Warehouse Storage Optimization

Dhruvil Dave
School Of Engineering And Applied Sciences
Ahmedabad University
Enrollment No.: AU1841003
email: dhruvil.d@ahduni.edu.in

Dhatri Kapuriya
School Of Engineering And Applied Sciences
Ahmedabad University
Enrollment No.: AU1841129
email: dhatri.k@ahduni.edu.in

Harvish Jariwala
School Of Engineering And Applied Sciences
Ahmedabad University
Enrollment No.: AU1841050
email: harvish.j@ahduni.edu.in

Nisarg Thoriya
School Of Engineering And Applied Sciences
Ahmedabad University
Enrollment No.: AU1841142
email: nisarg.t@ahduni.edu.in

***Abstract*—The objective of this project is to Optimise Warehouse Storage problems. We have used Amazon Bin Images Data set in order to apply machine learning algorithms on them and solve the optimisation problem. This project is divided into two phases. Initial phase consists of *Data Collection, Data pre-processing, Creation of Synthetic Data* and *Modelling the Sythetic Data* via certain Machine learning algorithms like Decision Tree, Random Tree Classifier and KNN. The latter phase consists of hyper parameter tuning of models. This was done via Optuna which is an automatic hyperparameter optimization software framework and H2O's AutoML. We have also tried hyperparameter tuning of Random Forest Classifier without using such a library later.**

***Index Terms*—Classification, Data pre-processing, Synthetic data, Decision tree, AutoML, H2O, Optuna, K-Nearest Neighbors Classifier, Random Forest Classifier, XGBoost**

I. Introduction

In any product based corporations the most important thing is warehouse storage optimisation. Optimisation increases efficiency facilitates proper coordination and maintains good control. When Warehouse Space Optimisation, the first thing is finding a suitable location. When choosing a location, planners should know which bin or container is empty and whether it could fit the dimensions of the product.

Most of the mega corporations uses robots to place their products into the right place. But the commands are driven manually. This can be replaced by machine learning techniques and machines/robots can decide themselves the optimal location for product.

A. Problem Statement

As a part of warehouse Storage optimisation, our main problem lies in finding the suitable bin or container for the incoming product and to place in a manner which would possibly accommodate more products in the given area.

"Model is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

T : classify which bin the product will be placed into
E : current state of bins (vacancy measure)
P : Classification accuracy, whether the product is placed in correct bin or not

For our project, we decided to use the Amazon Bin Images Dataset. This is originally a Computer Vision dataset. The Amazon Bin Image Dataset contains over 530,000 images and metadata from bins of a pod in an operating Amazon Fulfillment Center. We used the meta data of this data set and then

In further sections we have explained the Data Acquisition and its preprocessing process, modelling and finally hyper parameter tuning.

II. Literature Survey

Our goal in this project was to collect amazon bin data and provide best possible optimising machine learning algorithms for its storage. Hence, we reviewed certain algorithms and techniques taking place in the current scenario.

A. Optimising Warehouse Operation with Machine Learning on GPU

Calvin Seward and colleagues attempted to solve this problem using a neural network based approach. In it, they defined two problems: picker routing problem and order batching problem

Picker routing problem: If there's a person who wants to get to various points inside the warehouse, our goal is to find an optimal solution or a path for the person which is the most efficient. This is a special case of travelling salesman problem. To solve this, they created the okapi algorithm. This algorithm was linear in terms of number of isles but it was still slow for the next problem which is

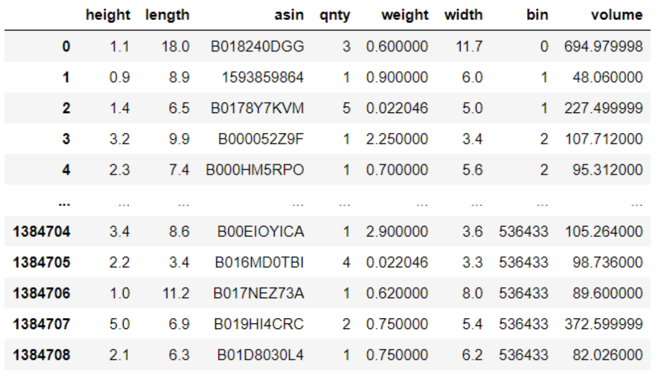
Order Batching problem: This is something just like the picker routing problem, however, in this problem, multiple people are trying to pick up multiple objects at the same time. So naturally, the performance of the algorithm worsens. So much so that it cannot be used anymore. So they tried to develop a neural network based Okapi algorithm and they observed significant performance improvement from seconds to milliseconds. [1]

B. Predicting Warehouse Storage

Another study was done by Michael Li in MIT. He tried to predict how long a shipment would stay in the warehouse using different methods which are listed in front of you. He had a relatively smaller dataset and managed to get decent results. In fact, he predicted he could provide savings of 10 million dollars. Many machine learning models such as Linear Regression, Gradient Boosting and fully connected neural network models were used for this. And acceptable prediction values were obtained in Naive Bayesian, Gradient Boosted Trees, Text Enhanced Linear Regression and 3-layer neural Networks. However, much details of this implementation are not available.

III. Implementation

For implementation of this storage optimisation problem we collected the metadata files of the Amazon bin images. These metadata files contained information about the image of the storage bin in which the products were stored. The most important task was to perform Exploratory Data Analysis. Since we were not going to use any of the deep learning techniques, we decided to drop the images entirely from our dataset and here we were left with almost 530,000 JSON files to be cleaned and preprocessed. We wrote a simple python script to automate the process of parsing the JSON file and updating the database with the fields. After this was done, we had a database that contained JSON blocks corresponding to each image. Now that is clean enough to work with. The data consists of seven columns each explaining the product features. They are height, length, asin(Amazon Standard Identification Number), qnty, weight, width and bin. After cleaning and merging all the metadata files we shaped the data using Pandas [2] and Numpy [3]. Here, we assume that the bin size are

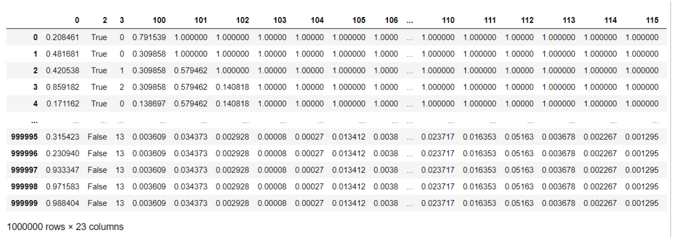


	height	length	asin	qnty	weight	width	bin	volume
0	1.1	18.0	B018240DGG	3	0.600000	11.7	0	694.979998
1	0.9	8.9	1593859864	1	0.900000	6.0	1	48.060000
2	1.4	6.5	B0178Y7KVM	5	0.022046	5.0	1	227.499999
3	3.2	9.9	B000052Z9F	1	2.250000	3.4	2	107.712000
4	2.3	7.4	B000HM5RPO	1	0.700000	5.6	2	95.312000
...	...	...	...	...	...	...	...	...
1384704	3.4	8.6	B00EIOYICA	1	2.900000	3.6	536433	105.264000
1384705	2.2	3.4	B016MD0TBI	4	0.022046	3.3	536433	98.736000
1384706	1.0	11.2	B017NEZ73A	1	0.620000	8.0	536433	89.600000
1384707	5.0	6.9	B019HI4CRC	2	0.750000	5.4	536433	372.599999
1384708	2.1	6.3	B01D8030L4	1	0.750000	6.2	536433	82.026000

Fig. 1. The dataset we obtained from amazon with added volume

equal and the volume column defines the occupied space in the respective bins.

We had a dataset with a single pattern arrangement. Machine learning model can performed more accurately with varied data. Hence, we created a synthetic dataset with the help of CTGAN. This dataset consists of 23 columns. These columns consists of volume of each input product,information if it has been placed or not, bin number in which it has been placed and the vacant space left in each of the 20 bins. We have such 1 millions products i.e. a dataset with dimensions 1000000 x 23.



	0	2	3	100	101	102	103	104	105	106	...	110	111	112	113	114	115
0	0.208461	True	0	0.791539	1.000000	1.000000	1.00000	1.00000	1.000000	1.0000	...	1.000000	1.000000	1.00000	1.000000	1.000000	1.000000
1	0.481681	True	0	0.309858	1.000000	1.000000	1.00000	1.00000	1.000000	1.0000	...	1.000000	1.000000	1.00000	1.000000	1.000000	1.000000
2	0.420538	True	1	0.309858	0.579462	1.000000	1.00000	1.00000	1.000000	1.0000	...	1.000000	1.000000	1.00000	1.000000	1.000000	1.000000
3	0.859182	True	2	0.309858	0.579462	0.140818	1.00000	1.00000	1.000000	1.0000	...	1.000000	1.000000	1.00000	1.000000	1.000000	1.000000
4	0.171162	True	0	0.138697	0.579462	0.140818	1.00000	1.00000	1.000000	1.0000	...	1.000000	1.000000	1.00000	1.000000	1.000000	1.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
999995	0.315423	False	13	0.003609	0.034373	0.002928	0.00008	0.00027	0.013412	0.0038	...	0.023717	0.016353	0.05163	0.003678	0.002267	0.001295
999996	0.230940	False	13	0.003609	0.034373	0.002928	0.00008	0.00027	0.013412	0.0038	...	0.023717	0.016353	0.05163	0.003678	0.002267	0.001295
999997	0.933347	False	13	0.003609	0.034373	0.002928	0.00008	0.00027	0.013412	0.0038	...	0.023717	0.016353	0.05163	0.003678	0.002267	0.001295
999998	0.971583	False	13	0.003609	0.034373	0.002928	0.00008	0.00027	0.013412	0.0038	...	0.023717	0.016353	0.05163	0.003678	0.002267	0.001295
999999	0.988404	False	13	0.003609	0.034373	0.002928	0.00008	0.00027	0.013412	0.0038	...	0.023717	0.016353	0.05163	0.003678	0.002267	0.001295

1000000 rows × 23 columns

We split this dataset into 95% train and 5% test which would give us 9,50,000 samples for train and 50,000 samples for test.

IV. Results

We then trained a few algorithms viz. Decision Tree, Random Forest, K-Nearest neighbours on this synthetic data set using Scikit learn [4]. The scores of these algorithms can be seen in the table below:

Algorithm	**Accuracy**
Decision Tree	82.46%
KNN (n_neighbours = 3)	83.41%
Random Forest (n_estimators = 20)	83.97%
Random Forest (n_estimators = 50)	84.05%
Random Forest (n_estimators = 100)	84.14%
KNN (n_neighbors = 5)	84.23%

After training these algorithms, we tried to implement Support Vector Classifier and obtained the following metrics

Algorithm	**Accuracy**
Linear Support Vector Classified (C=1)	82.69%
Linear Support Vector Classified (C=0.1)	82.82%

A. Optuna

After training these algorithms, we moved on to hyperparameter tuning. Firstly, we tried to explore a hyperparameter tuning library called Optuna and ran two algorithms on it:

1) *Random Forest Classifier*
   We ran 100 trials of Random forest Classifier with the following parameters:

Random Forest Classifier	**Accuracy: 84.643%**
n_estimators	17
max_depth	18.98

2) *XGBoost Classifier*
   XGBoost classifier was relatively slow on our machines, so we used a GPU to speed it up. However, this support was limited for other algorithms

XGBoost Classifier	**Accuracy: 84.876%**
n_estimators	78
max_depth	8
learning_rate	0.2747

B. *Auto-sklearn*

We also tried another framework called auto-sklearn and implemented XGBoost on it but only got an accuracy of 72.06% in it.

C. *H2O.ai AutoML*

AutoML can be used for automating the machine learning workflow, which includes automatic training and tuning of many models within a user-specified time-limit. One of the following stopping strategies (time or number-of-model based) must be specified. When both options are set, then the AutoML run will stop as soon as it hits one of either of these limits.

1) max_runtime_secs
2) max_models

model_id	**auc**
StackedEnsemble_AllModels	0.889618
StackedEnsemble_BestOfFamily	0.849501
XGBoost_2	0.88528
XGBoost_1	0.888208
GBM_4	0.848131
XGBoost_3	0.846808
GBM_3	0.846387
GBM_2	0.844955
GBM_1	0.84195
XRT_1	0.835325
DRF_1	0.824493

D. *Random Forest Classification by Classical Hyper Parameter Tuning*

Finally, we tried to fine tune the hyperparameters of Random Forest classifier without using any frameworks and obtained the following metrics. Firstly, we tried to fine tune the number of estimators for the algorithm and obtained the following plot:

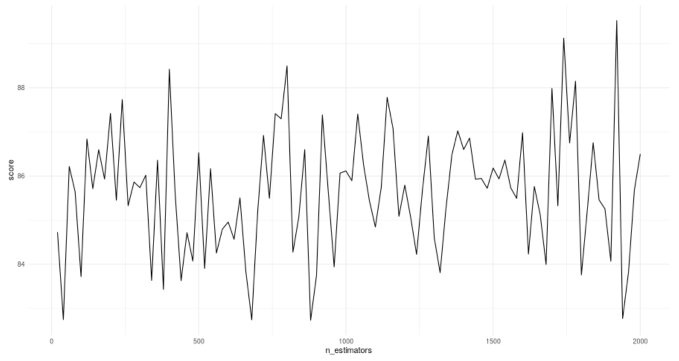


Fig. 2. RFC accuracy score for various estimators values

Then, we tried to fine tune the max_depth of Random Forest Classifier and obtained the following plot:

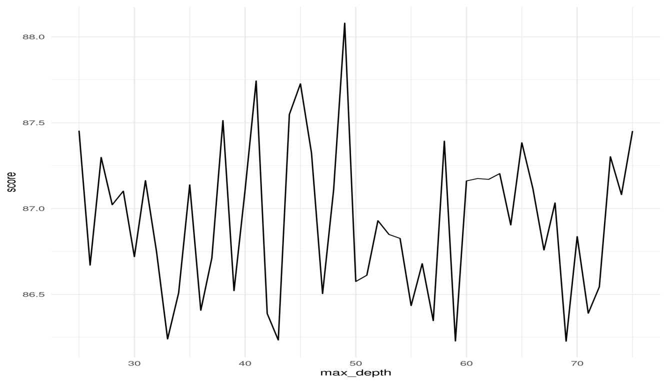


Fig. 3. RFC accuracy score at various depth

The default values for the parameters controlling the size of the trees (e.g. max_depth, min_samples_leaf, etc.) lead to fully grown and unpruned trees which can potentially be very large. The features are always randomly permuted at each split. Therefore, the best found split may vary, even with the same training data, max_features=n_features and bootstrap=False, if the improvement of the criterion is identical for several splits enumerated during the search of the best split. To obtain a deterministic behaviour during fitting, random_state has to be fixed.

E. *Confusion Matrix*

Confusion matrix states the instances when the predicted values and true values conflict or matches with each other. In this matrix, on the y axis we have the bin numbers as true values and on x axis numbers are presented as predicted values. The state when predicted value is equal to true value is on the diagonal and we see that there are quite a many successful placement of products. The first row, there are products which are placed no where. This row shows that when the product should be placed no where but it is placed in other bins. This is point where our model loses accuracy.

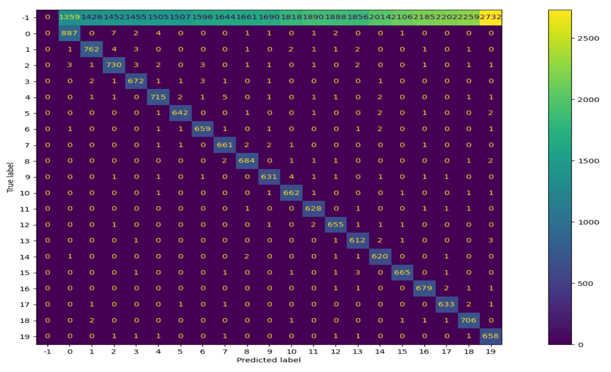


Fig. 4. Confusion Matrix

V. Conclusion

This project was an entirely different approach to the problem of solving the warehouse storage spaces optimization. This field is generally dominated by deep learning methods and classical machine learning methods are completely overlooked.

Through this project we understood the look of using a classical machine learning classifier to approach a complex problem like this with a very high number of classes in output. We tried and learnt various methods of the entire machine learning operations pipeline starting from data fetching, preprocessing, cleaning, feature engineering, modelling, exploratory data analysis, hyperparameter tuning and generating reproducible outputs.

In this process, we got to explore the various new libraries available that can highly speed this processes up on a GPU or a TPU and this learning will be very much helpful in our future endeavours as well. We aim to use this knowledge even further in any machine learning problems we encounter.

References

[1] K. Roodbergen and R. De Koster, "Routing order pickers in a warehouse with a middle aisle," *European Journal of Operational Research*, vol. 133, no. 1, pp. 32–43, 2001.

[2] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[3] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.