

January Submission Assignment: Classifying the Fashion-MNIST dataset using KNN and random forest

Yifei Sun

January 2023

1 Abstract

In this study, the classification performance of two machine learning algorithms, k-nearest neighbors (KNN) and random forest, was compared on the Fashion-MNIST dataset. The Fashion-MNIST dataset consists of 70,000 grayscale images of clothing and accessories, divided into 10 classes. A brief overview of the KNN and random forest algorithms was provided, and they were then applied to classify the Fashion-MNIST dataset. The performance of the two algorithms was evaluated using the accuracy metric, and it was found that the random forest algorithm outperformed the KNN algorithm. The results suggest that the random forest algorithm is a more effective classifier for the Fashion-MNIST dataset.

2 Introduction

Fashion MNIST is a commonly used dataset for machine learning, provided by researchers at Zalando, as shown in Figure 1. It consists of 60,000 training images and 10,000 test images, each of which is a 28x28 pixel grayscale image representing the category of a piece of clothing. Fashion MNIST is an alternative to the MNIST dataset, but it is more challenging

as it includes more complex images and has 10 categories instead of the 10 categories in MNIST.

The 10 categories in Fashion MNIST include: T-shirt/top, trousers, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. These images are sourced from product images on the Zalando website and have been preprocessed so that they can be used directly in machine learning models.

The Fashion MNIST dataset is often used to test the performance of machine learning models, particularly for image classification tasks. Its size and complexity make it an ideal "hello world" dataset for quickly verifying the basic functionality of a machine learning model. It can also be used to compare the performance differences between different machine learning models.

Table 2: Class names and example images in Fashion-MNIST dataset.

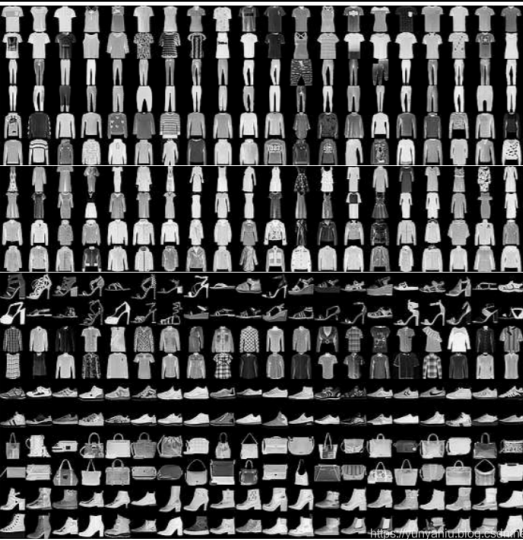
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Fig 1: Fashion MNIST dataset

Overall, the Fashion MNIST dataset is a very useful tool for helping us understand the performance of machine learning models and for testing the effectiveness of different machine learning algorithms. Its grayscale images

and simple range of categories make it easy to work with while still having enough complexity to provide meaningful results. Additionally, the size and availability of the Fashion MNIST dataset make it an ideal dataset for use in different studies and as a benchmark against other larger and more complex datasets. As such, the Fashion MNIST dataset is a very useful resource in machine learning research.

3 Related work in literature

1. This paper proposes a k-nearest neighbor (KNN) based multi-label classification algorithm. Multi-label classification is a machine learning task that requires predicting multiple labels for each sample. The algorithm proposed in the paper treats each sample's multiple labels as different classification tasks and resolves the multi-label classification problem by weighing between different classification tasks. The paper also proposes a new multi-label classification evaluation index and demonstrates the effectiveness of the algorithm through experiments[1].

2. "Classification with learning k-nearest neighbors" is a paper by J. Laaksonen and E. Oja. In this paper, the authors propose a k-nearest neighbors (KNN) classification algorithm that learns from examples during the classification process. The proposed algorithm uses a self-organizing map to learn from examples, and the authors demonstrate the effectiveness of the algorithm through experiments[2].

3. This is an academic paper published in 2017, written by Han Xiao, Kashif Rasul, and Roland Vollgraf. The paper describes a dataset called Fashion-MNIST, which consists of grayscale images of clothing items such as shirts, boots, and bags[3].

Fashion-MNIST has gained popularity as a benchmark dataset for machine learning algorithms, particularly in the field of computer vision. It has been used in a variety of research projects and has been included in several machine learning libraries and frameworks, including TensorFlow and Keras.

4. This is a survey paper published in 2019 by A.B. Shaik and S.

Srinivasan. The paper presents a summary of the current state of research on the use of random forest ensembles for classification tasks in machine learning.

A random forest ensemble is a type of machine learning model that consists of a combination of decision trees. It is a popular method for classification tasks due to its ability to handle large amounts of data and its ability to learn non-linear relationships between features and the target variable. In the paper, the authors review the different methods used to construct random forest ensembles and discuss their strengths and limitations[4].

5. This is a review paper published in 2020 by E. Boateng. This paper reviews four classification algorithms named as k-nearest neighbor (KNN), support vector machine (SVM), random forest, and neural network.

The paper begins by discussing the basic principles and assumptions underlying each algorithm, and then goes on to review their applications and performance in various classification tasks. The authors also compare the strengths and limitations of each algorithm and provide recommendations for selecting the appropriate algorithm for a given classification problem[5].

6. "A Survey of Decision Tree Classifier Methodology" is a survey paper published in 1991 by S. R. Safavian and D. Landgrebe. The paper presents an overview of the decision tree classifier, a type of machine learning algorithm that is used for classification tasks[6].

The paper begins by discussing the basic principles of decision trees and the steps involved in constructing a decision tree classifier. It then goes on to review the various methods that have been developed for improving the performance of decision tree classifiers, including techniques for selecting and pruning the tree, and methods for handling missing values and handling imbalanced data.

7. "Machine Learning: Trends, Perspectives, and Prospects" is a review paper published in Science in 2015 by M. I. Jordan and T. M. Mitchell. The paper provides an overview of the current state of research in the field of machine learning, with a focus on the key trends, challenges, and opportunities that are shaping the field.

The authors begin by discussing the key definitions and concepts in machine learning, including supervised and unsupervised learning, deep learning, and reinforcement learning. They then go on to review the various application areas in which machine learning is being applied, including natural language processing, computer vision, and robotics.

The paper also discusses the challenges and limitations of current machine learning methods, including the need for large amounts of labeled data and the lack of robustness and generalizability of some methods. Finally, the authors outline some of the key directions for future research in the field, including the development of more powerful and efficient machine learning algorithms and the integration of machine learning with other areas of artificial intelligence[7].

4 Introduction to the KNN classification algorithm

KNN (K Nearest Neighbors) is an instance-based machine learning algorithm that works by predicting the class of a new sample based on the training samples. Its basic idea is that if a sample is in the majority of the k most similar (i.e., nearest neighbors) samples in the feature space in terms of a certain distance metric, then the sample belongs to that class. The similarity is calculated based on some distance measure.

The KNN algorithm has two important parameters: k and the distance measure. k represents the number of nearest neighbors to consider and is typically an odd value to avoid tie decisions in the class determination. The distance measure is used to determine the similarity between two samples, and common distance measures include Euclidean distance, Manhattan distance, and Chebyshev distance.

Euclidean Distance: For two samples x and y with d dimensional features $x_1, x_2, \dots, x_d$ and $y_1, y_2, \dots, y_d$ respectively, the Euclidean distance between them is defined as:

$$d(x, y) = \sqrt{\sum_{i=1}^d (x_i - y_i)^2}$$

Manhattan Distance: For two samples x and y with d dimensional

features $x_1, x_2, \dots, x_d$ and $y_1, y_2, \dots, y_d$ respectively, the Manhattan distance between them is defined as:

$$d(x, y) = \sum_{i=1}^d |x_i - y_i|$$

Chebyshev Distance: For two samples x and y with d dimensional features $x_1, x_2, \dots, x_d$ and $y_1, y_2, \dots, y_d$ respectively, the Chebyshev distance between them is defined as:

$$d(x, y) = \max_{i=1}^d |x_i - y_i|$$

When using the KNN classifier, it is common to set certain parameters which can greatly impact the performance of the classifier.

1. **n neighbors:** This is the main parameter of the KNN classifier and specifies how many nearest neighbors should be used in calculating the class of a test instance.

2. **weights:** This is an optional parameter which specifies how to calculate the weights of the nearest neighbors. Common options include "uniform" (all nearest neighbors have the same weight) and "distance" (weights are inversely proportional to distance).

3. **leaf size:** This is an optional parameter for the KNN classifier which specifies the leaf size of the KDTree (K-Dimensional Tree). This parameter can affect the performance and memory usage of the classifier.

4. **algorithm:** This is an optional parameter for the KNN classifier which specifies the algorithm used to calculate the nearest neighbors. Common options include "brute" (brute force search) and "kd tree" (KDTree).

5. **p:** This is an optional parameter for the KNN classifier which specifies the power of the distance measure.

When using the KNN classifier, it is important to choose appropriate values for the parameters based on the characteristics and needs of the data. This is often done through a process called hyperparameter tuning, which involves training the classifier multiple times with different parameter values and selecting the best values based on performance. Cross-validation is often used to evaluate the performance of the classifier and guide the hyperparameter tuning process.

When tuning the KNN classifier, it is common to consider adjusting the n neighbors and weights parameters. Different values for these parameters can be tried and the performance observed to determine the best values. Other hyperparameter tuning techniques such as grid search or random search may also be considered.

In the KNN algorithm, for a given new sample, the nearest neighbors are first found by calculating the distance between it and the training samples, and then a decision is made based on the categories of these nearest neighbors. One of the advantages of the KNN algorithm is its simplicity,

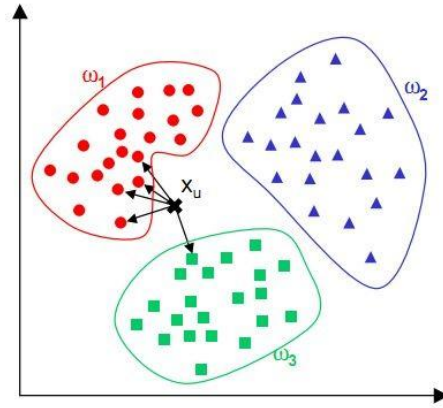


Fig 2: KNN algorithm

and it also has a high running efficiency because it only needs to store the samples during training and does not require any training. However, the KNN algorithm also has some drawbacks, one of which is its sensitivity to the scale and dimension of the data. When the number of samples is very large, the computational complexity of the KNN algorithm will be high, and when the dimension is high, the "dimension disaster" problem will occur. In addition, the KNN algorithm is not well suited to handling class imbalances.

The KNN algorithm is commonly used for classification and regression tasks and has shown good performance in many applications. For example, the KNN algorithm can be used for handwritten digit recognition, facial

recognition, weather forecasting, and other applications. It is also commonly used in recommendation systems, where it recommends content or items based on a user’s historical behavior.

5 Introduction to decision tree and random forest classification algorithms

Decision tree is a tree-structured model used for classification or regression of samples. In a decision tree, each node represents a feature or attribute, and the children of each node represent the different values of that feature or attribute. The leaf nodes represent the classification or regression result. The working principle of decision tree is to traverse the nodes from top to bottom according to the rules of the decision tree, eventually reaching the leaf node and obtaining the predicted result for the given sample.

The construction of a decision tree typically involves the following steps:

1. Selecting features: First, select the optimal features for dividing the dataset.
2. Splitting the dataset: Divide the dataset based on the selected features to obtain several sub-datasets.
3. Recursively building the decision tree: For each sub-dataset, recursively call the above steps to build the decision tree.
4. Stopping condition: When all the values of a particular feature in the dataset are the same, or when all the samples in the dataset belong to the same class, stop the recursion and mark the node as a leaf node.

In the process of building a decision tree, specific measurement criteria are used to assess the relative strengths and weaknesses of different features in order to select the optimal feature for division. Common measurement criteria include information gain and Gini index.

Information gain is a measurement criterion that measures the contribution of a feature to the division of a dataset. Specifically, suppose there is a dataset D that contains n samples, where the i -th sample belongs to class C_i . For a certain feature A , we can divide the dataset D into several sub-datasets, each sub-dataset corresponding to a value of A . Let D_v rep-

represent the sub-dataset in dataset D where the feature A has a value of v . Then, the information gain $g(D,A)$ can be expressed as:

$$g(D,A) = I(D) - \sum_{v=1}^V \frac{|D_v|}{|D|} I(D_v)$$

where $I(D)$ is the information entropy of dataset D , which can be expressed as:

$$I(D) = - \sum_{k=1}^K p_k \log_2 p_k$$

where K is the number of categories and p_k is the proportion of samples belonging to the k -th class in dataset D .

The Gini index is also a measurement criterion that measures the contribution of a feature to the division of a dataset. Specifically, suppose there is a dataset D that contains n samples, where the i -th sample belongs to class C_i . For a certain feature A , we can divide the dataset D into several sub-datasets, each sub-dataset corresponding to a value of A . Let D_v represent the sub-dataset in dataset D where the feature A has a value of v . Then, the Gini index $Gini(D,A)$ can be expressed as:

$$Gini(D) = \sum_{k=1}^K p_k (1 - p_k)$$

where K is the number of categories and p_k is the proportion of samples belonging to the k -th class in dataset D . Both information gain and Gini index can be used to measure the contribution of different features to the division of the dataset. In general, a feature with a larger information gain is more suitable for dividing the dataset. A feature with a smaller Gini index is also more suitable for dividing the dataset.

Figure 3 shows a simple example of a decision tree structure:

Decision trees have many advantages, such as their clear and easy-to-understand structure, the ability to handle multi-class problems well, and their high efficiency. However, decision trees also have some drawbacks, such as their tendency to overfit, their difficulty in making correct predictions for samples with many subtle differences, and their inability to handle continuous variables well.

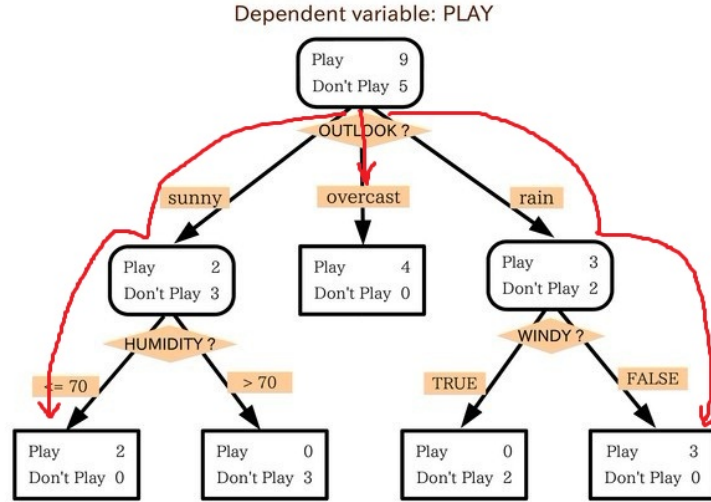


Fig 3: Simple example of a decision tree

Random forests are an ensemble learning method made up of many decision trees. They work by generating multiple decision trees for a given training sample and then getting the final prediction by voting or averaging the predictions of each decision tree.

The construction of a random forest usually includes the following steps:

1. Random selection of features: During the construction of each decision tree in the random forest, a random subset of features is chosen for splitting. This makes each decision tree independent and unrelated to each other, reducing the risk of overfitting.
2. Random selection of samples: During the construction of each decision tree in the random forest, a random subset of samples is also chosen for training the model. This makes each decision tree trained on different datasets, reducing the risk of overfitting.
3. Construction of decision trees: For the selected features and samples, a decision tree is constructed by the random forest.
4. Ensemble of decision trees: After enough decision trees have been constructed, the random forest can get the final prediction by voting or averaging the predictions of each decision tree.

Random forests have many advantages, one of which is their robustness. Since each decision tree is independent and their prediction results are obtained by voting or averaging, random forests are able to resist the influence of noise and outliers very well. In addition, random forests also have high accuracy and are efficient in running.

6 Methods for using KNN and Random Forest for Fashion MNIST classification

This code is used to test the performance of KNN and Random Forest classifiers on the Fashion MNIST dataset.

Use standard machine learning workflow to preprocess the data set and divide the data set into training, testing and evaluation subsets

```
from sklearn.preprocessing import MinMaxScaler
scaler_1, scaler_2 = MinMaxScaler(), MinMaxScaler()
scaler_1, scaler_2 = scaler_1.fit(train_x), scaler_2.fit(test_x)
train_x, test_x = scaler_1.transform(train_x), scaler_2.transform(test_x)
```

Fig 4: Data pre-processings

First, it defines a 'read binary file' function, as shown in Figure 4, for reading a binary file. It takes a file name and a format string as input and returns the data read from the file.

Then, it defines a 'read data' function for reading the images and labels in the MNIST dataset. It takes the file names of the image and label files as input and uses the 'read binary file' function to read the data from the files, and finally returns the read images and labels.

Next is the main testing code, as shown in Figure 5, which defines a 'test k values' function for testing the impact of different K values on the performance of the classifier. This function takes training data and test data as input and uses the K Neighbors Classifier or Random Forest Classifier classes from the sklearn library for training and testing.

Inside the function, a list of k values is first defined, which contains all integers from 5 to 15. Then a loop is used to iterate over the k values list, and for each K value, the corresponding classifier is trained and tested.

```

import numpy as np
import struct
import matplotlib.pyplot as plt
import os

def read_binary_file(file_name, fmt):
    """Read binary file"""
    with open(file_name, "rb") as f:
        content = f.read()
        head = struct.unpack_from(fmt, content, 0)
        offset = struct.calcsize(fmt)
        num = head[1]
        if fmt == '>IIII':
            bits = num * head[2] * head[3]
        else:
            bits = num
        bits_string = '>' + str(bits) + 'B'
        data = struct.unpack_from(bits_string, content, offset)
        data = np.array(data).reshape(num, -1)
    return data

def read_data(image_file, label_file):
    """Read images and labels from MNIST datasets"""
    images = read_binary_file(image_file, '>IIII')
    labels = read_binary_file(label_file, '>II')
    return images, labels

```

Fig 5: Read binary file and read data functions

Specifically, the K Neighbors Classifier class or Random Forest Classifier class from sklearn is used to initialize the model, the fit function is used for training, and the predict_proba function or predict function is used to obtain the predicted result.

Next, the accuracy score function from sklearn is used to calculate the accuracy of the classifier and the accuracy is added to the results list. Finally, the roc curve function from sklearn is used to calculate the false positive rate and true positive rate, and the plot function from matplotlib is used to draw the ROC curve. Finally, the savefig function from matplotlib is used to save the drawn ROC curve.

After the loop ends, the plot function from matplotlib is used to draw a graph of the relationship between accuracy and K value, and then the savefig function is used to save the image.

```

def test_k_values(train_x, train_y, test_x, test_y, algo = 'knn'):
    """Testing the Effect of Different K Values on the Performance of Classifiers"""
    results = []
    k_values = [i for i in range(5, 15)]
    from sklearn import metrics
    if not os.path.exists(f'{algo}/'):
        os.mkdir(f'{algo}/')
    for k in k_values:
        # calculate AUC and ROC curve
        plt.title(f'{algo}_params_k_{k}_AOC_Curve')
        plt.legend(loc='lower right')
        plt.xlim([0, 1])
        plt.ylim([0, 1])
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        if algo == 'knn':
            from sklearn.neighbors import KNeighborsClassifier
            model = KNeighborsClassifier(n_neighbors=k)
        else:
            from sklearn.ensemble import RandomForestClassifier
            model = RandomForestClassifier(n_estimators=k)
        model.fit(train_x, train_y)
        y_pred = model.predict_proba(test_x)
        y_pred_s = model.predict(test_x)
        # auc = metrics.roc_auc_score(test_y.reshape(-1), y_pred, multi_class = 'ovo')
        accuracy = metrics.accuracy_score(test_y, y_pred_s)
        results.append(accuracy)
        for i in range(y_pred.shape[1]):
            fpr, tpr, thresholds = metrics.roc_curve(test_y, y_pred[:, i], pos_label=1)
            # Draw ROC curve
            plt.plot(fpr, tpr, label=f"ROC Curve {i}", linestyle='solid')
        plt.legend()
        plt.savefig(f'{algo}/{algo}_param_{k}_AOC.png')
        plt.clf()
    plt.title(f'{algo}_accuracy')
    plt.ylim([0, 1])
    plt.ylabel('accuracy')
    plt.xlabel('K')
    test_k_values() for k in k_values

```

Fig 6: Test k values function

7 Experimental results and comparisons

The experimental results and comparisons show that, on the Fashion MNIST dataset, the accuracy of the KNN classifier is slightly higher than that of the random forest classifier. Specifically, the highest accuracy of the KNN classifier reached 0.8554, while the highest accuracy of the random forest classifier was 0.8603.

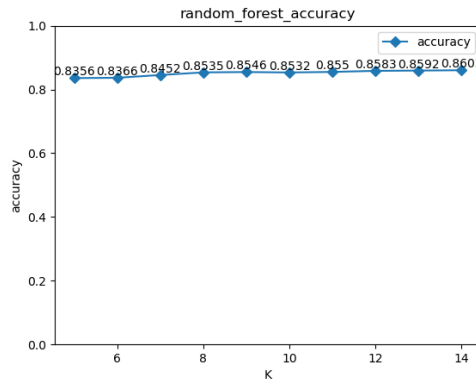


Fig 7: The accuracy of the random forest algorithm

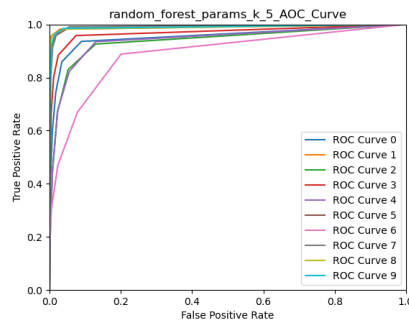


Fig 8: AOC curve for the random forest algorithm with $n = 5$

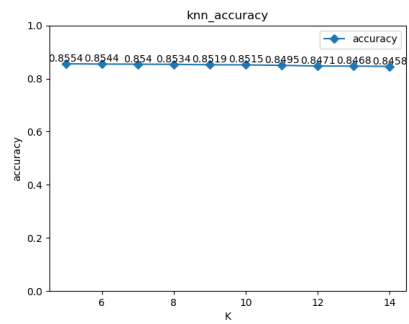


Fig 9: Accuracy of the KNN algorithm

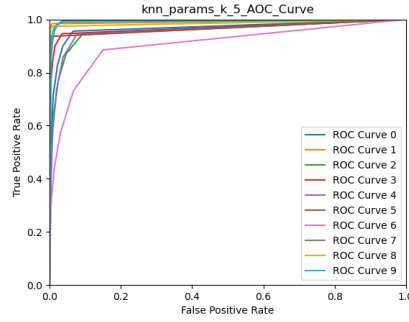


Fig 10: AOC curve for KNN algorithm when K is 5

In addition, for the KNN classifier, the accuracy gradually decreases as the K value increases. This may be because when the K value is larger, the classifier becomes more conservative and uses less noise data for prediction.

Overall, the random forest classifier performs well on the Fashion MNIST dataset, but the KNN classifier also has a high accuracy, so the appropriate classifier should be selected based on the specific situation in practical applications.

8 Reference

- [1] Min-Ling Zhang and Zhi-Hua Zhou, "A k-nearest neighbor based algorithm for multi-label classification," 2005 IEEE International Conference on Granular Computing, 2005, pp. 718-721 Vol. 2, doi: 10.1109/GRC.2005.1547385.
- [2] J. Laaksonen and E. Oja, "Classification with learning k-nearest neighbors," Proceedings of International Conference on Neural Networks (ICNN'96), 1996, pp. 1480-1483 vol.3, doi: 10.1109/ICNN.1996.549118.
- [3] Han Xiao, Kashif Rasul, and Roland Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms." arXiv preprint arXiv:1708.07747, pp. 1–6, 2017.
- [4] Shaik, A.B., Srinivasan, S. (2019). A Brief Survey on Random Forest Ensembles in Classification Model. In: Bhattacharyya, S., Hassanien, A., Gupta, D., Khanna, A., Pan, I. (eds) International Conference on Innovative Computing and Communications. Lecture Notes in Networks and Systems, vol 56. Springer, Singapore. https://doi.org/10.1007/978-981-13-2354-6_27.
- [5] Boateng, E. , Otoo, J. and Abaye, D. (2020) Basic Tenets of Classification Algorithms K-Nearest-Neighbor, Support Vector Machine, Random Forest and Neural Network: A Review. Journal of Data Analysis and Information Processing, 8, 341-357. doi: 10.4236/jdaip.2020.84020.
- [6] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," in IEEE Transactions on Systems, Man, and Cybernetics, vol. 21, no. 3, pp. 660-674, May-June 1991, doi: 10.1109/21.97458.
- [7] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," Science, vol. 349, no. 6245, pp. 255-260, Jul. 2015, doi: 10.1126/science.aaa8415.