Stacked Ensemble Models versus Deep Learning in Federated Contexts

Hira Sardar

Abstract: Federated Learning (FL) is a distributed machine learning paradigm that facilitates collaborative model training across multiple decentralized devices while preserving data privacy. Unlike traditional centralized learning, where data is aggregated in a central server, FL enables model training directly on client devices, with only model updates shared across the network. This study delves into the implementation of FL using a modified ResNet-50 model, adapted for tabular data, and evaluates its performance in a federated context. We explore key concepts such as Federated Averaging (FedAvg), client-server architecture, and privacy preservation techniques. The experimental analysis involves preprocessing a dataset, performing exploratory data analysis, and segmenting the data for federated training. The results highlight the challenges in model generalization, as indicated by negative R-squared values and poor performance metrics, suggesting the need for further optimization in FL architectures and methodologies.

1. Introduction

Federated Learning (FL) is an emerging machine learning paradigm that enables collaborative model training across multiple decentralized devices or servers without directly sharing data. Unlike traditional centralized approaches, where data is gathered in a central server, FL allows models to be trained locally on client devices, with only model updates being shared. This decentralized approach helps preserve data privacy and is particularly beneficial in scenarios where data is sensitive or cannot be moved due to legal or privacy constraints.

Research into Federated Learning is essential for understanding its underlying principles, benefits, and the techniques commonly employed in this field. To gain a solid foundation in FL, one should begin by exploring reliable sources such as academic papers, textbooks, and reputable online articles. Key concepts to focus on include:

- **Federated Averaging (FedAvg)**: A widely used algorithm in FL, where model updates from different clients are averaged to form a global model. This method helps reduce communication overhead and ensures that the global model is a reflection of the collective knowledge of all participating clients.
- **Client-Server Architecture**: The typical architecture in FL where a central server coordinates with multiple clients (devices) that locally train models using their data. The server aggregates these updates to improve the global model iteratively.
- **Privacy Preservation in FL**: Since data remains on the client devices, FL inherently provides a layer of privacy. However, additional techniques such as differential privacy, secure multi-party computation, and homomorphic encryption are often employed to further protect sensitive information.

Understanding these concepts is crucial for anyone looking to delve into Federated Learning, as they form the foundation upon which more advanced methodologies and applications are built. As the field continues to evolve, staying informed about the latest research and developments is vital for leveraging FL in real-world scenarios.

2. Experimental Analysis

2.1. Data Preprocessing

The dataset was initially loaded from an Excel file using the `pandas` library. After loading, any missing values were dropped using the `dropna()` function to ensure a clean dataset for further processing. To prepare the data for machine learning models, we scaled all numerical features using the `StandardScaler` from the `sklearn.preprocessing` module. This step normalized the data, which is crucial for optimizing the performance of machine learning models, particularly those that rely on gradient descent.

2.2. Exploratory Data Analysis

We performed exploratory data analysis (EDA) by visualizing the relationships between various features and the target variable, GSNR_1. Scatter plots were created using `matplotlib` to inspect the relationships between power, NLI (Non-Linear Interference), ASE (Amplified Spontaneous Emission), frequency, the number of spans, and the total distance with GSNR_1. This analysis provided insights into the correlations and potential influences of these features on GSNR_1.

2.3. Data Segmentation

The dataset was segmented based on the unique values in the ”No. Spans” column. Each subset of the data corresponding to a unique span value was saved as a separate CSV file. This step allowed us to create different datasets that could be used to train separate models for different clients in a federated learning context.

2.4. Data Loading

We utilized the `torch.utils.data.DataLoader` to load the data for training, validation, and testing. The input features and target variables were converted to PyTorch tensors, and the data was loaded into batches of 64 samples each. Three separate clients were defined using the first three datasets, while the fourth dataset was used for validation, and the fifth dataset was reserved for testing.

2.5. Model Architecture

We modified the pre-trained ResNet-50 model from `torchvision.models` to adapt it for tabular data. The first convolutional layer of ResNet was altered to accept a single channel input, and the final fully connected layer was replaced with a custom sequence of fully connected layers. The majority of the pre-trained ResNet layers were frozen to retain the learned representations, while the new layers were left unfrozen to allow training on our specific task.

2.6. Model Training

For each client, we initialized a separate instance of the modified ResNet model. The training process was conducted for 5 epochs using the Adam optimizer and Mean Squared Error (MSE) as the loss function. During training, the model's performance was evaluated on the validation set to monitor overfitting and adjust hyperparameters accordingly.

2.7. Feature Extraction and Data Preparation

To leverage the power of deep learning for feature representation, we utilized a pre-trained ResNet model to extract features from our dataset. The process was as follows:

1) **Feature Extraction:**
   - Extracted features from training data for each client using the ResNet model:
     - `train_features_client1`
     - `train_features_client2`

  - `train_features_client3`
- Extracted features from validation and test datasets:
  - `val_features`
  - `test_features`

2) **Concatenation:**
   - Combined all training features into a single dataset `X_train` by concatenating features from all clients along the first axis.
   - Combined all training targets into a single array `y_train` by concatenating targets from all clients.

3. Evaluation of Model involving Sacking Ensemble and Resnet

Code Explanation

The code provided performs a stacking ensemble learning approach using three different base regressors: RandomForestRegressor, GradientBoostingRegressor, and XGBoostRegressor. The steps are detailed as follows:

1) **Feature Extraction:** Features are extracted from the training, validation, and test datasets using a pre-trained ResNet model. The features from three clients are extracted separately and later combined to form a complete training set.
2) **Concatenation of Features and Targets:** The extracted features from the three clients are concatenated to form the complete feature set `X_train`. Similarly, the target values from the three clients are concatenated to form `y_train`.
3) **Base Learners Training:** Three different base learners (RandomForestRegressor, GradientBoostingRegressor, and XGBoostRegressor) are initialized and trained using the combined training features and targets.
4) **Stacking Process:** The predictions from the base learners are stacked together as new features (`X_meta`) to train a meta-learner. The meta-learner used is a simple Linear Regression model.
5) **Validation Predictions and Evaluation:** The trained meta-learner is used to make predictions on the validation set, and the performance is evaluated using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2).
6) **Test Predictions and Evaluation:** The meta-learner is also used to predict the outcomes on the test set, and the results are evaluated using the same metrics.

Results

The performance of the stacking model on the validation and test datasets is summarized in Table I. The metrics used to evaluate the model include Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2). The negative R-squared values indicate that the model is performing worse than a horizontal line (mean of the data), implying that the stacking approach might not be the best fit for this particular problem.

TABLE I
MODEL PERFORMANCE ON VALIDATION AND TEST SETS

Metric	Validation Set	Test Set
MSE	1.3991	2.8968
MAE	0.8794	1.3142
RMSE	1.1828	1.7020
R^2	-4.0635	-0.7036

The negative R^2 values indicate that the model is performing poorly, which might be due to

overfitting or an inappropriate model selection. Further analysis or alternative models may be necessary to improve performance.

3.1. Conclusion

The experiments conducted demonstrate that while deep learning models like ResNet can effectively extract meaningful features from complex datasets, careful consideration must be given to model selection and ensemble strategies. The standalone ResNet model outperformed the stacked ensemble in this scenario, suggesting that the additional layers of modeling may have introduced unnecessary complexity. Future work should focus on optimizing model architectures, exploring different ensemble techniques, and conducting thorough hyperparameter tuning to enhance predictive performance.

4. Evaluation of Model having Resnet

In this section, we evaluate the performance of our trained RESNET model using various metrics, including Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the Coefficient of Determination (R^2). The evaluation is performed on both the validation and test datasets.

The following code defines a function `evaluate_model` that takes a trained model, a data loader, and target values as inputs. It computes predictions and compares them against the actual target values using the specified metrics.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

def evaluate_model(model, data_loader, targets):
    model.eval()  # Set model to evaluation mode
    preds = []
    actuals = []

    with torch.no_grad():
        for features, target in data_loader:
            outputs = model(features).squeeze(1)
            preds.append(outputs.cpu().numpy())
            actuals.append(target.cpu().numpy())

    preds = np.concatenate(preds, axis=0)
    actuals = np.concatenate(actuals, axis=0)

    mse = mean_squared_error(actuals, preds)
    mae = mean_absolute_error(actuals, preds)
    rmse = np.sqrt(mse)
    r2 = r2_score(actuals, preds)

    return mse, mae, rmse, r2
```

The function returns the MSE, MAE, RMSE, and R^2 values after evaluating the model on the given dataset.

4.1. Evaluation on Validation and Test Sets

We evaluate the model on both the validation and test sets using the `evaluate_model` function. The results are summarized in the following table:

Dataset	MSE	MAE	RMSE	R^2
Validation Set	0.3019	0.5032	0.5494	-0.0924
Test Set	2.9954	1.2462	1.7307	-0.7616

TABLE II
MODEL EVALUATION RESULTS

4.2. Interpretation of Results

From the table, we observe the following:

- The Mean Squared Error (MSE) is significantly lower on the validation set compared to the test set, indicating that the model may not generalize well to unseen data.
- The Mean Absolute Error (MAE) follows a similar trend, suggesting that the model's predictions on the test set are less accurate on average.
- The Root Mean Squared Error (RMSE) is also higher on the test set, further emphasizing the model's poor generalization.
- The R^2 value is negative for both the validation and test sets, with a much lower value on the test set. This indicates that the model fails to capture the variance in the target variable, particularly on the test set.

The negative R^2 value suggests that the model performs worse than a simple mean prediction, especially on the test set. This poor performance indicates the need for further tuning or selecting a different model architecture.

5. Conclusion

The conducted experiments demonstrate that while deep learning models like ResNet can extract valuable features from complex datasets, careful consideration of model selection and ensemble strategies is crucial in a federated learning context. The modified ResNet model exhibited better performance than the stacking ensemble, indicating that additional layers of modeling complexity might lead to overfitting and reduced generalization. The negative R-squared values observed in the evaluation suggest that the current approach may not be optimal for the specific problem at hand. Future work should focus on optimizing the model architecture, exploring alternative ensemble techniques, and conducting comprehensive hyperparameter tuning to enhance predictive performance in federated learning scenarios.