The telecommunication industry has witnessed a significant rise in customer churn, which refers to the phenomenon where customers switch from one service provider to another. Customer churn not only leads to revenue loss but also affects the overall profitability and growth of telecom companies. To understand and address this issue, extensive analysis of customer churn patterns is crucial.

In this context, I use the Telecom Customer Churn dataset, a comprehensive collection of relevant customer data that can be used for churn prediction and analysis. The dataset encompasses various factors that influence customer behavior, including demographic information, service features, contract details, payment methods, and internet service types.

FNN is used here to predict potential customer churning rate. A feedforward neural network is an artificial neural network in which the connections between nodes do not form cycles. The opposite of a feedforward neural network is an iterative neural network that traverses specific paths. A feedforward model is the simplest form of neural network because information is processed in only one direction. Data can pass through multiple hidden nodes, but it always travels in one direction, never the other.

The code provided demonstrates the construction and training of a Feedforward Neural Network (FNN) using TensorFlow and Keras. The construction process of a feedforward neural network (FNN) involves several key components and terms. Let's break down the process and explain each term:

1. Network Architecture and Hyperparameters:

- The code starts by defining the network architecture using the `create\_fnn\_model` function. It creates a sequential model using the `Sequential` class from Keras. The model consists of one or more dense layers with the specified number of units and activation functions.

- The hyperparameters such as the input dimension, output dimension, different network architectures, activation functions, and learning rates are defined.

2. Model Training and Evaluation Loop:

- The code enters a nested loop that iterates over different combinations of network architectures, activation functions, and learning rates.

network architectures: [32] represents a network with a single hidden layer containing 32 neurons.[50, 64] represents a network with two hidden layers, the first layer containing 50 neurons and the second layer containing 64 neurons.[27, 128, 32] represents a network with three hidden layers, with 27 neurons in the first layer, 128 neurons in the second layer, and 32 neurons in the third layer.

An activation function determines the output of a neuron given its inputs. It introduces non-linearities into the network, allowing the FNN to model complex relationships between inputs and outputs. The code uses popular activation functions: 'relu', 'sigmoid', and 'tanh'. 'relu': The Rectified Linear Unit (ReLU) activation function, which sets negative values to zero and keeps positive values unchanged. 'sigmoid': The sigmoid activation function, which squashes the output between 0 and 1, making it suitable for binary classification problems. 'tanh': The hyperbolic tangent activation function, which squashes the output between -1 and 1, suitable for problems with outputs ranging from negative to positive values.

Different Learning Rates: The learning\_rates variable defines different learning rates for the optimization algorithm used during training. Learning rate determines the step size at which the FNN adjusts its weights and biases based on the computed gradients. By defining different learning rates, the code explores the effect of the learning rate on the training process and the convergence of the FNN. A higher learning rate may result in faster convergence but risk overshooting the optimal weights, while a lower learning rate may lead to slower convergence but potentially more accurate results. In the given code, three learning rates are considered: 0.01, 0.001, and 0.0001.

- Inside the loop:

- A model is created based on the current architecture using the `create\_fnn\_model` function.

- The model is compiled with the chosen learning rate and the Adam optimizer using the `compile` method. The loss function is set to `'sparse\_categorical\_crossentropy'`, suitable for multi-class classification.

- The data is split into training and validation sets using `train\_test\_split` from scikit-learn.

- The model is trained on the training set using the `fit` method. It runs for 10 epochs with a batch size of 32.

- The model is evaluated on the validation set using the `evaluate` method, and the validation accuracy is calculated.

- If the current model achieves a higher accuracy than the previous best accuracy, the best accuracy is updated, and the best model, architecture, activation function, and learning rate are updated.

3. Print Results:

- After the training and evaluation loop completes, the code prints the results for each model configuration, including the architecture, activation function, learning rate, and validation accuracy.

4. Evaluate the Best Model on the Test Set:

- Once the training loop is finished, the code uses the best model obtained during the loop to make predictions on the test set (`X\_test`).

- The predicted probabilities for each class are obtained using the `predict` method, and the predicted class labels (`y\_pred`) are obtained by selecting the class with the highest probability using `argmax`.

5. Calculate Classification Metrics and Print the Summary of the Optimal Model:

- The code calculates classification metrics (precision, recall, F1-score, and support) using the `classification\_report` function from scikit-learn.

Precision: Precision measures the proportion of correctly predicted positive instances out of the total instances predicted as positive. For class 0, the precision is 0.83, indicating that 83% of the instances predicted as class 0 were actually class 0. For class 1, the precision is 0.64, meaning that 64% of the instances predicted as class 1 were actually class 1.

Recall: Recall (also known as sensitivity or true positive rate) measures the proportion of actual positive instances that are correctly predicted. For class 0, the recall is 0.90, indicating that 90% of the actual class 0 instances were correctly predicted as class 0. For class 1, the recall is 0.51, meaning that only 51% of the actual class 1 instances were correctly predicted as class 1.

F1-Score: The F1-score is the harmonic mean of precision and recall. It provides a balanced measure of the model's accuracy, considering both false positives and false negatives. For class 0, the F1-score is 0.86, indicating a good balance between precision and recall. For class 1, the F1-score is 0.57, suggesting that there may be a trade-off between precision and recall for class 1 predictions.

Accuracy: Accuracy measures the overall correctness of the model's predictions. The accuracy achieved on the test set is 0.79, meaning that the model correctly predicts the class labels for approximately 79% of the instances in the test set.

- Finally, the code prints a summary of the best model, including the architecture, activation function, and learning rate.