CS 205 Artificial Intelligence

Projec 2 Feature Selection with Nearest Neighbor

| Zelai Fang | zfang052 | 862395114 | Hengshuo Zhang | hzhan402 | 862395234 |

Github Link: https://github.com/July-Fang2000/CS205_Project2

Introduction

This project aims to assist us in identifying the optimal feature subset for predicting model performance. Selecting appropriate features not only enhances the predictive capability of the model but also reduces its complexity, improves computational efficiency, and facilitates a deeper understanding of the inherent structure of the data. In this project, we are specifically focusing on implementing two popular feature selection methods, namely Forward Selection and Backward Elimination, for nearest neighbor classifiers.

The project we are developing will accept .txt and .csv files as input file types for the database, which can contain features and labels of the dataset. Users can decide whether to use Forward Selection or Backward Elimination according to their needs. The algorithm needs to evaluate the performance of each feature subset to identify the best one.

Algorithm

Nearest Neighbor Algorithm

In this project, we employ the 1-NN (1-Nearest Neighbor) algorithm, where an instance is classified by a majority vote of its nearest neighbor. The algorithm calculates the Euclidean distance between a new instance and all training samples, determining the closest instance. The new instance is then assigned the label of the nearest training sample. The NN algorithm is particularly suitable for feature selection as it distinctly indicates the influence of each feature on prediction accuracy.

Feature Selection

Forward Selection

In forward selection, the program starts with an empty set of features. For each feature not in the selected feature set, the algorithm evaluates the classification accuracy upon adding the feature to the selected feature set. This evaluation is done using the nearest neighbor algorithm where the feature set used is the currently selected feature set plus the additional feature. The process terminates when the classification accuracy no longer increases or when all features have been selected, and it returns the feature set with the highest classification accuracy.

Backward Elimination

Backward elimination is a greedy strategy that can be considered as the reverse of forward selection. The program starts with a subset containing all feature items. For each feature in the selected set, the algorithm evaluates the classification accuracy when the feature is removed from the selected set. This evaluation is done using the nearest neighbor algorithm. In each iteration, a feature is discarded from the current feature subset, after testing all selected features, it chooses the one whose removal resulted in the least decrease in classification accuracy. The process terminates when the classification accuracy no longer increases or when all features have been eliminated, and it returns the feature set with the highest classification accuracy.

Data Handling

The program will be applicable to datasets with features of varying sizes, making it a versatile feature selection program. Initially, the program will read dataset files of .txt or .csv types. It will separate the labels from the features in the dataset and then standardize the feature values to ensure they have similar scales. The program will preprocess the data in the dataset, calculate the total number of features in the dataset, and appropriately initialize the feature sets for the forward selection or backward elimination algorithms.

Code Implementation

`main.py`

Initiates the program by reading the data file and selected feature selection method, then accordingly invokes preprocessing and feature selection functionalities.

`method.py`

Defines the `NearestNeighborClassifier`, providing methods for calculating distance, finding nearest neighbors, predicting labels, and calculating accuracy. The `search` function implements the feature selection methods, used for computing every feature subset.

- Defines `accuracy_with_features`, a method for calculating accuracy when using feature subsets, which assists in feature selection.

`preprocess.py`

Used for data preprocessing tasks, including reading data from different file types and standardizing feature values.

Results

Forward Selection

Dataset	Subset	Accuracy
CS170_small_Data__27.txt	{10,1,4}	96.1%
CS170_small_Data__32.txt	{3,5}	96.4%
CS170_small_Data__33.txt	{8,3}	97.6%
CS170_large_Data__30.txt	{11,18}	96.7%
CS170_large_Data__32.txt	{3,6}	96.8%
CS170_large_Data__33.txt	{4,10}	97.7%
CS170_XXXlarge_Data__17.txt	{16,17}	97.1%
data.csv	{28,14,22,24,18,20,7,23,21,25,8,16}	98.1%

Backward Elimination

Dataset	Subset	Accuracy
CS170_small_Data__27.txt	{1,10}	95.9%
CS170_small_Data__32.txt	{3,5}	96.4%
CS170_small_Data__33.txt	{3,8}	97.6%
CS170_large_Data__30.txt	{11,18}	96.7%
CS170_large_Data__32.txt	{3,6}	96.8%
CS170_large_Data__33.txt	{4,10}	97.7%
CS170_XXXlarge_Data__17.txt	{3,4,5,6,10,12,13,14,15,18,20,21,22,23,24,26,27,28,29,33,34,35,36,37,38,40,45,47,51,52,53,55,58,61,63,64,66,69,70,71,72,73,78,79,80}	74.5%
data.csv	{1,2,3,4,5,7,8,11,12,13,14,16,17,18,19,20,21,24,25,26,27,28,30}	97.5%

Processing a Real-World Classification Dataset

Data Resource Link: https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data

In the selection of external data, we utilize the Breast Cancer Wisconsin (Diagnostic) dataset from Kaggle, originally sourced from the UCI Machine Learning Repository. The features of this dataset are computed from digitized images of fine needle aspirates (FNA) of a breast mass, with the characteristics describing cell nuclei. Each instance in the dataset corresponds to an individual image, with the `diagnosis` attribute serving as the label, classifying the cancer as either Malignant (M) or Benign (B).

The dataset is named "data.csv", wherein the first column represents patient ID, the second column signifies the diagnostic attribute, and the following columns from the third to the thirty-second encapsulate the feature attributes. The 30 features are three different parameter details for ten real-valued features, including mean values, standard error, and worst or maximum values.

Feature Information

1. id: continuous
2. diagnosis: String (M = malignant, B = benign)
3. Ten real-valued features:
   - radius: continuous
   - texture: continuous
   - perimeter: continuous
   - area: continuous
   - smoothness: continuous
   - compactness: continuous
   - concavity: continuous
   - concave points: continuous
   - symmetry: continuous
   - fractal dimension: continuous

Prior to the commencement of search operations, we first perform z-normalization on the data, normalizing continuous values to ensure that features with larger numerical ranges do not disproportionately influence the nearest neighbor algorithm. Additionally, we re-encode the `diagnosis` label of M and B using integer values for easier classification, with malignant = 1 and benign = 2.

After employing feature selection algorithms of forward selection and backward elimination, we find that when using forward selection for feature selection, the final result retains 12 feature attributes with an accuracy of 98.1%. When using backward elimination for feature selection, the final result retains 23 feature attributes with an accuracy of 97.5%. The accuracy of both methods is very similar. For forward selection, choosing fewer features indicates that certain attributes have a stronger impact on the diagnosis of cancer, effectively summarizing necessary information from the dataset. Meanwhile, for backward elimination, the retention of more features suggests that the removal of any additional attributes could potentially degrade the performance of the classifier.

Conclusion

The project demonstrates the potential of nearest neighbor classifiers, especially when paired with forward selection and backward elimination feature selection methods. The high accuracy of both synthetic and real datasets demonstrates the practical relevance and wide applicability of the project. Furthermore, it deepens our understanding of feature selection techniques, which are crucial in the field of machine learning tasks for high-dimensional data.

Reference

[1] *Russell, S., & Norvig, P. (2009). Artificial Intelligence: A Modern Approach (3rd ed.). Upper Saddle River, NJ: Pearson.*

[2] *Keogh, D. (2023). Introduction to Artificial Intelligence [PowerPoint slides]. CS205: AI Course.*

[3] *Microsoft. (n.d.). Normalize data (Azure Machine Learning). Retrieved from https://learn.microsoft.com/en-us/azure/machine-learning/component-reference/normalize-data?view=azureml-api-2*

[4] *UCI Machine Learning Repository. (n.d.). Breast Cancer Wisconsin (Diagnostic) Data Set. Retrieved from https://www.kaggle.com/uciml/breast-cancer-wisconsin-data*

Trace of Sample Output

Forward Selection <File: CS170_small_Data__32.txt>

```
Welcome to Zelai Fang and Hengshuo Zhang Feature Selection Algorithm.
Type in the name of the file to test: CS170_small_Data__32.txt
Type the number of the algorithm you want to run.
     1) Forward Selection
     2) Backward Elimination
1
This dataset has 10 features(not including the class attribute), with 1000 instances.
Running nearest neighbor with all 10 features, using "leaving-one-out" evaluation, I get an accuracy of 77.6%
Beginning search.
Using feature(s) { 1 } accuracy is 69.8 %
Using feature(s) { 2 } accuracy is 70.2 %
Using feature(s) { 3 } accuracy is 84.8 %
Using feature(s) { 4 } accuracy is 71.4 %
Using feature(s) { 5 } accuracy is 71.3 %
Using feature(s) { 6 } accuracy is 70.1 %
Using feature(s) { 7 } accuracy is 70.0 %
Using feature(s) { 8 } accuracy is 70.5 %
Using feature(s) { 9 } accuracy is 69.4 %
Using feature(s) { 10 } accuracy is 71.9 %
Feature set { 3 } was best, accuracy is 84.8 %

Using feature(s) { 3,1 } accuracy is 86.6 %
Using feature(s) { 3,2 } accuracy is 85.6 %
Using feature(s) { 3,4 } accuracy is 85.5 %
Using feature(s) { 3,5 } accuracy is 96.4 %
Using feature(s) { 3,6 } accuracy is 83.0 %
Using feature(s) { 3,7 } accuracy is 84.8 %
Using feature(s) { 3,8 } accuracy is 84.9 %
Using feature(s) { 3,9 } accuracy is 85.0 %
Using feature(s) { 3,10 } accuracy is 84.0 %
Feature set { 3,5 } was best, accuracy is 96.4 %
```

```
**Using feature(s) { 3,5,1 } accuracy is 95.4 %**
Using feature(s) { 3,5,2 } accuracy is 93.8 %
Using feature(s) { 3,5,4 } accuracy is 93.8 %
Using feature(s) { 3,5,6 } accuracy is 94.8 %
Using feature(s) { 3,5,7 } accuracy is 94.2 %
Using feature(s) { 3,5,8 } accuracy is 94.0 %
Using feature(s) { 3,5,9 } accuracy is 94.0 %
Using feature(s) { 3,5,10 } accuracy is 94.1 %
(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set { 3,5,1 } was best, accuracy is 95.4 %

… steps omitted here

Using feature(s) { 3,5,1,9,10,7,8,2 } accuracy is 79.4 %
Using feature(s) { 3,5,1,9,10,7,8,4 } accuracy is 80.5 %
Using feature(s) { 3,5,1,9,10,7,8,6 } accuracy is 79.1 %
(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set { 3,5,1,9,10,7,8,4 } was best, accuracy is 80.5 %

Using feature(s) { 3,5,1,9,10,7,8,4,2 } accuracy is 78.3 %
Using feature(s) { 3,5,1,9,10,7,8,4,6 } accuracy is 78.9 %
(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set { 3,5,1,9,10,7,8,4,6 } was best, accuracy is 78.9 %

Using feature(s) { 3,5,1,9,10,7,8,4,6,2 } accuracy is 77.6 %
(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set { 3,5,1,9,10,7,8,4,6,2 } was best, accuracy is 77.6 %

Finished search!! The best feature subset is { 3,5 }, which has an accuracy of
96.4 %
```

Backward Elimination <File: CS170_small_Data__33.txt>

```
Welcome to Zelai Fang and Hengshuo Zhang Feature Selection Algorithm.
Type in the name of the file to test: CS170_small_Data__33.txt
Type the number of the algorithm you want to run.
    1) Forward Selection
    2) Backward Elimination
2
This dataset has 10 features(not including the class attribute), with 1000
instances.
Running nearest neighbor with all 10 features, using "leaving-one-out"
evaluation, I get an accuracy of 77.8%
Beginning search.
Using feature(s) { 2,3,4,5,6,7,8,9,10 } accuracy is 78.9 %
Using feature(s) { 1,3,4,5,6,7,8,9,10 } accuracy is 79.0 %
Using feature(s) { 1,2,4,5,6,7,8,9,10 } accuracy is 76.8 %
Using feature(s) { 1,2,3,5,6,7,8,9,10 } accuracy is 78.3 %
Using feature(s) { 1,2,3,4,6,7,8,9,10 } accuracy is 79.2 %
Using feature(s) { 1,2,3,4,5,7,8,9,10 } accuracy is 79.5 %
Using feature(s) { 1,2,3,4,5,6,8,9,10 } accuracy is 77.1 %
Using feature(s) { 1,2,3,4,5,6,7,9,10 } accuracy is 70.5 %
Using feature(s) { 1,2,3,4,5,6,7,8,10 } accuracy is 80.3 %
Using feature(s) { 1,2,3,4,5,6,7,8,9 } accuracy is 78.2 %
Feature set { 1,2,3,4,5,6,7,8,10 } was best, accuracy is 80.3 %

Using feature(s) { 2,3,4,5,6,7,8,10 } accuracy is 80.2 %
Using feature(s) { 1,3,4,5,6,7,8,10 } accuracy is 82.8 %
Using feature(s) { 1,2,4,5,6,7,8,10 } accuracy is 76.0 %
Using feature(s) { 1,2,3,5,6,7,8,10 } accuracy is 82.7 %
Using feature(s) { 1,2,3,4,6,7,8,10 } accuracy is 80.2 %
Using feature(s) { 1,2,3,4,5,7,8,10 } accuracy is 82.3 %
Using feature(s) { 1,2,3,4,5,6,8,10 } accuracy is 78.6 %
Using feature(s) { 1,2,3,4,5,6,7,10 } accuracy is 73.8 %
Using feature(s) { 1,2,3,4,5,6,7,8 } accuracy is 82.2 %
Feature set { 1,3,4,5,6,7,8,10 } was best, accuracy is 82.8 %

Using feature(s) { 3,4,5,6,7,8,10 } accuracy is 83.4 %
Using feature(s) { 1,4,5,6,7,8,10 } accuracy is 76.9 %
```

```
Using feature(s) { 1,3,5,6,7,8,10 } accuracy is 83.7 %
Using feature(s) { 1,3,4,6,7,8,10 } accuracy is 83.3 %
Using feature(s) { 1,3,4,5,7,8,10 } accuracy is 83.8 %
Using feature(s) { 1,3,4,5,6,8,10 } accuracy is 81.9 %
Using feature(s) { 1,3,4,5,6,7,10 } accuracy is 74.4 %
Using feature(s) { 1,3,4,5,6,7,8 } accuracy is 84.1 %
Feature set { 1,3,4,5,6,7,8 } was best, accuracy is 84.1 %

… steps omitted here

Using feature(s) { 5,7,8 } accuracy is 83.1 %
Using feature(s) { 3,7,8 } accuracy is 95.0 %
Using feature(s) { 3,5,8 } accuracy is 92.3 %
Using feature(s) { 3,5,7 } accuracy is 73.3 %
Feature set { 3,7,8 } was best, accuracy is 95.0 %

Using feature(s) { 7,8 } accuracy is 85.4 %
Using feature(s) { 3,8 } accuracy is 97.6 %
Using feature(s) { 3,7 } accuracy is 74.2 %
Feature set { 3,8 } was best, accuracy is 97.6 %

Using feature(s) { 8 } accuracy is 82.4 %
Using feature(s) { 3 } accuracy is 76.8 %
(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set { 8 } was best, accuracy is 82.4 %

Using feature(s) {  } accuracy is 18.4 %
(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set {  } was best, accuracy is 18.4 %

Finished search!! The best feature subset is { 3,8 }, which has an accuracy of
97.6 %
```

Code

Github Link: https://github.com/July-Fang2000/CS205_Project2

method.py

```
import numpy as np


# Define the Nearest Neighbor Classifier
class NearestNeighborClassifier:
    def __init__(self, datas, labels):
        # Initialize the classifier with data and labels
        self.datas = datas
        self.labels = labels

    @staticmethod
    def calculate_distances(data, datas):
        # Compute and return the Euclidean distance between data points
        return np.sum((datas - data) ** 2, axis=1)

    def find_nearest(self, index):
        # Compute distances from the given data point to all others
        # Replace the distance to itself with infinity to avoid picking itself
        # Return the index of the nearest data point
        distances = self.calculate_distances(self.datas[index], self.datas)
        distances[index] = float('inf')
        return np.argmin(distances)

    def predict(self, index):
        # Predict the label of the given data point by looking at its nearest neighbor
        return self.labels[self.find_nearest(index)]

    def accuracy(self):
        # Compute and return the accuracy of the classifier
        correct = 0
        for i in range(len(self.datas)):
            if self.labels[i] == self.predict(i):
                correct += 1
        return round((correct / len(self.datas)) * 100, 1)

    def accuracy_with_features(self, features):
         # Compute and return the accuracy of the classifier when only a subset of features is
used
        datas_sub = self.datas[:, features]
        sub_classifier = NearestNeighborClassifier(datas_sub, self.labels)
        return sub_classifier.accuracy()
```

```
def search(method, classifier):
      # Depending on the method chosen by the user, call the appropriate feature selection
function
    feature_size = len(classifier.datas[0])
    instance_size = len(classifier.datas)
        print("This dataset has {} features(not including the class attribute), with {}
instances."
          .format(feature_size, instance_size))
    print("Running nearest neighbor with all {} features, "
          "using \"leaving-one-out\" evaluation, I get an accuracy of {}%"
          .format(feature_size, classifier.accuracy()))
    print("Beginning search.")
    if method == "1":
        feature_selection(classifier, feature_size, method="forward")
    elif method == "2":
        feature_selection(classifier, feature_size, method="backward")
    else:
        print("Choose Wrong Method! Please type 1 or 2!")


def explore_features(feature_set, size, method):
     # This function generates all possible combinations of feature sets by adding or removing
one feature at a time.
    # 'feature_set' is the current set of features,
    # 'size' is the total number of features,
    # 'method' is the selection method ('forward' or 'backward').

    explored_sets = []  # Initialize an empty list to store all explored feature sets

    if method == "forward":
        # In forward selection, we add one feature at a time
        for i in range(size):
            cur_set = list(feature_set)  # Create a copy of the current feature set
            if i not in cur_set:  # If the feature is not already in the set
                cur_set.append(i)  # Add it
                explored_sets.append(cur_set)  # Add the new set to the list of explored sets
    elif method == "backward":
        # In backward selection, we remove one feature at a time
        for i in range(size):
            cur_set = list(feature_set)  # Create a copy of the current feature set
            if i in cur_set:  # If the feature is in the set
                cur_set.remove(i)  # Remove it
                explored_sets.append(cur_set)  # Add the new set to the list of explored sets
    else:
        raise ValueError("Invalid method. Choose 'forward' or 'backward'.")

    return explored_sets  # Return all explored feature sets
```

```
def feature_selection(classifier, size, method):
    # This function performs the actual feature selection, based on the chosen method (forward
or backward).
    # 'classifier' is the classifier used for evaluating feature subsets,
    # 'size' is the total number of features,
    # 'method' is the selection method ('forward' or 'backward').

    # Check if the method is valid
    if method not in ['forward', 'backward']:
        raise ValueError("Invalid method. Choose 'forward' or 'backward'.")

    max_accuracy = -2  # Initialize the maximum accuracy to a very low value
    feature_set = list(range(size)) if method == 'backward' else []  # Initialize the feature
set based on the method

    # Loop through each feature
    for i in range(size):
        curr_accuracy = -1  # Initialize the current maximum accuracy to a very low value
        sets = explore_features(feature_set, size, method)  # Get all possible feature sets
for this iteration

        # Loop through each possible feature set
        for expanded in sets:
            accuracy = classifier.accuracy_with_features(expanded)  # Compute the accuracy for
this feature set
            print("Using feature(s) {", ','.join(map(str, [feature + 1 for feature in
expanded])), "} accuracy is",
                  accuracy, "%")
            if accuracy > curr_accuracy:  # If this feature set is better than the current
best
                curr_accuracy = accuracy  # Update the current maximum accuracy
                feature_set = expanded  # Update the current best feature set

        if curr_accuracy > max_accuracy:  # If the current best feature set is better than the
overall best
            max_accuracy = curr_accuracy  # Update the maximum accuracy
            max_set = feature_set  # Update the best feature set
        else:
            print("(Warning, Accuracy has decreased! Continuing search in case of local
maxima)")

        print("Feature set {", ','.join(map(str, [feature + 1 for feature in feature_set])),
"} was best, accuracy is",
              curr_accuracy, "%\n")

    # Print the best feature set and its accuracy
```

```
    print("Finished search!! The best feature subset is {", ','.join(map(str, [feature + 1 for feature in max_set])),
          "}, which has an accuracy of", max_accuracy, "%\n")
```

preprocess.py

```
import numpy as np


# Define a function to read data from a .txt file
def read_txt(filename):
    # Initialize empty lists for data and labels
    datas = []
    labels = []
    # Open the file for reading
    with open("./dataset/" + filename, 'r') as file:
        # Loop through each line in the file
        for line in file:
            # Split the line into a list of values
            data = line.split()
            # The first value is the label, append it to the labels list
            labels.append(float(data[0]))
            # The rest of the values are the features, append them to the datas list
            datas.append([float(x) for x in data[1:]])
    # Return the data and labels
    return datas, labels


# cite: https://learn.microsoft.com/en-us/azure/machine-learning/component-reference/normalize-data?view=azureml-api-2
# def normalized_data(data):
#         datas_normalized = 2 * (data - np.min(data, axis=0)) / (np.max(data, axis=0) - np.min(data, axis=0)) - 1
#     return datas_normalized

# Define a function to standardize the data
def standard_data(data):
    # Subtract the mean and divide by the standard deviation
    datas_standard = (data - np.mean(data, axis=0)) / np.std(data, axis=0)
    # Return the standardized data
    return datas_standard
```

main.py

```
import pandas as pd
import os
from preprocess import *
from method import *

print("Welcome to Zelai Fang and Hengshuo Zhang Feature Selection Algorithm.")

# Get the file name from the user
filename = input("Type in the name of the file to test: ")

# Check the file extension of the input file
_, file_extension = os.path.splitext(filename)

# Get the algorithm choice from the user
method = input("Type the number of the algorithm you want to run.\n"
               "     1) Forward Selection\n"
               "     2) Backward Elimination\n")

# Initialize empty lists for data and labels
datas = []
labels = []

# Read the data based on its file extension
if file_extension == ".txt":
    # If it's a .txt file, use read_txt function from preprocess.py
    datas, labels = read_txt(filename)
elif file_extension == ".csv":
    # If it's a .csv file, use pandas to read the csv file
     # Drop the first and the last column, and map 'M' and 'B' to 1 and 2 respectively in the
first column
    data = pd.read_csv("./dataset/"+filename)
    data = data.drop(data.columns[[0, -1]], axis=1)
    data[data.columns[0]] = data[data.columns[0]].map({'M': 1, 'B': 2})
    data = data.values
    labels = data[:, 0]
    datas = data[:, 1:]

# Standardize the data
datas = standard_data(datas)

# Instantiate a Nearest Neighbor Classifier with the data and labels
classifier = NearestNeighborClassifier(datas, labels)

# Perform the feature selection based on the user's choice of method
search(method, classifier)
```