Delivery Time Classification Using Neural Networks: A Predictive Modeling Approach

Introduction

Timely delivery in logistics is critical for customer satisfaction and operational efficiency. With the rise of online food, grocery, and e-commerce services, predicting whether a delivery will be “Fast” or “Delayed” is of immense business value. This project aims to use machine learning—specifically, a feedforward neural network—to classify delivery speed based on factors like traffic, weather, distance, and priority. Traditional statistical methods like Logistic Regression will also be applied for comparison.

We start with cleaning and preprocessing the dataset, followed by defining the target variable by converting continuous delivery times into binary classes using the dataset's mean as a threshold. Feature selection and encoding are applied to prepare the data for modeling. The chosen neural network is trained using the Keras API with TensorFlow backend. The model's performance is evaluated using accuracy, precision, recall, and F1-score.

Additionally, model tuning is performed to explore deeper architectures, optimize learning rate, and adjust hyperparameters. The evaluation phase compares the neural network with Logistic Regression to understand strengths and weaknesses in each. The final section provides data-backed recommendations on improving delivery speed prediction and optimizing the logistics chain using AI solutions.

Problem Statement and Objective

The primary objective of this project is to predict whether a delivery will be fast or delayed using supervised machine learning, focusing on artificial neural networks. Delivery time is affected by multiple dynamic factors such as traffic conditions, weather variations, order distance, and customer priority level. These variables form the feature space for our model.

Real-world delivery datasets often contain inconsistencies and skewed distributions. Instead of predicting the exact time (a regression task), we simplify the objective into a binary classification problem: deliveries below the average delivery time are labeled "Fast" (0), and others as "Delayed" (1). This conversion allows us to treat the problem as a binary classifier suitable for models like logistic regression or neural networks with a sigmoid activation in the output layer.

The project is not just about classification but also about evaluating the effectiveness of deep learning over traditional models. By comparing both, we aim to discover whether the neural network captures nonlinear patterns better and whether it justifies the added complexity. Our broader goal is to develop a model that can be deployed into real-time logistics systems for proactive decision-making.

Data Preprocessing

The dataset includes features such as Delivery_Time, Traffic_Conditions, Weather_Conditions, Order_Distance, and Order_Priority. The target variable is derived from the Delivery_Time column. A threshold was created using the mean delivery time, labeling entries below the mean as "Fast" (0) and those above or equal as "Delayed" (1). This binary classification format is more manageable for real-time deployment and simplifies the learning task.

Categorical variables like Traffic_Conditions and Weather_Conditions were label-encoded or one-hot encoded, depending on cardinality. All numerical features, such as distance, were normalized using StandardScaler to ensure even feature contribution to gradient descent during training.

The dataset was split into training and test sets in an 80:20 ratio. Feature correlation was visualized through heatmaps to inspect multicollinearity and potential feature engineering. Null values were handled by imputation where applicable, or rows were dropped if data was insufficient.

Dimensionality was kept low to avoid the curse of dimensionality in neural networks. This also ensured faster convergence. Additional features like time of day or day of week could be engineered in future improvements to enrich the feature space.

Model Architecture

A feedforward neural network (also known as a multi-layer perceptron or MLP) was implemented using the Keras library with TensorFlow backend. The model architecture includes:

- **Input layer** based on the number of features (after encoding)
- **Hidden Layer 1:** 64 neurons, ReLU activation
- **Hidden Layer 2:** 32 neurons, ReLU activation
- **Hidden Layer 3:** 16 neurons, ReLU activation
- **Output Layer:** 1 neuron with Sigmoid activation for binary output

This architecture was selected to capture both linear and nonlinear relationships. The ReLU function enables the network to model complex patterns without vanishing gradient issues. The sigmoid activation at the output ensures output in the range [0, 1], suitable for binary classification.

The network was compiled using the Adam optimizer with a learning rate of 0.001. The loss function was binary_crossentropy, optimal for binary classification tasks. The model was trained over 50 epochs with a batch size of 16. EarlyStopping was used to prevent overfitting by monitoring validation loss and halting training once performance plateaued.

Evaluation Metrics Explained

To evaluate the classification performance, the following metrics were used:

- **Accuracy:** The proportion of correct predictions over total predictions. While informative, it may not reflect the true performance on imbalanced datasets.
- **Precision:** The ratio of true positives to all positive predictions. High precision ensures that predicted "Delayed" deliveries are usually correct.
- **Recall:** The ratio of true positives to all actual positive cases. High recall ensures that most delayed deliveries are correctly identified.
- **F1-score:** Harmonic mean of precision and recall. This is particularly useful when there's a trade-off between the two.

These metrics were calculated on both the training and validation datasets. The neural network gave a test accuracy of approximately 53%, precision and recall near 0.5, which is only slightly better than random guessing. This indicates the need for model tuning or possibly more informative features.

Neural Network Results and Interpretation

After training the neural network with default hyperparameters, the model yielded the following results on the test set:

🔍 **Neural Network Evaluation**

Accuracy: 0.525

Precision: 0.5789473684210527

Recall: 0.5

F1-score: 0.5365853658536586

While these values are not high, they indicate that the model is learning some patterns but lacks consistency. The difference between precision and recall suggests a slight class imbalance or insufficient training.

Confusion matrix visualization showed moderate success in predicting delayed deliveries but more false positives for fast ones. This behavior may stem from limited data size or insufficiently distinct features. Adding time-based or location-specific variables could significantly improve this.

The model is likely underfitting, indicating that more complex architectures or data preprocessing enhancements are needed. Nonetheless, this baseline performance provides a reference point for further tuning and comparison with other models.

Logistic Regression Comparison

As a benchmark, Logistic Regression was trained on the same preprocessed dataset. Being a linear model, it provides insight into the minimum performance we can expect with simple techniques.

Logistic Regression achieved:

🔍 **Logistic Regression Evaluation**

Accuracy: 0.575

Precision: 0.6086956521739131

Recall: 0.6363636363636364

F1-score: 0.6222222222222222

These results reveal that Logistic Regression performed nearly as well as the neural network, albeit with simpler computation and interpretability. The neural network only slightly outperformed it, suggesting that the additional complexity of deep learning was not fully utilized due to limited feature richness or data volume.

The similarity in performance implies that either the problem is linearly separable or that current features lack the depth to benefit from non-linear modeling. This insight highlights the importance of feature engineering and the diminishing returns of deep models on small or weakly informative datasets.

Model Tuning and Optimization

To improve model performance, several hyperparameters were tuned:

- Increased number of neurons to 64/32/16 in hidden layers
- Changed optimizer to RMSprop and SGD (though Adam performed best)
- Learning rate adjusted between 0.001 and 0.0001
- Added Dropout layers to prevent overfitting (not significantly effective here)

Additionally, techniques like batch normalization and L2 regularization were tested. These reduced training time but didn't drastically improve metrics.

The biggest gains were observed when feature scaling and EarlyStopping were applied correctly. Epoch count was also increased to 100 with patience of 10. The model performed slightly better, reaching ~55% accuracy.

However, these gains plateaued quickly, reaffirming that the dataset likely needs better feature signals. The model may also benefit from larger datasets or external data (e.g., historical delays, driver rating, route congestion).

Future tuning can involve automated techniques like GridSearchCV for traditional models or KerasTuner for neural networks, which systematically explore hyperparameter combinations.

Key Findings

1. **Delivery Speed Prediction is Difficult:** Simple models like logistic regression perform similarly to neural networks, suggesting that the existing features may not strongly determine delivery speed.
2. **Neural Network Marginally Better:** The deep learning model captured slightly better patterns but was limited by input quality.
3. **Evaluation Metrics Reveal Class Confusion:** While recall and precision were balanced, neither was high, indicating a lack of clear signal in the dataset.
4. **Hyperparameter Tuning Has Limited Impact Alone:** Changing neurons/layers helped only marginally. Feature quality matters more.
5. **Binary Threshold Based on Mean is Simple Yet Effective:** Using the mean delivery time for labeling works well for binary classification, but more dynamic thresholds may yield better results.
6. **Data Volume and Feature Depth are Crucial:** The models likely underperform due to limited feature space or insufficient data quantity.

Recommendations

Based on our findings, here are key actionable recommendations:

1. **Enrich Feature Space:** Add more variables like time of day, delivery zone, driver experience, and customer type. This will help models better capture delay factors.
2. **Collect More Data:** The current dataset is relatively small. Larger datasets will improve generalization and training.
3. **Deploy Hybrid Models:** Combine neural networks with decision trees (e.g., Neural Boosting or ensemble methods) for better performance.
4. **Monitor Predictions in Real-Time:** Use prediction probabilities to trigger delivery alerts when a delay is likely.
5. **Use AutoML or KerasTuner:** Automate hyperparameter search to systematically improve architecture and training parameters.
6. **Model Explainability:** Use SHAP or LIME to understand feature contributions, especially before deploying in production.
7. **Incremental Learning:** Update the model periodically as new delivery records come in, ensuring relevance and freshness.