Brandon Polk
CSC 578 Final
Kaggle Documentation
Kaggle User: Brandon Polk

Introduction

The goal of this notebook is to develop a recurrent neural network that will forecast traffic volume on an interstate in Minneapolis/St.Paul. The data is an hourly time series recorded by John Hogue Social Data Science and General Mills. The recorded data contains the westbound traffic volume on interstate 94 between Minneapolis and St.Paul along with 8 other variables (holiday, temperature, amount of rain, amount of snow, percentage of cloud cover, general description of weather, a detailed description of weather, and the datetime hour when the sample was collected. There are a total of 48,204 samples collected from 2013-2018.

Data Exploration / Pre-processing

The first view of the data distribution is a 5-number summary that includes count, frequency, unique value information about the categorical variables (holiday, weather_main, and weather_description). There are a few things to note in the data that should be addressed based on the summary. First, temperature is measured in kelvins, so a temp of 0 is absolute zero. Absolute zero is not a possible temperature in this context, so any sample with this value is likely missing data and must be dealt with appropriately. To address this, all zero temperatures were replaced with the mean temperature recorded at that hour of all samples in the data with a recorded temperature above 0. Second, there is a severe outlier in rain_1h. It has a max value of 9831.3 mm which would be an excessive amount of rainfall in a single hour, far beyond what would be considered possible. Again, this is replaced with the mean amount of rainfall recorded at that hour in the rest of the data. Lastly, there are several categorical variables that may need to be transformed into numerical representations to be used in the predictive RNN model. For the holiday variable all non-holidays are encoded as 0 and all holidays are encoded as 1. The variable is generalized this way because individual holidays are infrequent events (once a year), even with several years of data, there are likely too few occurrences for the model to adequately pick up patterns specific to an individual holiday. The weather_main and weather_description are redundant variables. They convey the same information but with different levels of detail, weather_main being the more general of the two. The redundancy suggests that only one of these variables should be kept within the data, but which to keep is an important question. Given that these variables need to be converted into numerical representations using dummy variables, to avoid issues of a sparse data set, fewer features maybe preferred at the expense of deeper model learning. However, given that we have many data samples relative to the number of variables available, sparsity issues should be limited. Additionally, a model with enough depth should be more likely to pick many of these details in the data. To resolve this, 3 variations of the dataset were evaluated. One dataset containing all dummy variables for weather_main and weather_description, one containing dummy variables for only weather_main, and one containing only dummy variables for weather_description. The baseline model will be run on each of the datasets and the best performing dataset will be used in model optimization.

Similarly to the tutorial, the datetime for each sample will be converted into signals representing the time of day and the time of year. This is done so that the dates can be used by the model, giving it the chance to pick up cyclical patterns in the data related to days (time of day) and years (time of year). The datetimes were converted using sin and cos functions to express the time of day or year as a cyclical period. The resulting sin and cos signals for day and year were added to the dataset as new variables.

The final part of pre-processing involves splitting the data into training, validation, and testing data and normalizing the values so that they are similarly scaled. The last 5000 records are used as test data, and the rest of the data is split 80-20 training-validation. The data is then normalized to values between 0 and 1 using min-max normalization. The tutorial originally used z-score normalization to address this issue, however, we are using different data here and that form of normalization works best on data that largely follows a gaussian distribution, which this data decidedly does not. The traffic data contains many categorical variables encoded by 0 or 1 and are not evenly distributed. As a result, z-normalization may still produce a difference in scale between variables that we do not want. Therefore min-max normalization was chosen. To note, normalization was not applied to the time of day/year variables to maintain their integrity and because they already exist at an acceptable scale. Also, data was only normalized using the values of the training data. This is done to ensure that the model does not inadvertently pick up information about the validation and test data during training.

Time Series Implementation

The WindowGenerator() class from the timeseries tutorial was used to process the data for use in the RNN with a few modifications. Some of the variables in this class and its methods were hardcoded for use in the tutorial. These areas were generalized for use in this notebook. Additionally, the make_dataset() method was modified so that it no longer uses the 'shuffle' parameter of the keras.preprocessing timeseries_dataset_from-array() method and uses a batch size of 24. A batch size of 24 was used to better match the daily cyclical nature of the data. Through a few trials, this was demonstrated to be more likely to produce better model performance.

The convenience function, compile_and_fit(), from the tutorial was also used in this implementation with some modifications. The patience for early stopping was increased to 3 and the maximum number of epochs was increased to 50. This was done to give the model enough time to train thoroughly without wasting too much time.

Baseline Model

The purpose of this project is to develop and evaluate an RNN model tasked with forecasting traffic volume 2 timesteps (2-hours in this case) into the future with a timeseries dataset. Because the purpose is the optimization of an RNN, the baseline model used to compare results is the simplest RNN model possible consisting of a single RNN layer with 1 unit and a dense layer of 1 unit for output. Model performance will be compared against the results of this baseline model.

As mentioned earlier, the baseline model was used on each of the 3 datasets. The dataset that produced the best result over a few trials was the dataset containing the weather_description variables, but no the weather_main variables. This dataset is what will be used in the optimization going forward.

Optimization Approach

The process of optimization involved evaluating individual hyperparameters one at a time with several (at least 3) different options. The best performing hyperparameters were kept and compiled into a single model. This approach was a good way to see how the model performance changed as a result of different hyperparameters. The primary model hyperparameters evaluated this way was batch size, number of recurrent units, number of stacked recurrent layers, and the use of a 1D convolution layer. Other hyperparameters were evaluated as well such as activation functions, kernel size in convolution layers, kernel initializers, dense layers, and more.

Batch sizes of 12, 24, 32, and 48 were evaluated. A batch size of 24 appeared to produce the best results. A batch size of 12 often resulted in a model learning faster by the end of each epoch but achieved an overall worse performance than a batch size of 24. Batch sizes of 32 and 48 both failed to reduce MAE as well as 24.

One interesting thing of note while running trials on the different models is that generally it seemed as though using a number of units in a layer that corresponds to a 24-hour day cycle produced better results. This observation was applied to both the best model and alternative models.

Alternative Model

The alternative model was an exploration into the observation that using a number of units in a layer that aligns with a 24-hour cycle (12, 24, 36, 48, etc.) tended to produce better performing models. This model was structured to be much deeper than the best model. The alternative model consists of 1 1D convolution layer (24 units and a kernel size of 3), 4 RNN-LSTM layers with and increasing number of units (12, 24, 36, 48), and a dense layer with 1 unit for output with weights initialized at 0. At its best, this model produced a validation MAE of .0443 which is nearly identical to the best performance of the best model. As with all the optimization models, it was trained on batch sizes of 24 for up to 50 epochs.

Best Model

The best model was a simpler implementation of alternative model. The model was trained on a batch size of 24 for up to 50 epochs and consists of a 1D convolution layer (12 units, kernel size 3), 2 RNN-LSTM layers with an increasing number of units (24, 48), and a dense layer with 1 unit for output with weights initialized at 0. This model produced a MAE of .0442. I believe this model performed so much better than my other attempts because it was able to pick up on the cyclical daily traffic patterns and generalize it well given other information about things such as weather. When I submitted the test predictions to Kaggle, this model achieved a score of 304. My goal was to produce a model below 300, but similarly to what happened in assignment 6, my models hit a wall right before hitting my target.

Comments / Reflections

There are a number of interesting aspects of assignment, including the data itself. The data had some significant issues including some outliers, redundant categorical data, and several gaps in recorded data, including a 6-8 month long gap between 2014 and 2015, and missing data, particularly in the snow_1h variable. How one deals with these issues has significant impact on model performance. With the exception of the most extreme outliers, I did not address much of the inconsistency in the data, opting in to leave in the notable gap in the data in hopes that there was still enough data to produce a strong model. There were likely also cases of missing data that I simply could not address. For example, there are years long gaps in the rain and snow variables, but its impossible to know if an individual observation recording a 0 for those values is missing data or if it recorded no rain or snow. I do wonder if a more extensive or different approach to data preparation will have significantly improved model performance.

The tutorial for this assignment was informative but difficult to follow. It was not well organized, particularly the WindowGenerator class object. There were also several variables in the code that were not explained or never even referenced making it tough to decipher what was going without backtracking. The tutorial helped get me through the project but was by far the worst aspect of this assignment. Another general observation I came across trying to understand neural networks, hyperparameters, etc. is that nearly all of the reference materials out there seem to talk about what something is, but much less why it's there or why we need it and almost never when it's most appropriate to use. Knowing what something is or how to implement it is not the same as understanding when to use it and how to apply it to greatest effect.