Optimizing diabetes Prediction using Perceptron and Gradient-Based Optimizer.s

Anonymous submission

Paper ID

Abstract

Using the Pima Indian Diabetes dataset, I analyzed the performance of multiple optimization techniques for diabetes prediction, including SGD, Adam, and RMSProp. I have assessed the performance of various optimizers on a Perceptron model and show how the optimization strategies affect model accuracy, precision, recall, and F1 score. The findings demonstrate how optimization tactics influence model performance and convergence in binary classification problems. I have also included the learning curves and confusion matrices to help you visualize model performance.

1. Introduction

Diabetes is a huge worldwide health concern that requires early detection for better management and outcomes. Machine learning systems have showed promise for predicting diabetes using clinical data. The Perceptron model, a fundamental technique in the field of supervised learning, is particularly promising for binary classification tasks such as diabetes prediction. However, the optimization approach used during training can have an impact on the efficacy of a Perceptron model.

In this study, I analyzed the effects of three gradient-based optimization algorithms—SGD, Adam, and RMSProp—on the performance of a Perceptron model used to forecast diabetes from medical data. By comparing different optimizers, I hope to provide a light on the differences between how each technique influences model training and performance measures. I will demonstrate it through thorough testing how optimizer selection can result in varied degrees of precision and convergence.

2. Related Work

Existing research highlights the use of various machine learning techniques, such as Support Vector Machines (SVMs), Logistic Regression, and Random Forests, in diabetes prediction. These studies show an increased interest in using complex methodologies to improve forecast accuracy. However, the study of optimization algorithms, particularly Adam and RMSProp, in the context of simpler models like the Perceptron is underrepresented.

Previous research, [1]has shown that the choice of optimization technique has a significant impact on the performance of neural network models, as different algorithms have unique convergence speed, stability, and generalization capacities. The research seeks to fill this gap by systematically comparing the efficacy of various optimization methods used to the Perceptron model in predicting diabetes, so contributing to a better understanding of their significance in medical data analysis.

3. Methodology

3.1. Dataset Description

The Pima Indian Diabetes dataset serves as the basis for our analysis. This dataset contains 768 samples, each of these are recognized by eight clinical features, including glucose concentration, blood pressure, skin thickness, insulin levels, body mass index, and age. The goal variable is a binary value that shows whether the patient has diabetes or not. This large dataset serves as an appropriate platform for assessing the Perceptron model's predictive capabilities as well as the accuracy of various optimization strategies.

3.2. Data Preprocessing

Data preprocessing is a crucial step in machine learning that has a large impact on model performance. In my research, I used a StandardScaler to normalize the input features. This method normalizes the dataset by removing the mean and scaling to unit variance, ensuring that all features contribute equally to distance calculation during the optimization phase. Normalization is especially significant, [2] for gradient-based optimizers since it increases the convergence rate and general stability of the training process.

Data preprocessing includes normalizing the feature set to have a mean of zero and a standard deviation of one.

This was required to ensure that each attribute contributed equally to the model's learning process, especially for scale-sensitive models like neural networks.

3.3. Model Setup

I used the Perceptron model as an initial classifier, utilizing the features of the SGDClassifier from the Scikit-learn module. I also focused on three optimization techniques for comparison:

1.Stochastic Gradient Descent (SGD): A popular optimization strategy that iteratively changes model parameters using the gradient of the loss function obtained on a single sample or mini-batch. It brings variability into the optimization process, which may result in faster convergence but increased variation in updates.
2. Adam: This optimizer combines the advantages of two other SGD extensions: AdaGrad and RMSProp. Adam adjusts the learning rate for each parameter based on the gradients first and second moments, achieving a balance between convergence speed and stability. It is especially useful in circumstances where data is sparse or the optimization landscape is complex.
3. RMSProp: This approach is designed to address the drop in learning rates that might occur with traditional gradient descent algorithms. RMSProp computes a moving average of the squared gradients and divides the learning rate by the root of this average. This change provides for more consistent convergence across multiple parameters while also minimizing the oscillations that are common during training.
To achieve optimal performance for each optimization strategy, I have fine-tuned the hyperparameters such as the learning rate and regularization penalties. The model was trained by iterating through the dataset several times and adjusting weights depending on gradients generated from the loss function.

3.3.1 Perceptron:

The Perceptron is one of the earliest and simplest types of linear classifiers, designed to classify linearly separable data. Its decision boundary is a hyperplane, [**?**] hence it is unsuitable for datasets with complicated patterns, such as the Pima Indians dataset.

The Perceptron was implemented using the Scikit-learn Perceptron class. GridSearchCV was used to tune hyperparameters, including penalty (to regulate regularization) and alpha (to control learning rate). Training was carried out using max(_)iter=1000 and a tolerance of 1e-3.

The Perceptron iteratively adjusts its weights to reduce classification error, [3] however due to intrinsic constraints in representing non-linearity, it struggled to attain high accuracy on this dataset.

3.3.2 Neural Network:

I used Keras to create a fully connected neural network. The architecture consists of:

1.Input Layer: Accepting eight input features.
2.Hidden Layer: Three completely connected layers of 32, 64, and 32 neurons each, followed by the ReLU activation function to add nonlinearity.
3.Dropout Layer: To avoid the overfitting, I used a 50(%) of dropout rate after the hidden layers.
4.Output Layer: A single neuron with a sigmoid activity function predicts a binary outcome (diabetes or no diabetes).

I trained the Neural Network with three distinct optimizers: SGD, Adam, and RMSprop, and then analyzed their performance. Training was completed in 100 epochs with a batch size of 32. Binary cross-entropy was employed as the loss function, with accuracy serving as the performance metrics.

3.3.3 Hyper-parameter tunning:

I used GridSearchCV from the sklearn.model selection library to tune the model's hyperparameters and improve its performance. This method enables the systematic investigation of hyperparameter combinations while optimizing model performance through cross-validation.

1.Perceptron Model Tunning: In this perceptron model, I have defined the parameter grid that includes, max(_)iter, penalty and alpha.

a. max(_)iter : It controls the maximum number of iterations of the algorithm. I have tested the values of 500 and 1000.
b. penalty: This regularisation procedure was changed with the parameters 'l2','l1', and also no penalty. It reduces the overfitting by penalizing large coefficients.
c. alpha: This parameter represents the learning rate was evaluated with 0.0001 and 0.001 values to check how much the weights are altered in relation to the loss gradient.

2. Stochastic Gradient Descent(SGD) Classifier Tuning: Similarly, I optimized the SGD Classifier, focussing on different hyperparameters such as loss, penalty, alpha, and learning(_)rate.And then, the result indicates that the SGD classifier benefits from appropriate learning rate modification during training, together with L2 regularization and the hinge loss function, which is especially useful for large margin classifiers.

3.4. Model Evaluation

The model is evaluated based on the following metrics.

1. Accuracy: It is the percentage of correct predictions.
2. Precision: It is defined as the number of true positives divided by the total number of true and false positives.
3. Recall: The number of true positives divided by the sum of true positives and false negatives.
4. F1-Score: It is the harmonic mean of precision and recall.
Confusion matrices are also used to visualize the performance of each model.

4. Results

4.1. Perceptron Result:

The Perceptron has obtained an accuracy of 62.34 (%) on the test set. Despite hyperparameter optimization, the model failed to capture the non-linear correlations between characteristics and outcomes, resulting in poor performance on the more complicated parts of the dataset. The confusion matrix demonstrated that the Perceptron had a significant false negative rate, indicating that it was particularly unsuccessful at detecting diabetic patients.

4.2. Neural Network:

The best Neural Network design uses the RMSprop optimizer and attained an accuracy of 72.73(%)significantly exceeding the Perceptron. However, while the accuracy was greater, the recall was very low at 0.13, indicating that the model was missing a huge portion of actual diabetic cases. This is a common problem when working with imbalanced datasets.

4.3. Precision, Recall, and F1-Score:

For the RMSprop-that is optimized using Neural Network had a precision of 1.00, indicating that every patient predicted as diabetic was truly diabetic. However, the low recall resulted in an F1-score of 0.23, highlighting the imbalance between positive and negative classes. Table 1 depicts the confusion matrix for Neural Network performance.

Metric	Perceptron	Neural Network(RMSProp)
Accuracy	62.34(%)	72.73(%)
Precision	0.75	1.00
Recall	0.23	0.13
F1-Score	0.35	0.23

Table 1. Confusion matrix for the best Neural Network.

4.4. Analysis of Results

The RMSProp optimizer scored the maximum accuracy of 72.73(%), demonstrating its ability to navigate the optimization environment for this particular dataset. This shows that RMSProp's flexible learning rate and changing the average of gradients created better learning than other approaches.

However, the Adam optimizer achieved a somewhat lower accuracy of 66.88(%), which could be attributed to its adaptive features being less effective in this setting. The SGD optimizer achieved an accuracy of 68.83(%), demonstrating its potential but also emphasizing the difficulties associated with adjusting its hyper parameters to attain optimal results.

The confusion matrix for each model gives additional information about missclassification patterns, which is critical for understanding model limitations in properly predicting diabetes.

5. Learning Curves Interpretation

Learning curves are graphs that illustrate the link between training set size and model performance (accuracy or loss).

5.0.1 Perceptron Learning Curves:

Both training and validation accuracy are likely to stop at a lower level, indicating that the model is too simple to fit the dataset effectively. This suggests that the Perceptron is unsuitable for this nonlinear situation and has a limited capacity to capture nuances in the data.

5.0.2 Neural Network learning Curves:

The training accuracy increases rapidly, while the validation accuracy stabilizes sooner. If you notice a significant difference between these curves in this data, this may indicate overfitting. However, regularization helps to mitigate this effect, and the validation accuracy remains somewhat near to the training accuracy.

6. Confusion Matrix

The confusion matrix provides an information based on the performance of a classification model, specifically how well the model predicted classes relative to the true labels. Here is a breakdown of the confusion matrix.

	Predicted Positive	Predicted Negative
True Positive	5	50
True Negative	1	98

Table 2. Confusion Matrix of the Model's Predictions

6.1. Metrics Calculation from the Confusion metrics

1.Accuracy:

$$TP+TN/TP+TN+FP+FN = 5+98/5+98+1+50 = 0.669 \tag{1}$$

2.Precision:

$$TP/TP+FP = 5/5+1 = 5/6 = 0.833 \tag{2}$$

This indicates that when the model predicts positive, it is correct about 83.3(%)of the time.

3.Recall:

$$TP/TP+FN = 5/5+50 = 5/55 = 0.091 \tag{3}$$

4.F1-Score:

$$2X(PrecisionXRecall/Precision + Recall) = 0.166 \tag{4}$$

7. Conclusion

1.Perceptron Model: I have evaluated the performance of two machine learning models, that is a perceptron and neural network on a binary classification problem on a binary classification problem using the Pima Indians Diabetes dataset. The goal was to estimate the likelihood of diabetes based on several medical characteristics.
2. Model Performance: The accuracy results indicates that the neural network exceeded the Perceptron, with higher accuracy because of its more complex architecture, which allows for better learning of non-linear patterns in data. This is supported by the accuracy comparison graph, which clearly displays the strengths of each model.

In conclusion, both the Perceptron and Neural Network models were effective in predicting diabetes. I determined the best settings for model accuracy by adjusting hyperparameters. The neural network consistently outperformed the perceptron, indicating its ability to handle complex categorization tasks.

References

[1] Hazem M. El-Bakry Abeer El-Sayyid El-Bashbishy. Pediatric diabetes prediction using deep learning. *scientific reports*, 14(4206):3103, 2024. 1

[2] Filbert H.Junwono King Hann Lim W.k.Wong Boon Feng Wee, Saveethya Sivakumar. Diabetes detection based on machine learning and deep learning approaches. *springerLink*, 83(4206):24153–24185, 2023. 1

[3] Shai Shalev-Shwartz. The perceptron algorithm. (6):642–644, 1959. 2

7.1. Link to the github

https://github.com/Khurrat123/deep-learnig-asignment1