Learning From Differences

Oliver Linger

March 20, 2024

Final-Year Project - B.Sc. in Computer Science

Supervisor: Dr. Derek Bridge

Department of Computer Science

University College Cork

Abstract

Declaration of Originality

In signing this declaration, you are confirming, in writing that the submitted work is entirely your own original work, except where it clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award.

1. This body of work is all my own, unless clearly stated otherwise, with the full and proper accreditation;
2. With respect to my work: none of it has been submitted to any educational institution contributing in any way to an educational award.
3. With respect to anyone else's work: all diagrams, text, code, or ideas have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or print.

Signed

Date: March 20, 2024

Acknowledgements

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning encompasses various paradigms and methodologies aimed at enabling computational systems to learn from data and make predictions or decisions without explicit programming. Among the diverse approaches in machine learning, model-based and instance-based learning represent two fundamental strategies, each with distinct advantages and disadvantages.

1.1 Model-Based Learning

Model-based learning involves the construction of explicit representations of the underlying relationships within the data. These representations, often referred to as models, capture the patterns and structures present in the dataset and can be used to make predictions or infer insights from new, unseen data points. For instance, artificial neural networks (ANNs) exemplify model-based learning by discerning trends and patterns in data.

1.1.1 Advantages of Model-Based Learning:

1. **Generalization:** Well-constructed models have the ability to generalize patterns learned from the training data to unseen instances, thereby making accurate predictions on new data samples. This generalization capability is essential for robust performance across diverse datasets and real-world scenarios. For example, a well-trained regression model can accurately predict housing prices in different regions based on historical data. The book [2] provides insights into the generalization capabilities of model based learning.

2. **Robustness:** Models learn underlying patterns and assumptions within data, making them more robust to noisy outliers. By capturing trends within data, models can effectively mitigate noise, enhancing their reliability in real-world scenarios. The book [2] also shows the ability of models to learn underlying patterns within data, making them far more robust to noisy data.

3. **Efficiency:** Once trained, model-based systems can quickly return predictions, making them suitable for real-time applications where rapid decision-making is essential. The journal [5] offers an insight how the efficiency of model based learning can be implemented to make predictions on big data.

1.1.2 Disadvantages of Model-Based Learning:

1. **Sensitivity to Assumptions:** Models often rely on simplifying assumptions about data distribution and relationships. Deviations from these assumptions can significantly degrade performance, leading to inaccurate predictions or biased outcomes. For example, a model trained to predict house prices based on 2017 data may inaccurately predict house prices for 2023. The book "Machine Learning: A Bayesian and Optimization Perspective" by Sergios Theodoridis and Konstantinos Koutroumbas emphasizes the crucial role assumptions play in model sensitivity, stressing their significance in machine learning applications [12]

2. **Interpretability and Decision-Making:** Certain models, like Neural Networks, lack transparency in decision-making despite their high accuracy. In critical domains like healthcare and finance, interpretability is crucial for validating decisions and ensuring trust in the system. This disadvantage of lack of transparency is depicted well in the journal [11].

3. **Limited Flexibility:** Traditional model-based algorithms struggle to capture complex, non-linear relationships in high-dimensional or unstructured data, limiting their performance in tasks with intricate patterns or diverse feature interactions.

1.2 Instance-Based Learning

Instance-based learning, also known as memory-based learning or lazy learning, eschews explicit model construction in favor of storing and manipulating instances or examples from the training data. Instead of deriving general rules or representations, instance-based methods make predictions based on the similarity between new instances and those observed in the training set. A classic example of instance-based learning is the k-nearest neighbors (KNN) algorithm. The advantages and disadvantages discussed with instance based learning are seen in the paper [1].

1.2.1 Advantages of Instance-Based Learning:

1. **Flexibility:** Instance-based methods adapt to the characteristics of the training data, making them suitable for tasks with complex, non-parametric relationships or evolving patterns. They can capture nuances and irregularities without imposing rigid assumptions.
2. **Explainability:** The transparency of instance-based learning enables analysts to trace predictions back to specific examples in the training set, facilitating interpretability and trust in the system. The journal [11] highlights this issue for black box model based learning such as neural nets.

1.2.2 Disadvantages of Instance-Based Learning:

The disadvantages discussed with instance based learning are discussed in the paper [1].

1. **Computational Complexity:** Storing and manipulating the entire training dataset can lead to high memory consumption and computational overhead during prediction, limiting scalability.
2. **Susceptibility to Noise and Redundancy:** Instance-based approaches are sensitive to noisy or irrelevant features, which can lead to suboptimal performance or overfitting. Redundant instances or outliers may distort similarity measures, compromising model robustness.

1.2.3 Motivation for Blended Learning

The advantages and disadvantages of both model-based and instance-based learning highlight the need for a versatile and robust approach to machine learning. While model-based learning excels in generalization and robustness, it lacks transparency and struggles with complex relationships. Instance-based learning offers flexibility and explainability but suffers from computational complexity and susceptibility to noise.

The motivation for blended learning arises from leveraging the strengths of both approaches while mitigating their weaknesses. Blended methods aim to capitalize on the advantages of model-based and instance-based learning, circumventing their limitations. By integrating the interpretability of instance-based learning with the accuracy of model-based systems, blended learning offers a promising solution for critical domains like healthcare and finance. The journal [11] provides further evidence that it is important to move towards interpretability for our models going forward, so that critical systems are interpretable and repairable by humans going forward.

In recent years, neural networks have gained significant attention in regression tasks [9]. Inspired by case-based reasoning principles [10], researchers are exploring predictive ensemble models based on these principles [13]. Deviating from conventional approaches, some researchers advocate for training neural networks on differences between sets of features to enhance efficiency and transparency [4].

By adopting a blended approach, machine learning systems can achieve greater interpretability, reliability, and performance across diverse applications.

Chapter 2

Literature Review

2.1 Introduction

The exploration of alternative methodologies for enhancing the performance of neural networks in regression tasks has garnered significant attention within the machine learning community in recent years. One innovative approach, inspired by the principles of case-based reasoning (CBR), involves training neural networks to predict differences between problems based on disparities between features rather than the features themselves. This departure from the conventional supervised learning paradigm aims to leverage the inherent advantages of learning from differences, potentially leading to more interpretable and efficient models.

The study under consideration [4] conducts a systematic factorial study to investigate the efficacy of this approach across various datasets and experimental conditions. The findings suggest that neural networks trained on differences demonstrate comparable or even superior performance to those trained using traditional methods, while requiring significantly fewer epochs for convergence. In this section of the literature review, we consider the utilization of case-based reasoning for regression tasks, with a specific focus on the implementation of KNN (K-Nearest Neighbors) and ANN (Artificial Neural Network) frameworks.

As we delve further into the literature review, we extend our exploration of case-based reasoning beyond regression tasks to include classification challenges. The study by Ye et al. [14], titled "Learning Adaptations for Case-based Classification," offers insights into the application of CBR in classification tasks, shedding light on recent advancements and methodologies. Through a detailed analysis of various adaptation strategies and neural network-based approaches, we aim to uncover the effectiveness of CBR in solving classification problems, thereby providing a comprehensive understanding of its utility across different machine learning tasks.

2.2 KNN + ANN for Regression, Utilizing CBR

The exploration of alternative methodologies for enhancing the performance of neural networks in regression tasks has attracted considerable attention within the machine learning community in recent years. One innovative approach, inspired by the principles of case-based reasoning (CBR), involves training neural networks to predict differences between problems based on disparities between features rather than the features themselves. This departure from the conventional supervised learning paradigm aims to leverage the inherent advantages of learning from differences, potentially leading to more interpretable and efficient models. The study under consideration [4] conducts a systematic factorial study to investigate the efficacy of this approach across various datasets and experimental conditions. The findings suggest that neural networks trained on differences demonstrate comparable or even superior performance to those trained using traditional methods, while requiring significantly fewer epochs for convergence. In this section of the literature review, we consider the utilization of case-based reasoning for regression tasks, with a specific focus on the implementation of KNN (K-Nearest Neighbors) and ANN (Artificial Neural Network) frameworks.

2.2.1 Learning From Differences Paper

The paper by [4] introduces the innovative concept of training a neural network based on the disparities between a case and its closest neighbors. Titled "A Factorial Study of Neural Network Learning from Differences for Regression" [4], the study aims to present novel learning methodologies geared towards enhancing performance and fostering a more interpretable learning process. The research evaluates three model variations: a benchmark neural network, a learning from differences model, and an augmented learning from differences model incorporating the original context. The context here refers to the base case, from which differences with its nearest neighbors are calculated.

According to [4], the study's key findings underscore a significant increase in accuracy when incorporating contextual information in the learning process. Moreover, comparable or superior results were achieved with fewer training epochs compared to the basic neural network. This enhancement is attributed partly to the broader training dataset that encompasses neighbors greater than 1. Notably, the paper advocates for a hybrid approach, combining learning from differences with direct feature-based learning, which yields superior results.

The related work for the paper [4] has addressed the use of neural networks in the case-based reasoning process. These papers have focused on exploiting the base case using the idea that adaptation of information is acquirable from the differences between pairs [6]. Since then, various different approaches have been used. More recently, in relation to using neural networks to predict the differences between problems, a few interesting points have been investigated. These points show the ability of a neural network to correct the solution of the most similar retrieved case. These preliminary works expressed in the paper showed that the use of differences plus context yielded superior results [3]. The previous studies examined in the paper [4] did not examine the impact of different parameters during the CBR (case-based reasoning) experiments. The paper [4] made an effort to explore the effect of epochs in the learning process.

The graphical representations in the paper highlight a distinct trend: rapid attainment of high accuracy levels with the learning from differences method, particularly when contextual information is included. Conversely, the basic neural network requires a substantially larger number of epochs to achieve comparable accuracy, if not worse. This disparity suggests the efficiency of the learning from differences approach.

The paper suggests a marginal performance enhancement and a substantial reduction in training epochs through the adoption of the differences method and inclusion of contextual information. However, it is crucial to acknowledge the time overhead associated with retrieving and training using similar cases.

2.2.2 Algorithms

The training algorithm for learning from differences is described in Algorithm 1, while the prediction algorithm is outlined in Algorithm 2. Additionally, a variant of the training algorithm with context included is presented in Algorithm 3, and the corresponding prediction algorithm is provided in Algorithm 4.

Context which is utilized in 3, and 4. Context is the case used to retrieve (Nearest Neighbors). In the training algorithm it is X_i^{train}, and in prediction it is X_j^{test}. These are included in the training, and prediction in Variant 2.

The paper [4] has an associated library where they have implemented a regression-based machine learning model.

Algorithm 1 Training Algorithm for Learning from Differences

```
Input: Dataset $D$, neural network model, number of neighbors $n$, number of epochs $E$
 1: Split $D$ into training set $D^{train}$ and test set $D^{test}$ with a 80%-20% split
 2: Initialize $\Delta D^{train}$ as an empty list
 3: for each case $C_i = (X_i^{train}, y_i^{train}) \in D^{train}$ do
 4:     Retrieve $n$ similar cases from $D^{train}$ using nearest neighbors
 5:     for each similar case $(X_j^{train}, y_j^{train}) \in D^{train}$ do
 6:         Compute the differences: $\Delta D_{ij}^{train} = (X_i^{train} - X_j^{train}, y_i^{train} - y_j^{train})$
 7:     end for
 8: end for
 9: Train the MLPRegressor neural network model using $\Delta D^{train}$ over a few epochs for
    later predictions.
10: Return Trained neural network model $TrainedNN()$
```

Algorithm 2 Prediction Algorithm for Learning from Differences

```
Input: Test dataset $D^{test}$, trained neural network model $TrainedNN()$, number of neigh-
bors $n$
  for each case $X_j^{test} \in D^{test}$ do
      Retrieve $n$ similar cases from $D^{train}$ using nearest neighbors
      Initialize empty list $y^{pred}predictions$
      for each similar case $(X_i^{train}, y_i^{train}) \in D^{train}$ do
          Compute the differences: $\Delta X_{ij}^{test} = X_j^{test} - X_i^{train}$
          Use the trained neural network to predict: $TrainedNN(\Delta X_{ij}^{test}) = \Delta X_{ij}^{pred}$
          Adapt neighbor's $y$ values: $y^{pred}predictions_j = X_i^{train} + \Delta X_{ij}^{pred}$
      end for
      Average the adapted $y$ values: $y_j^{pred} = \frac{1}{n}\sum_{i=1}^{n} y^{pred}predictions$
      Compare $y_j^{pred}$ to $y_j^{test}$ to evaluate accuracy
  end for
```

Algorithm 3 Training Algorithm for Learning from Differences with context

```
Input: Dataset $D$, neural network model, number of neighbors $n$, number of epochs $E$
 1: Split the dataset $D$ into training set $D^{train}$ and test set $D^{test}$ using an 80%-20% split.
 2: Initialize $\Delta D^{train}$ as an empty list.
 3: for each case $C_i = (X_i^{train}, y_i^{train}) \in D^{train}$ do
 4:     Retrieve $n$ similar cases from $D^{train}$ using nearest neighbors.
 5:     for each similar case $(X_j^{train}, y_j^{train}) \in D^{train}$ do
 6:         Compute the differences, with concatenated context : $\Delta D_{ij}^{train} = ((X_i^{train} -$
    $X_j^{train}) : X_i^{train}, y_i^{train} - y_j^{train})$.
 7:         Append $\Delta D_{ij}^{train}$ to $\Delta D^{train}$.
 8:     end for
 9: end for
10: Train the MLPRegressor neural network model using $\Delta D^{train}$.
11: return Trained neural network model $TrainedNN()$.
```

Algorithm 4 Prediction Algorithm for Learning from Differences with context

```
Input: Test dataset $D^{test}$, trained neural network model $TrainedNN()$, number of neighbors $n$
    for each case $X_j^{test} \in D^{test}$ do
        Retrieve $n$ similar cases from $D^{train}$ using nearest neighbors
        Initialize $y_j^{pred}$ as an empty list
        for each similar case $(X_i^{train}, y_i^{train}) \in D^{train}$ do ‘
            Initialize empty list $y^{pred}predictions$
            Compute the differences: $\Delta X_{ij}^{test} = ((X_j^{test} - X_i^{train}) : X_j^{test})$
            Use the trained neural network to predict: $\Delta X_{ij}^{pred} = TrainedNN(\Delta X_{ij}^{test})$
            Adapt neighbor's $y$ values: $y^{pred}predictions_j = y_i^{train} + \Delta X_{ij}^{pred}$
        end for
        Average the adapted $y$ values: $y_j^{pred} = \frac{1}{n}\sum_{i=1}^{n} y_i^{pred}$
        Compare $y_j^{pred}$ to $y_j^{test}$ to evaluate accuracy
    end for
```

The new model proposed in the paper is tested using the factorial study methodology. This process involved finding a network structure through experimental means for each dataset. It entailed training and testing using a wide range of hyperparameters to determine the optimal settings.

Despite its contributions, the paper falls short by confining its attention to regression tasks and not considering classification tasks and exploring additional variations of the learning from differences model beyond context inclusion. Incorporating original values for the base case in the generated differences dataset provides a valuable framework for understanding the observed disparities. While the model with contextual inclusion demonstrates slight improvements over the base model in learning from differences, a more comprehensive examination of the methodology is warranted for further refinement and enhancement.

The paper tested three models using the same factorial methodology: a base version, which is a basic neural network trained empirically; a differences model employing the (CBH) Case-based Heuristic for predictions, (CBH) guides how cases are retrieved with respect to the base case, and how the retrieved cases are adapted to produce a prediction, (CBH) relies on the adaptation of previous problems to solve the current issue; and a differences model with its base context included. The results yielded interesting insights, with similar performance for all three models and a slight improvement in the differences + context model. Notably, the differences and differences + context models required significantly fewer epochs to achieve accuracy compared to the basic neural network, which achieved similar accuracy after more epochs.

Another consideration in the paper [4] was the number of neighbors for both training and prediction. The study revealed no consistent trend across datasets, suggesting dataset-specific requirements.

In conclusion, the learning from differences method achieved equal or higher accuracy compared to the base neural network model.

2.3 Learning Adaptations for Case-based classification

The paper by Ye et al. [14], titled "Learning Adaptations for Case-based Classification," investigates the application of Case-Based Reasoning (CBR) for classification tasks. It explores recent advancements in CBR, particularly focusing on replacing traditional rule-based learning with Case-Based Heuristic (CBH) network models for adaptation. The study introduces a novel model comprising three variations: segmentation of adaptation knowledge based on the classes of the source cases, training a single neural network on differences between problem solutions, and adapting from an ensemble of source cases followed by a majority vote.

The "NN-CDH" method, as proposed in the paper, represents a novel classification approach that operates on pairs of cases, where one serves as the source and the other as the target. These pairs form the basis of adaptation knowledge, which is learned through neural network models. The methodology adopted in [14] emphasizes learning adaptation knowledge through case pairs processed by neural networks.

The study introduces the first neural network-based approach to classification, termed "C-NN-CDH" or Classification with Neural Network-based Case Difference Heuristic. This approach employs neural networks to learn adaptation knowledge from pairs of cases. According to the findings, this methodology outperforms traditional statistical models, as evidenced in [7].

The first approach involves segmenting the dataset by class and training a neural network for all segmented differences set between base cases. This method offers faster training but slightly less accuracy, according to the paper. The use of these CBR methods provides at least two benefits: inertia-free lazy learning and the ability to assess different cases when running classification. Inertial free means that the model does not make assumptions based on the underlying structure of the data, and lazy learning pertains to an algorithm that will postpone generalization until it is certain a new instance needs to be classified. It will only classify as the data comes; it won't try and learn or adapt preemptively.

In the process of CBR, adaptation is often treated as the most difficult. Various methods have been developed by academics over the years to tackle the issue. For example, Leake et al. [8] extrapolate a method in which the case base adaptation cases are populated from previously successful adaptations. Another approach is to have a stored base case and query and retrieve the case most similar to it, and adapting the retrieved case to the base case [3].

In the paper by Ye et al. [14], several design questions had to be addressed. How to approach case differences and how to select case pairs for training. The paper in question [14] considers standard pair selection methods as well as a new training per selection approach based on class-class classification.

The paper [14] also addresses the application of (CDH) method for classification. It discusses the value distance metric, which is a probabilistic method to measure the similarity that allows the comparison of nominal values in a single-dimensional space.

2.3.1 An NN-CDH Approach for Classification

To achieve adaptation, a neural network is integrated into the process. The source and target cases become case pairs, and their adaptation is learned through a neural network. The neural network computes the difference between the source and target pairs. This difference calculation is a crucial step in the Case-Based Reasoning (CBR) system. Once the difference is computed, the predicted result obtained from the neural network is applied to the source solution. This adapted solution is then considered as the final result of the classification process.

The NN-CDH approach represents a fusion of neural network techniques with the principles of Case-Based Reasoning, offering a dynamic and adaptable method for classification tasks. When calculating the difference between the problem and the solution values, it is important to implement a difference function. This presents an interesting challenge for nominal values. The method presented in [14] uses an implicit calculation through machine learning techniques. This replaces traditional (CDH) case difference heuristic approaches for difference calculations. Through the use of a neural network, the paper's method hopes to include the base context in the differences' calculation. This method for generating differences has been aptly named ("C-CDH") case difference heuristic approach for classification.

When calculating the difference, an issue arrives in that we cannot explicitly calculate differences through subtraction, which was the case in the previous paper by Learning

From Differences [4]. An implicit way of learning from differences is required. In the paper by Ye et al. [14], this is called "C-CDH".

This presents an interesting challenge for nominal values. The method presented in [14] uses an implicit calculation through machine learning techniques. When discussing the Neural Network-based Case-Driven Hierarchical (NN-CDH) Approach described in [14], notation will be homogenized with the algorithms from the previous paper.

For clarification when going through the algorithms, Retrieved cases = Source Cases, Target case = The case which we are attempting to predict its y label values.

Multiple variants were created with the paper by Ye et al. [14].

2.3.2 Variant 0: Non-Network C-CDH

Variant 0, as described in [14], serves as a foundational test bed implementation for classification tasks. The algorithms presented here outline the process of building and utilizing a dictionary of case pairs, where each pair consists of a source case and its corresponding target case. Through the use of nearest neighbors, the algorithm identifies similar cases within the training dataset and organizes them based on their source solution. Subsequently, during prediction, the algorithm retrieves the nearest similar cases for each test case and utilizes the corresponding target solutions for classification. This approach embodies the essence of case-based reasoning, where solutions are adapted from similar past cases to classify new instances effectively. The following algorithms detail the steps involved in training and predicting using Variant 0's case-based heuristic approach.

Algorithm 5 Variant 0, Classification Using Case Based Heuristic, Training

```
Input: Dataset $D$, number of neighbors 1
  Split $D$ into training set $D^{train}$ and test set $D^{test}$ with an 80%-20% split
  Create $CasePairs()$ dictionary.
  for each case $C_i = (X_i^{train}, y_i^{train}) \in D^{train}$ do
      Retrieve 1 similar cases from $D^{train}$ using nearest neighbors.
      for each similar case $(X_j^{train}, y_j^{train}) \in D^{train}$ do
          $CasePairs(y_j^{train}).append([X_j^{train} : X_i^{train}, y_i^{train}])$
      end for
  end for
```

Algorithm 6 Variant 0, Classification Using Case Based Heuristic, Prediction

```
Input: Test dataset $D^{test}$, $CasePairs()$
1: Initialize Empty list $y^{pred}$
2: for each case $X_j^{test} \in D^{test}$ do
3:     Retrieve 1 similar cases from $D^{train}$ using nearest neighbors
4:     Initialize $y_j^{pred}$ as an empty list
5:     for each similar case $(X_i^{train}, y_i^{train}) \in D^{train}$ do
6:         Retrieve $r = 1NearestNeighbor(CasePairs(y_i^{train}), [X_i^{train} : X_j^{test}])$
7:         $y_j^{pred} = r$
8:     end for
9: end for
10: return $y^{pred}$
```

2.3.3 Variant 1 - 2: C-NN-CDH

Variants 1 - 2, known as C-NN-CDH, represent a pivotal advancement in classification methodologies by integrating neural networks into the case-based heuristic framework. In this variant, the traditional case-based heuristic approach is augmented with the power of neural networks to adapt solutions from similar cases. The algorithms presented here delineate the training and prediction processes, where a neural network model learns adaptation knowledge from case pairs derived from the training dataset. Unlike Variant 0, which relies solely on case similarity for adaptation, C-NN-CDH leverages the expressive capacity of neural networks to capture intricate relationships between cases and their solutions. Through the fusion of case-based reasoning principles with neural network techniques, Variant 1 - 2 offers a dynamic and adaptable approach to classification tasks, capable of learning complex adaptation patterns and improving classification accuracy.

2.3.4 Variant 3 - 5: C-NN-CDH

Variants 3 - 5 of the classification neural network using case-based heuristics introduce a novel approach to classification by grouping case pairs based on their source solutions. In this framework, multiple specialized adaptation neural networks are employed, each trained to adapt cases of specific solutions to all solutions. Inspired by the ensemble of adaptations for classification approach, these variants share a common grouping and training method but differ in their testing strategies. The algorithms delineate the training process, where case pairs are organized and used to train specialized neural networks for adaptation. The variation arises in the prediction algorithm, where different strategies are employed to classify test instances based on the learned adaptation knowledge. The following section details the nuances of each variant's

Algorithm 7 Variant 1 - 2: Classification Using Neural Net Case Based Heuristic (Training)

```
Input: Dataset $D$, neural network model, number of neighbors $n$, number of epochs $E$
  Split $D$ into training set $D^{train}$ and test set $D^{test}$ with an 80%-20% split
  Create $CasePairs$ list.
  for each case $C_i = (X_i^{train}, y_i^{train}) \in D^{train}$ do
      Retrieve 1 similar case from $D^{train}$ using nearest neighbors.
      for each similar case $(X_j^{train}, y_j^{train}) \in D^{train}$ do
          if Variant 1 then
              $CasePairs$.append($[X_j^{train} : X_i^{train}, y_i^{train}]$)
          else
              if Variant 2 then
                  $CasePairs$.append($[X_j^{train} : X_i^{train} : y_j^{train}, y_i^{train}]$)
              end if
          end if
      end for
  end for
  Train the neural network model using $CasePairs()$
  return Trained neural network model $TrainedNN()$.
```

Algorithm 8 Variant 1 - 2: Classification Using Neural Net Case Based Heuristic (Prediction)

```
Input: Test dataset $D^{test}$, trained neural network model $TrainedNN()$
 1: Initialize Empty list $y^{pred}$
 2: for each case $X_j^{test} \in D^{test}$ do
 3:     Retrieve 1 similar case from $D^{train}$ using nearest neighbors
 4:     for each similar case $(X_i^{train}, y_i^{train}) \in D^{train}$ do
 5:         Use the trained neural network to predict: $y_j^{pred} = TrainedNN([X_i^{train} : X_j^{test} : y_i^{train}])$
 6:     end for
 7: end for
 8: return $y^{pred}$
```

prediction methodology and discusses their implications for classification accuracy and adaptability.

Algorithm 9 Variant 3 - 5: Classification Neural Network Using Case Based Heuristic (Training)

```
Input: Dataset $D$, number of neighbors 1
  Split $D$ into training set $D^{train}$ and test set $D^{test}$ with an 80%-20% split
  Create $CasePairs()$ dictionary.
  Create $AdaptNN()$, a dictionary to store each Neural Network
  List of unique Classes $UniqueClasses$
  for each case $C_i = (X_i^{train}, y_i^{train}) \in D^{train}$ do
      Retrieve 1 similar case from $D^{train}$ using nearest neighbors.
      for each similar case $(X_j^{train}, y_j^{train}) \in D^{train}$ do
          $CasePairs(y_j^{train}).append([X_j^{train} : X_i^{train}, y_i^{train}])$
      end for
  end for
  for each unique $Class$ in $UniqueClasses$ do
      Create a new Classification Neural Network, trained on Case Pairs
      $AdaptNN(Class).fit(CasePairs(Class))$
  end for
  return Trained neural network models dictionary $AdaptNN()$.
```

In variant (5), the prediction method utilizes a Class-to-Class approach for retrieving neighbors. This method retrieves the most similar neighbor for each class from the training dataset. Once the nearest neighbors for each class are identified, the corresponding specialized neural networks associated with those classes are employed to generate predictions for the target instance. Subsequently, a majority vote is conducted among the predictions from the specialized neural networks to determine the final classification outcome.

It is worth considering re-implementing the algorithm to allow for the retrieval of multiple neighbors (N) from each class during the prediction stage. This adjustment could potentially lead to improved results, particularly when dealing with large and varied datasets.

Variant (5) stands out as the most successful approach in the study, primarily due to its Class-to-Class retrieval of neighbors during the prediction stage. By employing multiple specialized neural networks, the variant reduces training time while effectively leveraging implicit learning through differences for classification tasks. This method demonstrates its efficacy in adapting to diverse datasets and achieving robust classification performance.

Algorithm 10 Variant 3 - 5: Classification Using Neural Net Case-Based Heuristic (Prediction)

```
Input: $D^{test}$, $AdaptNN()$, $UniqueClasses$, and $CasePairs()$
  Initialize an empty list $y^{pred}$
  for each case $X_j^{test} \in D^{test}$ do
      if Variant 3 then
          Retrieve 1 similar case from $D^{train}$ using nearest neighbors
          for each similar case $(X_i^{train}, y_i^{train}) \in D^{train}$ do
              $y_j^{pred} = AdaptNN(y_i^{train}).predict([X_i^{train} : X_j^{test}])$
          end for
      else
          if Variant 4 then
              Create an empty list $predictions$
              Retrieve $N$ similar cases from $D^{train}$ using nearest neighbors
              for each similar case $(X_j^{train}, y_j^{train}) \in D^{train}$ do
                  $predictions.$append$(Adapt(y_i^{train}).predict([X_i^{train} : X_j^{test}]))$
              end for
              Perform a majority vote on $predictions$
              $y_j^{pred} = MajorityVote(predictions)$
          else
              if Variant 5 then
                  Create an empty list $preds$
                  for each unique class $Class$ in $UniqueClasses$ do
                      Retrieve 1 similar case from $CasePairs(Class)$
                      for each similar case $(X_i^{CasePair}, y_i^{CasePair}) \in CasePairs(Class)$ do
                          for each $(X_z^{train}, X_q^{train}) \in X_i^{CasePair}$ do
                              $preds.append(AdaptNN(Class).predict(X_z^{train} : X_j^{test}))$
                          end for
                      end for
                      Perform a majority vote on $preds$
                      $y_j^{pred} = MajorityVote(preds)$
                  end for
              end if
          end if
      end if
  end for
  return $y^{pred}$
```

2.3.5 Evaluation Summary

Within the paper [14] the experiments conducted on two different datasets provided answers to several key questions.

The effectiveness of neural network learning adaptation knowledge was demonstrated through consistent performance of one or more C-NN-CDHs, which often outperformed the neural network classifier. Despite the possibility of neural networks learning to discard the source problem and rely solely on the target problem, the non-zero weights associated with the source problem and significant performance differences between C-NN-CDHs and the neural network indicate that C-NN-CDHs effectively learn adaptation knowledge. This suggests that if a neural network can handle the classification task directly, it can also learn adaptation knowledge or the relation between pairs of cases in the task domain.

Surprisingly, variant (1) performed almost identically to variant (2), which also considers the source solution in adaptation. This suggests that the source solution, heavily coupled with the source problem, may not provide additional useful information for adaptation.

In terms of segmenting case pairs by source case solution, variants (1, 2) generally outperformed variants (3, 4) in accuracy on most datasets. This could be because a single adaptation neural network in (1, 2) is well-trained with all pairs of cases, while a specialized adaptation neural network in (3, 4) is trained with segmented examples. However, variants (3, 4) showed faster convergence during training due to training on segmented examples.

The impact of the ensemble of adaptations was investigated, with EAC-NN-CDH (variant (4)) performing similarly to its counterpart variant (3) without ensemble. This suggests that the generalization power of C-NN-CDH produces stable predictions that an ensemble version does not significantly alter.

The usefulness of the class-to-class approach for adaptation was demonstrated by C2C-NN-CDH (variant (5)), which performed differently from and often better than the other C-NN-CDHs. C2C-NN-CDH reaches its prediction by collecting evidence from diverse source cases from all classes, which can provide more global support, especially when there are multiple classes. Moreover, the C2C approach offers the possibility of explanation with contrastive evidence.

2.3.6 Conclusions on Learning adaptations for case based classification: A Neural Network approach

The conclusion of the paper [14] being that (variant (5)) is worth investigating using its C2C-NN-CDH approach to classification. As it yielded the most promising results when compared to all the other variants tested in the paper. The grouping of the dataset based on class outcome and training a specialized neural network, has its benefits in terms of speedup. However, the retrieval of N cases across all classes during testing, has shown to yield better results, in a way that allows an interpretation of the results.

Chapter 3

Design

Overview

3.1 Introduction and Motivation

The design process begins with an examination of the paper [4], which lays out the fundamental algorithm for the learning from differences method. Our primary objective is to enhance and expand upon this algorithm to create a more dynamic and adaptable version. Inspired by the modular design philosophy of Scikit-learn's GitHub repository, our aim is to develop a model that not only mirrors the rigorous hyperparameter testing capabilities of Scikit-learn but also extends beyond, offering increased flexibility and functionality.

Following the implementation of the base models, our focus shifts to introducing variations to optimize the machine learning models for both the `LingerRegressor` and `LingerClassifier`. These variations aim to improve training efficiency and accuracy, thereby enhancing overall model performance. The names chosen for the custom regressor and classifier are `LingerRegressor` and `LingerClassifier`, respectively. These models are designed and implemented to utilize the training from differences approach.

A crucial aspect of this endeavor is to ensure that the introduced variations are easily testable, akin to the comprehensive hyperparameter testing framework already established. This approach enables easy analysis of the models and their variations, facilitating informed decision-making regarding their effectiveness and suitability for specific tasks.

3.2 Linger Regression Base Model: `LingerRegressor`

3.2.1 Objective

The Linger Regression Base Model integrates principles of Case-Based Reasoning (CBR) into regression analysis to enhance interpretability and create a performative model. The objectives include:

- Developing a dynamic regression model using differences between a case and its nearest neighbors, adapting retrieved solutions to the target solution.
- Ensuring compatibility with Scikit-learn for accessibility, ease of implementation, and extensive hyperparameter testing.
- Enabling effective interpretation of predictions for users.
- Implementing the learning from differences algorithm with enhancements for improved performance.
- Creating an easily extendable code base to allow for variations.

Its intended use case is for regression problems that could benefit from a more transparent learning process.

3.2.2 Architecture

The `LingerRegressor` combines K-nearest neighbors (KNN) and `MLPRegressor` from Scikit-learn for flexibility and accuracy. A K-nearest neighbors (KNN) is used in conjunction with a `MLPRegressor` in the fit phase of the machine. During the predict phase, a K-nearest neighbors (KNN) is used once again with the now trained `MLPRegressor` to get the differences' prediction or K neighbors.

3.2.3 Components

- KNN module: Computes nearest neighbors and extracts differences during both training and prediction phases.
- `MLPRegressor`: Offers flexibility in activation functions, solvers, and hyperparameters for regression.

- Supplementary Features: Weighted KNN, Context Addition, Additional Results Column, and Duplication Based on Distance. Each supplementary feature can be activated through a hyperparameter.

3.2.4 Parameterization

Extensive parameterization allows fine-tuning of KNN, `MLPRegressor`, and additional features. All the same hyperparameters are available as in a conventional (KNN) and neural network, with additional hyperparameters for `LingerRegressor`.

3.2.5 Training and Prediction

Training

Given dataset D, split it into D^{train} and D^{test} with an 80%-20% split through random selection. For each case $C_i = (X_i^{train}, y_i^{train}) \in D^{train}$, retrieve n similar cases from D^{train}, $(X_j^{train}, y_j^{train}) \in D^{train}$ using Scikit-learn's neighbors module. Compute the differences between C_i and its neighbors and append them to ΔD^{train} as $(X_i^{train} - X_j^{train}, y_i^{train} - y_j^{train})$.

Algorithm 11 Training Algorithm for Learning from Differences

```
1: Split $D$ into training set $D^{train}$ and test set $D^{test}$ with an 80%-20% split
2: Initialize $\Delta D^{train}$ as an empty list
3: for each case $C_i = (X_i^{train}, y_i^{train}) \in D^{train}$ do
4:     Retrieve $n$ similar cases from $D^{train}$ using nearest neighbors
5:     for each similar case $(X_j^{train}, y_j^{train}) \in D^{train}$ do
6:         Compute the differences: $\Delta D_{ij}^{train} = (X_i^{train} - X_j^{train}, y_i^{train} - y_j^{train})$
7:         Append $\Delta D_{ij}^{train}$ to $\Delta D^{train}$
8:     end for
9: end for
10: Train the MLPRegressor neural network model using $\Delta D^{train}$ over a few epochs for
    later predictions.
11: Return Trained neural network model $TrainedNN()$
```

Prediction

During prediction, for each case X_j^{test}, retrieve n similar cases from D^{train}. Compute the differences $\Delta X_{ij}^{test} = X_j^{test} - X_i^{train}$ for each neighbor. Use the trained neural network to predict ΔX_{ij}^{test}, resulting in ΔX_{ij}^{pred}. For each set of retrieved neighbors from D^{train}, calculate $y_i^{train} + \Delta X_{ij}^{pred}$ and add each n neighbors adapted y values. Once summed, divide by n to obtain y_j^{pred}. Compare y_j^{pred} to y_j^{test} to evaluate accuracy.

Algorithm 12 Prediction Algorithm for Learning from Differences

```
Input: Test dataset $D^{test}$, trained neural network model $TrainedNN()$, number of neighbors $n$
  Initialize empty list $y^{pred}$
  for each case $X_j^{test} \in D^{test}$ do
      Initialize empty list $y_j^{pred}$
      Retrieve $n$ similar cases from $D^{train}$ using nearest neighbors
      Initialize empty list $y^{pred} predictions$
      for each similar case $(X_i^{train}, y_i^{train}) \in D^{train}$ do
          Compute the differences: $\Delta X_{ji}^{test} = X_j^{test} - X_i^{train}$
          Use the trained neural network to predict: $TrainedNN(\Delta X_{ji}^{test}) = \Delta X_{ji}^{pred}$
          Adapt neighbor's $y$ values: $y_{ji}^{pred} = y_i^{train} + \Delta X_{ji}^{pred}$
      end for
      Average the adapted $y$ values: $y_j^{pred} = \frac{1}{n} \sum_{i=1}^{n} y_{ji}^{pred}$
  end for
  Compare $y^{pred}$ to $y^{test}$ to evaluate accuracy
```

3.2.6 Evaluation

Evaluation metrics include Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2) to assess model accuracy and performance.

3.2.7 Optimization

To optimize the model, we consider various strategies:

- Hyperparameter Tuning: Grid search or randomized search cross-validation for optimal hyperparameters.
- Feature Engineering: Experimentation with different feature combinations to improve model performance.
- Regularization Techniques: L1 and L2 regularization to prevent overfitting.

3.2.8 Limitations and Future Work

The Linger Regression Base Model has several limitations:

- Sensitivity to Noise: The model may produce inaccurate predictions when faced with noisy or irrelevant data.
- Interpretability: While the model enhances interpretability compared to black-box models, it may still be challenging to interpret in complex scenarios.

Future work includes:

- Enhancing Noise Robustness: Investigating methods to improve model performance in the presence of noisy data.
- Interpretability Improvements: Exploring techniques to enhance model interpretability for complex datasets.

3.3 Linger Classification Base Model: `LingerClassifier`

3.3.1 Objective

The objective of the Linger Classification Base Model (`LingerClassifier`) is to extend the learning from differences methodology to classification tasks. The primary goals include:

- Developing a versatile classification model that utilizes differences between cases for prediction.
- Ensuring compatibility with Scikit-learn for seamless integration and extensive hyperparameter testing.
- Providing interpretable predictions to facilitate user understanding.
- Implementing enhancements to the learning from differences algorithm for improved classification performance.
- Designing a modular and extensible codebase to support future variations and enhancements.

The `LingerClassifier` is tailored for classification problems where interpretability and adaptability are critical.

3.3.2 Architecture

Similar to the regression counterpart, the `LingerClassifier` integrates K-nearest neighbors (KNN) and Scikit-learn's `MLPClassifier` for classification tasks. During training, KNN is used to compute nearest neighbors, and during prediction, the differences are used to adapt the class labels.

3.3.3 Components

The key components of the `LingerClassifier` include:

- KNN Module: Computes nearest neighbors and extracts differences during training and prediction phases.
- `MLPClassifier`: Offers flexibility in activation functions, solvers, and hyperparameters for classification.
- Additional Features: Similar to the regression model, features like weighted KNN, context addition, additional results' column, and duplication based on distance can be activated through hyperparameters.

3.3.4 Parameterization

The model provides extensive parameterization options for fine-tuning KNN, `MLPClassifier`, and additional features. Hyperparameters for the classification model mirror those available in standard KNN and neural networks, with additional parameters specific to `LingerClassifier`.

3.3.5 Training and Prediction

The training and prediction algorithms for the `LingerClassifier` are similar to those of the `LingerRegressor`, with adjustments made to accommodate classification tasks.

Training Algorithm

During training, the algorithm computes differences between cases and their nearest neighbors, similar to the regression model. However, the adaptation of class labels is based on the majority class among the nearest neighbors.

Algorithm 13 Training Algorithm for `LingerClassifier`

```
Split the dataset $D$ into training set $D^{train}$ and test set $D^{test}$ using an 80%-20% split.
Initialize $\Delta D^{train}$ as an empty list.
for each case $C_i = (X_i^{train}, y_i^{train}) \in D^{train}$ do
    Retrieve $n$ similar cases from $D^{train}$ using nearest neighbors.
    for each similar case $(X_j^{train}, y_j^{train}) \in D^{train}$ do
        Compute the differences: $\Delta D_{ij}^{train} = (X_i^{train} - X_j^{train}, y_i^{train} - y_j^{train})$.
        Append $\Delta D_{ij}^{train}$ to $\Delta D^{train}$.
    end for
end for
Train the MLPClassifier neural network model using $\Delta D^{train}$.
return Trained neural network model $TrainedNN()$.
```

Prediction Algorithm

During prediction, the algorithm adapts class labels based on the majority class among the nearest neighbors' labels.

Algorithm 14 Prediction Algorithm for Learning from Differences for Classification

```
Input: Test dataset $D^{test}$, trained neural network model $TrainedNN()$, number of neighbors $n$
 1: Initialize empty list $y^{pred}$
 2: for each case $X_j^{test} \in D^{test}$ do
 3:     Retrieve $n$ similar cases from $D^{train}$ using nearest neighbors
 4:     Initialize empty list $y_j^{pred}$
 5:     for each similar case $(X_i^{train}, y_i^{train}) \in D^{train}$ do
 6:         Compute the differences: $\Delta X_{ji}^{test} = X_j^{test} - X_i^{train}$
 7:         Use the trained neural network to predict: $TrainedNN(\Delta X_{ji}^{test}) = \Delta X_{ji}^{pred}$
 8:         Adapt neighbor's $y$ value: $yy_{ji}^{pred} = y_i^{train} + \Delta X_{ji}^{pred}$
 9:     end for
10:     Perform majority voting on $y^{pred} predictions$ to obtain $y_j^{pred}$
11: end for
12: Compare $y^{pred}$ to $y^{test}$ to evaluate accuracy
```

3.3.6 Evaluation

Evaluation metrics for the `LingerClassifier` include accuracy, precision, recall, F1-score, and confusion matrix analysis. These metrics provide insights into the model's performance and its ability to classify instances accurately.

3.3.7 Optimization

To enhance the performance of the `LingerClassifier`, several optimization strategies are considered:

- **Hyperparameter Tuning**: Utilize grid search or randomized search cross-validation to fine-tune the hyperparameters for optimal classifier performance.
- **Feature Engineering**: Experiment with different feature combinations and transformations to uncover informative patterns in the data and improve classifier accuracy.
- **Regularization Techniques**: Apply L1 and L2 regularization methods to the classifier to mitigate overfitting and enhance generalization ability.

These optimization strategies aim to maximize the predictive capability of the `LingerClassifier` model.

3.3.8 Limitations and Future Work

The `LingerClassifier` shares similar limitations and avenues for future work as the regression model, including sensitivity to noise and interpretability challenges. Future work may focus on improving noise robustness and enhancing interpretability for complex classification tasks.

3.3.9 Conclusion

The `LingerClassifier` offers a novel approach to classification tasks by integrating the learning from differences methodology. Its modular design and extensible architecture make it well-suited for a wide range of classification problems, particularly those where interpretability and adaptability are paramount.

3.4 Variations

3.4.1 Variation 1: Addition of Context

Objective

The objective of adding context is to incorporate the original attribute values of the source cases into the generated differences array. This ensures that the new datasets used for fitting and prediction contain both the original attribute values of the source cases and the differences between them and their nearest neighbors. This may aid in the learning process and improves the model. Its design is the same for both the `LingerRegressor` and `LingerClassifier`.

Implementation for Training

Algorithm 15 Training Algorithm for Learning from Differences, with context included in `LingerClassifier` and `LingerRegressor`

```
1: Split the dataset $D$ into training set $D^{train}$ and test set $D^{test}$ using an 80%-20% split.
2: Initialize $\Delta D^{train}$ as an empty list.
3: for each case $C_i = (X_i^{train}, y_i^{train}) \in D^{train}$ do
4:     Retrieve $n$ similar cases from $D^{train}$ using nearest neighbors.
5:     for each similar case $(X_j^{train}, y_j^{train}) \in D^{train}$ do
6:         Compute the differences, with concatenated context: $\Delta D_{ij}^{train} = ((X_i^{train} - X_j^{train}) : X_i^{train}, y_i^{train} - y_j^{train})$.
7:         Append $\Delta D_{ij}^{train}$ to $\Delta D^{train}$.
8:     end for
9: end for
10: Train the MLPClassifier or MLPRegressor neural network model using $\Delta D^{train}$.
11: return Trained neural network model $TrainedNN()$.
```

Implementation for Predict

Algorithm 16 Prediction Algorithm for Variant 1, context included.

```
Input: Test dataset $D^{test}$, trained neural network model $TrainedNN()$, number of neighbors $n$
 1: Initialize $y^{pred}$ as an empty list.
 2: for each case $X_j^{test}$ in $D^{test}$ do
 3:     Retrieve $n$ similar cases from $D^{train}$ using nearest neighbors.
 4:     Initialize $y_j^{pred}$ as an empty list.
 5:     for each similar case $(X_i^{train}, y_i^{train})$ in $D^{train}$ do
 6:         Compute the differences: $\Delta X_{ji}^{test} = ((X_j^{test} - X_i^{train}) : X_j^{test})$.
 7:         Use the trained neural network to predict: Trained NN$(\Delta X_{ji}^{test}) = \Delta X_{ji}^{pred}$.
 8:         Adapt neighbor's $y$ values: $y_{ji}^{pred} = y_i^{train} + \Delta X_{ji}^{pred}$.
 9:     end for
10:     if Regression then
11:         Average the adapted $y$ values: $y_j^{pred} = \frac{1}{n} \sum_{i=1}^{n} y_j^{pred}$.
12:     else if Classification then
13:         Perform majority voting on $y_j^{pred}$ to obtain $y_j^{pred}$.
14:     end if
15: end for
16: Compare $y_j^{pred}$ to $y_j^{test}$ to evaluate accuracy.
```

3.4.2 Variation 2: Duplication Based on Distance

Objective

A novel variation introduced in this study involves duplicating differences based on distance, aiming to amplify the influence of neighbors closely resembling the base case. This innovative approach represents a departure from conventional methods in the learning from differences paradigm. By magnifying the impact of differences that exhibit stronger correlation with the base case, this technique effectively mitigates the influence of outliers, leading to more robust model performance.

This novel approach is specifically integrated into the fitting process of both the `LingerClassifier` and `LingerRegressor` models. By emphasizing the significance of closely resembling neighbors, this technique enhances the model's ability to discern meaningful patterns in the data, thereby improving its predictive accuracy and generalization capabilities.

1. **Normalize Distances**: Normalize the distances from the base case to its nearest neighbors represented by $Distances_{ij}$. We apply a logarithmic transformation to the distances: $Distances_{ij} = \log_e(Distances_{ij} + 1)$, with the addition of 1 to prevent a logarithm of 0.

2. **Determine Maximum and Minimum Distances**: Determine the maximum distance $MaxDistance$ and minimum distance $MinDistance$ in $Distances_{ij}$.

3. **Compress Distances**: Compress the distances within $Distances_{ij}$ to a range between 0 and 10 using the formula:

$$Distances_{ij} = \text{round}\left(10 - \frac{(Distances_{ij} - MinDistance)}{(MaxDistance - MinDistance) \times 10}\right)$$

4. **Duplicate Items**: Duplicate each item ΔD_{ij}^{train} by the corresponding distance in $Distances_{ij}$. It's important to note that this duplication process can significantly increase the size of the dataset and, consequently, the training time overhead.

Implementation for Training

Algorithm 17 Duplication Based on Distance in `LingerClassifier` and `LingerRegressor`

Input: Distances $Distances_{ij}$, Base dataset ΔD^{train}

```
1:  Normalize Distances:
2:  for each distance $Distances_{ij}$ do
3:      $Distances_{ij} \leftarrow \log_e(Distances_{ij} + 1)$      ▷ Apply logarithmic transformation
4:  end for
5:  Determine Maximum and Minimum Distances:
6:  $MaxDistance \leftarrow$ maximum value of $Distances_{ij}$
7:  $MinDistance \leftarrow$ minimum value of $Distances_{ij}$
8:  Compress Distances:
9:  for each distance $Distances_{ij}$ do
10:     $Distances_{ij} \leftarrow \text{round}\left(10 - \frac{(Distances_{ij} - MinDistance)}{(MaxDistance - MinDistance) \times 10}\right)$
11: end for
12: Duplicate Items:
13: for each item $\Delta D_{ij}^{train}$ in $\Delta D^{train}$ do
14:     Duplicate $\Delta D_{ij}^{train}$ by the corresponding distance in $Distances_{ij}$
15: end for
```

3.4.3 Variation 3: Addition of Distance Column

Objective

Variation 3 introduces a novel approach by incorporating the distance between each base case and its nearest neighbor into both the training and prediction phases of the neural network. This innovative method aims to leverage the proximity information to enhance the learning process and predictive performance of the model.

By integrating distance information, the neural network gains insight into the spatial relationships between data points. This allows the model to potentially learn associations between differences and their corresponding distances, leading to improved accuracy and effectiveness in both training and prediction tasks.

The motivation behind this variation lies in its potential to provide the model with a deeper understanding of the underlying data structure. By considering distance as a factor in the learning process, the model can better capture nuanced patterns and relationships, resulting in more robust and reliable predictions.

Implementation for Training

In the fit phase, the distances between each base case D_i^{train} and its nearest neighbor in D^{train} are computed and stored in $Distances_{ij}$. To incorporate these distances into the training data, each distance in $Distances_{ij}$ is concatenated with the corresponding difference ΔD^{train}. This results in a new dataset where each difference is paired with its corresponding distance, ensuring a one-to-one ratio between distances and differences.

Algorithm 18 Training Algorithm for Learning from Differences, with Distance Column

Input: Dataset D, Neural network model, Number of neighbors n, Number of epochs E

```
 1: Split $D$ into training set $D^{train}$ and test set $D^{test}$ with an 80%-20% split
 2: Initialize $\Delta D^{train}$ and $Distances$ as empty lists
 3: for each case $C_i = (X_i^{train}, y_i^{train}) \in D^{train}$ do
 4:     Retrieve $n$ similar cases from $D^{train}$ using nearest neighbors
 5:     for each similar case $(X_j^{train}, y_j^{train}) \in D^{train}$ do
 6:         Compute the differences: $\Delta D_{ij}^{train} = (X_i^{train} - X_j^{train}, y_i^{train} - y_j^{train})$
 7:         Compute the distance: $Distances_{ij} = ||X_i^{train}, X_j^{train}||$ (Euclidean distance)
 8:         Concatenate $Distances_{ij}$ as a new attribute to $\Delta D_{ij}^{train}$
 9:     end for
10: end for
11: Train the MLPRegressor neural network model using $\Delta D^{train}$ over $E$ epochs
12: return Trained neural network model $TrainedNN()$.
```

Implementation for Predict

In the prediction phase, distances between each test case X_j^{test} and its nearest neighbors in D^{train} are stored in $Distances_{ji}$. These distances are concatenated with the corresponding differences ΔX^{test}, ensuring a one-to-one ratio between distances and differences in the prediction dataset.

Algorithm 19 Prediction Algorithm for Learning from Differences with Additional Distance Column

Input: Test dataset D^{test}, Trained neural network model, Number of neighbors n, $TrainedNN()$

```
1: Initialize $\Delta D^{train}$ and $Distances$ as empty lists
2: for each case $X_j^{test} \in D^{test}$ do
3:     Retrieve $n$ similar cases from $D^{train}$ using nearest neighbors
4:     for each similar case $(X_i^{train}, y_i^{train}) \in D^{train}$ do
5:         Compute the differences: $\Delta X_{ij}^{test} = (X_j^{test} - X_i^{train})$
6:         Compute the distance: $Distances_{ji} = ||X_j^{test}, X_i^{train}||$ (Euclidean distance)
7:         Add $Distances_{ji}$ as a new attribute to $\Delta X_{ji}^{test}$
8:         Use the trained neural network to predict: $TrainedNN(\Delta X_{ji}^{test}) = \Delta X_{ji}^{pred}$
9:         Adapt neighbor's $y$ values: $y_j^{pred} = y_i^{train} + \Delta X_{ji}^{pred}$
10:    end for
11:    if Regression then
12:        Average the adapted $y$ values: $y_j^{pred} = \frac{1}{n}\sum_{i=1}^{n} y_j^{pred}$
13:    else if Classification then
14:        Perform majority voting on $y_j^{pred}$ to obtain $y_j^{pred}$
15:    end if
16:    Compare $y_j^{pred}$ to $y_j^{test}$ to evaluate accuracy
17: end for
```

3.5 Learning From differences Implicitly

3.5.1 Introduction and Motivation

This design segment aims to implement and extend the algorithms proposed by Ye et al. [14] for both classification and regression tasks. The primary objective is to develop a unified framework where a single set of algorithms can be employed for both classification and regression tasks, leveraging implicit learning from differences within case pairs.

As outlined in the literature review (see Section 2), the algorithms (Algorithm 9, Algorithm 10) proposed by Ye et al. were chosen as the basis for our model implementation. Variant 5, as demonstrated in the paper, exhibited superior performance among the variants explored, prompting its adoption in our implementation.

Moreover, while the paper focuses on classification tasks, it overlooks the potential application of implicit difference learning within case pairs for regression tasks. Thus, our secondary motivation is to explore the adaptability of Variant 5 to regression models. We seek to investigate whether the framework outlined in Algorithm 9 and Algorithm 10 can be extended and optimized for regression tasks effectively.

By extending the application of Variant 5 to regression tasks, we aim to provide a comprehensive and versatile framework for both classification and regression tasks, harnessing the power of implicit learning from differences within case pairs.

3.6 Linger Implicit Classification Base Model: `LingerImplicitClassifier`

3.6.1 Objective

The objective of the Linger Implicit Classification Base Model (`LingerImplicitClassifier`) is to implement the learning from differences methodology seen in [14] for classification tasks. The primary goals include:

- Developing a versatile classification model that learns through implicit differences in case pairs.
- Ensuring compatibility with Scikit-learn for seamless integration and extensive hyperparameter testing.
- Providing interpretable predictions to facilitate user understanding, for a variety of tasks.
- Implementing enhancements to the learning from differences algorithm for improved classification performance through hyperparameter testing.
- Designing a modular and extensible codebase to support future variations and enhancements.

The `LingerImplicitClassifier` is tailored for classification problems where interpretability and adaptability are critical, Where explicit differences may not be easily calculable. It adds another layer of interpretability, as both the target case and retrieved case are visible within the case pair which the Neural Nets are trained on.

3.6.2 Architecture

The `LingerImplicitClassifier` leverages a hybrid architecture incorporating K-nearest neighbors (KNN) and Scikit-learn's `MLPClassifier` to tackle classification tasks. In the training phase, KNN plays a pivotal role in computing nearest neighbors, thereby facilitating the creation of case pairs. During the prediction phase, KNN is again utilized to retrieve the most similar cases from each class.

The process of generating case pairs for predictions involves KNN's functionality to identify the nearest neighbors. These neighbors serve as crucial elements in forming case pairs, enabling the classifier to make informed predictions based on similarities observed within the dataset.

The integration of KNN with the `MLPClassifier` offers a comprehensive approach to classification tasks, combining the strengths of both algorithms. While KNN excels in identifying similarities and forming case pairs, the `MLPClassifier` contributes by providing a powerful neural network-based classification model capable of learning intricate patterns and relationships within the data.

By harnessing the complementary capabilities of KNN and the `MLPClassifier`, the `LingerImplicitClassifier` aims to achieve robust and accurate classification results across diverse datasets, making it a versatile tool for various machine learning applications.

3.6.3 Components

The key components of the `LingerClassifier` include:

- KNN Module: Computes nearest neighbors and extracts differences during training and prediction phases.
- `MLPClassifier`: Offers flexibility in activation functions, solvers, and hyperparameters for classification.
- Additional Features: Class to class pairs, random N selected neighbors, and N nearest neighbors.

3.6.4 Parameterization

The model provides extensive parameterization options for fine-tuning KNN, `MLPClassifier`, and additional features. Hyperparameters for the classification model mirror those available in standard KNN and neural networks, with additional parameters specific to `LingerClassifier`.

3.6.5 Training and Prediction

The training and prediction algorithms for the `LingerImplicitClassifier`.

Training Algorithm

Algorithm 20 Training Algorithm for `LingerImplicitClassifier`

```
Input: Dataset $D$, number of neighbors 1
  Split $D$ into training set $D^{train}$ and test set $D^{test}$ with an 80%-20% split
  Create $CasePairs()$ dictionary.
  Create $AdaptNN()$, a dictionary to store each Neural Network
  List of unique Classes $UniqueClasses$
  for each case $C_i = (X_i^{train}, y_i^{train}) \in D^{train}$ do
      Retrieve 1 similar case from $D^{train}$ using nearest neighbors.
      for each similar case $(X_j^{train}, y_j^{train}) \in D^{train}$ do
          $CasePairs(y_j^{train}).append([X_j^{train} : X_i^{train}, y_i^{train}])$
      end for
  end for
  for each unique $Class$ in $UniqueClasses$ do
      Create a new Classification Neural Network, trained on Case Pairs
      $AdaptNN(Class).fit(CasePairs(Class))$
  end for
  return Trained neural network models dictionary $AdaptNN()$.
```

Prediction Algorithm

During the prediction phase, various methods exist for retrieving neighbors within each class. These methods can be categorized into three variants:

1. **Variant 1:** Selecting the N nearest neighbors from each class.

2. **Variant 2:** Selecting N random items from each class.
3. **Variant 3:** Selecting the single nearest neighbor from each class.

Each variant may be suitable depending on the specific characteristics of the dataset and the objectives of the prediction task.

Algorithm 21 Prediction Algorithm for `LingerImplicitClassifier`

```
Input: Dataset $D^{test}$, Dictionary $AdaptNN()$, Set $UniqueClasses$, and Dictionary $CasePairs()$
  Initialize an empty list $y^{pred}$
  for each case $X_j^{test} \in D^{test}$ do
      Create an empty list $preds$
      for each unique class $Class$ in $UniqueClasses$ do
          if variant 1 then
              Retrieve $N$ nearest neighbors from $CasePairs(Class)$
          else if variant 2 then
              Select $N$ random items from $CasePairs(Class)$
          else if variant 3 then
              Select the single nearest neighbor from $CasePairs(Class)$
          end if
          for each selected case $(X_i^{CasePair}, y_i^{CasePair})$ do
              for each $(X_z^{train}, X_q^{train}) \in X_i^{CasePair}$ do
                  $preds.append(AdaptNN(Class).predict(X_z^{train} : X_j^{test}))$
              end for
          end for
          Perform a majority vote on $preds$
          $y_j^{pred} = MajorityVote(preds)$
      end for
  end for
  return $y^{pred}$
```

3.6.6 Optimization

To enhance the performance of the `LingerImplicitClassifier`, several optimization strategies are considered:

- **Hyperparameter Tuning**: Utilize grid search or randomized search cross-validation to fine-tune the hyperparameters for optimal classifier performance.

- **Feature Engineering**: Experiment with different feature combinations and transformations to uncover informative patterns in the data and improve classifier accuracy.
- **Regularization Techniques**: Apply Regularization techniques to improve accuracy, and reduce noise.

3.6.7 Limitations and Future Work

The `LingerImplicitClassifier` including sensitivity to noise. Future work may focus on improving noise robustness and enhancing interpretability for complex classification tasks.

3.6.8 Conclusion

The `LingerImplicitClassifier` offers a novel approach to classification tasks by integrating the learning from differences implicitly through case pairs. It performs well for a variety of classification tasks. That could benefit from a more interpretable learning methodology

3.7 Linger Implicit Regression Base Model: `LingerImplicitRegressor`

3.7.1 Objective

The objective of the `LingerImplicitRegressor` is to pioneer a learning methodology seen in [14] tailored specifically for regression tasks, known as Learning from Differences. The primary objectives encompass:

- Creation of a flexible regression model capable of learning implicitly from differences observed in pairs of cases.

- Adaptation of regression tasks into the Learning from Differences framework to unlock its potential.
- Seamless integration with Scikit-learn to ensure compatibility and enable extensive hyperparameter optimization.
- Provision of interpretable predictions crucial for user comprehension across a wide spectrum of regression scenarios.
- Continuous enhancement of the Learning from Differences algorithm through rigorous hyperparameter testing to achieve superior regression performance.
- Designing a modular and extensible codebase to accommodate future variations and improvements.

The `LingerImplicitRegressor` is designed to address regression challenges where explicit differences may be elusive, prioritizing interpretability and adaptability. It implements the classification methodology described in [14] but facilitates its use for regression tasks. This is a novel approach. It provides added transparency by exposing both the target case and the retrieved case within the case pair, facilitating a deeper understanding during model training.

3.7.2 Architecture

The `LingerImplicitRegressor` utilizes a hybrid architecture integrating K-nearest neighbors (KNN) with Scikit-learn's `MLPClassifier`. In the training phase, KNN plays a pivotal role in computing nearest neighbors, thereby facilitating the creation of case pairs. During the prediction phase, KNN is again utilized to retrieve the most similar cases from each class.

The process of generating case pairs for predictions involves KNN's functionality to identify the nearest neighbors. These neighbors serve as crucial elements in forming case pairs, enabling the regressor to make informed predictions based on similarities observed within the dataset.

The integration of KNN with the `MLPClassifier` offers an interesting approach to regression tasks, combining the strengths of both algorithms. It places regression values within specified ranges, offering a new type of solution to classic regression issues. While KNN excels in identifying similarities and forming case pairs, the `MLPClassifier` contributes by providing a powerful neural network-based classification model capable of learning intricate patterns and relationships within the data. Classifying regression values.

By harnessing the complementary capabilities of KNN and the `MLPClassifier`, the `LingerImplicitRegressor` aims to achieve robust and accurate regression results across diverse datasets while using a classifier, making it a versatile tool for various machine learning applications. This new approach to regression tasks that allows for categorization of regression results. It processes a regression task and converts it to a classification task, this could be useful for problems where ranges of values are highly important. This would make certain regression tasks more interpretable.

3.7.3 Components

The key components of the `LingerRegressor` include:

- KNN Module: Computes nearest neighbors and extracts differences during training and prediction phases.
- `MLPClassifier`: Offers flexibility in activation functions, solvers, and hyperparameters for classification.
- Additional Features: Class to class pairs, random N selected neighbors, and N nearest neighbors.

3.7.4 Parameterization

The model provides extensive parameterization options for fine-tuning KNN, `MLPClassifier`, and additional features. Hyperparameters for the classifier model mirror those available in standard KNN and neural networks, with additional parameters specific to `LingerRegressor`.

3.7.5 Training and Prediction

The training and prediction algorithms for the `LingerImplicitRegressor`.

Training Algorithm

Algorithm 22 Training Algorithm for `LingerImplicitRegressor`

```
Input: Dataset $D$, number of neighbors 1, $N$ Classes
  Split $D$ into training set $D^{train}$ and test set $D^{test}$ with an 80%-20% split
  Create $CasePairs()$ dictionary.
  Create $AdaptNN()$, a dictionary to store each Neural Network
  Convert training data $y$ values into classes, using the algorithm 24
  $D^{train}y, unique\_ranges, y^{\max}_{\text{train}}, y^{\min}_{\text{train}} = RegToClassConverterFunction(D^{train}, N)$
  Convert test $y$ values based on the train $y$ value range. So that a new range is the
  same for both.
  $D^{test}y, unique\_ranges, y^{\max}_{\text{test}}, y^{\min}_{\text{test}} = RegToClassConverterFunction(D^{train}, N, y^{\max}_{\text{train}}, y^{\min}_{\text{train}})$
  List of unique Classes $UniqueClasses$
  for each case $C_i = (X_i^{train}, y_i^{train}) \in D^{train}$ do
    Retrieve 1 similar case from $D^{train}$ using nearest neighbors.
    for each similar case $(X_j^{train}, y_j^{train}) \in D^{train}$ do
      $CasePairs(y_j^{train}).append([X_j^{train} : X_i^{train}, y_i^{train}])$
    end for
  end for
  for each unique $Class$ in $UniqueClasses$ do
    Create a new Classification Neural Network, trained on Case Pairs
    $AdaptNN(Class).fit(CasePairs(Class))$
  end for
  return Trained neural network models dictionary $AdaptNN()$.
```

Prediction Algorithm

During the prediction phase for regression, various methods exist for retrieving neighbors within each class. These methods can be categorized into three variants:

1. **Variant 1:** Selecting the N nearest neighbors from each class.
2. **Variant 2:** Selecting N random items from each class.
3. **Variant 3:** Selecting the single nearest neighbor from each class.

Algorithm 23 Prediction Algorithm for `LingerImplicitRegressor`

```
Input: Dataset $D^{test}$, Dictionary $AdaptNN()$, Set $UniqueClasses$, and dictionary $CasePairs()$
  Initialize an empty list $y^{pred}$
  for each case $X_j^{test} \in D^{test}$ do
      Create an empty list $preds$
      for each unique class $Class$ in $UniqueClasses$ do
          if variant 1 then
              Retrieve $N$ nearest neighbors from $CasePairs(Class)$
          else if variant 2 then
              Select $N$ random items from $CasePairs(Class)$
          else if variant 3 then
              Select the single nearest neighbor from $CasePairs(Class)$
          end if
          for each selected case $(X_i^{CasePair}, y_i^{CasePair})$ do
              for each $(X_z^{train}, X_q^{train}) \in X_i^{CasePair}$ do
                  $preds.append(AdaptNN(Class).predict(X_z^{train} : X_j^{test}))$
              end for
          end for
          Perform a majority vote on $preds$
          $y_j^{pred} = MajorityVote(preds)$
      end for
  end for
  return $y^{pred}$
```

Regression to Classification converter function

This function serves to categorize regression y target values for training and validation. It returns a human-readable range with their corresponding numeric prediction. This ensures that the results are interpretable to humans at the end of the process. The variable equal division is used to signify whether samples should be equally distributed into N classes. If there are Z samples in dataset D then each class should have $\approx \frac{Z}{N}$ per class.

Algorithm 24 Converter algorithm, regression to classification

Input: Dataset D, number of segments N, optional: $y_{\max}$, $y_{\min}$, Equal division

```
1:  $(X, y) \in D$
2:  Get the minimum and maximum values of the labels
3:  Ensure that the max and min are used from the test set so that $y$ values are classified
    the same in both fit and predict.
4:  **if** Equal division **then**
5:      Divide the data into $N$ segments based on nearly equal counts
6:      $quantiles \leftarrow$ np.linspace$(0, 1, N + 1)$
7:      **if** unique_ranges is None **then**
8:          $unique\_ranges \leftarrow$ np.quantile$(y, quantiles)$
9:      **end if**
10:     $categories \leftarrow$ np.digitize$(y, unique\_ranges[: -1])$
11:     $category\_counts \leftarrow$ Counter$(categories)$
12:     **Return** $categories$, $unique\_ranges$, $y_{\min}$, $y_{\max}$
13: **else**
14:     **if** $y_{\max}$ and $y_{\min}$ are provided **then**
15:         Leave $y_{\max}$ and $y_{\min}$ as is
16:     **else**
17:         $y_{\max} \leftarrow$ MAX$(y)$
18:         $y_{\min} \leftarrow$ MIN$(y)$
19:     **end if**
20:     Calculate the range for each segment for $y$ values
21:     segmentRange $\leftarrow \frac{y_{\max} - y_{\min}}{N}$
22:     Initialize an empty list: $categories$
23:     Initialize an empty dictionary: $unique\_ranges$
24:     $current\_index \leftarrow 0$
25:     **for** each value $val$ in $y$ **do**
26:         **for** $i$ in range$(N)$ **do**
27:             **if** $i < N - 1$ **then**
28:                 **if** $y_{\min} + i \times \text{segmentRange} \leq \text{val} < y_{\min} + (i + 1) \times \text{segmentRange}$ **then**
29:                     $category \leftarrow (y_{\min} + i \times \text{segmentRange}) - (y_{\min} + (i + 1) \times$
    $\text{segmentRange})$
30:                     **if** category not in $unique\_ranges$ **then**
31:                         $unique\_ranges[category] \leftarrow current\_index$
32:                         $current\_index \leftarrow current\_index + 1$
33:                     **end if**
34:                     Append $unique\_ranges[category]$ to $categories$
35:                     **break**
36:                 **end if**
37:             **else**
38:                 **if** $y_{\min} + i \times \text{segmentRange} \leq \text{val} \leq y_{\max}$ **then**
39:                     $category \leftarrow (y_{\min} + i \times \text{segmentRange}) - (y_{\max})$
```

3.7.6 Optimization

To enhance the performance of the `LingerImplicitRegressor`, several optimization strategies are considered:

- **Hyperparameter Tuning**: Utilize grid search or randomized search cross-validation to fine-tune the hyperparameters for optimal model performance.
- **Feature Engineering**: Experiment with different feature combinations and transformations to uncover informative patterns in the data and improve regressor accuracy.
- **Regularization Techniques**: Apply regularization techniques to improve accuracy and reduce noise.

3.7.7 Limitations and Future Work

One of the limitations of the `LingerImplicitClassifier` is its sensitivity to noise, particularly when labeling values based on the maximum and minimum. However, opting to evenly distribute samples into classes can mitigate this sensitivity. For future research, exploring alternative methods to bucket regression values based on diverse distributions could be an intriguing avenue.

3.7.8 Conclusion

The `LingerImplicitClassifier` offers a novel approach to regression tasks by integrating the learning from differences implicitly through case pairs. It performs well for a variety of regression tasks, while utilizing a learning from differences classification model. The number of segments greatly affects the outcome of this model.

Chapter 4

Implementation

Chapter 5

Evaluation

Chapter 6

Conclusions

Chapter 7

Experiments

7.1 Experiment 1: Abalone dataset Linger Regression

7.2 Experiment 2: House Prices Dataset Regression

7.3 Experiment 3: Glass Dataset Classification

7.4 Experiment 4: Raisin Dataset Classification

7.5 Experiment 5: Car Safety Dataset Classification

Chapter 8

Results

Bibliography

[1] David W Aha, Dennis Kibler, and Marc K Albert. Instance-based learning algorithms. *Machine learning*, 6:37–66, 1991.

[2] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[3] Susan Craw, Nirmalie Wiratunga, and Ray C Rowe. Learning adaptation knowledge to improve case-based reasoning. *Artificial intelligence*, 170(16-17):1175–1192, 2006.

[4] Mathieu D'Aquin, Emmanuel Nauer, and Justine Lieber. A factorial study of neural network learning from differences for regression. In Mark T. Keane and Nirmalie Wiratunga, editors, *Case-Based Reasoning Research and Development. ICCBR 2022. Lecture Notes in Computer Science*, volume 13405, page PageNumbersHere. Springer, Cham, 2022.

[5] Preeti Gupta, Arun Sharma, and Rajni Jindal. Scalable machine-learning algorithms for big data analytics: a comprehensive review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 6(6):194–214, 2016.

[6] Kathleen Hanney and Mark T Keane. Learning adaptation rules from a case-base. In *European Workshop on Advances in Case-Based Reasoning*, pages 179–192. Springer, 1996.

[7] Vahid Jalali, David Leake, and Najmeh Forouzandehmehr. Learning and applying case adaptation rules for classification: An ensemble approach. In *IJCAI*, pages 4874–4878, 2017.

[8] David B Leake and Andrew Kinley. Acquiring case adaptation knowledge: A hybrid approach. 1996.

[9] Leorey Marquez, Tim Hill, Reginald Worthley, and William Remus. Neural network models as an alternative to regression. In *Proceedings of the*

Twenty-Fourth Annual Hawaii International Conference on System Sciences, volume 4, pages 129–135. IEEE, 1991.

[10] Christopher K Riesbeck and Roger C Schank. *Inside case-based reasoning.* Psychology Press, 2013.

[11] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206–215, 2019.

[12] Sergios Theodoridis. *Machine learning: a Bayesian and optimization perspective.* Academic press, 2015.

[13] Ian Watson and Farhi Marir. Case-based reasoning: A review. *The knowledge engineering review*, 9(4):327–354, 1994.

[14] Xiaomeng Ye, David Leake, Vahid Jalali, and David J Crandall. Learning adaptations for case-based classification: A neural network approach. In *Case-Based Reasoning Research and Development: 29th International Conference, ICCBR 2021, Salamanca, Spain, September 13–16, 2021, Proceedings 29*, pages 279–293. Springer, 2021.