Model Selection

Our output is ratio of from views to submission which is numerical variable always less in the range of [0,1] While selecting modeling technique to build the regressor, we need to consider the following aspects:

1. Although what we want to predict here is a number, in essence it's a classification problem where ratio denotes the probability of form being viewed is submitted. Thus, we need a regression technique which will output values between 0 and 1 or classification technique which can take probability as output variable
2. Approximates the training data as closely as possible.
3. Generalizable for unseen data (Avoids overfitting)
4. Has good runtime performance i.e. it takes less time to run inference and occupies small disk size
5. The aim of modeling exercise. If the sole objective of model building is to predict the output where machine learning model act as black box, then we try to fit a model which gives the best possible performance on test data under given cost constraints. However, if the objective of model building is not just predicting the outcome but also interpretability and determining which features have an impact on the output, then we select model which is interpretable and less complex

Considering these we choose Random Forest Regressor (& not Random forest classifier) as technique to predict the output.

1. We cannot use tradition regression techniques such as Linear or SVM because it's not guaranteed that the predicted value will be less than 1. Also, we cannot use classification techniques which require classes as output and not probability of classes. As such we cannot use SVM classifier here because for SVM we will need well defined classes and not probability. Although decision tree or random forest as classification model output class probabilities, we cannot use decision tree or random forest classifier because while training we would require classes and not ratio or probability for node partition. However, Random forest as regression model outputs value which is always in the range of the training data output variable because the leave nodes of the individual trees output value which is mean of the values of target variable in that node. Other option would be logistic regression because the binary cross entropy loss used allows us to use probability values as target variable while training. However, random forest has its own advantages over logistic regression as discussed below.
2. Random Forest is inherently robust to overfitting as it fits a large number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.
3. Random Forest naturally does feature selection as the decision trees partitioning tries to minimize the error/impurity of the node. Nodes with the greatest decrease in impurity happen at the start of the trees, while nodes with the least decrease in impurity occur at the end of trees. Thus, by pruning trees below a particular node, we can create a subset of the most important features. Random Forest has been used in many cases for selecting features. Since the modeling exercise focuses on ranking features based on their importance, we select Random Forest in our case.

4. Random forest also captures non-linear decision boundaries which is not possible with logistic regression, Naive Bayes or Linear Support vector Machines
5. The number of trees can be increased till the model stops overfittings and generalizes pretty well to unseen data. Nowadays, fitting large number of trees is not computationally expensive because of easy accessibility parallel computing. In fact, since random forest can fit trees in a parallel manner, training time required for Random forest is very low on new system with multi core cpus. As a result, Random forest easily scales to dataset having large number of rows or if there are moderately large number of columns.
6. Random forest can handle variables of different scales as it's based on tree partitioning algorithm which is immune to the absolute value of the variable. Only ranking of values matters. Although for the current sample data, every variable lies within the same range, the future data can have variables which have different range of values
7. Random forest also handles outliers well because each tree is built on random subset of data. Correlation between output variable is also well handled because for each split we randomly select only subset of variables.

Hyperparameter Tuning:

It's critical to use optimal values of hyper parameters of the model so that model generalizes to unseen data properly. We use grid search cross validation in sklearn to do hyperparameter tuning. The hyper parameters we consider for tuning are:

1. Number of trees
2. Depth of trees
3. Ratio of number of features to be selected for each node splitting

We just don't look at the error but also the time require for inference as the model is going to handle large number of rows for inference in the production environment. So, if a random forest with 100 trees is giving error slightly more than random forest with 200 trees but the time required for inference is much higher for random forest with 200 trees, we will go with model with 100 trees.

Infrastructure Architecture:

The model needs to scale to handle an increasing amount of API request queries. We have 15000 requests per minute which can even increase to million per minutes as the company grows.

I decide to divide the infrastructure into 5 components such that each component is independent of other and can be scaled depending on the number of API requests We use docker containers for each service and for handling large number of API requests we configure AWS ECS service for the inference component. The components communicate with each other through AWS SNS and AWS SQS.

Model Training component:

The training is done using sklearn package in python and the source code is wrapped in docker container. It allows a better portability, since our data team members are not working with the same version of libraries. Once the model is ready to go into production (when it reaches the minimum accuracy level required), a service will load all the necessary data (in our case it is pkl model file) into S3 databases. We definitely intend to improve our model: for every new version, the service will erase the previous objects in AWS and replace them by the new ones.

Application builder:
once new model objects are loaded, a message is published through AWS SNS and an AWS Lambda function is triggered. It launches the final unit and integration test and eventually deploys our application through Jenkins, an open source automation server.

Flask web server
This holds our flask application. The role of web server is to manage communication from client to the model server. Doing so, it handles the request from a client and publishes the request message to SNS topic which in turn pushes the message to SQS queue which subscribed to it. Model server continuously polls this SQS queue for new input for which prediction has to be made. Flask web applications continuously polls another SQS queue for prediction output from the model server and then sends the output to client. Flask based web server will stay "on hold" until it receives the answer from model server, so we make sure a request is only processed once. Thus we have two SNS topics

1. Input SNS flask web server to publish the input and
2. Output SNS for model server to publish the predicted output.

Corresponding to the two SNS topic we have two SQS queues.

1. Input SQS: The model server continuously polls this SQS queue for new input for which prediction has to be made
2. Output SQS: Flask web server continuously polls one SQS queues for the new predicted output

Model Server:
This where the actual model inference takes place. The inference source code is wrapped inside docker image which is hosted on AWS ECR registry. We use AWS elastic container service (ECS) to scale out/ scale in depending on the number of API requests Here based on the amount of messages in the input SQS queue, ECS automatically spawns AWS ec2 instances which runs the docker containers from inference docker images hosted on AWS ECR. Each container constantly loops on the queue to look for new data to predict by our random forest model and sends the output in JSON format to output SNS topic which in turn pushes the response to output SQS queue. In our case, prediction can take a few seconds (while it only takes a few milliseconds for the producer to send a message to the Queue

The monitoring:
In addition to the amount of requests handled by our API, we have to monitor many metrics within our application to quickly detect unusual values, bugs or any other anomaly. We mainly use ELK stack to build interactive and dynamic dashboards and Grafana to monitor the infrastructure resource usage.