Högskolan Kristianstad
291 88 Kristianstad
044 250 30 00
www.hkr.se

DA380A Machine Learning

Project Report:

Greedy Cloud Selection Deployment on Microservices

Group 2

Students: Nedim Kanat, Omar Zarifa

10 / 11 / 2024

Table of contents

Dataset

For this project, we have been assigned the "**Greedy Cloud Selection Deployment on Microservices**" dataset (ID: 007), which consists of 400,000 rows and 11 columns, capturing various parameters essential for deploying microservices-based applications across multiple cloud environments. [1]

Problem

Our aim was to analyze the latency variations across different geographic regions and cloud providers, in a production environment. We chose this problem since latency directly impacts user experience and application performance.

We think that this dataset is valuable to solve this problem since it has a large coverage from multiple giant cloud providers such as AWS, Azure, Google Cloud, IBM Cloud, and various geographic regions. This wide coverage makes it a great fit for understanding how these factors influence latency in actual use.

Approach

Check and handle Missing Data

We began by checking for any missing- and duplicate values for data-reparation, to make sure that we don't input any null- or noisy values into the models, which would break or cause bias. Luckily, we had neither missing nor duplicate data.

Check data types

Thereafter, we observed the data types that were being used in order to decide which features will require encoding, because ML models do not understand how to process data-types other than numbers. We found that the "*microservice name"*, "*cloud provider"*, "*region"* and "*environment"* features were all of type `object` - which meant that we had to encode/transform these fields for the model

we'll use later on. Our choice of encoding was "one-hot encoding" because the values were categorical and had no order-significance.

Filtering for Real-World Accuracy

Since our aim was to focus on the data from production environments, we decided to remove all non-production instances, which was about 66% of the total data. Our reasoning was that the data from controlled environments (test/dev) would cause noise and/or bias for the models and yield incorrect/faulty results, hence it would be better to process real-world data instead.

Checking for Outliers

The data which we cleaned had about 1.2% outliers. We decided to remove these instances since they made up a small portion of the whole dataset and had very little effect on the model performance. We used "matplotlib.pyplot" and "seaborn" to paint the chart shown in figure 1.

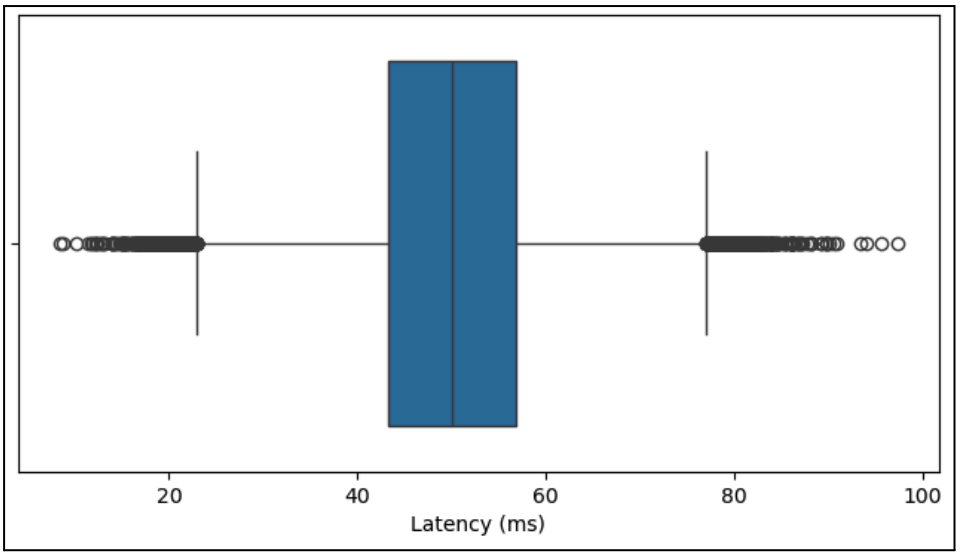

Fig. 1. Showing outliers

Correlation Analysis

After checking for outliers in the data, we observed a correlation matrix in order to get some insight into how features might be related and whether we could get rid of more redundant features to simplify the data. For this step, we used “matplotlib.pyplot”. However, it turned out that there was no linear correlation between the features and the target variable, Latency.

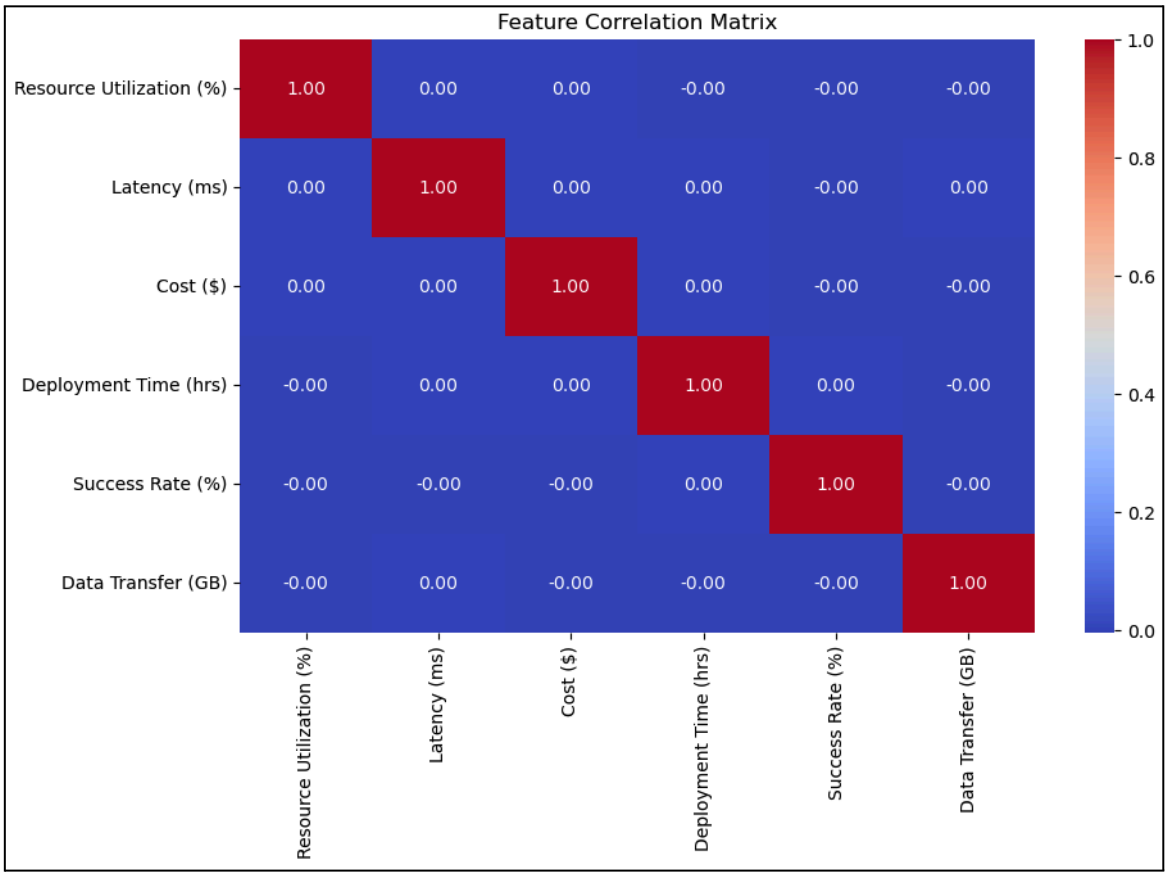

Fig. 2. Correlation matrix

Normalization

In order to avoid the "larger-value" bias where the model assumes larger numerical values to be of higher importance, we normalized certain features: “Latency, Resource Utilization, Cost, Deployment Time, Success Rate, Data Transfer”. To achieve this, we used “sklearn.preprocessing.StandardScaler” to apply z-score normalization.

Check for correlation with DecisionTree

Since we were unable to find any meaningful linear-correlation, we decided to investigate this further by applying some models that are capable of capturing non-linear relationships between features, such as **DecisionTreeRegressor** which yielded *2%* accuracy for training and *-2%* for test data, with 3 cross-validation results of [*-2.9% -3.9% -3%*] and an MSE of *92.6*. These results displayed very poor performance which meant that the algorithm/model failed to capture a meaningful pattern in the data.

We then attempted to use **GridSearchCV** to find better parameters for the **DecisionTreeRegressor**, but that gave about the same results as the un-tuned version. The tuned version yielded *2%* accuracy for training, *-0.3%* for testing with an MSE of *91.03* which is not significant change. For this part we used “sklearn.tree.DecisionTreeRegressor” for the algorithm/model
“sklearn.model_selection.GridSearchCV” for fine-tuning
and “sklearn.model_selection.train_test_split, cross_val_score” to prepare and test the data.

Feature Importance on dataset

According to the tuned **DecisionTreeRegressor** model's feature importance graph, "Data Transfer" is of highest importance. This makes sense because the amount of time it would take for a service to complete a task would depend on how much data it has to process (among other factors, such as how much resources are available to the service)

After that comes cost, which might have predictive power in terms of the quality of service that the provider gives for a certain price. For example, providers usually offer higher-cost-better alternatives aside from lower-cost-limited alternatives where lower-cost-limited ones are prone to being queued, put in systematic-hibernation, or traffic congestion.

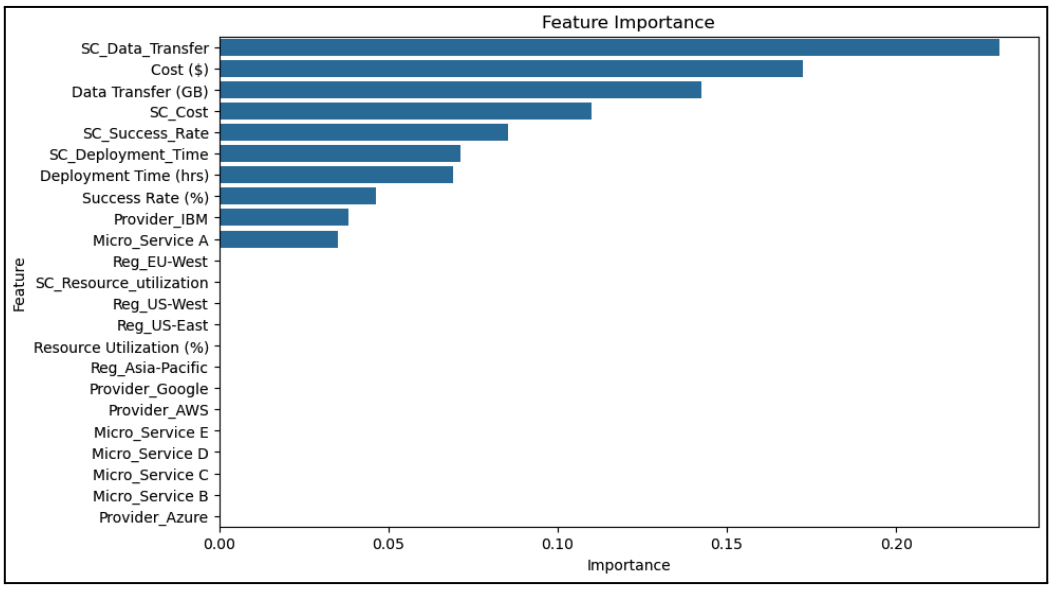

Fig. 3: Feature importance for the dataset

RandomForestRegressor

We tried training a random forest regression model, hoping for better scores. However, this model also failed to generalize well even though the training score was quite high. Test scores turned out to be significantly low. training accuracy: *85%*, test accuracy: *-4%*, and test MSE: *1.016*, which indicates overfitting.

PolynomialFeatures + RandomForestRegressor

As a last option, we combined polynomial features and random forest regressor. However, there were no improvements in the scores. training accuracy: *79%*, test accuracy: *-12%*, and test MSE: *1.16*, which indicates persistent overfitting.

For this part, We made use of the following modules/libs:
“sklearn.preprocessing.PolynomialFeatures”,
“sklearn.ensemble.RandomForestRegressor”,
“sklearn.model_selection.train_test_split`,
”sklearn.metrics.mean_squared_error, r2_score”

Discussion

A key challenge was the lack of correlation with the target variable, in the dataset. This required us to explore nonlinear models but still gave very limited output. We also had difficulties with lowering the overfitting which occurred in our models. We have learned that the real world data will not always carry meaningful patterns, which could be used for solving problems through the usage of models.

Conclusion

This project aimed to analyze latency variations across different geographic regions and cloud providers, which we believed would help us build models that could predict such information. We began by preprocessing the data, cleaning and transforming as needed such as searching for missing values, looking for duplicates and encoding categorical features. It turned out that the correlation analysis showed no significant linear-relationships between the latency and other existing features, which gave us the idea that there might be non-linear, complex relations between features. For that reason, we attempted to capture complex relationships in the data by using DecisionTreeRegressor, DecisionTreeRegressor + GridSearch tuning, RandomForestRegressor and RandomForestRegressor + PolyFeatures. However, the models that we used have failed to generalize well on the test sets unlike the training sets.

The feature-importances gave the idea that "resource utilization", "cost" and "data transfer" might have some predictive power but the models weren't able to make useful predictions which indicates that the dataset is missing other

important data which would be related to latency. For example, network congestion, local link conditions or perhaps retransmission attempts between networks.

We have come to the conclusion that the current dataset is not enough for the models to capture meaningful patterns and make valuable predictions for latency variations between services across various regions. As for possible future work, it would be useful to gather more types of data such as the state of local network/links, bandwidth capacity, and more - which could improve prediction rates for the models mentioned above.

References

1. Kaggle. (2024). **Greedy Cloud Selection Deployment on Microservices**. https://www.kaggle.com/datasets/nickkinyae/greedy-cloud-selection-deployment-on-microservices