VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Machine Learning - CO3117

Assignment

Iris Flower Classification
using several Machine Learning models

Instructor: Nguyen Duc Dung
Class: CC01
Student: Tran Anh Kiet - 2152147

HO CHI MINH CITY, DECEMBER 2023

Contents

1 Introduction

Irises influenced the design of the French fleur-de-lis, are commonly used in the Japanese art of flower arrangement known as Ikebana, and underlie the floral scents of the "essence of violet" perfume. They're also the subject of this well-known machine learning project, in which you must create an ML model capable of sorting irises based on five factors into one of three classes, Iris Setosa, Iris Versicolour, and Iris Virginica.

To get started, the data set below includes 50 instances of each of the three iris classes for a total of 150 instances. While one of the classes is linearly separable, the other two are not. Our goal is to create a model capable of classifying each iris instance into the appropriate class based on four attributes: sepal length, sepal width, petal length, and petal width.

Figure 1: Iris Setosa

Figure 2: Iris Versicolour

Figure 3: Iris Virginica

2 Literature Review

Discuss previous work and findings related to the Iris classification problem. For classifying, i use 3 polular model in machine learning

2.1 Machine Learning Models Used in This Project

1. **K-nearest Neighbors** The K-Nearest Neighbors (K-NN) algorithm is one of the simplest machine learning algorithms, based on the supervised learning technique. It assumes similarity between the new case/data and available cases, categorizing the new case into the most similar available category. K-NN stores all available data and classifies new data points based on similarity, making it a convenient method for classifying new data into suitable categories.

   K-NN can be used for both regression and classification, though it is primarily used for classification problems. It is a non-parametric algorithm, meaning it does not make any assumptions about the underlying data distribution. Also known as a lazy learner algorithm, K-NN does not immediately learn from the training set but rather stores the dataset and performs actions upon classification.

   For example, consider two categories, Category A and Category B. If we have a new data point x1, K-NN helps determine whether this data point belongs to Category A or B. The process can be visualized in the diagram below:

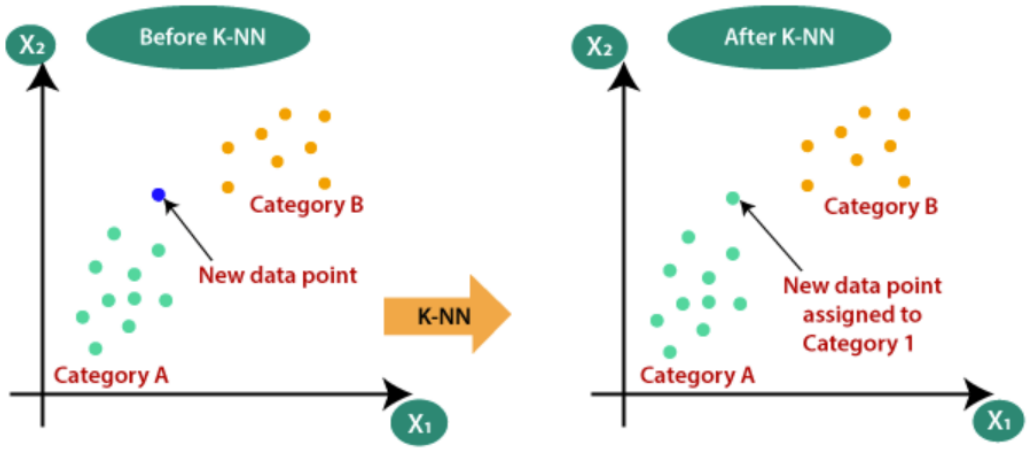

Figure 4: Illustration of K-NN Classification

How Does K-NN Work? The K-NN algorithm operates as follows:

(a) Select the number K of neighbors.
(b) Calculate the Euclidean distance of K neighbors.
(c) Take the K nearest neighbors as per the Euclidean distance.
(d) Count the number of data points in each category among these K neighbors.
(e) Assign the new data points to the category with the maximum number of neighbors.
(f) The model is ready.

Suppose we have a new data point that needs categorization. First, we choose the number of neighbors (k=5). Next, we calculate the Euclidean distance between the data points, as shown below:

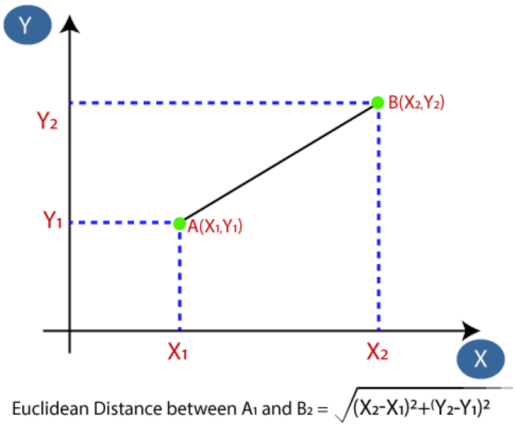

Figure 5: Euclidean Distance Calculation in K-NN

After calculating the Euclidean distance, we identify the nearest neighbors. For instance, if there are three nearest neighbors in Category A and two in Category B, as illustrated in the image below, the new data point would most likely belong to Category A.

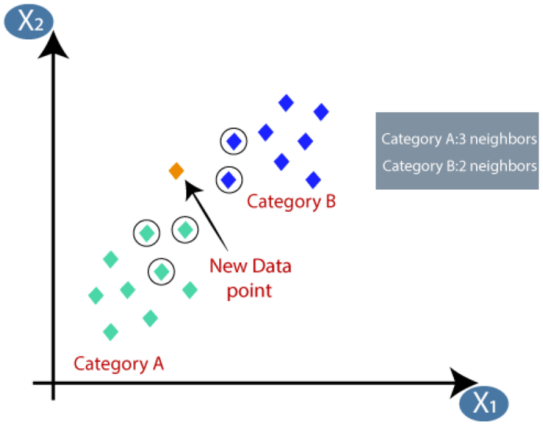

Figure 6: Identifying Nearest Neighbors in K-NN

2. **Random Forest** Random Forest is a supervised learning algorithm known for its versatility. It constructs a "forest" which is an ensemble of decision trees, typically trained using the bagging method. This method involves combining multiple learning models to improve the overall result.

   Random Forest is effective because it constructs multiple decision trees and merges them to achieve a more accurate and stable prediction. This algorithm is advantageous as it can be applied to both classification and regression problems, which are prevalent in many machine learning systems.

   Random Forest in Classification and Regression In classification, which is often

considered a fundamental aspect of machine learning, Random Forest operates effectively. The following figure illustrates a Random Forest model with two trees:

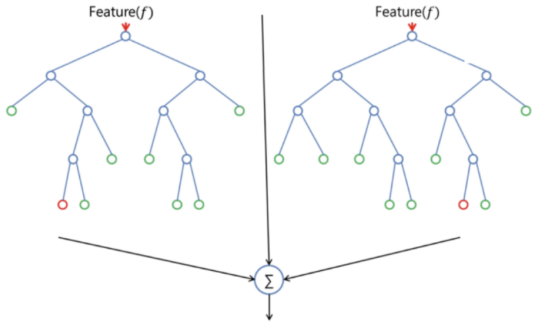

Figure 7: Example of a Random Forest Model with Two Trees

Random Forest shares many hyperparameters with decision trees and bagging classifiers. It simplifies the process by combining these elements into a single classifier. The classifier can be adjusted to handle regression tasks as well, using the regressor class of Random Forest.

The algorithm introduces additional randomness into the model during tree construction. Instead of searching for the most important feature while splitting a node, it searches among a random subset of features. This approach enhances diversity and typically yields a better model.

In a Random Forest classifier, only a random subset of features is considered for splitting a node. Trees can be made even more random by using random thresholds for each feature rather than the best possible thresholds, as is done in standard decision trees.

Random Forest Models vs. Decision Trees Random Forest, while a collection of decision trees, differs significantly in its approach. A decision tree creates a set of rules from a training dataset, which it then uses to make predictions. For example, in predicting online advertisement clicks, a decision tree will generate rules based on past click data and associated features.

In contrast, Random Forest selects random observations and features to construct several decision trees and then averages their predictions. This method helps mitigate overfitting, which is a common issue with "deep" decision trees. Random Forest achieves this by creating random subsets of features, building smaller trees, and then combining them. However, it's important to note that this approach may increase computational time and doesn't always prevent overfitting, depending on the number of trees constructed.

3. **Logistic Regression**

2.2 Confusion Matrix

A confusion matrix is a table typically used to describe the performance of a classification model on a set of test data for which the true values are known. It visualizes the model's accuracy by comparing the actual target values with the predictions made by the machine learning model.

Consider the following example of a confusion matrix:

The key components of a confusion matrix are:

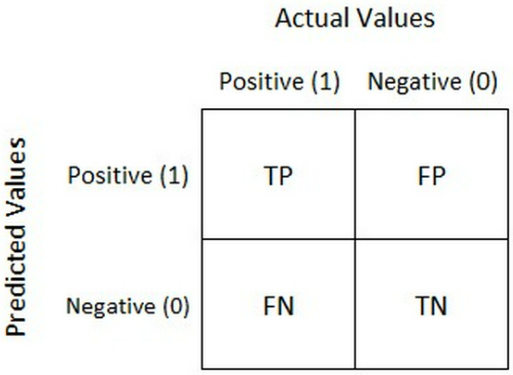

Figure 8: Confusion Matrix Example

- **True Positive (TP)**: Instances where the model correctly predicts the positive class.
- **True Negative (TN)**: Instances where the model correctly predicts the negative class.
- **False Positive (FP)**: Instances where the model incorrectly predicts the positive class, also known as a Type I error.
- **False Negative (FN)**: Instances where the model incorrectly predicts the negative class, also known as a Type II error.

In the case of a multi-class classification problem, such as the Iris dataset, the confusion matrix will have dimensions of $n\_classes \times n\_classes$. Each entry (i, j) in the matrix corresponds to the number of instances where class i was predicted as class j.

2.3 Classification Report

The classification report presents several key metrics for each class, which are essential for evaluating the performance of a classification model:

- **Precision**: This metric is the ratio of correctly predicted positive observations to the total predicted positives. It addresses the question, "Of all instances labeled as positive by the model, how many are actually positive?" Precision is calculated as TP / (TP + FP), where TP is the number of true positives and FP is the number of false positives.
- **Recall** (also known as Sensitivity or True Positive Rate): Recall is the ratio of correctly predicted positive observations to all observations in the actual class. It answers the question, "Of all the instances that are actually positive, how many were correctly labeled by the model?" It is calculated as TP / (TP + FN), where FN is the number of false negatives.
- **F1-Score**: This score is the weighted harmonic mean of precision and recall. It considers both false positives and false negatives, making it a robust measure that shows the balance between precision and recall. The F1-Score is calculated as 2 * (Precision * Recall) / (Precision + Recall).

- **Support**: This indicates the number of actual occurrences of each class in the specified dataset. For instance, if Iris-setosa has a support of 11, it means there are 11 instances of Iris-setosa in the test set that the model has attempted to predict.
- **Accuracy**: Accuracy represents the ratio of correctly predicted observations to the total observations. It is expressed as (TP + TN) / Total, where TN is the number of true negatives.
- **Macro Average**: This is the average of precision, recall, and F1-score calculated between classes, without considering class imbalance. It gives equal weight to each class.
- **Weighted Average**: In contrast to the macro average, the weighted average of precision, recall, and F1-score is calculated between classes while taking into account the number of instances in each class. This approach weights each class proportionally to its presence in the dataset.

3 Methodology

3.1 Data Collection

The dataset was obtained from the UCI Machine Learning Repository. For more information, visit the UCI Iris Dataset page.
This is one of the earliest datasets used in the literature on classification methods and widely used in statistics and machine learning. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are not linearly separable from each other.

Predicted attribute: class of iris plant.

This is an exceedingly simple domain.

3.2 Data Preprocessing

1. **Loading the Dataset:** The dataset is loaded into a pandas DataFrame. This dataset includes features like sepal length, sepal width, petal length, and petal width, along with the class labels.

2. **Feature Selection:** The features (sepal length, sepal width, petal length, and petal width) are separated from the class labels. This is done by splitting the DataFrame into X (features) and y (labels).

3. **Splitting the Dataset:** The dataset is split into training and test sets. *train_test_split()*. This is a common practice in machine learning to evaluate the model on data that it hasn't seen during training. The test size is set to 20% of the total dataset.

4. **Feature Scaling:** Standardization of features using *StandardScaler()*.

The Python code for these steps is as follows:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pd.read_csv(url, names=names)

# Feature Selection
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Splitting the Dataset
X_train, X_test, y_train
y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

3.3 Model Training

The training of different models in this project is outlined as follows:

1. **K-nearest Neighbors (K-NN)**: For K-NN, the number of neighbors is set to 5.
   - **For Classification:** The K-NN model is trained (fit) on the training data (X_train, y_train). The trained model is then used to make predictions on the test set (X_test). The model's performance is evaluated using a confusion matrix and a classification report, which provide insights into accuracy and other metrics. The 'model.predict(X_test)' method is used to apply the algorithm to each point in the test set, making predictions based on the similarity of these points to the training set.
   - **For Regression:** Two approaches can be used for weight assignment:
     (a) *Uniform Weights* ('weights='uniform''): Each of the 'k' nearest neighbors contributes equally to the prediction, regardless of their distance from the query point. For instance, in a KNN regression with k=5 and uniform weights, the predicted value is the average of the values of the 5 nearest neighbors.
     (b) *Distance Weights* ('weights='distance''): Neighbors contribute to the prediction according to their distance from the query point. Closer neighbors have a greater influence on the prediction than those farther away, with the predicted value typically being a weighted average of the nearest neighbors' values, where weights are inversely proportional to the distance.
2. **Random Forest**: The number of random tree is set to 100, we take 40% of data for testing
   (a) Classification: Bootstrapping: Random Forest creates multiple decision trees using random subsets of training data, a process known as bootstrapping.
   Feature Randomness: Each tree is built using a random subset of features, increasing diversity and robustness among the trees.
   Building Decision Trees: Trees are formed by repeatedly splitting data based on features, using criteria like Gini impurity or entropy.
   Independence of Trees: Trees grow independently, reducing correlation and capturing a wide range of data patterns.
   Aggregation of Results: The forest combines tree predictions, usually through majority voting, to improve accuracy and generalization.
   Handling Overfitting: By averaging multiple trees, Random Forest reduces overfitting compared to individual decision trees.
   (b) **Random Forest Regressor**: In this analysis, we focus on the Iris Setosa species from the Iris dataset, specifically examining the relationship between the sepal length and the sepal width. A Random Forest Regressor, consisting of 100 decision trees, is employed to model this relationship. The dataset is first filtered to include only the data points corresponding to Iris Setosa. Subsequently, 'sepal-length' is used as the feature to predict 'sepal-width'. The data is split into training and test sets, with the training set used to fit the Random Forest model. The model's performance is then visualized through a plot, which displays both the actual data points and the predictions made by the regressor. This approach provides insights into how sepal length influences sepal width in the Iris Setosa species and demonstrates the application of Random Forest Regression on a subset of a dataset.

4 Results and Discussion

First, let plot all the data:

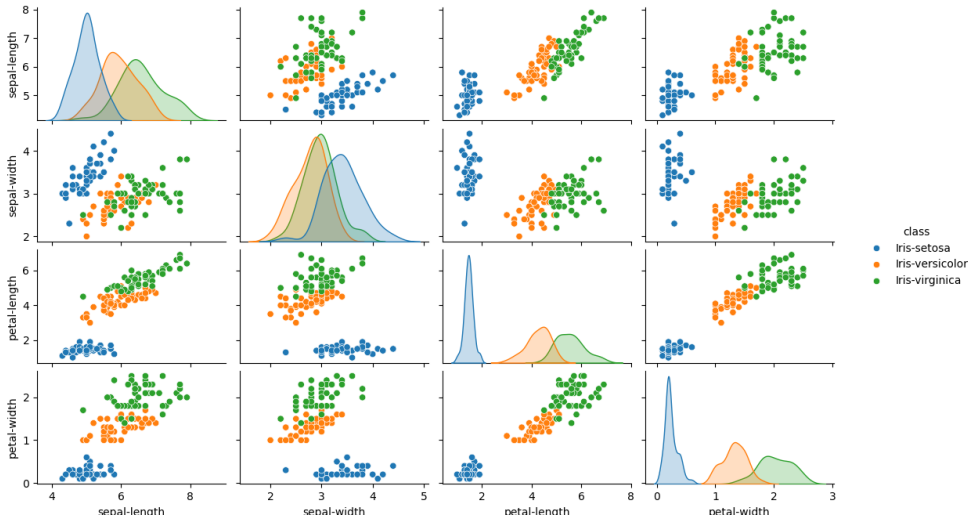

Figure 9: Data with feature

4.1 Model Performance and Analysis

4.1.1 K-nearest neighbors

1. **Classifier Analysis**: The final KNN classifier achieved an impressive accuracy of 97% on the test dataset, comprising 30 samples. This high accuracy indicates the model's effectiveness in classifying iris species based on their features.

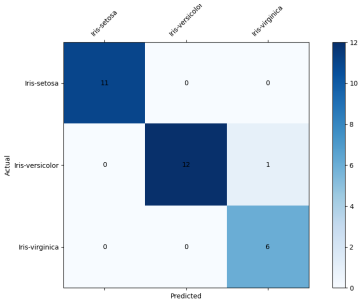

Figure 10: Confusion matrix

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.92	0.96	13
Iris-virginica	0.86	1.00	0.92	6
accuracy			0.97	30
macro avg	0.95	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

Figure 11: Output result

The confusion matrix further details the model's performance, revealing that only one instance of the Versicolor species was incorrectly classified. This insight is crucial for understanding the model's strengths and limitations in distinguishing between species.

2. **Regression Analysis**: For a regression perspective, the 'petal-length' feature was used to predict the species, converting species names to numerical values.

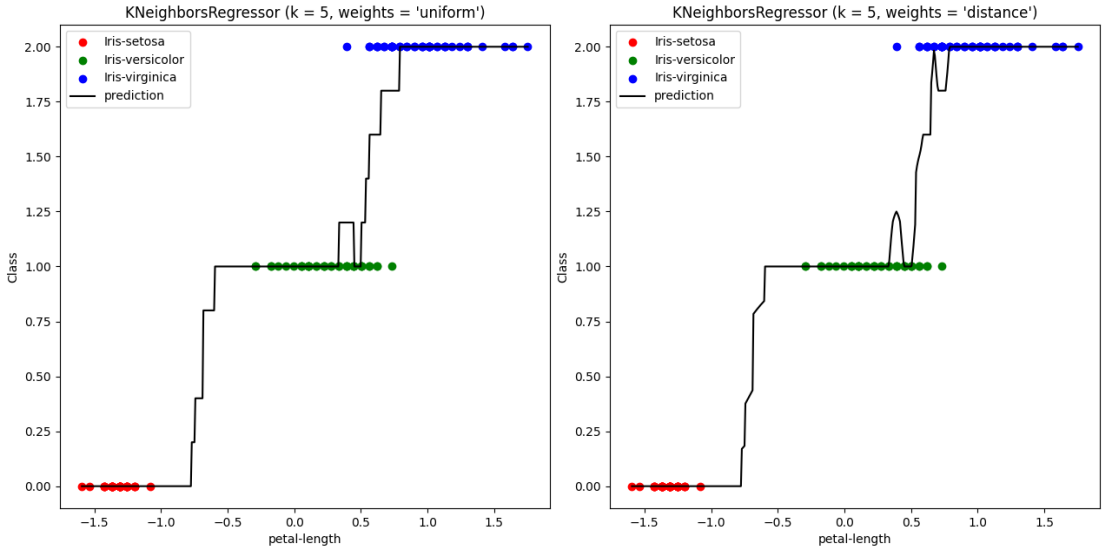

Figure 12: Final result

The results showed distinct clusters for each species after standard scaling of the data. Petal lengths below -1 were classified as Iris Setosa, lengths between -0.5 and 1 as Versicolor, and lengths above 0.5 as Iris Virginica. This demonstrates the relationship between petal length and species, highlighting the model's ability to discern patterns within the data.

4.1.2 Random Forest Analysis

The Random Forest algorithm was applied in two distinct modes: classification and regression. This section outlines the results obtained from both approaches.

1. **Classification Analysis** The classification model was evaluated using a confusion matrix, which is depicted in Figure 13. The model achieved an overall prediction accuracy of 97%. However, it is noteworthy that there were misclassifications, specifically one instance of Iris virginica and one of Iris versicolor being incorrectly classified.

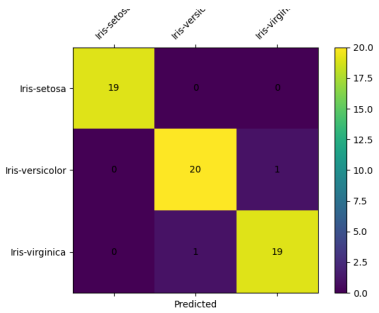

Figure 13: Confusion matrix for Random Forest Classifier

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	19
Iris-versicolor	0.95	0.95	0.95	21
Iris-virginica	0.95	0.95	0.95	20
accuracy			0.97	60
macro avg	0.97	0.97	0.97	60
weighted avg	0.97	0.97	0.97	60

Figure 14: Classification output results

2. **Regression Analysis** In the regression analysis, the focus was on predicting the sepal width based on the sepal length specifically for the Iris Setosa species. The Random Forest Regressor, though effective, exhibited limitations in extrapolating values beyond the range of the training data. The model generally predicted values close to the maximum of the training set range when confronted with inputs outside this range.

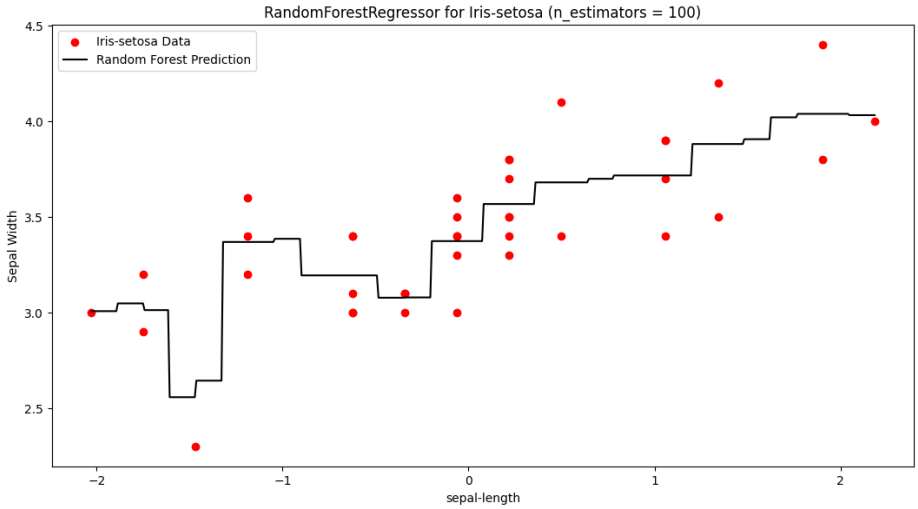

Figure 15: Random Forest Regression Analysis of Sepal Width Based on Sepal Length for Iris Setosa

5 Conclusion

Github link for this project: ML-Iris-Classification
Summarize the main findings, the implications of your work, and any future work.

6 References

https://machinelearningcoban.com/2017/01/08/knn/
https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning