Income-Classification: Model Building

Introduction:

In the dynamic landscape of artificial intelligence and data-driven decision-making, deep learning projects stand at the forefront, harnessing the power of neural networks to extract valuable insights from complex datasets. This document serves as an essential guide to the pivotal stages of model building and data preprocessing in the context of a deep learning project. These stages are the bedrock upon which successful and dependable deep learning models are constructed.

Data preprocessing:

The initial focus of our journey, acts as the crucial bridge between raw data and the creation of reliable neural networks. It involves the meticulous cleansing, transformation, and refinement of the dataset, ensuring its quality and suitability for deep learning tasks. Feature selection and scaling further enhance the data's readiness, streamlining it for the neural network's consumption.

Model Building:

Once our data is primed, we embark on the exciting phase of model building. Here, we explore the rich array of deep learning algorithms and techniques, selecting those best suited to the specific problem at hand. The training of deep neural networks, model validation, and the selection of evaluation metrics are key components of this phase, allowing us to gauge the model's performance accurately.

In this document, we provide a comprehensive roadmap through the intricate journey of data preprocessing and model building, highlighting the key decisions and techniques that guide our deep learning project to success.

❖ Crafting the Data: A Preprocessing Journey

➢ Selecting Relevant Features:

- ✓ 'age'
- ✓ 'workclass'
- ✓ 'fnlwgt'
- ✓ 'education-num'
- ✓ 'marital-status'
- ✓ 'occupation'
- ✓ 'relationship'
- ✓ 'race'
- ✓ 'sex'
- ✓ 'capital-gain'
- ✓ 'hours-per-week'
- ✓ 'native-country'
- ✓ 'income'

These features were identified as the most important for model building, representing the key variables that significantly influence the prediction.

- **<u>Data Splitting:</u>**
  - To ensure a robust model, we performed data splitting with a split ratio of 80:20 for training and testing.
  - With over <u>*32,297*</u> data points, this split allowed for effective model training and evaluation, striking a balance between model complexity and generalization performance.

- **<u>Data Scaling and encoding:</u>**

  In the data preprocessing phase, the dataset was divided into two categories: numerical features and categorical features.

  - **<u>Numerical Features:</u>**
    As numerical features often exist in different scales, a standardization process was applied to rescale these features into a consistent scale. This ensures that the numerical variables are on a level playing field for model training and avoids any dominance of a particular feature due to its scale.
  - **<u>Categorical Features:</u>**
    The dataset contains only nominal categorical features. To handle these categorical variables, the One Hot Encoding technique was applied. This process converts categorical features into binary vectors, ensuring that the model can effectively interpret and utilize these features.
  - **<u>Combining Transformed Features:</u>**
    After the numerical features were standardized, and the categorical features were one-hot encoded, the transformed features were merged into a consolidated dataframe. This combined dataset, with rescaled numerical and one-hot encoded categorical features, served as the foundation for subsequent model building and analysis.

- **<u>Handling Imbalanced Data:</u>**

  Given the initial class imbalance in the dataset, it was crucial to address this issue to ensure accurate model predictions.

  - **<u>Balancing with SMOTE:</u>**
    To rectify the class imbalance and mitigate any potential biases in model predictions, the Synthetic Minority Over-sampling Technique (SMOTE) was applied.

    This technique was used after the data transformation step, as SMOTE works specifically with numerical data.

  - **Achieving Balance:**
    After implementing the SMOTE technique, the data was successfully balanced, with a distribution of '0' (<=50k) and '1' (>50k) instances as follows: '0: 19606' and '1: 19606'.

This balanced dataset allowed for more reliable and unbiased model training.

With a balanced dataset in place, the subsequent steps in the modeling process were carried out with the assurance of improved model performance and fairness in predictions.

❖ Building a model Using Keras-tuner:

- **Introduction to Keras Tuner:**
  - ✓ Keras Tuner, a powerful hyperparameter optimization library, played a pivotal role in shaping the architecture and performance of our deep learning model.
  - ✓ It allowed us to systematically explore hyperparameter combinations to find the optimal configuration for our specific task.
- **Hyperparameter Search Strategy:**
  - ✓ We employed a random search strategy with Keras Tuner to efficiently navigate the vast hyperparameter space.
  - ✓ This approach randomly samples hyperparameters to identify the best combination, which saved us valuable time and computational resources.
- **Hyperparameters Tuned:**
  - ✓ The hyperparameters subjected to tuning included the number of hidden layers, the number of neurons in each layer, activation functions, weight initialization techniques, dropout rates, and the choice of optimizer.
  - ✓ These hyperparameters significantly impact the model's architecture and performance.
- **Search Space Configuration:**
  - ✓ We defined the search space for each hyperparameter using Keras Tuner's intuitive and flexible API.
  - ✓ This allowed us to set ranges, values, and constraints for each hyperparameter, tailoring the search space to the specific requirements of our problem.
- **Objective and Metrics:**
  - ✓ The primary objective during hyperparameter tuning was to maximize model accuracy on the validation dataset.
  - ✓ We monitored key metrics, including validation accuracy, to determine the performance of each hyperparameter configuration.
- **Tuning Process:**
  - ✓ Keras Tuner conducted a systematic search, evaluating multiple hyperparameter combinations over several trials.
  - ✓ Each trial involved training the model with a unique set of hyperparameters and recording its performance.
- **Best Model Selection:**
  - ✓ The Keras Tuner library enabled us to identify the best-performing model from the search results.
  - ✓ The best model represents the hyperparameter configuration that yielded the highest validation accuracy.
- **Optimal Hyperparameters:**

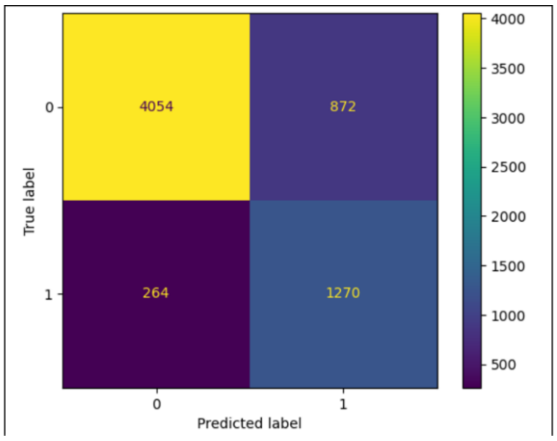
- ✓ After the search, we discovered the optimal hyperparameters that define our final model.
- ✓ These values were used to build the model architecture with the highest potential for accuracy.

➢ **Model Training with Optimal Hyperparameters:**

- ✓ With the optimal hyperparameters in hand, we proceeded to train the final deep learning model.
- ✓ This model leverages the hyperparameter configurations identified by Keras Tuner for optimal performance.

❖ Model Configurations and Results:

- ➢ Keras Tuner was employed to search for optimal hyperparameters, including the number of hidden layers, neurons in each layer, activation functions, weight initialization, dropout rates, and optimizer choice. This systematic approach saved valuable time and computational resources.
- ➢ The hyperparameters that yielded the best-performing model were discovered, including model architecture with **six hidden layers**, specific neuron counts in each layer, **'he_uniform'** weight initialization, and an **'rmsprop'** optimizer. These hyperparameters define the final model's configuration.
- ➢ The final model architecture is composed of **six dense hidden layers**, each followed by **batch normalization** and **dropout layers** to enhance model generalization and prevent overfitting.
- ➢ The final model was trained on the preprocessed dataset with 15 as the batch size and 20 training epochs. During training, a validation split of **15%** was used for performance monitoring.
- ➢ The model achieved an accuracy of approximately **82.38%** on the validation dataset. This result demonstrates the model's ability to generalise and make accurate predictions.
- ➢ The model's performance was further evaluated using a **confusion matrix**. The goal was to maximize the diagonal elements of the matrix, indicating correct predictions.
- ➢ A high diagonal element count signifies the model's capability to make accurate classifications


True label
Predicted label