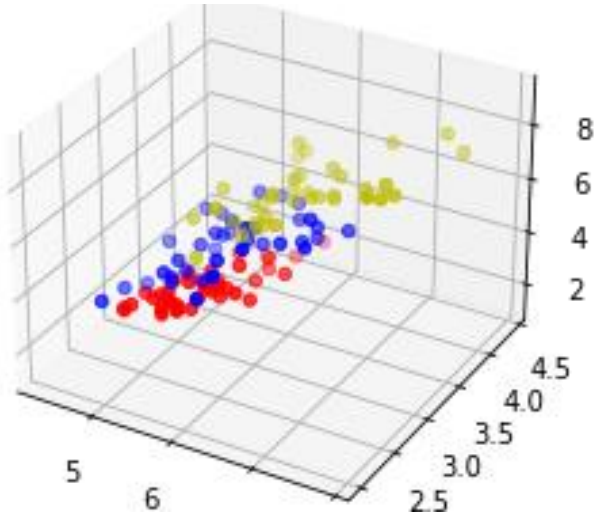


These graphs represent information about Petal. In a scatter plot, the y-axis represents length, and the x-axis represents width. In the 'Petal Scatter plot', you can see that setosa is relatively far away from other points, at the same time, versicolor and virginica are located quite far away from each other compared to ‘Sepal Scatter plot’. In the 'Pepal Length' box plot on the right, the three categories show differences, and also, in the 'Petal Width box plot' at the bottom, you can see that three categories do not share the box plot range a lot, especially in setosa.

Result of KNN & Interpretation

All of four modeling I used k=3. I left 30 pieces of data (20% of the total) as the test set and 120 pieces as the train set. First, we will run KNN using all columns using the code presented in the module. From the second time, we will test after standardizing the columns. Although we don't think it will have much of an effect because our data doesn't have large numbers or large range differences between columns, we will proceed thinking of practicing standardization in Python. Second, we will proceed with KNN using only the width and length of the petal, where the data was clearly distinguished in the visualization. Third, let's compare the results after running KNN with the width and length of sepals, which had similar categories in visualization.

Result 1: module code (using all, four column)
Scatter 3D plot (Trainset's Sepal Length, Sepal Width, Sum of Petal Length & Width)



1. This graph has three axes, and as written in the title, the first axis is Trainset's Sepal Length, the second axis is Sepal Width, and the third axis is Sum of Petal Length & Width.
2. The reason for using Sum is presumed to be to reduce dimensionality.
3. You can see that the three categories overlap quite a bit.
4. As seen in the scatter plot, setosa appears to be slightly distant from other categories, and versicolor and virginica appear to be quite close.

K-Nearest Neighbor Modeling Result 1: Using all (four) column

index	result of votes	label	data	correct
0	1	1	[6.1 3.0 4.6 1.4]	True
1	0	0	[4.5 2.3 1.3 0.3]	True
2	1	1	[6.6 2.9 4.6 1.3]	True
…		…		…
26	1	2	[4.9 2.5 4.5 1.7]	False
27	0	0	[5.8 4.0 1.2 0.2]	True
28	1	1	[5.8 2.6 4.0 1.2]	True
29	2	2	[7.1 3.0 5.9 2.1]	True

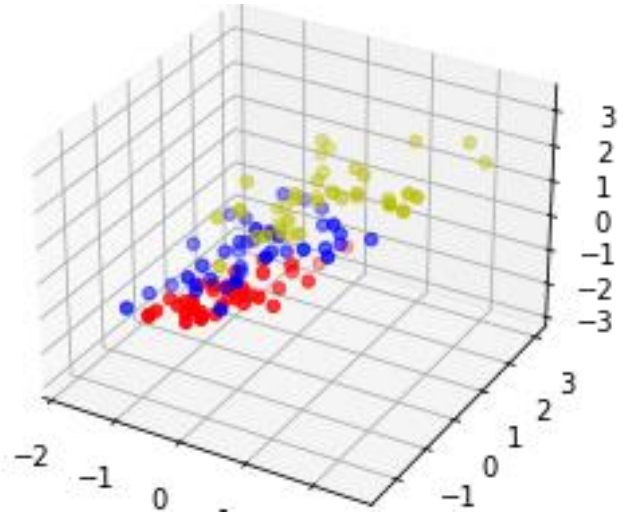
Number of correct predictions (True): 29
Number of incorrect predictions (False): 1
Accuracy: 0.97

1. The table above shows the results of the KNN model using all columns (Sepal Length & Width, and Petal Length & Width).
2. Testset has 30, which is 80% of the total data. The result of vote determines the labels of neighbors

using k=3, that is, three points arbitrarily set in the function. label is the label given to the testset.

3. The two columns are divided into three categories with numbers from 0 to 2.
4. The ‘data’ shows the data used for this.
5. Lastly, I added separate codes for correct, 'number of~', and 'accuracy' below. I checked whether the vote and label matched using KNN and output True if they matched. I counted the numbers of True and False and then calculated the Accuracy. At this time, because there are three categories, only the overall accuracy could be obtained, and if you look at each category, other performance analyzes can also be performed.
6. The index 26 was voted as (1, versicolor), but it was (2, virginica). This confirms that setosa, which we confirmed in the visualization, is relatively distant from the others, but versicolor and virginica are sometimes difficult to distinguish with our length & width data.
7. I can confirm that False appear only in categories 1 and 2.

Result 2: with Standardization (using all, four column)
Scatter 3D plot (Trainset’s Sepal Length, Sepal Width, Sum of Petal Length & Width)



1. This was visualized after standardizing all the data using StandardScaler in the 'sklearn.preprocessing' library.
2. There is more data gathered between -1 and 1 than in the previous graph. Setosa (red) and other categories seem relatively close.

K-Nearest Neighbor Modeling Result 2: with Standardization Using all (four) column

index	result of votes	label	data	correct
0	1	1	[0.36 -0.62 0.58 0.04]	True
1	0	0	[-0.12 1.65 -1.11 -1.13]	True
…		…		…
26	1	2	[1.08 -1.30 1.199 0.81]	False
27	0	0	[1.20 0.29 1.26 1.46]	True

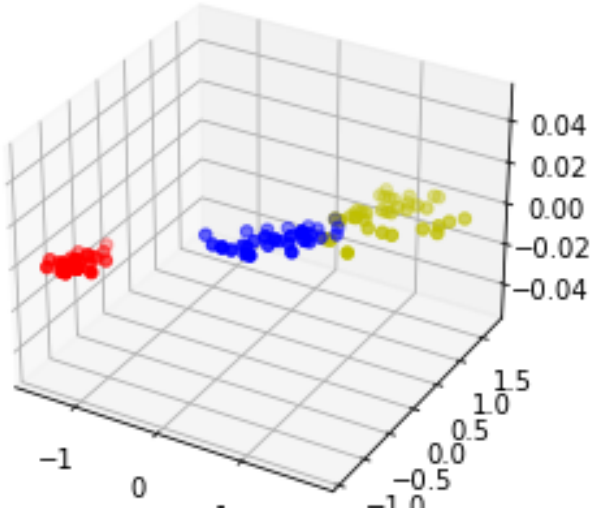
Number of correct predictions (True): 29
Number of incorrect predictions (False): 1
Accuracy: 0.97

1. The results were the same as before when all columns were used without standardization.
2. Since standardization is a process of processing and reflecting existing data, in this case, the range of width and length is not wide, only 4 columns are used, and there is not much data in total (150), so it is believed that it did not have a significant impact.
3. KNN is a very popular algorithm based on distances (typically Euclidean distance). The prediction considers the k nearest neighbors to a given point in the space of the features. Just like SVM, even

KNN requires working with normalized data (Gianluca, 2022).

Result 3: Using two Petal column only with standardization

Scatter 3D plot (Petal Length and Width)



1. It uses only two columns of Petal; it can be expressed in two dimensions. However, to utilize existing code and compare with other graphs, I will make it appear as if it appears on a 3D plane.
2. The data is more clustered between -1 and 1. In particular, the petal width looks relatively denser because the minimum was 0.1 and the maximum was 2.5. Petal was the factor that best distinguished the three categories, so you can see that the dots appear far apart.

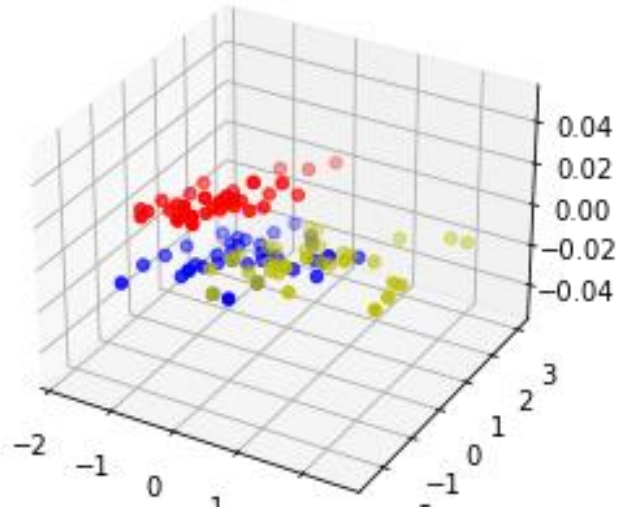
K-Nearest Neighbor Modeling Result 3: with Standardization Using Petal

index	result of votes	label	data	correct
0	1	1	[0.58 0.04]	True
1	0	0	[-0.11 -1.13]	True
2	1	1	[1.82 1.46]	True
…		…		…
15	2	1	[0.58 0.55]	False
…		…		…
26	1	2	[1.20 0.81]	False
27	0	0	[1.26 1.46]	True

Number of correct predictions (True): 28
Number of incorrect predictions (False): 2
Accuracy: 0.93

1. This is the result when only petal length and width are used.
2. The number of incorrectly predicted rows increased by one. The accuracy has become lower than before.
3. I thought petal was the factor that could best classify categories, so I thought the same results might come out, but the results were worse than when I used more columns.
4. However, it was not significantly different from the existing results, and it was found that Petal acted as an important KNN voter.
5. I can confirm that all False appear in categories 1 and 2.

Result 4: Using two Sepal column only with standardization
Scatter 3D plot (Sepal Length and Width)



1. It uses only two columns of Sepal; it can be expressed in two dimensions. As with Petal, I used existing code for continuity and comparison.
2. The data is more clustered between -1 and 1. Sepal shows some overlap as the two columns seem to have quite a bit of gray area except for setosa.

K-Nearest Neighbor Modeling Result 4: with Standardization Using Sepal

index	result of votes	label	data	correct
0	2	1	[0.36 -0.62]	False
1	1	0	[-0.12 1.65]	False
13	1	2	[-1.09 0.06]	False
15	2	1	[0.60 0.51]	False
17	1	2	[-0.25 -1.30]	False
21	2	1	[0.36 -0.17]	False
22	1	2	[-0.97 0.74]	False
26	1	2	[1.08 -0.17]	False

Number of correct predictions (True): 22
Number of incorrect predictions (False): 8
Accuracy: 0.73

1. The percentage of False rose sharply. There are 8 False, which is 7 more than when all columns were first used, and 6 more than when only Petal was used.
2. Accuracy dropped to 73%. Previously, problems appeared in both categories 1 and 2. However, I was able to check one case whose index is 1 and voted as 1 (vote) and actual label as 0.

Result by label

iris	Number of label	False Cnt	True Cnt	Precision, Recall
0	7	1	6	100%, 86%
1	11	3	8	62%, 73%
2	12	4	8	73%, 67%

1. The three categories showed differences in Precision and Recall. Among those predicted to be positive, all were 0, showing 100%.
2. 62% of those predicted to be 1 turned out to be false. 67% of actual 2s were predicted to be false.

Answering Questions

1. What was the overall accuracy of the model?

I have run KNN through a total of 4 methods. I use all four columns, add standardization, use only two columns that clearly distinguished the categories in visualization, and use only two columns that did not distinguish the categories well. First, the results of using all four columns showed an accuracy of 97%, if accuracy is measured across all columns. Out of a total of 30 test sets, 1 was found to be False. The second experiment showed the same 97% result even after standardization.
When only Petal was used, two values were measured as False and the results were lower than when the entire column was used, but the results were quite similar. Finally, when Sepal alone was used, accuracy dropped significantly. Through these four comparisons, I wanted to measure what consists of big data and think about what method should be used when computational energy is limited.

Result of 4 KNN with k=3

	Accuracy	True	False	Total
1. Using all four column	97%	29	1	30
2. Standardization with four column	97%	29	1	30
3. Petal two column with standardization	93%	28	2	30
4. Sepal two column with standardization	73%	22	8	30

2. What was the accuracy of each type of iris?

0: setosa 1: versicolor 2: virginica.
In trials 1 and 2, all cases that appeared as False were cases where the actual number 2 was mistakenly judged to be 1.The False added in Trial 3 was a case of mistaking it for 2 when it was actually 2. As seen in the visualization, you can see that 1 and 2 are not clearly distinguished in the iris data. Finally, Trial 4 had more False results due to insufficient data, mainly in Trials 1 and 2. In reality, 2 or 4 were incorrectly judged to be 1. In reality, 3 for 1 person were mistakenly judged to be 2. There was once a case where something that was actually 0 was mistakenly judged to be 1.

3. Would you classify the model as a good model or not?

I think it is a good model in terms of execution 1 and 2. This is because the accuracy was 97% and the categories were accurately classified using only 4 columns within a small dataset of 150. And even when done with just two columns, Petal, it represented a good model. There were many elements that could be distinguished in the data, and I think the machine learning algorithm called KNN made good use of them to perform predictive analysis.

Conclusion

KNN is the most basic machine learning model. A machine learning model is a program that can find patterns or make decisions from a previously unseen dataset (databricks, 2023).

I use iris data with K Nearest Neighbor modeling, which is the starting data with machine learning in Python. Rather than just doing one type of modeling, I checked how the KNN results changed by adding and excluding columns and performing standardization. In the future, I will consider how to preprocess various data within the machine learning algorithm, what machine learning to apply, and how to measure performance.

Reference

Ikeoluwa, Adegbite. (2020, December 18). Iris Flower Classification. Medium. Retrieved from https://peaceadegbite1.medium.com/iris-flower-classification-60790e9718a1

Gianluca Malato, (2022, June 12). Which models require normalized data?. Medium. Retrieved from https://towardsdatascience.com/which-models-require-normalized-data-d85ca3c85388#:~:text=KNN%20is%20a%20very%20popular,the%20space%20of%20the%20features.&text=Just%20like%20SVM%2C%20even%20KNN%20requires%20working%20with%20normalized%20data.

RANJEET, JAIN. (2019). Visualization -> Machine learning -> Deep learning. Kaggle. Retrieved from https://www.kaggle.com/code/ranjeetjain3/visualization-machine-learning-deep-learning

SuperStormer. (2021, November 25). How to convert a Scikit-learn dataset to a Pandas dataset. stackoverflow. Retrieved from https://stackoverflow.com/questions/38105539/how-to-convert-a-scikit-learn-dataset-to-a-pandas-dataset

apathak092. (2022, June 27). How To Convert Sklearn Dataset To Pandas Dataframe In Python. geeksforgeeks. Retrieved from https://www.geeksforgeeks.org/how-to-convert-sklearn-dataset-to-pandas-dataframe-in-python/

Manthan, Ghasadiya. (2023, May 10). How to create a seaborn correlation heatmap in Python?. tutorialspoint. Retrieved from https://www.tutorialspoint.com/how-to-create-a-seaborn-correlation-heatmap-in-python#:~:text=We%20use%20Seaborn's%20heatmap%20method,method%20to%20display%20the%20heatmap.

Stephen, Gruppetta. (2023). Visualizing Data in Python Using plt.scatter(). RealPython. Retrieved from https://realpython.com/visualizing-python-plt-scatter/

Plotly. (2023). Graph Objects in Python. Retrieved from https://plotly.com/python/graph-objects/

integratedots. (2019, June 4). Determine the number of Iris species with k-Means. Retrieved from https://www.integratedots.com/determine-number-of-iris-species-with-k-means/

kashishlohiya5. (2023, January 10). Python – Basics of Pandas using Iris Dataset. geeksforgeeks. Retrieved from https://www.geeksforgeeks.org/python-basics-of-pandas-using-iris-dataset/

databricks. (2023). Machine Learning Models. Retrieved from https://www.databricks.com/glossary/machine-learning-models

Appendix (Python):

```
# In[15]:
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from collections import Counter

iris = datasets.load_iris()
iris_data = iris.data
iris_labels = iris.target


# In[16]:
np.random.seed(42)
indices = np.random.permutation(len(iris_data))
n_training_samples = 30
trainset_data = iris_data[indices[:-n_training_samples]]
trainset_labels = iris_labels[indices[:-n_training_samples]]
testset_data = iris_data[indices[-n_training_samples:]]
testset_labels = iris_labels[indices[-n_training_samples:]]


# In[17]:
X = []
for iclass in range(3):
    X.append([[], [], []])
    for i in range(len(trainset_data)):
        if trainset_labels[i] == iclass:
            X[iclass][0].append(trainset_data[i][0])
            X[iclass][1].append(trainset_data[i][1])
            X[iclass][2].append(sum(trainset_data[i][2:]))

colours = ("r", "b", "y")

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

for iclass in range(3):
    ax.scatter(X[iclass][0], X[iclass][1], X[iclass][2], c=colours[iclass])

plt.show()


# In[18]:
def distance(instance1, instance2):
    instance1 = np.array(instance1)
    instance2 = np.array(instance2)

    return np.linalg.norm(instance1 - instance2)
```

```
# In[19]:
def get_neighbors(training_set,
                  labels,
                  test_instance,
                  k,
                  distance = distance):

    distances = []
    for index in range(len(training_set)):
        dist = distance(test_instance, training_set[index])
        distances.append((training_set[index], dist, labels[index]))
    distances.sort(key=lambda x: x[1])
    neighbors = distances[:k]
    return(neighbors)

# In[20]:
def vote(neighbors):
    class_counter = Counter()
    for neighbor in neighbors:
        class_counter[neighbor[2]] += 1
    return class_counter.most_common(1)[0][0]

# In[21]:
true_count = 0
false_count = 0

for i in range(n_training_samples):
    neighbors = get_neighbors(trainset_data,
                              trainset_labels,
                              testset_data[i],
                              3,
                              distance=distance)
    print("index: ", i,
          ", result of votes: ", vote(neighbors),
          ", label: ", testset_labels[i],
          ", data: ", testset_data[i])

    correct_prediction = vote(neighbors) == testset_labels[i]
    print("correct: ", correct_prediction)

    # Update counters based on correct_prediction
    if correct_prediction:
        true_count += 1
    else:
        false_count += 1

# Print the counts of True and False predictions
print("Number of correct predictions (True):", true_count)
print("Number of incorrect predictions (False):", false_count)
print("Accuracy: ", (true_count)/((true_count)+(false_count)))

# In[22]:
for i in range(n_training_samples):
```

```
    neighbors = get_neighbors(trainset_data,
                              trainset_labels,
                              testset_data[i],
                              3,
                              distance=distance)

    correct = 0
    if vote(neighbors) == testset_labels[i]:
        correct = correct + 1


# ## Visualization & Understanding Dataset
# In[23]:
df=iris
print(df)


# In[24]:
import pandas as pd
import numpy as np

# In[25]:
df = pd.DataFrame(iris.data[:, [0, 1, 2, 3]], columns=iris.feature_names[0:])


# In[26]:
df.head()


# In[27]:
from sklearn.datasets import load_iris


# In[28]:
iris_data = load_iris()


# In[29]:
df = pd.DataFrame(data=iris_data.data, columns=iris_data.feature_names)


# In[30]:
df.head()

# In[31]:
df['target'] = pd.Series(iris_data.target)

# In[32]:
df

# In[33]:
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
```

```
# In[34]:
df

# In[35]:
df=df.drop(['target'], axis=1)

# In[36]:
df.describe()

# In[37]:
df.info()

# In[38]:
Species = df['species'].unique()
Species

# In[39]:
import plotly.graph_objs as go
import plotly.offline as py

# In[40]:
species_count = df['species'].value_counts()
data = [go.Bar(
    x = species_count.index,
    y = species_count.values,
    marker = dict(color = ["red", "yellow", "blue"]))]

py.iplot(data)

# In[41]:
import seaborn as sns

# In[42]:
iris_corr_matrix = df.corr()
print(iris_corr_matrix)

# In[43]:
sns.heatmap(iris_corr_matrix, cmap='coolwarm', annot=True)

# ### Scatter Plotting
# In[44]:
setosa = go.Scatter(x = df['sepal width (cm)'][df.species =='setosa'], y = df['sepal
length (cm)'][df.species =='setosa']

, mode = 'markers', name = 'setosa')
versicolor = go.Scatter(x = df['sepal width (cm)'][df.species =='versicolor'], y =
df['sepal length (cm)'][df.species =='versicolor']

, mode = 'markers', name = 'versicolor')
virginica = go.Scatter(x = df['sepal width (cm)'][df.species =='virginica'], y =
df['sepal length (cm)'][df.species =='virginica']

, mode = 'markers', name = 'virginica')
```

```
data = [setosa, versicolor, virginica]
fig = dict(data=data)
py.iplot(fig, filename='styled-scatter')

# ### box with sepal width
# In[45]:
setosa_box_sep_w = go.Box(y=df['sepal width (cm)'][df.species =='setosa'],
boxmean=True, name= 'setosa')
versicolor_box_sep_w = go.Box(y=df['sepal width (cm)'][df.species =='versicolor'],
boxmean=True, name= 'versicolor')
virginica_box_sep_w = go.Box(y=df['sepal width (cm)'][df.species =='virginica'],
boxmean=True, name= 'virginica')
data = [setosa_box_sep_w, versicolor_box_sep_w, virginica_box_sep_w]

py.iplot(data)

# ### box with sepal length
# In[46]:
setosa_box_sep_l = go.Box(y=df['sepal length (cm)'][df.species =='setosa'],
boxmean=True, name= 'setosa')
versicolor_box_sep_l = go.Box(y=df['sepal length (cm)'][df.species =='versicolor'],
boxmean=True, name= 'versicolor')
virginica_box_sep_l = go.Box(y=df['sepal length (cm)'][df.species =='virginica'],
boxmean=True, name= 'virginica')
data = [setosa_box_sep_l, versicolor_box_sep_l, virginica_box_sep_l]

py.iplot(data)

# In[47]:
setosa = go.Scatter(x = df['petal width (cm)'][df.species =='setosa'], y = df['petal
length (cm)'][df.species =='setosa']

, mode = 'markers', name = 'setosa')
versicolor = go.Scatter(x = df['petal width (cm)'][df.species =='versicolor'], y =
df['petal length (cm)'][df.species =='versicolor']

, mode = 'markers', name = 'versicolor')
virginica = go.Scatter(x = df['petal width (cm)'][df.species =='virginica'], y =
df['petal length (cm)'][df.species =='virginica']

, mode = 'markers', name = 'virginica')
data = [setosa, versicolor, virginica]
fig = dict(data=data)
py.iplot(fig, filename='styled-scatter')


# ### box with petal width

# In[48]:
setosa_box_pet_w = go.Box(y=df['petal width (cm)'][df.species =='setosa'],
boxmean=True, name= 'setosa')
versicolor_box_pet_w = go.Box(y=df['petal width (cm)'][df.species =='versicolor'],
boxmean=True, name= 'versicolor')
```

```
virginica_box_pet_w = go.Box(y=df['petal width (cm)'][df.species =='virginica'],
boxmean=True, name= 'virginica')
data = [setosa_box_pet_w, versicolor_box_pet_w, virginica_box_pet_w]

py.iplot(data)
# ### box with petal length

# In[49]:
setosa_box_pet_l = go.Box(y=df['petal length (cm)'][df.species =='setosa'],
boxmean=True, name= 'setosa')
versicolor_box_pet_l = go.Box(y=df['petal length (cm)'][df.species =='versicolor'],
boxmean=True, name= 'versicolor')
virginica_box_pet_l = go.Box(y=df['petal length (cm)'][df.species =='virginica'],
boxmean=True, name= 'virginica')
data = [setosa_box_pet_l, versicolor_box_pet_l, virginica_box_pet_l]

py.iplot(data)
# ## KNN with Standardization
# In[50]:
iris = datasets.load_iris()
iris_data = iris.data
iris_labels = iris.target

# In[51]:
np.random.seed(42)
indices = np.random.permutation(len(iris_data))
n_training_samples = 30
trainset_data = iris_data[indices[:-n_training_samples]]
trainset_labels = iris_labels[indices[:-n_training_samples]]
testset_data = iris_data[indices[-n_training_samples:]]
testset_labels = iris_labels[indices[-n_training_samples:]]
# In[52]:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
trainset_data_std = scaler.fit_transform(trainset_data)
testset_data_std = scaler.transform(testset_data)

# In[53]:
X = []
for iclass in range(3):
    X.append([[], [], []])
    for i in range(len(trainset_data_std)):
        if trainset_labels[i] == iclass:
            X[iclass][0].append(trainset_data_std[i][0])
            X[iclass][1].append(trainset_data_std[i][1])
            X[iclass][2].append(sum(trainset_data_std[i][2:]))

colours = ("r", "b", "y")

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

for iclass in range(3):
```

```
    ax.scatter(X[iclass][0], X[iclass][1], X[iclass][2], c=colours[iclass])

plt.show()

# In[54]:
def distance(instance1, instance2):

    instance1 = np.array(instance1)
    instance2 = np.array(instance2)

    return np.linalg.norm(instance1 - instance2)

# In[55]:
def get_neighbors(trainset_data_std,
                  labels,
                  test_instance,
                  k,
                  distance = distance):

    distances = []
    for index in range(len(trainset_data_std)):
        dist = distance(test_instance, trainset_data_std[index])
        distances.append((trainset_data_std[index], dist, labels[index]))
    distances.sort(key=lambda x: x[1])
    neighbors = distances[:k]
    return(neighbors)

# In[56]:
def vote(neighbors):
    class_counter = Counter()
    for neighbor in neighbors:
        class_counter[neighbor[2]] += 1
    return class_counter.most_common(1)[0][0]

# In[57]:
true_count = 0
false_count = 0

for i in range(n_training_samples):
    neighbors = get_neighbors(trainset_data_std,
                              trainset_labels,
                              testset_data_std[i],
                              3,
                              distance=distance)
    print("index: ", i,
          ", result of votes: ", vote(neighbors),
          ", label: ", testset_labels[i],
          ", data: ", trainset_data_std[i])

    correct_prediction = vote(neighbors) == testset_labels[i]
    print("correct: ", correct_prediction)

    # Update counters based on correct_prediction
```

```
    if correct_prediction:
        true_count += 1
    else:
        false_count += 1

# Print the counts of True and False predictions
print("Number of correct predictions (True):", true_count)
print("Number of incorrect predictions (False):", false_count)
print("Accuracy: ", (true_count)/((true_count)+(false_count)))

# ### KNN with Petal
# In[58]:
iris = datasets.load_iris()
iris_data = iris.data
iris_labels = iris.target

# In[59]:
petal_data = iris_data[:,2:4]
print(petal_data)

# In[76]:
np.random.seed(42)
indices = np.random.permutation(len(petal_data))
n_training_samples = 30
trainset_data = petal_data[indices[:-n_training_samples]]
trainset_labels = iris_labels[indices[:-n_training_samples]]
testset_data = petal_data[indices[-n_training_samples:]]
testset_labels = iris_labels[indices[-n_training_samples:]]

# In[77]:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
trainset_data_std = scaler.fit_transform(trainset_data)
testset_data_std = scaler.transform(testset_data)

# In[78]:
trainset_data_std

# In[62]
X = []
for iclass in range(3):
    X.append([[], [], []])
    for i in range(len(trainset_data_std)):
        if trainset_labels[i] == iclass:
            X[iclass][0].append(trainset_data_std[i][0])
            X[iclass][1].append(trainset_data_std[i][1])
            X[iclass][2].append(sum(trainset_data_std[i][2:]))

colours = ("r", "b", "y")

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

```
for iclass in range(3):
    ax.scatter(X[iclass][0], X[iclass][1], X[iclass][2], c=colours[iclass])

plt.show()

# In[63]:
def distance(instance1, instance2):

    instance1 = np.array(instance1)
    instance2 = np.array(instance2)

    return np.linalg.norm(instance1 - instance2)

# In[64]:
def get_neighbors(trainset_data_std,
                  labels,
                  test_instance,
                  k,
                  distance = distance):

    distances = []
    for index in range(len(trainset_data_std)):
        dist = distance(test_instance, trainset_data_std[index])
        distances.append((trainset_data_std[index], dist, labels[index]))
    distances.sort(key=lambda x: x[1])
    neighbors = distances[:k]
    return(neighbors)

# In[65]:
def vote(neighbors):
    class_counter = Counter()
    for neighbor in neighbors:
        class_counter[neighbor[2]] += 1
    return class_counter.most_common(1)[0][0]

# In[66]:
true_count = 0
false_count = 0

for i in range(n_training_samples):
    neighbors = get_neighbors(trainset_data_std,
                              trainset_labels,
                              testset_data_std[i],
                              3,
                              distance=distance)
    print("index: ", i,
          ", result of votes: ", vote(neighbors),
          ", label: ", testset_labels[i],
          ", data: ", trainset_data_std[i])

    correct_prediction = vote(neighbors) == testset_labels[i]
    print("correct: ", correct_prediction)
```

```
    # Update counters based on correct_prediction
    if correct_prediction:
        true_count += 1
    else:
        false_count += 1

# Print the counts of True and False predictions
print("Number of correct predictions (True):", true_count)
print("Number of incorrect predictions (False):", false_count)
print("Accuracy: ", (true_count)/((true_count)+(false_count)))

# ### KNN with Sepal

# In[67]:
iris = datasets.load_iris()
iris_data = iris.data
iris_labels = iris.target

# In[68]:
sepal_data = iris_data[:,0:2]
print(sepal_data)

# In[79]:
np.random.seed(42)
indices = np.random.permutation(len(sepal_data))
n_training_samples = 30
trainset_data = sepal_data[indices[:-n_training_samples]]
trainset_labels = iris_labels[indices[:-n_training_samples]]
testset_data = sepal_data[indices[-n_training_samples:]]
testset_labels = iris_labels[indices[-n_training_samples:]]

# In[80]:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
trainset_data_std = scaler.fit_transform(trainset_data)
testset_data_std = scaler.transform(testset_data)

# In[81]:
X = []
for iclass in range(3):
    X.append([[], [], []])
    for i in range(len(trainset_data_std)):
        if trainset_labels[i] == iclass:
            X[iclass][0].append(trainset_data_std[i][0])
            X[iclass][1].append(trainset_data_std[i][1])
            X[iclass][2].append(sum(trainset_data_std[i][2:]))

colours = ("r", "b", "y")

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

for iclass in range(3):
```

```
    ax.scatter(X[iclass][0], X[iclass][1], X[iclass][2], c=colours[iclass])

plt.show()

# In[82]:
def distance(instance1, instance2):

    instance1 = np.array(instance1)
    instance2 = np.array(instance2)

    return np.linalg.norm(instance1 - instance2)

# In[83]:
def get_neighbors(trainset_data_std,
                  labels,
                  test_instance,
                  k,
                  distance = distance):

    distances = []
    for index in range(len(trainset_data_std)):
        dist = distance(test_instance, trainset_data_std[index])
        distances.append((trainset_data_std[index], dist, labels[index]))
    distances.sort(key=lambda x: x[1])
    neighbors = distances[:k]
    return(neighbors)

# In[84]:
def vote(neighbors):
    class_counter = Counter()
    for neighbor in neighbors:
        class_counter[neighbor[2]] += 1
    return class_counter.most_common(1)[0][0]


# In[85]:
true_count = 0
false_count = 0
for i in range(n_training_samples):
    neighbors = get_neighbors(trainset_data_std,
                              trainset_labels,
                              testset_data_std[i],
                              3,
                              distance=distance)
    print("index: ", i,
          ", result of votes: ", vote(neighbors),
          ", label: ", testset_labels[i],
          ", data: ", trainset_data_std[i])

    correct_prediction = vote(neighbors) == testset_labels[i]
    print("correct: ", correct_prediction)

    # Update counters based on correct_prediction
```

```
    if correct_prediction:
        true_count += 1
    else:
        false_count += 1

# Print the counts of True and False predictions
print("Number of correct predictions (True):", true_count)
print("Number of incorrect predictions (False):", false_count)
print("Accuracy: ", (true_count)/((true_count)+(false_count)))
```