Machine Learning Assignment-2

Pei-Chun Chen

Nation Cheng Kung University

Abstract—**This paper explores the nuanced application of kernel methods in ensemble learning, specifically in integrating them into the ensemble of deep learning-based non-tree weak learners. Kernel methods serve as a versatile framework to enhance the predictive performance of the ensemble by combining diverse learners. The paper examines the theoretical foundations, adaptability, and unique challenges of incorporating deep learning-based non-tree weak learners. Experimental results demonstrate the effectiveness of this approach across various datasets, showcasing the potential for improved ensemble learning through kernel methods.**

I. INTRODUCTION

Ensemble learning, a powerful technique that combines multiple learners, is the focal point of this paper's exploration into the application of kernel methods. Kernel methods are introduced as a flexible framework, capable of capturing complex relationships within data and providing an effective means of amalgamating outputs from individual learners.

Particular attention is directed towards ensembles composed of deep learning-based non-tree weak learners. While deep learning excels at capturing intricate patterns, integrating such models into an ensemble introduces unique challenges. This paper aims to bridge the theoretical foundations of kernel methods with the practical implications of employing deep learning-based non-tree weak learners in ensemble settings.

Through a comprehensive examination of theoretical foundations, experimental outcomes, and real-world applications, this research contributes to a deeper understanding of the role of kernel methods in ensemble learning. Simultaneously, it sheds light on the potential advantages achievable by integrating deep learning-based non-tree weak learners into the ensemble structure.

II. ALGORIYHM

A. Kernel Method

- Concept and Detailed Description:

  Kernel methods are a class of algorithms widely employed in machine learning to address non-linear patterns within data. At the core of kernel methods lies a crucial concept known as the "kernel function," which facilitates the implicit mapping of input data to a higher-dimensional space without the explicit calculation of new feature vectors.

  This mapping process to a higher-dimensional space is achieved through the utilization of kernel functions, such as radial basis function (RBF) or polynomial kernels. These functions effectively quantify the similarity between pairs of data points in the original space, capturing intricate relationships that may be challenging to express in lower dimensions.

  In mathematical terms, the kernel trick involves employing kernel functions to compute the inner products of mapped data points in the higher-dimensional space without explicitly executing the mapping process. Essentially, this approach replaces the explicit computation of transformed feature vectors with a computationally efficient calculation based on the kernel function.

  The essence of kernel methods lies in their ingenious use of the "kernel trick," allowing for the direct computation of inner products in a higher-dimensional space without the need for explicit data projection. This means we can calculate the similarity of data points in the original space through kernel functions, bypassing the necessity of knowing their specific locations in the higher-dimensional space.

  In summary, the beauty of kernel methods is their ability to operate implicitly in a high-dimensional space without explicitly projecting data into that space. This approach enables computational efficiency, avoiding the complexities and computational costs associated with explicitly transforming data into high-dimensional feature vectors. This allows kernel methods to maintain computational efficiency when dealing with intricate patterns in data.

- Kernel Function Varieties:

  ➢ Linear Kernel:

  Description: The linear kernel is one of the simplest kernel functions, computing the inner product of input features in the original space.

  Application: Suitable for linearly separable data and scenarios where a linear decision boundary is effective.

  ➢ Polynomial Kernel:

  Description: The polynomial kernel introduces non-linearity through polynomial transformations of the input features.

  Application: Useful when underlying patterns exhibit polynomial relationships, allowing the algorithm to capture complex structures.

  ➢ Gaussian Radial Basis Function (RBF) Kernel:

  Description: The RBF kernel maps data into a high-dimensional space using a Gaussian distribution centered at each data point.

  Application: Effective in capturing complex, non-linear relationships and suitable for a wide range of applications.

  ➢ Sigmoid Kernel:

  Description: The sigmoid kernel employs hyperbolic tangent functions to create non-linear transformations of the input data.

Application: Suitable for scenarios where the relationships between features are sigmoidal in nature.

- Laplacian Kernel:

Description: The Laplacian kernel, also known as the double-exponential kernel, is based on the Laplace distribution.

Application: Useful for emphasizing data points within a certain radius, providing robustness to outliers.

- Integration with Other Algorithms:

Kernel methods can be seamlessly integrated with other machine learning algorithms to enhance model performance and versatility. Some common integration strategies include:

- Support Vector Machines (SVM):

Integration: Kernel methods, especially in SVM, play a crucial role in handling non-linear classification tasks by mapping data into higher-dimensional spaces. The choice of kernel influences the decision boundary shape.

- Principal Component Analysis (PCA):

Integration: Kernel PCA extends traditional PCA by incorporating kernel tricks, allowing for non-linear dimensionality reduction. This integration aids in capturing complex relationships within the data.

- K-Means Clustering:

Integration: Kernelized versions of K-Means clustering leverage the advantages of kernel functions to handle non-linearly separable clusters. This integration enhances the clustering performance in complex data scenarios.

- Decision Trees and Random Forests:

Integration: Kernel methods can be integrated with decision trees and random forests by serving as feature transformers, enabling the algorithms to capture non-linear patterns more effectively.

- Neural Networks:

Integration: Kernel methods can be applied to certain layers of neural networks to introduce non-linearity and improve the network's ability to learn complex mappings.

- Summary:

Kernel methods, leveraging diverse kernel functions, offer a versatile approach to capture non-linear patterns in data without explicit feature computation. Their adaptability and seamless integration across algorithms make them indispensable in enhancing machine learning models.

B. *Deep Learning-Based Non-Tree Weak Learners*

- Introduction:

In recent years, deep learning has achieved significant success across various fields. However, the focus on non-tree weak learners based on deep learning has been growing, diverging from traditional tree-based structures. This emerging class of learners presents unique concepts, intriguing features, and challenges that hold substantial implications for addressing specific problems and enhancing model robustness.

- Concept:

Traditional machine learning methods often employ tree-based structures for weak learners, such as decision trees or random forests. In contrast, non-tree weak learners based on deep learning introduce the concepts of neural networks into weak learner construction. The core idea involves utilizing the hidden layers of deep neural networks as weak learners for learning and feature extraction, deviating from traditional tree-based approaches.

- Noteworthy Aspects:

Advantages of Feature Learning: Deep learning models are renowned for their powerful feature learning capabilities. Introducing this ability into weak learner construction enables models to better capture complex features in the data, thereby enhancing model performance.

Model Interpretability: Compared to some complex deep learning models, non-tree weak learners may offer better interpretability. This is due to their relatively simpler structures, facilitating easier interpretation of the model's prediction results, which is crucial for certain application scenarios.

Flexibility: Non-tree weak learners based on deep learning exhibit greater structural flexibility, allowing adjustments based on specific tasks and data characteristics, thereby enhancing the model's adaptability.

- Challenges:

Risk of Overfitting: Overfitting is a common concern in deep learning models and remains a challenge for non-tree weak learners. Techniques such as regularization can be employed to mitigate the risk of overfitting.

Model Interpretability: While non-tree weak learners exhibit better interpretability compared to some complex deep learning models, there is still room for improvement in terms of interpretability. One approach is to introduce highly interpretable deep learning structures or combine them with other models with better interpretability for modeling.

- Conclusion:

Non-tree weak learners based on deep learning represent an emerging direction in the field of machine learning with potential rich applications. Their unique advantages in solving specific problems, enhancing model performance, and improving interpretability highlight their significance. However, addressing challenges related to data requirements, computational resources, and interpretability enhancement is essential for the continued development and broader application of this approach.

C. *Ensemble Learning: Revolutionizing Model Robustness and Generalization*

- Concept:

  The core idea behind ensemble learning is to aggregate the predictions of multiple weak learners, harnessing their predictive capabilities to outperform individual learners. These weak learners may come from different algorithms, feature sets, or the same algorithm trained with different parameters. In the realm of non-tree weak learners based on deep learning, this combination can encompass predictions from various deep network structures, achieving a greater degree of diversity.

- Challenges and Solutions:

  Computational Resource Requirements: Combining multiple models may require additional computational resources. Solutions include optimizing algorithms, parallel computing, and utilizing distributed computing resources.

  Reduced Model Interpretability: The complexity of ensemble models may decrease overall interpretability. Solutions involve using interpretable weak learners or employing model interpretation techniques like LIME (Local Interpretable Model-agnostic Explanations) to enhance interpretability.

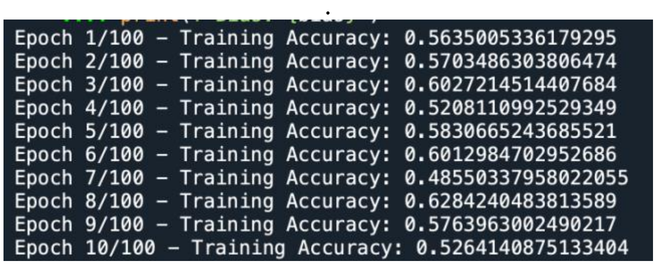
- Summary:

  Ensemble learning, a powerful tool for enhancing model performance and generalization, exhibits significant potential in the realm of non-tree weak learners based on deep learning. While its strengths include model diversity and performance improvement, careful consideration of factors like computational resources and model interpretability is essential for effective implementation.

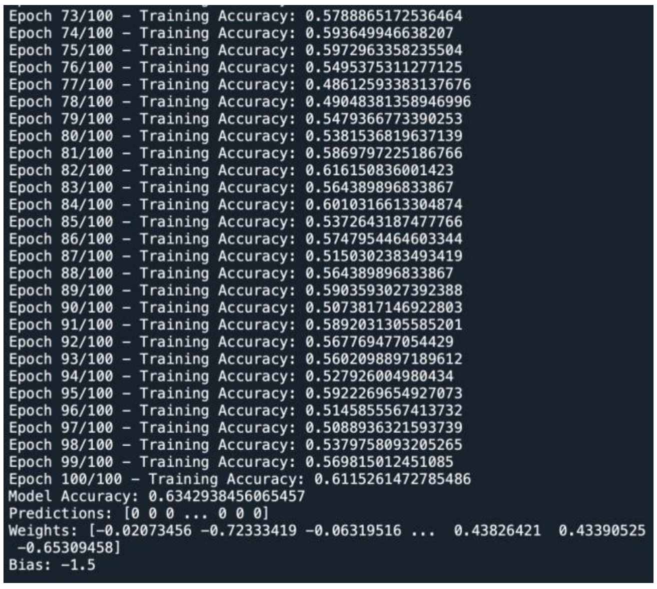
III. EXPERMENTS

Building upon the prior steps and omitting preprocessing details, the focus now shifts to training the perceptron and k-nearest neighbors (KNN) models using a kernel method. The introduction of the rbf_kernel function facilitates the computation of pairwise distances. In this context, the pairwise_dists variable is derived by summing the squared values of each row in X1, adding them to the sum of squared values of each row in X2, and subtracting twice the dot product of X1 and the transpose of X2. The resultant distances are then exponentiated with a negative gamma factor, yielding the radial basis function (RBF) kernel. This rbf_kernel is subsequently employed during the model training phase to compute inner products.

A. *Kernelized percentron*



```
Epoch 1/100 - Training Accuracy: 0.5635005336179295
Epoch 2/100 - Training Accuracy: 0.5703486303806474
Epoch 3/100 - Training Accuracy: 0.6027214514407684
Epoch 4/100 - Training Accuracy: 0.5208110992529349
Epoch 5/100 - Training Accuracy: 0.5830665243685521
Epoch 6/100 - Training Accuracy: 0.6012984702952686
Epoch 7/100 - Training Accuracy: 0.48550337958022055
Epoch 8/100 - Training Accuracy: 0.6284240483813589
Epoch 9/100 - Training Accuracy: 0.5763963002490217
Epoch 10/100 - Training Accuracy: 0.5264140875133404
```



```
Epoch 73/100 - Training Accuracy: 0.5788865172536464
Epoch 74/100 - Training Accuracy: 0.593649946638207
Epoch 75/100 - Training Accuracy: 0.5972963358235504
Epoch 76/100 - Training Accuracy: 0.5495375311277125
Epoch 77/100 - Training Accuracy: 0.48612593383137676
Epoch 78/100 - Training Accuracy: 0.49048381358946996
Epoch 79/100 - Training Accuracy: 0.5479366773390253
Epoch 80/100 - Training Accuracy: 0.5381536819637139
Epoch 81/100 - Training Accuracy: 0.5869797225186766
Epoch 82/100 - Training Accuracy: 0.616150836001423
Epoch 83/100 - Training Accuracy: 0.564389896833867
Epoch 84/100 - Training Accuracy: 0.6010316613304874
Epoch 85/100 - Training Accuracy: 0.5372643187477766
Epoch 86/100 - Training Accuracy: 0.5747954464603344
Epoch 87/100 - Training Accuracy: 0.5150302383493419
Epoch 88/100 - Training Accuracy: 0.564389896833867
Epoch 89/100 - Training Accuracy: 0.5903593027392388
Epoch 90/100 - Training Accuracy: 0.5073817146922803
Epoch 91/100 - Training Accuracy: 0.5892031305585201
Epoch 92/100 - Training Accuracy: 0.567769477054429
Epoch 93/100 - Training Accuracy: 0.5602098897189612
Epoch 94/100 - Training Accuracy: 0.527926004980434
Epoch 95/100 - Training Accuracy: 0.5922269654927073
Epoch 96/100 - Training Accuracy: 0.5145855567413732
Epoch 97/100 - Training Accuracy: 0.5088936321593739
Epoch 98/100 - Training Accuracy: 0.5379758093205265
Epoch 99/100 - Training Accuracy: 0.569815012451085
Epoch 100/100 - Training Accuracy: 0.6115261472785486
Model Accuracy: 0.6342938456065457
Predictions: [0 0 0 ... 0 0 0]
Weights: [-0.02073456 -0.72333419 -0.06319516 ...  0.43826421  0.43390525
 -0.65309458]
Bias: -1.5
```

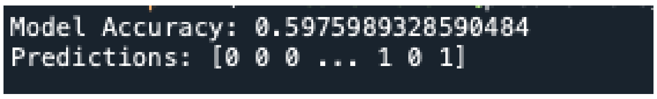
After applying the kernelized perceptron, the accuracy of this model is 0.63. Comparing each epoch with the model without kernelization, we observe that the performance scores tend to converge towards a specific value, indicating the potential discovery of a separable hyperplane in a higher-dimensional space.

B. *Kernelized K-NN*

The Initially, it ensures the data type of the input X_test is converted to floating-point. Subsequently, it calculates the similarity between X_test and self.X_train using the Radial Basis Function (RBF) kernel, a technique for mapping data to a higher-dimensional space, particularly useful for handling nonlinear problems.

During the prediction process, a loop iterates through each sample in X_test, computing the squared Euclidean distance from all samples in self.X_train. The algorithm then selects the indices of the smallest distances, representing the nearest neighbors, and retrieves their class labels. Finally, it determines the most common class label among the neighbors, considering it as the predicted class label.

In summary, this approach utilizes a kernel method to map data to a higher-dimensional feature space for better capturing nonlinear relationships. The KNN algorithm operates in this mapped space to find the nearest neighbors, ultimately predicting the class labels for X_test.



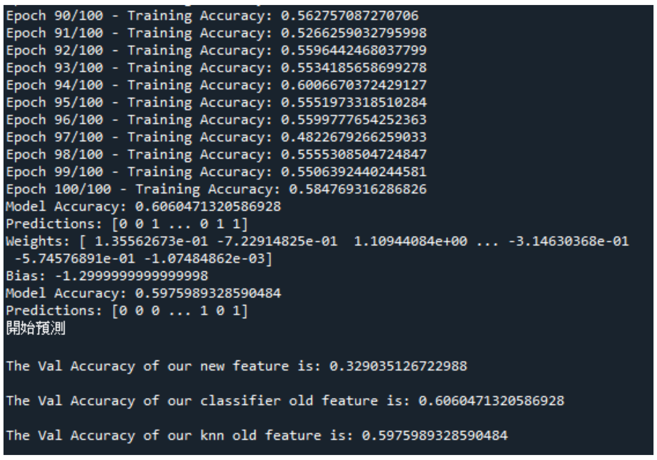
```
Model Accuracy: 0.5975989328590484
Predictions: [0 0 0 ... 1 0 1]
```

I successfully ran the KNN, which originally used Euclidean distance, with a kernel method and obtained an accuracy.

C. *Stacking*

I re-ran the previously used k-NN with Manhattan distance, and then created a new feature by using predictions from Kernelized Perceptron and K-NN. I utilized this new feature with the KNN_manhattan_Classifier as the base for training. Unexpectedly, the performance did not improve;

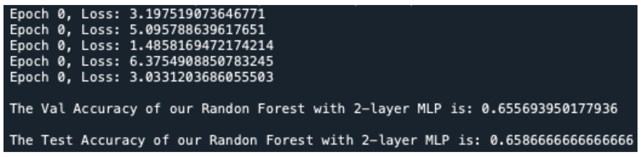
instead, it decreased. I speculate that since both models are based on K-NN, the stacking did not yield significant additional learning, resulting in limited effectiveness.



```
Epoch 90/100 - Training Accuracy: 0.562757087270706
Epoch 91/100 - Training Accuracy: 0.5266259032795998
Epoch 92/100 - Training Accuracy: 0.5596442468037799
Epoch 93/100 - Training Accuracy: 0.5534185658699278
Epoch 94/100 - Training Accuracy: 0.6006670372429127
Epoch 95/100 - Training Accuracy: 0.5551973318510284
Epoch 96/100 - Training Accuracy: 0.5599777654252363
Epoch 97/100 - Training Accuracy: 0.4822679266259033
Epoch 98/100 - Training Accuracy: 0.5555308504724847
Epoch 99/100 - Training Accuracy: 0.5506392440244581
Epoch 100/100 - Training Accuracy: 0.584769316286826
Model Accuracy: 0.6060471320586928
Predictions: [0 0 1 ... 0 1 1]
Weights: [ 1.35562673e-01 -7.22914825e-01  1.10944084e+00 ... -3.14630368e-01
 -5.74576891e-01 -1.07484862e-03]
Bias: -1.2999999999999998
Model Accuracy: 0.5975989328590484
Predictions: [0 0 0 ... 1 0 1]
開始預測

The Val Accuracy of our new feature is: 0.329035126722988

The Val Accuracy of our classifier old feature is: 0.6060471320586928

The Val Accuracy of our knn old feature is: 0.5975989328590484
```

D. *Ensemble of Deep Learning-Based Non-Tree Weak Learners: Simple 2-layer Multi-Layer Perceptron*

I used Random Forest as a base and replaced its tree components with 2-layer Multilayer Perceptrons (MLPs). The process was fascinating, revealing the ability to dismantle and modify complex models like Random Forest. It was a valuable experience. The final results show the validaion and testing accuracies as depicted in the figures below.



```
Epoch 0, Loss: 3.197519073646771
Epoch 0, Loss: 5.095788639617651
Epoch 0, Loss: 1.4858169472174214
Epoch 0, Loss: 6.3754908850783245
Epoch 0, Loss: 3.0331203686055503

The Val Accuracy of our Randon Forest with 2-layer MLP is: 0.655693950177936

The Test Accuracy of our Randon Forest with 2-layer MLP is: 0.6586666666666666
```

Additionally, I printed out the loss for each epoch to facilitate observation.

REFERENCES

[1] https://www.kaggle.com/code/zubinrelia/car-insurance-claim-classifier