ST3189: Assessed Coursework Project. (J01499)

Exploring Machine Learning Techniques on Iris Dataset

Table of Contents:

Iris Dataset :

The data set contains 3 classes with 50 instances each, and 150 instances in total, where each class refers to a type of iris plant.

Class : Iris Setosa,Iris Versicolour, Iris Virginica

The format for the data: (sepal length, sepal width, petal length, petal width)

Iris Setosa

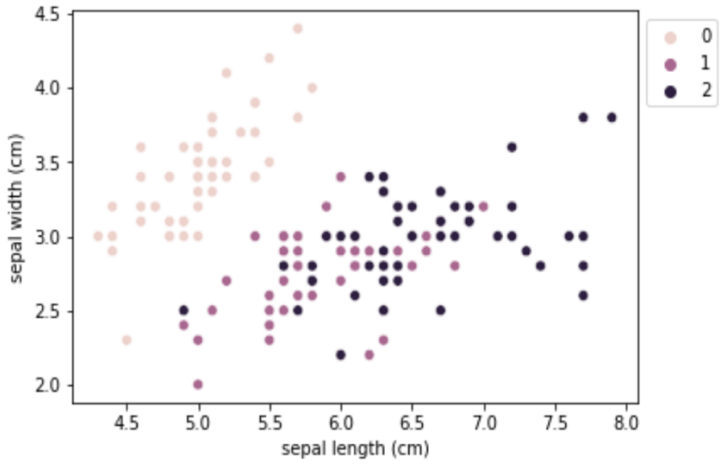
Iris Versicolor

Iris Virginica

We will be training our models based on these parameters and further use them to predict the flower classes.

TASK1 : Unsupervised Learning

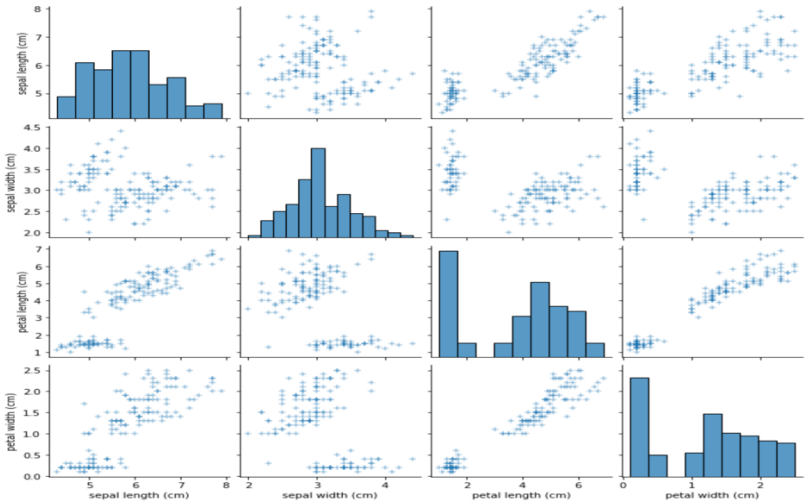
Visualizing the data using matplotlib:



From the above plot, we can infer that –

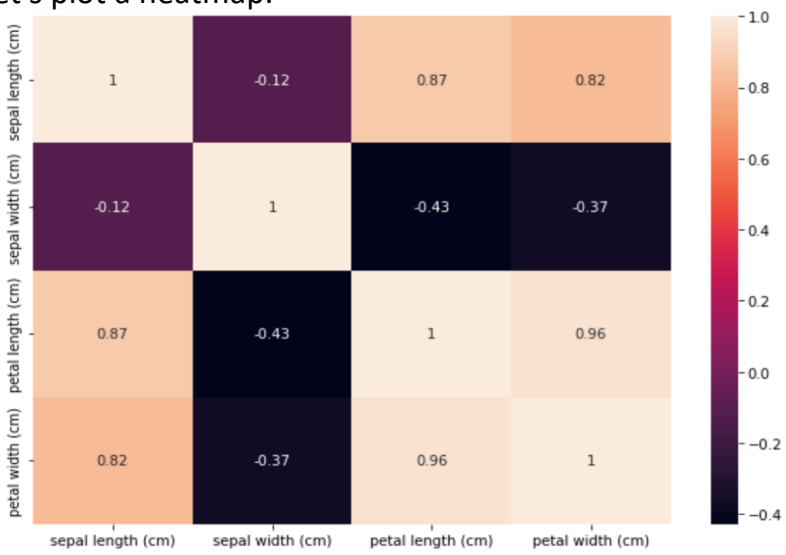
- Species Setosa has smaller sepal lengths but larger sepal widths.
- Versicolor Species lies in the middle of the other two species in terms of sepal length and width
- Species Virginica has larger sepal lengths but smaller sepal widths.

Let's plot all the column's relationships using a pairplot. It can be used for multivariate analysis.



- From this visualization, we can tell that iris-setosa is well separated from the other two flowers.
- And iris virginica is the longest flower and iris setosa is the shortest.

Let’s plot a heatmap.



- From this visualization, we can tell that iris-setosa is well separated from the other two flowers.
- And iris virginica is the longest flower and iris setosa is the shortest.

Petal width and petal length have high correlations.

- Petal length and sepal width have good correlations.
- Petal Width and Sepal length have good correlations.

Determing the number of clusters:

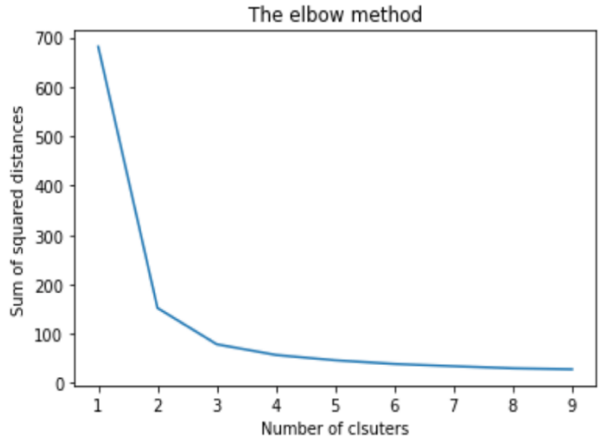
The Elbow Method

The Elbow method runs k-means clustering on the dataset for a range of values for k (say from 1-10) and then for each value of k computes an average score for all clusters. By default, the distortion score is computed, the sum of square distances from each point to its assigned center.

When these overall metrics for each model are plotted, it is possible to visually determine the best value for k. If the line chart looks like an arm, then the “elbow” (the point of inflection on the curve) is the best value of k. The “arm” can be either up or down, but if there is a strong inflection point, it is a good indication that the underlying model fits best at that point.

We use the Elbow Method which uses Within Cluster Sum Of Squares (WCSS) against the the number of clusters (K Value) to figure out the optimal number of clusters value. (Wikipedia)SS

Now, we visualize the Elbow Method so that we can determine the number of optimal clusters for our dataset.

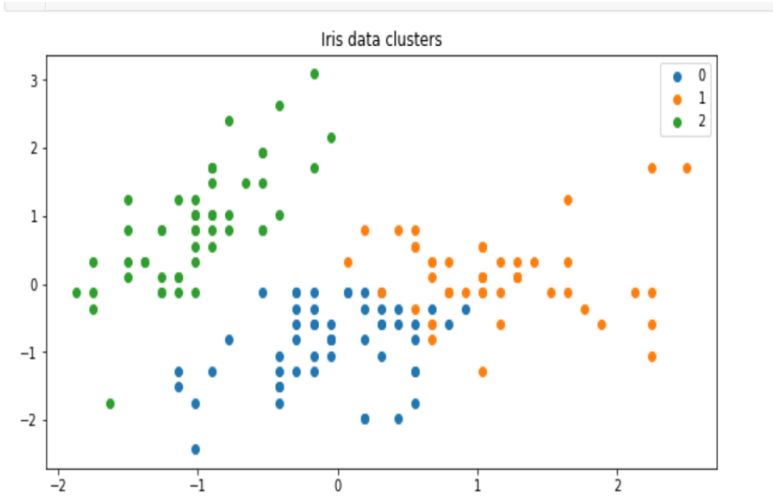


- It is clear, that the optimal number of clusters for our data are 3, as the slope of the curve is not steep enough after it.
- When we observe this curve, we see that last elbow comes at k = 3, it would be difficult to visualize the elbow if we choose the higher range.

Clustering

Now we will build the model for creating clusters from the dataset. We will use n_clusters = 3 i.e. 3 clusters as we have determined by the elbow method, which would be optimal for our dataset.

Our data set is for unsupervised learning therefore we will use fit_predict()



Analysis

Analyzing Data using the above graph becomes much more easier as it gives us a visual aid for better understanding of the data. Kmeans has divided the dataset into 3 clusters based on Annual income and the spending scores of the individual customers. The following clusters are created by the model,

- Iris-versicolor(0)
- Iris-setosa(1)
- Iris-virginica(2)

TASK2 : Regression

This involved the implementation of various regression techniques using scikit-learn library.

Creating Dataframes

```
# creating pd dataframes
iris_df = pd.DataFrame(data= iris.data, columns = iris.feature_names)
target_df = pd.DataFrame(data= iris.target,columns = ['species'])
def converter(specie):
    if specie == 0:
        return'setosa'
    elif specie == 1:
        return 'versicolor'
    else:
        return 'virginica'
target_df['species'] = target_df['species'].apply(converter)

#concatenate the dataframes
iris_df = pd.concat([iris_df,target_df],axis=1)
iris_df.describe()

```

Organising the datasets into dataframes facilitates better results and analysis, with separate structures for features and target variables.

This preparation step ensures that the dataset is structured properly for smooth implementation of machine learning algorithms and explorations.

Splitting the dataset

Since our process involve training and testing ,We should split our dataset.

```
# splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

x_train contains the training features
x_test contains the testing features
y_train contains the training label
y_test contains the testing labels

Evaluating the Model

Let's Evaluate our model performance by calculating the residual sum of squares:

- Calculate the Root Mean squared Error
- R-squared

Regression Techniques And Results

- Linear Regression: We employed this technique to establish a direct linear relationship between the independent and dependent variables within the Iris dataset.We will make predictions and train the model with predict and fit function.

In [26]:
```
# Linear Regression
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
linear_reg_pred = linear_reg.predict(X_test)
linear_reg_r2 = r2_score(y_test, linear_reg_pred)
linear_reg_rmse = mean_squared_error(y_test, linear_reg_pred, squared=False)
print(f"Linear Regression - R-squared: {linear_reg_r2}, RMSE: {linear_reg_rmse}")

```

```
Linear Regression - R-squared: 1.0, RMSE: 9.315293853352739e-16
```

Result:
Achieved a perfect R-squared score of 1.0 and an incredibly low Root Mean Squared Error (RMSE) of 9.315293853352739e-16.

- Decision Tree Regression: Utilizing a decision tree algorithm, this technique was employed to derive simple decision rules from the dataset's features, achieving high predictive performance with a remarkable r squared score.

```
In [27]:
# Decision Tree Regression
decision_tree_reg = DecisionTreeRegressor()
decision_tree_reg.fit(X_train, y_train)
decision_tree_reg_pred = decision_tree_reg.predict(X_test)
decision_tree_reg_r2 = r2_score(y_test, decision_tree_reg_pred)
decision_tree_reg_rmse = mean_squared_error(y_test, decision_tree_reg_pred, squared=False)
print(f"Decision Tree Regression - R-squared: {decision_tree_reg_r2}, RMSE: {decision_tree_reg_rmse}")

Decision Tree Regression - R-squared: 0.9917906123237397, RMSE: 0.07527726527090811
```

Result:

Demonstrated high performance with an R-squared value of 0.9917906123237397 and a low RMSE of 0.07527726527090811.

- **Random Forest Regression:** Leveraging an ensemble of decision trees, we used this method to harness the collective predictive power of multiple trees, resulting in a robust model with an impressive

```
In [28]:
# Random Forest Regression
random_forest_reg = RandomForestRegressor()
random_forest_reg.fit(X_train, y_train)
random_forest_reg_pred = random_forest_reg.predict(X_test)
random_forest_reg_r2 = r2_score(y_test, random_forest_reg_pred)
random_forest_reg_rmse = mean_squared_error(y_test, random_forest_reg_pred, squared=False)
print(f"Random Forest Regression - R-squared: {random_forest_reg_r2}, RMSE: {random_forest_reg_rmse}")

Random Forest Regression - R-squared: 0.9955157909986481, RMSE: 0.0556354203722758
```

Result:

Demonstrated high performance with an R-squared value of 0.9955157909986481 and a low RMSE of 0.0556354203722758.

Conclusion

- The regression models applied to predict 'sepal length' from the Iris dataset displayed remarkable performance.
- The Linear Regression model notably achieved a perfect R-squared score, indicating an ideal fit to the data.
- Both Decision Tree and Random Forest models also exhibited high accuracy, showcasing their potential for precise predictions on this dataset.

TASK3 : Classification

Create the Model (Classification)

Utilized Models:

- Logistic Regression

```
# Logistic Regression
logistic_reg = LogisticRegression()
logistic_reg.fit(X_train, y_train)
logistic_reg_pred = logistic_reg.predict(X_test)
```

- Random Forest Classifier

```
# Random Forest Classifier
random_forest_clf = RandomForestClassifier()
random_forest_clf.fit(X_train, y_train)
random_forest_pred = random_forest_clf.predict(X_test)
```

- Gradient Boosting Classifier

```
# Gradient Boosting Classifier
gradient_boosting_clf = GradientBoostingClassifier()
gradient_boosting_clf.fit(X_train, y_train)
gradient_boosting_pred = gradient_boosting_clf.predict(X_test)
```

Model Implementation:

- Employed logistic regression, random forest, and gradient boosting classifiers to predict iris species based on given features within the dataset.
- Trained the models on the training set and evaluated their performance on the test set.
- Performed evaluation metrics to find accuracy, precision, recall and F1 score.

Model Performance:

```
Logistic Regression Metrics:
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-score: 1.0000

Random Forest Classifier Metrics:
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-score: 1.0000

Gradient Boosting Classifier Metrics:
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-score: 1.0000
```

- All three models exhibited perfect accuracy, precision, recall, and F1-scores of 1.0000 on the test set, showcasing exceptional predictive capabilities.

- The high performance across all metrics suggests these models effectively captured patterns within the dataset, accurately predicting iris species based on given features.

Predicted Values

We can also compare the values that our model predicted with the actual values.

Comparison for Logistic Regression:

	True Values	Predicted Values
73	1	1
18	0	0
118	2	2
78	1	1
76	1	1

Comparison for Gradient Boosting Classifier:

	True Values	Predicted Values
73	1	1
18	0	0
118	2	2
78	1	1
76	1	1

Comparison for Random Forest Classifier:

	True Values	Predicted Values
73	1	1
18	0	0
118	2	2
78	1	1
76	1	1

The predictions line up almost perfectly, and only once the model incorrectly predicted that a flower belonged to class 1 when it really belonged to class 2.

Conclusion

Since these models show the same excellent results, we could look into things like how easy it is to explain them or how quick they are to make predictions to decide which one to go with. This extra look helps in picking the best model for our specific needs beyond just their accuracy. Given the equal performance, further analysis could focus on other aspects like model interpretability or computational efficiency to finalize the model selection.

END☺