Traffic Severity Prediction Using Multilayer Perceptron (MLP)

Sulthan Syuza Prabowo - 2502020075

Rendy Gunawan - 2502024666

Problem description

Traffic is the movement of vehicles, pedestrians, cyclists, and other road users on a network of roads. Traffic can be affected by various factors, such as weather, road conditions, accidents, construction, events, and demand. These factors can impact traffic in different ways, such as causing delays, congestion, diversions, closures, or collisions.

Traffic can also impact the driving experience of road users, depending on their expectations, preferences, and goals. For example, some drivers may prefer to avoid traffic and take alternative routes, while others may not mind traffic and enjoy the scenery or listen to music. Some drivers may have a tight schedule and need to reach their destination on time, while others may have more flexibility and can adjust their plans accordingly. Some drivers may find traffic stressful and frustrating, while others may find it relaxing and calming.

Therefore, traffic is a complex phenomenon that involves multiple factors and perspectives. Understanding how traffic works and how it affects the driving experience can help road users make better decisions and improve their safety and satisfaction.

Solution features

Pandas is a Python library that provides data structures and tools for data analysis and manipulation. It allows you to work with tabular data, such as CSV files, Excel spreadsheets, or SQL databases, and perform operations such as filtering, grouping, aggregating, merging, or

reshaping. Pandas also supports various types of data, such as numerical, categorical, datetime, or text, and handles missing or invalid values.

Sklearn is a Python library that provides a range of machine learning algorithms and utilities for data preprocessing, model selection, evaluation, and deployment. Some of the functions in sklearn are:

- train_test_split: This function splits a dataset into training and testing subsets, optionally with a given ratio or size. This is useful for evaluating the performance of a machine learning model on unseen data.

- LabelEncoder: This function encodes categorical labels into numerical values, such as 0, 1, 2, etc. This is useful for converting text labels into a format that can be used by machine learning algorithms.

- OneHotEncoder: This function encodes categorical features into binary vectors, where each possible value is represented by a 0 or 1 in a separate column. This is useful for creating dummy variables or avoiding the ordinality problem of label encoding.

Keras is a Python library that provides a high-level interface for building and training neural networks. It supports various types of layers, optimizers, activation functions, loss functions, and metrics. Some of the functions in keras are:

- to_categorical: This function converts a class vector (integers) to a binary class matrix (one-hot encoding). This is useful for preparing the target variable for a classification problem.

- Sequential: This function creates a linear stack of layers that can be added one by one. This is useful for creating simple neural network architectures without complex connections or branches.

- Dense: This function creates a fully connected layer that takes an input and produces an output by applying a linear transformation and an activation function. This is useful for creating the basic building blocks of a neural network.

- EarlyStopping: This function creates a callback that stops the training process when a given metric (such as validation loss or accuracy) stops improving for a number of epochs. This is useful for preventing overfitting or saving computational resources.

- load_model: This function loads a saved model from a file or a directory. This is useful for reusing or deploying a trained model.

Matplotlib is a Python library that provides visualization tools for creating various types of plots and charts. It supports different styles, formats, and interactive features. One of the modules in matplotlib is:

- plt: This module provides a simple interface for plotting data using matplotlib's default settings and conventions. It allows you to create figures, axes, labels, legends, titles, and annotations with minimal code.

Solution design architecture

The model architecture that is used for this project is the Multilayer Perceptron architecture, a deep learning model.

A Multilayer Perceptron (MLP) is a type of artificial neural network that consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. Each node in one layer connects with a certain weight to every node in the following layer.

MLPs are universal function approximators. They can capture and represent complex patterns and relationships in the data, which makes them suitable for many prediction tasks.

The MLP can learn from the historical traffic data and make predictions about future traffic patterns. The reason why MLP is suitable for this task is that traffic patterns can be complex and non-linear, which are the types of patterns that MLPs are good at capturing.

Experiments

The first part of the code defines an `EarlyStopping` callback. This is a way to stop the training process early if the model isn't improving, which can save a lot of time. The `monitor` parameter is set to `'val_loss'`, which means that the model's performance is evaluated based on its loss on the validation set. The `patience` parameter is set to `3`, which means that the training will stop if the validation loss doesn't improve for 3 epochs. The `mode` parameter is set to `'min'`, which means that training will stop when the monitored quantity has stopped decreasing.

The second part of the code trains the model using the `fit` method. The `X_train` and `y_train` parameters are the features and target variable for the training data. The `epochs` parameter is set to `1000`, which means that the model will go through the training data up to 1000 times. The `batch_size` parameter is set to `32`, which means that the model will be updated after every 32 samples. The `callbacks` parameter is a list that includes the `early_stopping` callback we defined earlier. The `validation_split` parameter is set to `0.2`, which means that 20% of the training data will be set aside and used as a validation set.

```
# Define the early stopping criteria
early_stopping = EarlyStopping(monitor='val_loss', patience=3, mode='min')

# Train the model
history = model.fit(X_train, y_train, epochs=1000, batch_size=32, callbacks=[early_stopping], validation_split=0.2)
```

The `model.evaluate()` function is a method in Keras used to evaluate the performance of a trained model. This function returns the loss value and metrics values for the model in test mode. In this case, the model is evaluated on the `X_test` and `y_test` data, which are the features and target variable for the test data, respectively. The function returns two values: `loss` and `accuracy`.

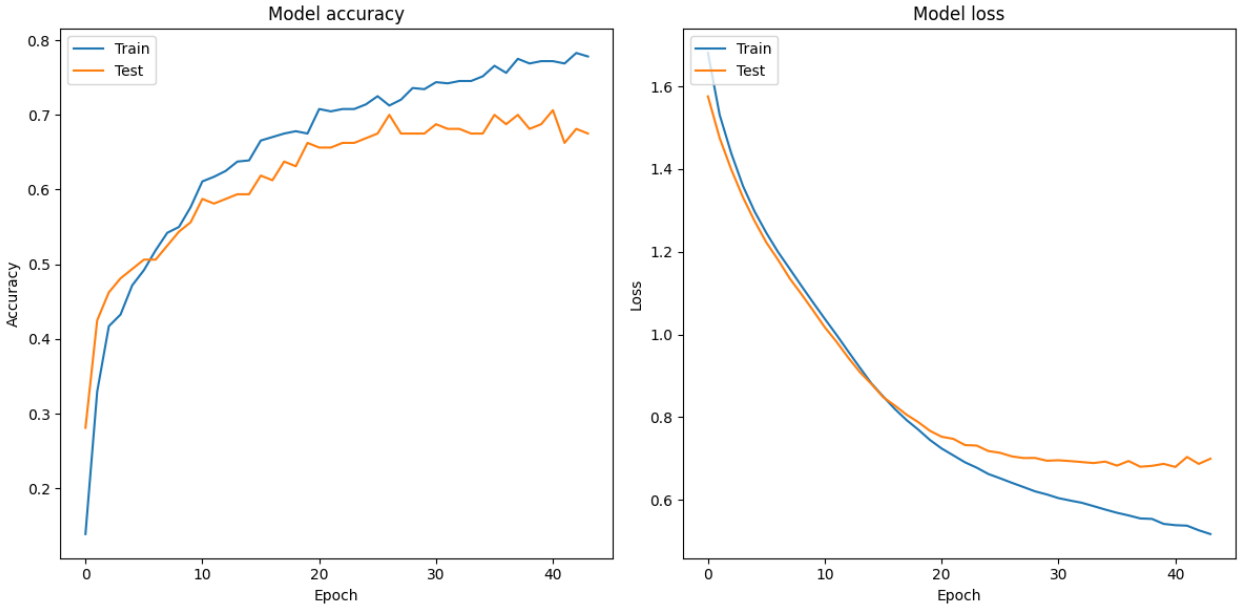
The `loss` is a number indicating how well the model's predictions match the true values. Lower loss values are better, with zero being the best possible loss.

The `accuracy` is a percentage indicating the proportion of correct predictions made by the model out of all predictions. Higher accuracy values are better, with 100% being the best possible accuracy.

The `print()` function is then used to display the accuracy of the model on the test data. The `f-string` (formatted string literal) is a way to embed expressions inside string literals, using curly braces `{}`. The expressions will be replaced with their values when the string is printed. In this case, the expression is `accuracy * 100`, which calculates the accuracy as a percentage. The resulting string will be something like `Test Accuracy: 95%`, assuming an accuracy of 0.95.

```python
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy * 100}%')
```

```
Epoch 44/1000
20/20 [==============================] - 0s 2ms/step - loss: 0.5175 - accuracy: 0.7781 - val_loss: 0.6997 - val_accuracy: 0.6750
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
c:\Users\Lynn\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\engine\training.py:3103: UserWarning: You are sav
  saving_api.save_model(
7/7 [==============================] - 0s 1ms/step - loss: 0.6121 - accuracy: 0.7500
Test Accuracy: 75.0%
```



Program manual

Upon executing the cell for predicting the traffic, the user is asked to input values for the model to predict.

```python
# User input
roadblock = input("Enter roadblock: ")
time = input("Enter time: ")
weather = input("Enter weather: ")
infrastructure = input("Enter infrastructure: ")
type = input("Enter type: ")
volume = input("Enter volume: ")
```

The various factors that can affect traffic severity can be found on traffic_dataset_generator.py.

```python
# Define the categories and their corresponding points
incidents = {"none": 0, "minor-accident": 1, "major-accident": 2, "partial-roadwork": 1, "full-roadwork": 2, "partial-blockage": 1, "full
time = {"early-morning": 1, "morning": 2, "afternoon": 1, "evening": 2, "night": 1}
weather = {"clear": 0, "foggy": 1, "light-rain": 1, "heavy": 2, "snow/storm": 3}
road_infrastructure = {"good": 0, "average": 1, "bad": 2}
types_of_roads = {"freeway": 0, "highways": 1, "arterial": 1, "collector": 2, "local": 2}
traffic_volume = {"0-249": 0, "250-499": 1, "500-749": 2, "750-1000": 3, "1000+": 4}
```

Once the user has inputted all the values, the program will output the predicted severity.

```
1/1 [==============================] - 0s 41ms/step
The predicted severity is: very-low
```

Github link and video link

Github: https://github.com/SulthanTriesToCode/Traffic-Predict

Video: https://drive.google.com/file/d/1VwRN_zINqV-2SEXMaAIxWHXRvT5CZE1U/view?usp=sharing