Quantitative Data Analysis of Hourly Rental Bikes Count

Part 1

Random Forest

Random forests is an ensemble learning method for classification, regression that operates by constructing a multitude of decision trees at training time.

Why?

1. Can tackle non linear data
2. Ability to deal with missing data
3. Robust to outliers
4. Scalable across large datasets (big data)
5. Flexible, easy-to-use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is a very handy algorithm because the default hyper-parameters (lesser hyper-parameters, straightforward to understand) it uses often produce a good prediction result.
6. Random Forest algorithms are better in overcoming overfitting by reducing the variance of the decision trees.
7. Can be used for both regression and classification tasks.
8. Ability to select and rank features with respect to their importance to the model

---

Mean Absolute Deviation

Train split: 70%
Test split: 30%

MAD Train: 0.60513
MAD Test: 0.9369

Quantitative metrics after feature selection

MAD Test: 0.7832

Code Maintenance

1. Problem statement, acceptance criteria, expectations has to be illustrated in detail
2. Standard code repository (git) has to be maintained
3. Updating repository with new code has to be done after review and approval from peers
4. Standard code practices has to be adhered
5. Updates has to be tried out locally before merging to the main branch

Part 2

Scaling properties for incremental learning

As dataset size increases we can work on few of the hyper-parameters for optimisation:

1. **n_estimators** = number of trees in the forest
2. **max_features** = max number of features considered for splitting a node
3. **max_depth** = max number of levels in each decision tree
4. **min_samples_split** = min number of data points placed in a node before the node is split
5. **min_samples_leaf** = min number of data points allowed in a leaf node
6. **bootstrap** = method for sampling data points (with or without replacement)

Couple of problems that we can run into with increasing size of data and possible solutions:

1. Models need re-training or fine-tuning every time dataset grows —> Automate a ML pipeline in the cloud to fine-tune every time new data comes in
2. Data storage —> Data storage can be tackled with compression, tiering and deduplication; tools such as Hadoop, NoSQL etc can be used
3. Lack of understanding big data —> Workshops, seminars, training programs etc
4. Data integration when comes from multiple sources —> Use data integration tools like MS SQL, Oracle Data Service Integrator, IBM InfoSphere etc
5. Big data tool selection —> Big data consulting
6. Data security —> Data encryption, segregation, big data security tools etc
7. High dimensionality —> Dimension reduction

To handle big data, we have few solutions from research on Random Forests that we can use:

1. Parallel Random Forest: https://arxiv.org/pdf/1810.07748.pdf
2. Mondrian Forests: https://arxiv.org/abs/1406.2673

Technologies that help Random Forest to tackle scaling properties

Apache Spark provides cross validation facility that prevents manual tuning of hyper-parameters which can be a lot of work.

Limits and drawbacks of new approach:

1. *Memory requirements go high*
2. *Longer training time*