Classifying Iris Types with Machine Learning Models

Mina Kemmer-Lee, Justin Talamantes, Amber Aeschbach

Overview

- We used a dataset from Kaggle that included 150 measurements of sepal/petal length and width for three different species of irises.
- Our goal was to create a classification model that accurately predicted the species of Iris based off of the length and width of the sepals and petals.
- After creating our model, we created a web interface that gathers inputs for length and width and displays the species of iris based on those values.
- We used SciKit-Learn, pandas, HTML/CSS, Flask, SQL database, Matplotlib, and Tableau

Models we tried

- Features:
  - 3 different Iris Flower Species:
    - Iris-Setosa
    - Iris-Versicolor
    - Iris-Virginica
  - Lengths and widths of both the Sepals and the Petals of the flowers in centimeters
- Random Forest Classification:
  - We tried different test data sizes (0.4, 0.3 and 0.2) but decided to stick with our original size (0.4) as it performed very accurately with only a little margin of error, rather than either of the other two we tried as they seemed a little too accurate
- Logistic Regression:
  - We also trained and tested a logistic regression model, which yielded a 100% accuracy rate.

Results (Random Forest Classification)

- We were able to get very high accuracy in our classification report, probably due to the data being straightforward and consistent between species.

Accuracy: 0.9833333333333333
Classification Report:

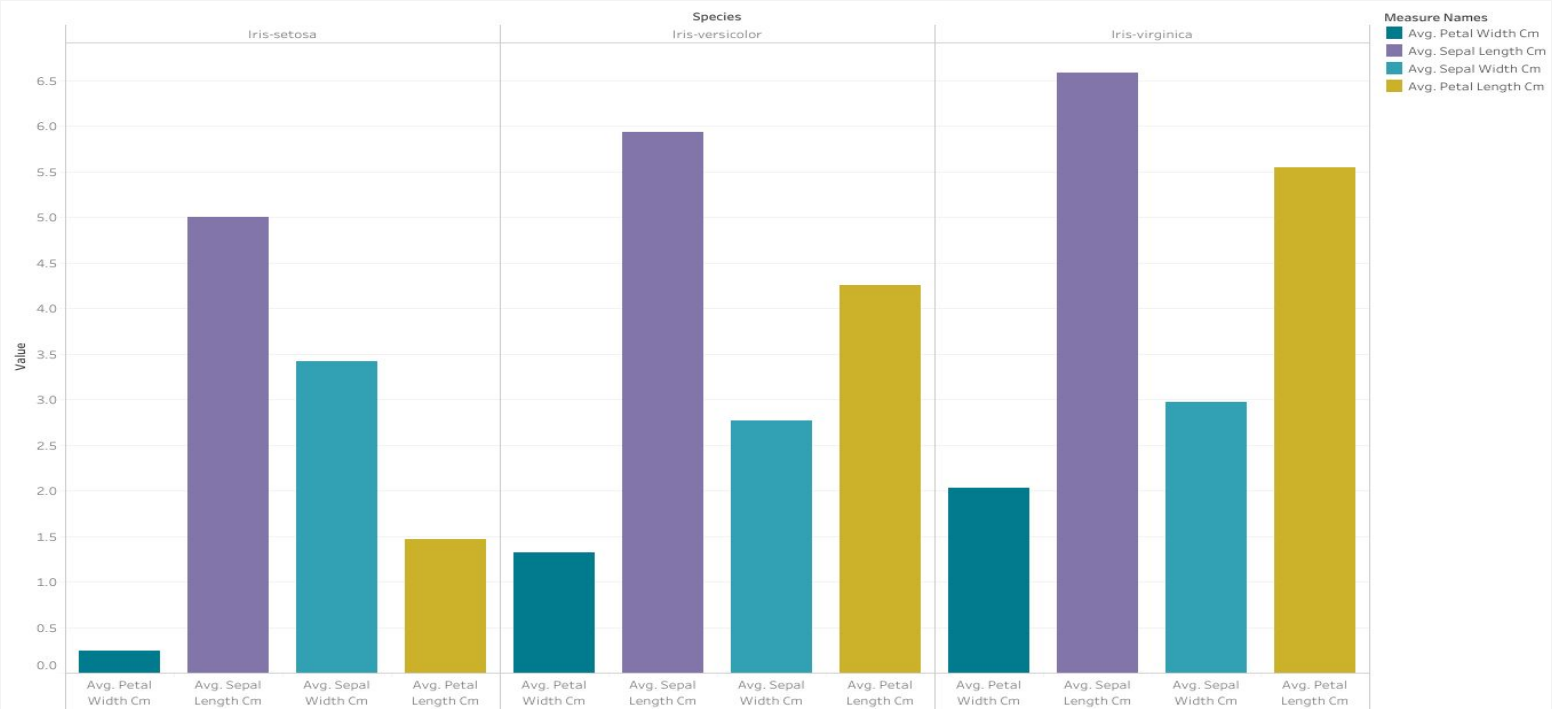
	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	23
Iris-versicolor	0.95	1.00	0.97	19
Iris-virginica	1.00	0.94	0.97	18
accuracy			0.98	60
macro avg	0.98	0.98	0.98	60
weighted avg	0.98	0.98	0.98	60

Confusion Matrix:

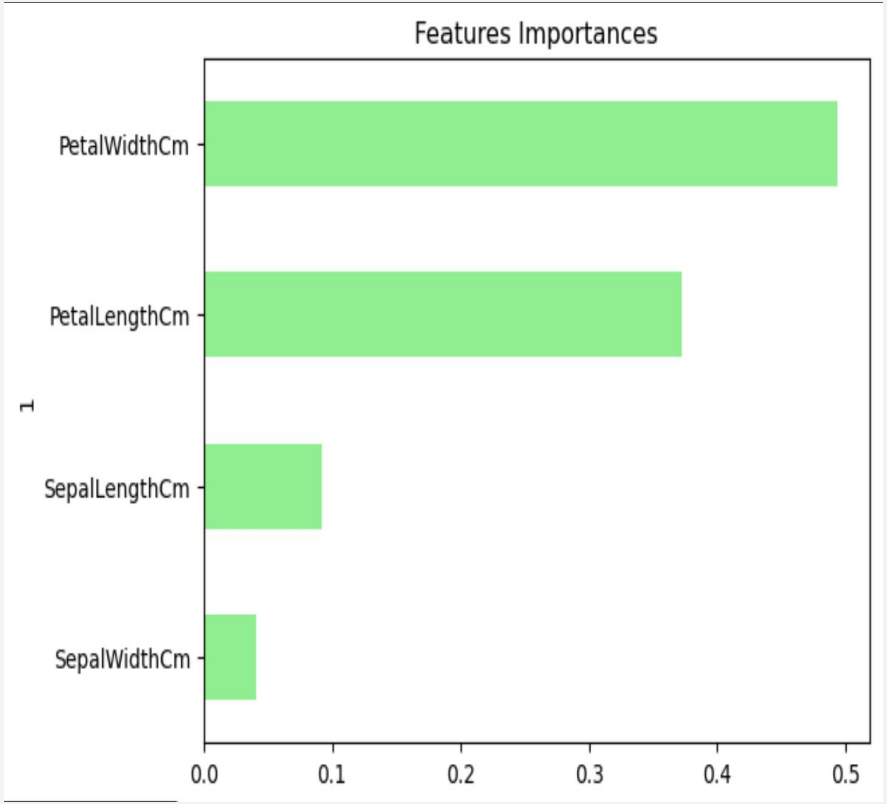
```
[[23  0  0]
 [ 0 19  0]
 [ 0  1 17]]
```

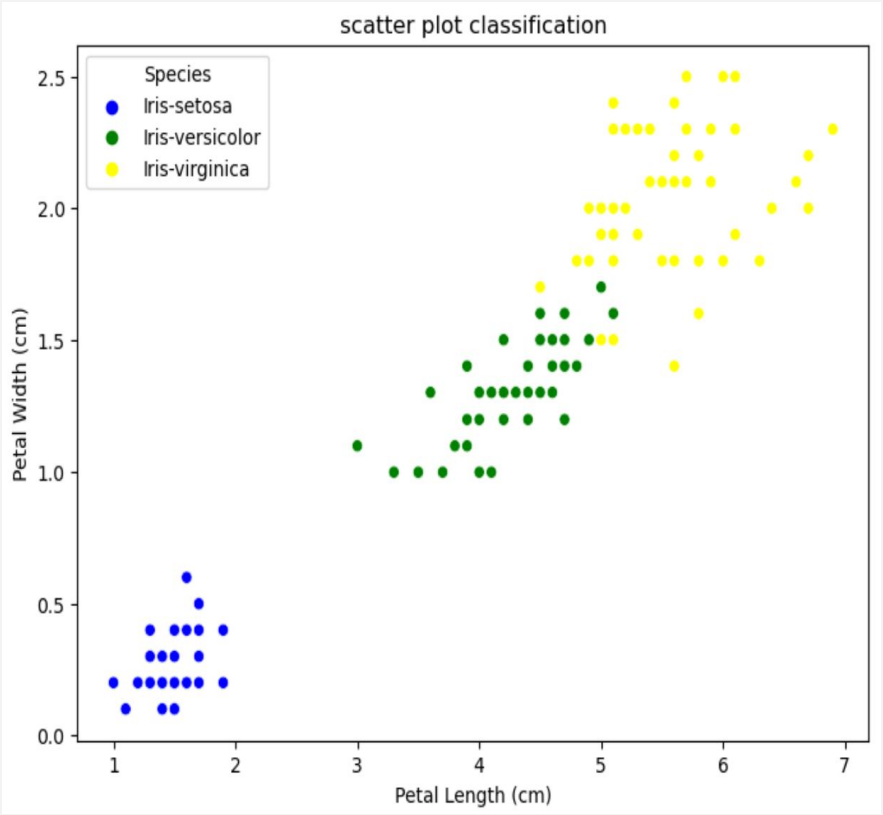
Tableau

- Looking at avg sepal and petal widths and lengths across species



Matplotlib


Features Importances
PetalWidthCm
PetalLengthCm
SepalLengthCm
SepalWidthCm
1
0.0
0.1
0.2
0.3
0.4
0.5


scatter plot classification
Species
Iris-setosa
Iris-versicolor
Iris-virginica
Petal Width (cm)
Petal Length (cm)
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7

Web Interface

- For our web interface, we used HTML/CSS.
- We set up the flask app to connected to our SQL Database that holds all the Iris Flower data.
- Most of the routes in the app were pretty basic besides our Model Predictor tab, which was coded to take user given measurements and provide a predicted Iris Species using our Random Forest model based off of those inputs given.
- We used joblib to connect our machine learning model from our notebook to our app

Incorporating the model into a Flask App

- Using JobLib we were able to ‘dump’ the model from the Collab Notebook as a .joblib file

```
pip install joblib
```

```
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (1.3.2)
```

```
[13] import joblib
```

```
[14] joblib.dump(model, 'model.joblib')
```

```
['model.joblib']
```

- We then were able to load the model.joblib right into our Flask App

```
# Load Machine Learning Model
model= joblib.load("model.joblib")
```

Model Predicting using User Inputs

- We created a list of values that a user can input from our web page, we started by testing known values from our dataset (right)

```
#example of web interface inputs
sepal_length = 5.9
sepal_width = 3.0
petal_length = 5.1
petal_width = 1.8

input_data = [[sepal_length, sepal_width, petal_length, petal_width]]
predicted_species = model.predict(input_data)
print("Predicted Species:", predicted_species[0])
```

```
Predicted Species: Iris-virginica
```

```
fetch("/Predictor", {
    method: "POST",
    headers: {
        "Content-Type": "application/json"
    },
    body: JSON.stringify({
        sepal_length: sepalLength,
        sepal_width: sepalWidth,
        petal_length: petalLength,
        petal_width: petalWidth
    })
})
.then(response => response.json())
.then(data => {
    // Display the prediction result
    predictionResult.textContent = `Prediction: ${data.prediction}`;
})
.catch(error => {
    console.error("Error:", error);
    predictionResult.textContent = "An error occurred during prediction.";
});
```

- In the html for the Model Predictor, after it gathers the inputs given, it sends the list of values to the model.joblib using an API to then return the predictionResult (aka the predicted species)

The End!
Thank you