Predict Bike Sharing Demand with AutoGluon

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

Dear Student,

I am really impressed with the amount of effort you've put into the project. You deserve applaud for your hardwork!

🎉 Finally, Congratulations on completing this project. You are one step closer to finishing your Nanodegree.

Wishing you good luck for all future projects 🎉

Some general suggestions

Use of assertions and Logging:

- Consider using Python assertions for sanity testing - assertions are great for catching bugs. This is especially true of a dynamically type-checked language like Python where a wrong variable type or shape can cause errors at runtime
- Logging is important for long-running applications. Logging done right produces a report that can be analyzed to debug errors and find crucial information. There could be different levels of logging or logging tags that can be used to filter messages most relevant to someone. Messages can be written to the terminal using print() or saved to file, for example using the Logger module. Sometimes it's worthwhile to catch and log exceptions during a long-running operation so that the operation itself is not aborted.

Debugging:

- Check out this guide on debugging in python

Reproducibility:

- Reproducibility is perhaps the biggest issue in machine learning right now. With so many moving parts present in the code (data, hyperparameters, etc) it is imperative that the instructions and code make it easy for anyone to get exactly the same results (just imagine debugging an ML pipeline where the data changes every time and so you cannot get the same result twice).
- Also consider using random seeds to make your data more reproducible.

Optimization and Profiling:

- Monitoring progress and debugging with Tensorboard: This tool can log detailed information about the model, data, hyperparameters, and more. Tensorboard can be used with Pytorch as well.
- Profiling with Pytorch: Pytorch's profiler can be used to break down profiling information by operations (convolution, pooling, batch norm) and identify performance bottlenecks. The performance traces can be viewed in the browser itself. The profiler is a great tool for quickly comparing GPU vs CPU speedups for example.

Loading the Dataset

✓ **Student uses the kaggle cli with the kaggle API token to download and unzip the Bike Sharing Demand dataset into Sagemaker Studio (or local development).**

✅ Kaggle API is used to download the Bike Sharing dataset.

✅ The dataset is unzipped successfully

⚠️ Additional Suggestions:

- Please remember to mask your credentials such as `kaggle_username` and `kaggle_key`. You may replace them with random letters after completing the project. Leaving them exposed will allow anyone to gain full control over your account with your permission.

```
# Fill in your user name and key from creating the kaggle account and API token file
import json
kaggle_username = [redacted]
kaggle_key = [redacted]

# Save API token the kaggle.json file
with open("/root/.kaggle/kaggle.json", "w") as f:
    f.write(json.dumps({"username": kaggle_username, "key": kaggle_key}))
```

✓ **Student uses Panda's `read_csv()` function to load the train/test/and sample submission file into DataFrames. Once loaded, they can view the dataframe in their jupyter notebook.**

✅ Pandas' `read_csv()` function is used to load in the datasets from the corresponding `csv` files

✅ Records from the dataset are sampled and displayed

💡 Suggestion - Sampling the data using `sample()` function is ideally better than just viewing head data using `head()`

Feature Creation and Data Analysis

✓ **Student uses data from one feature column and extract data from it to use in a new feature column.**

Data is extracted from a feature column and used to generate a new feature set for prediction 👍

Note - This process is called feature engineering. Basically, the goal of feature engineering is modify your data such that it is better suited for the task. This may involve modifying a feature's type, combining several features into a new feature or splitting a feature into multiple features that lead to better representation of the data.

💡 Additional Reading -

- You can read more about feature engineering at this blog
- tsfresh library is quite useful in calculating time series characteristics. It also contains built-in functions for feature evaluation in regression/classification tasks.

✓ **Student creates a matplotlib image showing histograms of each feature column in the train dataframe.**

✅ Notebook includes a plot depicting histograms of individual feature columns in the dataset

This step is more generally referred to as EDA (shot for Exploratory Data Analysis). Performing EDA on the dataset helps us develop a complete understanding of the data especially when working on projects where we are not aware of the characteristics of training data. In case of image classification, it's usually a good idea to visualize atleast a small subset of the images.

- 💡 Refer to this blog post for more details on Exploratory Data Analysis

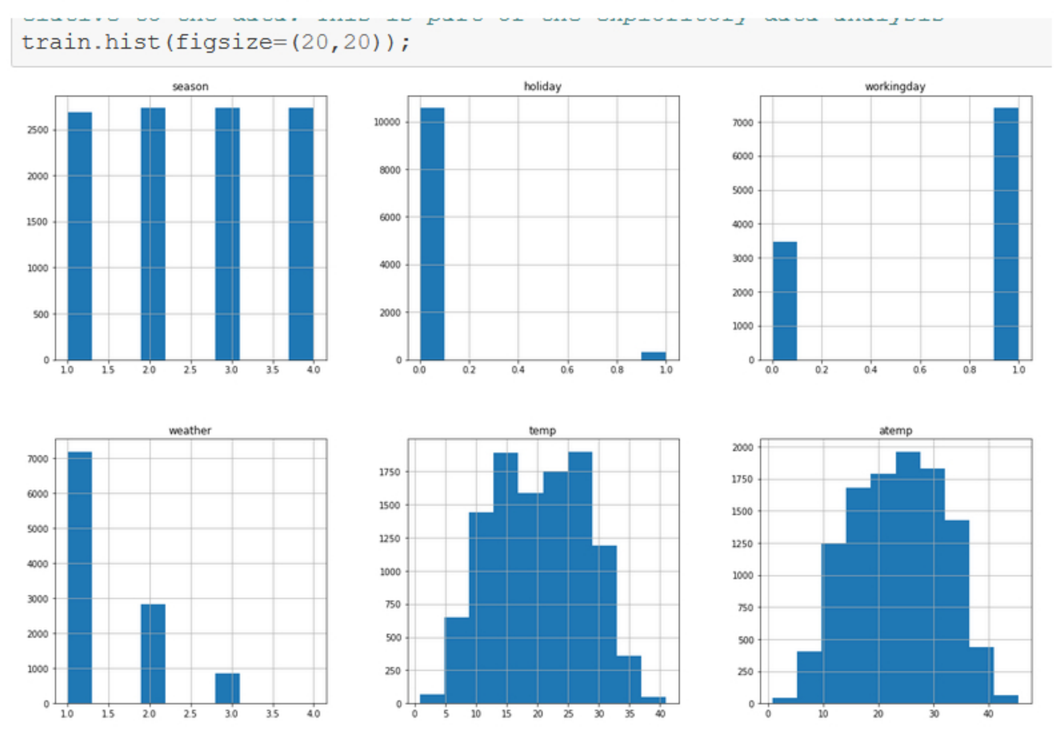
---

📖 Brief notes on Histograms:

- Histograms are best suited for looking at the distribution of numerical variables
- The values on the X-axis in the histogram are numerical
- The histogram's X-axis is a Cartesian coordinate axis along which values cannot be changed

---

💡 Suggestion - You can set the `figsize` attribute to make your histogram plot cleaner



✓ **Student assigns category data types to feature columns that are typed as numeric values.**

✅ Since the original numerical columns actually contained data that was supposed to be categorical, doing the conversion will improve the model's performance.

Model Training With AutoGluon

✓ **Student uses the TabularPredictor class from AutoGluon to create a predictor by calling .fit().**

✅ The `TabularPredictor` has been invoked for creating a predictor.

✅ You've used `root_mean_squared_error` as the evaluation metric.

✅ "casual" and "registered" columns are not used for data fitting.

The `ignored_columns` argument allows you to specify column names that the predictor is not supposed to use for prediction.

- Check out the documentation for more details about `TabularPredictor`

✓ **Student provides additional arguments in the TabularPredictor .fit() function to change how the model uses hyperparameters for training.**

💡 For brief explanation of how it works, please check out this Knowledge post - https://knowledge.udacity.com/questions/736973

There are two crucial arguments that are to be passed to the `fit()` function:

- `time_limit` - Used to specify the number of seconds for which `fit()` should run. If it is not specified then execution will go on until all modes have completed training.
- `presets` - This is used to specify preset configurations for other arguments in fit(), instead of manually specifying each of them.

You can read more about fit function and its arguments in the documentation

✓ **Student uses the predictor created by fitting a model with TabularPredictor to predict new values from the test dataset.**

👍 Good to see that you printed the prediction values in your notebook for the purpose of sanity checking.

📖 Note - By default, Autogluon uses the best model for generating predictions. However, if you would like to generate predictions using a specific model for testing purposes, then you may use the following code snippet

```
predictor.predict(test_data, model='MODEL_NAME')
```

Compare Model Performance

✓ **Student uses the kaggle cli to submit their predictions from the trained AutoGluon Tabular Predictor to Kaggle for a public score submission.**

✅ The predictions are submitted to Kaggle via the cli interface

📖 Note - If you ever face trouble submitting your data via the CLI you can always upload it directly via the Kaggle website.

✓ **Student uses matplotlib or google sheets/excel to chart model performance metrics in a line chart. The appropriate metric will be derived from the either `fit_summary()` or `leaderboard()` of the predictor. Y axis is the metric number and X axis is each model iteration.**

✅ A plot of training scores is included in the notebook.

✓ **Student uses matplotlib or google sheets/excel to chart changes to the competition score. Y axis is the kaggle score and X axis is each model iteration.**

✅ A plot of Kaggle scores is included in the notebook.

Competition Report

✓ **The submitted report makes use of `fit_summary()` or `leaderboard()` to detail the results of the training run and shows that the first entry will be the "best" model.**

Both fit_summary() or leaderboard() function calls amay be used for generating the list of models based on their performance.

✓ **The submitted report discusses how adding additional features and changing hyperparameters led to a direct improvement in the kaggle score.**

✅ Good work discussing how adding features or tweaking hyperparameters affected your model's performance

📖 Note - Hyperparameter tuning is resource and time intensive task. So it's not easy to find the best set of hyperparameters for a model in a limited amount of time. Howeever, if you'd like to experiment further, I'd encourage you to spend additional time testing the performance with various sets of hyperparameters.

✓ **The submitted report contains a table outlining each hyperparameter uses along with the kaggle score received from each iteration.**

The report contains an explanation of why certain changes to a hyperparameter affected the outcome of their score.

👍 Your report contains a table detailing the hyperparameters tuned in the HyperParameter tuning section and a brief summary of the project. Good job!

💡 You can use Tables Generator website for generating data tables in markdown format