Nusuk dataset deep learning project

Nusuk is an App that provides reservation, awareness and guidance services during hajj and Umrah.

Feature engineering

is a crucial step in building machine learning models. The goal is to extract relevant information of the dataset to improve model performance. Here are some feature engineering ideas for the dataset:

1. **Date-Based Features:**
    - Extract year, month, and day from the 'Date' column.
    - Create binary features for weekends or weekdays.
2. **Seasonality:**
    - Use one-hot encoding for the 'Season' column.
3. **Permit Type and Gender Encoding:**
    - Perform one-hot encoding for 'Permit Type' and 'Gender' columns.
4. **Frequency-Based Features:**
    - Calculate the count of permits issued for each 'Permit Type' or 'Gender.'
    - Calculate the cumulative count of permits issued up to a certain date.
5. **Lag Features:**
    - Create lag features to capture historical trends. For example, you can calculate the permit count for the same month in the previous year.

Model Accuracy

To make the machine learning model as accurate as possible, consider the following:

1. **Data Preprocessing:**
    - Handle missing values appropriately, either by imputing or removing them.
    - Scale or normalize numerical features if necessary.
    - Encode categorical features using one-hot encoding or label encoding.
2. **Feature Selection:**
    - Use feature selection techniques to identify the most relevant features for your model. Features with low importance can be removed to simplify the model.
3. **Cross-Validation:**
    - Implement cross-validation to assess the model's performance more reliably. Techniques like k-fold cross-validation can help estimate model accuracy.
4. **Hyperparameter Tuning:**
    - Experiment with different machine learning algorithms and hyperparameters to find the best-performing model. Consider techniques like grid search or random search.
5. **Time Series Considerations:**
    - If your dataset has a time-series component, consider using time series models like ARIMA or Prophet for forecasting.
6. **Evaluation Metrics:**
    - Choose appropriate evaluation metrics based on the problem type (classification or regression) and business objectives. Common metrics include accuracy, precision, recall, F1-score, or RMSE.
7. **Feature Importance Analysis:**
    - Analyze feature importance to understand which features have the most impact on predictions.
8. **Regularization:**
    - Apply regularization techniques like L1 or L2 regularization to prevent overfitting.
9. **Domain Knowledge:**
    - Leverage domain knowledge to engineer domain-specific features and make informed decisions during preprocessing and modeling.

Feature Extraction

Feature extraction it provides meaningful and relevant information from the raw data. In the context of Nusuk dataset, here's how the extracted frequency-based features can be relevant to improving the machine learning model:

1. Count of Permits by Permit Type or Gender:
    - Relevance: Calculating the count of permits issued for each 'Permit Type' or 'Gender' allows your model to capture the distribution and prevalence of different types of permits or genders in your data. It provides insight into which categories are more or less common.
    - Use Case: This information can be valuable for models that need to distinguish between different types of permits or genders. For example, since our target is Permit Classification, knowing the frequency of each permit type can help your model understand which types are more likely to occur in certain situations.

2. Cumulative Count of Permits by Date:
    - Relevance: The cumulative count of permits issued up to a certain date reflects the historical trend of permit issuance. It can capture seasonality, trends, or any underlying patterns in the data.
    - Use Case: For time series forecasting or predictive modeling, this feature can be extremely valuable. It helps the model understand how the number of permits has evolved over time and can be used to predict future trends. For example, if we're going to forecast future permit counts, the cumulative count feature can serve as a predictor, allowing the model to capture the historical context.

Overall, these frequency-based features provide additional context and information of the machine learning model. They help the model identify patterns, trends, and relationships within the data, which can lead to more accurate predictions or classifications. Including these features in your model can improve its ability to generalize from historical data to make predictions on unseen data points.

Feature extraction (Lag features)

Lag features captures historical trends, are relevant in improving machine learning models in several ways:

1. **Seasonal Patterns**: Lag features can capture recurring seasonal patterns. For example, in a time series dataset, if there's a surge in permit counts during specific months each year (Ramadhan/ Eid Holiday), lag features can help the model account for this seasonality.
2. **Autocorrelation**: Lag features allow the model to capture autocorrelation within the data. If the number of permits issued in one month is related to the number issued in the previous month, including lag features can help the model account for this relationship.
3. **Improved Predictive Power**: By providing historical context, lag features can enhance the model's predictive power. For instance, knowing the permit count in the previous month could prove useful in predicting the count for the same month.
4. **Time-Dependent Relationships**: Lag features can help the model understand how past values of a variable impact its current value. This is particularly useful when dealing with time-dependent datasets.

By incorporating these lag features into the model, it can capture and leverage the historical patterns and relationships in your data, potentially leading to more accurate predictions and a better understanding of how past data influences the present.

Feature extraction (Means)

Mean of gender count, monthly count and permit count.

1. **Trend Analysis**: By calculating and utilizing the mean values, we can easily identify trends over time. If the monthly counts consistently deviate from the mean, it could indicate a significant change in user behavior, permit issuances, or other factors. Detecting trends early allows for timely adjustments and better decision-making.
2. **Anomaly Detection**: Mean values act as a reference point. When you calculate the mean for each column, you create a baseline for what is considered "normal." Any values significantly above or below the mean can be flagged as anomalies. This can help in identifying unusual patterns or outliers that might require investigation.

3. **Comparative Analysis:** Comparing actual values to the mean allows you to assess performance against historical averages. For instance, if 'Monthly Counts' are consistently above the mean, it might indicate growth or increased demand. Conversely, if counts are consistently below the mean, it might suggest a decline in activity.
4. **Data-Driven Insights:** Mean values provide a simple yet effective summary of the data. A mean-based summary can help in conveying insights in a more digestible manner.
5. **Performance Metrics:** Means can serve as performance metrics. For instance, if you're managing a facility or service, comparing 'Monthly Counts' to the mean can help gauge how well you're meeting demand or expectations. It provides a quick and understandable measure of performance.
6. **Quality Control:** Means can be used in quality control processes. If the 'Monthly Count' for a particular month falls far below the mean, it might trigger a review of data collection processes to ensure accuracy.

In essence, calculating and using the mean of these columns offers a straightforward yet powerful tool for data analysis, trend identification. It helps in making data-driven decision.

Overfitting possibilities - seasonality

Addressing the issue of possibility of overfitting due to seasonality:
When using one-hot encoding to convert categorical variables like seasons into binary columns, it's common to observe correlations among these binary columns. This is because the presence of a '1' in one column implies the absence (a '0') in all the other columns. In the model's case, the correlation arises because a specific month can only belong to one season.

Regarding the concern of overfitting, it's essential to consider the context and the goals of the analysis or model. Correlations between one-hot encoded categorical variables are not necessarily a sign of overfitting. They reflect the nature of the categorical data. In most machine learning models, multicollinearity (correlation between predictors) isn't a problem. However, it can affect the interpretability of the model coefficients.

Here are a few considerations:

In summary, whether to keep seasons as separate one-hot encoded columns or as a single column depends on the modeling goals, the specific algorithm used, and the trade-offs willing to make regarding interpretability and model complexity. Both approaches are valid and have their use cases. But in the context of classifying each permit it can prove useful.

Deep learning framework

1. **Data Splitting**:
    - Splitting dataset into training, validation, and test sets. Splits used are 70-80% for training 10-15% for validation, and 10-15% for testing.
2. **Data Scaling**:
    - Normalize or standardize your numeric features. Deep learning models often perform better when the input features have similar scales. Scaling techniques used is z-score scaling.
3. **Model Architecture**:
    - The model used is FNN (Feed Forward Network) to address the permit classification problem will be the most relevant.
    - Defining the number of layers, the number of neurons in each layer related to the number of features, activation functions, and any regularization techniques (e.g., dropout, L2 regularization) is also necessary for the model architecture.
4. **Model Compilation**:
    - Compile the model by specifying the loss function, optimizer, and evaluation metrics. The choice of these depends the problem which is classification.
5. **Model Training**:
    - Training the deep learning model using the training data. Monitoring the model's performance on the validation set to detect overfitting. Model may need to be adjusted using hyperparameters like learning rate or batch size to optimize training.
6. **Deployment**:
    - In the future, I plan to deploy he model, prepare it for production for next batch of data. This might involve converting it to a different format (e.g., TensorFlow Serving, ONNX), building APIs.
7. **Documentation and Reporting**:
    - Documenting entire process, data preprocessing steps, model architecture, hyperparameters, and evaluation results. Clear documentation is essential for future reference and collaboration.
8. **Iterate and Refine**:
    - Finally, Iteration and refining of the model based on the success of deployment.

Data Splitting:

Data splitting is a crucial step in deep learning model development.

Given the dataset, a combination of random split and time series split might be appropriate:

- Randomly splitting the dataset into training (70%) and testing (30%) sets to ensure model training and evaluation.

Feature Scaling Objective:

- The primary goal of feature scaling is to ensure that all features contribute equally to model training.
- It helps prevent features with larger scales from dominating features with smaller scales.

Scaling Technique

Standardization (Z-Score Scaling):

- In standardization, each feature is transformed to have a mean (average) of 0 and a standard deviation of 1.
- Formula: `z = (x - mean) / standard_deviation`
- Result: The transformed data has a mean of 0 and a standard deviation of 1.
- When to use: Standardization is preferred when the data distribution does not have a specific range or when there are potential outliers. It's commonly used for deep learning.
-

Splitting the data purpose

1. **Training Dataset (X_train, y_train)**:
    - **Purpose**: This dataset is used to train the machine learning model. The model learns patterns, relationships, and features in the data from this dataset.
    - **Usage**: During training, the model adjusts its internal parameters (weights and biases in the case of neural networks) using optimization algorithms to minimize the difference between its predictions and the actual target values (labels) in this dataset.
2. **Validation Dataset (X_val, y_val)**:
    - **Purpose**: This dataset is used during training to evaluate the model's performance and tune hyperparameters. It helps prevent overfitting by providing an independent set of data that the model hasn't seen during training.
    - **Usage**: The model's performance on the validation dataset is monitored during training. This performance is used to make decisions such as stopping training early if performance starts degrading (indicating overfitting) or selecting the best hyperparameters.

3. **Test Dataset (X_test, y_test)**:
    - **Purpose**: This dataset is completely unseen by the model during training and hyperparameter tuning. It serves as a final evaluation of the model's performance.
    - **Usage**: After the model is trained and validated, it is evaluated on the test dataset to assess how well it generalizes to new, unseen data. This evaluation provides an estimate of how well the model is expected to perform in a real-world scenario.

The key idea is to split the data into these three sets to ensure that the model is not only learning the training data but also generalizing well to new, unseen data. The training set is used to teach the model, the validation set is used to fine-tune its hyperparameters and monitor its performance, and the test set is used to provide a final, unbiased assessment of its performance.

For neural networks and deep learning models, this process remains the same, with the additional step of scaling or preprocessing the data and defining the neural network architecture before training.

Model architecture:

1. **Feedforward Neural Network (FNN):** An FNN is a specific type of ANN where the data flows in one direction, from the input layer to the output layer, without any loops or cycles. It's the most common and straightforward type of neural network. In an FNN, each neuron in a layer is connected to every neuron in the subsequent layer, and there are no feedback loops. This architecture is suitable for tasks like regression and classification.

Model evaluation metrics: #1

```
9/9 [==============================] - 0s 1ms/step
Accuracy: 1.0
Precision: 1.0
Recall (Sensitivity): 1.0
F1-Score: 1.0
```

An accuracy of 1.0, precision of 1.0, and recall of 1.0 indicate that the model is performing perfectly on the test data. This means that it's making correct predictions for all classes in your multiclass classification problem.

Overfitting

Although the training of the data may suggest no overfitting due to performance of the validation and test data set. Both the test and validation of the data set accuracies are 100%, it suggests that your model is not overfitting because it's able to generalize perfectly to unseen data as well. Using an FNN deep learning model architecture in relatively small datasets of 26 columns and 1719 rows. This raises a concern of overfitting. To address the issue of overfitting, applying **Regularization techniques to penalize large weights in the model**.

Addressing overfitting (Regularization):

1. **L1 and L2 Regularization (Weight Decay):** These techniques add a penalty term to the loss function, encouraging the model's weights to stay small. L1 regularization encourages sparsity in weights (some weights become exactly zero), while L2 regularization encourages small weights.
2. **Data Augmentation:** Having limited data, data augmentation techniques like random rotations, flips, and translations can artificially increase the dataset size and help the model generalize better.

experimenting with these techniques individually or in combination to find the best regularization strategy for the specific problem.

Regularized model

The training and validation results for the regularized model look promising. Here's a summary of what we can observe:

1. **Accuracy**: The training accuracy reaches 100%, which could indicate that the model has learned the training data almost perfectly. However, the more important metric is the validation accuracy, which is also very high and reaches 100%. This suggests that the model generalizes well to unseen data, which is a positive sign.
2. **Loss**: The training loss steadily decreases over the epochs, which indicates that the model is learning and improving. The validation loss follows a similar pattern, suggesting that the model is not overfitting the training data.
3. **Accuracy and Loss Stability**: The accuracy and loss curves appear stable, without large fluctuations, which is a good sign of a well-behaved model during training.

Based on these observations, it seems that the regularized model is performing well and has likely addressed overfitting. However, it's important to keep in mind that achieving 100% accuracy on both the training and validation sets might also suggest that the problem might be relatively easy for the model.

Comparing the results of regularized and non-regularized:

Non-Regularized Model:

- Test Loss: 2.3879416403360665e-05
- Test Accuracy: 1.0

For the non-regularized model:

- The test loss is extremely low, close to zero. This suggests that the model is making very accurate predictions on the test data, as the loss measures how well the predicted probabilities match the true labels.
- The test accuracy is 1.0, which means that the non-regularized model is achieving 100% accuracy on the test data. This is a perfect score, indicating that the model is correctly classifying all the samples in the test set.

Regularized Model:

- Test Loss (Regularized): 0.05216742679476738
- Test Accuracy (Regularized): 1.0

For the regularized model:

- The test loss is slightly higher compared to the non-regularized model but still relatively low. This indicates that the regularized model is also making accurate predictions, though the loss is higher than the non-regularized model.
- The test accuracy is again 1.0, which means that the regularized model is achieving 100% accuracy on the test data, just like the non-regularized model.

Fine-tuned model.

- Activation Function: ReLU
- Batch Size: 32
- L2 Regularization Strength: 0.01
- Learning Rate: 0.01
- Number of Hidden Units: 128

- Best Accuracy: 99.76%

The best-tuned model, after hyperparameter tuning and addressing overfitting, achieved a slightly lower accuracy but still very close to 100%.