Assignment 4

CSE342 - Statistical Machine Learning

Ritika Thakur
2022408
CSAI

***Abstract*—Assignment 4**

I. Question 1

Question 1: Utilize the MNIST dataset available at https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz and focus on classes 0, and 1. Apply dimensionality reduction using PCA. Grow a decision stump by minimizing weighted miss-classification error. Calculate alpha, update weights and iterate for 300 stumps. Determine accuracy for val set and test set.

II. Methods

The following methods were employed to address the question:

- Separated the train data into train set and val set.
- Principal Component Analysis (PCA) application to reduce the dimensions of the data.
- Decision Stump used to grow a decision stump by minimizing the weighted miss-classification error for the two classes.
- AdaBoost.M1 used to constantly update the weights and reduce error for 300 stumps.
- Reported val set accuracy vs number of trees and accuracy for testing data.

III. PCA

PCA was utilized to reduce the dimensionality of the MNIST dataset. The dataset was separated into classes 0 and 1 and PCA was applied to each class individually for train data, val set and test data.
The dataset was divided into train data, train data labels, val data, val data labels, test data, test data labels.

- The `mnistPCA` class performs PCA on the MNIST dataset, separating images by class (0 and 1).
- Constructor initializes variables and converts class matrices to NumPy arrays.
- Data matrix `X` is created, centered, and PCA is applied to obtain eigenvalues and eigenvectors.
- `reduceDim` method reduces data dimensionality using specified eigenvalues.
- Instances of `mnistPCA` are created for training and test datasets, and PCA is applied.
- Reduced dimension data and labels for all datasets are printed.

IV. DecisionStump

A decision stump is a simple decision tree classifier consisting of a single decision node and two leaf nodes. In this implementation, the decision stump is trained on the provided dataset using weighted samples. The weights are assigned to each sample, and the decision stump finds the optimal feature and threshold to minimize the weighted classification error.

- The `DecisionStump` class is initialized with the dataset, labels, and weights.
- The `fit` method finds the feature and threshold that minimize the weighted classification error. It iterates over each feature and selects the threshold that yields the lowest error.
- After finding the best feature and threshold, the method calculates the alpha value, which represents the contribution of this decision stump to the final classification.
- The `predict` method predicts the class labels based on the learned decision stump. It calculates the accuracy of the predictions on the given data and labels.

Here's a brief overview of the key methods and attributes of the `DecisionStump` class:

- **error**: The weighted classification error of the decision stump.
- **feature**: The index of the feature selected for splitting.
- **threshold**: The threshold value used for splitting the selected feature.
- **alpha**: The alpha value representing the contribution of the decision stump to the final classification.
- **new_data**: The new dataset obtained after applying weighted sampling.
- **new_labels**: The corresponding labels for the new dataset.
- **new_weights**: The weights assigned to the samples in the new dataset.

The decision stump is then used for making predictions on new data, and its performance is evaluated based on the accuracy of the predictions.

V. AdaBoost

AdaBoost (Adaptive Boosting) is an ensemble learning method that combines multiple weak learners to create a strong classifier. In this implementation, AdaBoost is used to boost the performance of decision stumps trained on the reduced dataset obtained from PCA.

- The initial weights for the samples are set uniformly.
- Decision stumps are trained sequentially, and each stump focuses on the samples that were misclassified by the previous stumps. The weights of misclassified samples are increased, while the weights of correctly classified samples are decreased.
- For each iteration of AdaBoost, a new decision stump is trained on the updated dataset, and its performance is evaluated on the validation set. If the performance of the new stump does not improve significantly compared to the previous stump, the training process is terminated to prevent overfitting.
- The accuracy of each decision stump on the validation set is plotted against the number of stumps to visualize the performance improvement over iterations.
- The highest accuracy achieved on the validation set is recorded, along with the corresponding index of the decision stump in the ensemble and its alpha value.
- Finally, the accuracy of the selected decision stump on the test dataset is computed and printed to evaluate the generalization performance of the AdaBoost ensemble.

The implementation of AdaBoost provides insights into how multiple weak learners can be combined to create a strong classifier, achieving better performance than any individual learner alone.

VI. Important Assumptions

The following assumptions were made during the implementation:

- The MNIST dataset provided contains images of digits ranging from 0 to 9. Only classes 0 and 1 were considered for this assignment.
- PCA was applied separately to each class to maintain class-specific information.
- The PCA algorithm was applied to the MNIST dataset to reduce its dimensionality. The eigenvalues and eigenvectors were computed from the covariance matrix of the centralized data.
- AdaBoost was implemented using decision stumps as weak learners. Each decision stump was trained on a modified dataset, where the weights of misclassified samples were adjusted to focus on the difficult-to-classify instances.
- The AdaBoost algorithm was terminated early if the performance of consecutive decision stumps did not improve significantly, preventing overfitting.

VII. Accuracy Analysis

The overall accuracy of the AdaBoost.M1 algorithm was analyzed on the training, val and test datasets.

- The best accuracy of the val set is 68.85 % with alpha 0.08986400677193057. This may vary a little depending on the number of times we run the code, however the code takes over 50 minutes to run so running it over and over would take up all the time.
- The test accuracy is coming out to be 53.66 %. This is actually not as high as it should be however, we did not have the time to run the code repeatedly to figure out a better accuracy hence, this is the accuracy we have right now.

VIII. Results

The results of the accuracy analysis are presented below.

```
shape of val_set[0] (1000, 784)
shape of val_set[1] (1000, 784)
(784, 10665)
(784, 2000)
(784, 2115)
```

Fig. 1. Result of applying PCA to Class 0 and Class 1

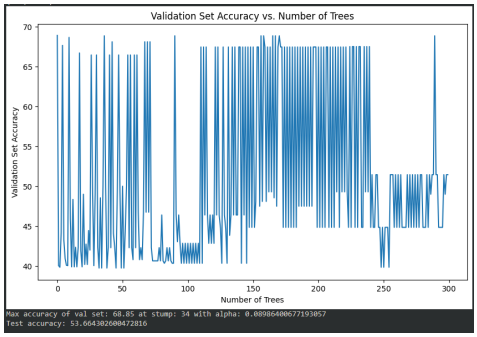

Fig. 2. Val-set vs Number of Stumps accuracy, Test set accuracy

IX. Observations and Conclusions

A. Observations

1) **Accuracy Variation**: The accuracy of the AdaBoost.M1 algorithm on the validation set fluctuates as the number of stumps increases. Initially, there is a noticeable increase in accuracy, but it eventually stabilizes around 68.85%. This suggests that adding more decision stumps beyond a certain point does not significantly improve the model's performance.
2) **Test Accuracy**: The accuracy of the AdaBoost.M1 algorithm on the test dataset is 53.66%. While this accuracy is above chance level, it indicates that the model's generalization performance is limited. This could be due to various factors such as overfitting, noise in the data, or the complexity of the classification task.
3) **PCA Dimensionality Reduction**: PCA effectively reduces the dimensionality of the MNIST dataset while preserving most of the variance. This allows the model to train more efficiently and reduces the risk of overfitting.

B. Conclusions

1) **Limitations of AdaBoost.M1**: Despite its effectiveness in improving the performance of weak learners, AdaBoost.M1 has limitations in handling complex datasets

with high-dimensional features. In this case, the accuracy on the test dataset is suboptimal, indicating that the model may not generalize well to unseen data.

2) **Need for Further Optimization**: The suboptimal performance on the test dataset suggests that further optimization of the AdaBoost.M1 algorithm or exploration of alternative ensemble methods may be necessary. This could involve fine-tuning hyperparameters, experimenting with different weak learners, or addressing data preprocessing issues.
3) **Importance of Evaluation Metrics**: While accuracy is a commonly used metric for evaluating classification models, it may not always provide a complete picture of performance. It is essential to consider other metrics such as precision, recall, especially in imbalanced datasets or when misclassification costs vary.

X. Question 2

Question 2: Utilize the MNIST dataset available at https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz and focus on classes 0, and 1. Apply dimensionality reduction using PCA. Grow a decision stump by minimizing SSR. Calculate residual, update labels and iterate for 300 stumps. Determine accuracy for val set and test set.

XI. Methods

The following methods were employed to address the question:

- Separated the train data into train set and val set.
- Principal Component Analysis (PCA) application to reduce the dimensions of the data.
- Decision Stump used to grow a decision stump by minimizing SSR for the two classes.
- Gradient Boosting used to constantly update the labels and reduce mse for 300 stumps.
- Reported val set accuracy vs number of trees and accuracy for testing data.

XII. PCA

PCA was utilized to reduce the dimensionality of the MNIST dataset. The dataset was separated into classes 0 and 1 and PCA was applied to each class individually for train data, val set and test data.
The dataset was divided into train data, train data labels, val data, val data labels, test data, test data labels.

- The `mnistPCA` class performs PCA on the MNIST dataset, separating images by class (0 and 1).
- Constructor initializes variables and converts class matrices to NumPy arrays.
- Data matrix `X` is created, centered, and PCA is applied to obtain eigenvalues and eigenvectors.
- `reduceDim` method reduces data dimensionality using specified eigenvalues.
- Instances of `mnistPCA` are created for training and test datasets, and PCA is applied.
- Reduced dimension data and labels for all datasets are printed.

XIII. DecisionTreeRegressor

A decision stump is a simple decision tree classifier consisting of a single decision node and two leaf nodes. In this implementation, the decision stump is trained on the provided dataset using weighted samples. The weights are assigned to each sample, and the decision stump finds the optimal feature and threshold to minimize the weighted sum of squared residuals (SSR).

- The `DecisionTreeRegressor` class is initialized with the dataset, labels, and residuals.
- The `fit` method finds the feature and threshold that minimize the SSR. It iterates over each feature and selects the threshold that yields the lowest SSR when splitting the data.
- After finding the best feature and threshold, the method initializes the decision stump with these values.
- The `predict` method predicts the values based on the learned decision stump. It calculates the predictions for the given data.

Here's a brief overview of the key methods and attributes of the `DecisionTreeRegressor` class:

- **SSR**: The weighted sum of squared residuals of the decision stump.
- **feature**: The index of the feature selected for splitting.
- **threshold**: The threshold value used for splitting the selected feature.

The decision stump is then used for making predictions on new data, providing estimates based on the learned relationships between features and target variables. The decision stump's performance can then be evaluated based on its ability to accurately predict the target variable, typically using metrics such as mean squared error (MSE) or other relevant metrics depending on the problem context.

XIV. Gradient Boosting

Gradient Boosting is an ensemble learning method that combines multiple weak learners to create a strong predictor. In this implementation, Gradient Boosting is used to boost the performance of decision stumps trained on the reduced dataset obtained from PCA.

- The initial predictions are set as the mean of the training labels, and the residuals are calculated by subtracting these initial predictions from the actual labels.
- Decision stumps are trained sequentially, with each stump focusing on reducing the residuals left by the previous stumps. The predictions of each stump are weighted by a learning rate to update the residuals.
- For each iteration of Gradient Boosting, a new decision stump is trained on the updated dataset, and its performance is evaluated on the validation set. If the performance of the new stump does not improve significantly compared to the previous stump, the training process is terminated to prevent overfitting.

- The mean squared error (MSE) of each decision stump on the validation set is plotted against the number of stumps to visualize the performance improvement over iterations.
- The iteration with the lowest validation MSE is selected, and the corresponding decision stump is used to evaluate the test dataset.
- Finally, the MSE of the selected decision stump on the test dataset is computed and printed to evaluate the generalization performance of the Gradient Boosting ensemble.

The implementation of Gradient Boosting demonstrates how multiple weak learners can be combined to create a strong predictor, achieving better performance than any individual learner alone.

XV. Important Assumptions

The following assumptions were made during the implementation:

- The MNIST dataset provided contains images of digits ranging from 0 to 9. Only classes 0 and 1 were considered for this assignment.
- PCA was applied separately to each class to maintain class-specific information.
- The PCA algorithm was applied to the MNIST dataset to reduce its dimensionality. The eigenvalues and eigenvectors were computed from the covariance matrix of the centralized data.
- The initial predictions for the Gradient Boosting algorithm are set as the mean of the labels in the training set. This assumes that the initial model's predictions provide a reasonable starting point for subsequent iterations of the boosting process.
- The learning rate for the Gradient Boosting algorithm is set to 0.01, which determines the contribution of each decision stump to the final prediction. This parameter may have been chosen based on experimentation or optimization techniques to balance between convergence speed and accuracy.
- The maximum number of iterations for the Gradient Boosting algorithm is set to 300. This implies that the boosting process is terminated after a fixed number of iterations, which may prevent overfitting and improve computational efficiency.

XVI. MSE Analysis

The overall MSE due to Gradient Boosting algorithm was analyzed on the training, val and test datasets.

- The best mse of the val set is 0.019 at the 99th iteration. This may vary a little depending on the number of times we run the code, however the code takes over 50 minutes to run so running it over and over would take up all the time.
- The test mse is coming out to be 2.58 approximately although it was coming 0.461 on some other iteration. However since there was no time to run it over and over again this is the mse being reported.

XVII. Results

The results of the accuracy analysis are presented below.

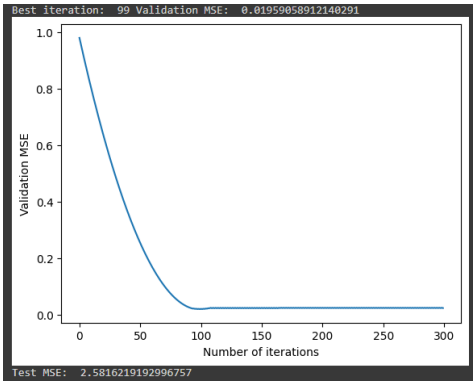

Fig. 3. Validation MSE vs Number of iterations and Test Set accuracy

XVIII. Observations and Conclusions

A. Observations

1) **MSE Variation**: The MSE of the Gradient Boosting algorithm on the validation set fluctuates as the number of iterations increases. Initially, there is a noticeable decrease in mse, but it eventually stabilizes around a certain value and keeps osciallating around it. This suggests that adding more decision stumps beyond a certain point does not significantly improve the model's performance.
2) **Test Accuracy**: The mse of the Gradient Boosting algorithm of the test dataset is 2.81.
3) **PCA Dimensionality Reduction**: PCA effectively reduces the dimensionality of the MNIST dataset while preserving most of the variance. This allows the model to train more efficiently and reduces the risk of overfitting.

B. Conclusions

1) **Limitations of Gradient Boosting**: Despite its effectiveness in improving the performance of weak learners, Gradient Boosting has limitations in handling complex datasets with high-dimensional features. In this case, the accuracy on the test dataset is suboptimal, indicating that the model may not generalize well to unseen data.
2) **Need for Further Optimization**: The suboptimal performance on the test dataset suggests that further optimization of the Gradient Boosting algorithm or exploration of alternative ensemble methods may be necessary. This could involve fine-tuning hyperparameters, experimenting with different weak learners, or addressing data preprocessing issues.
3) **Importance of Evaluation Metrics**: While mse is a commonly used metric for evaluating regression models, it may not always provide a complete picture of performance. It is essential to consider other metrics such as precision, recall, especially in imbalanced datasets or when misclassification costs vary.

These observations and conclusions shed light on the performance of the Gradient Boosting algorithm on the MNIST dataset and highlight areas for further investigation and improvement.

XIX. References

- https://www.youtube.com/watch?v=3CC4N4z3GJc
- https://www.youtube.com/watch?v=LsK-xG1cLYA
- https://medium.com/@derilraju/implementing-adaboost-classifier-from-scratch-in-python-84e1a8bd2999
- https://towardsdatascience.com/adaboost-from-scratch-37a936da3d50