**A Birds Eye View on Sulvo Predictor System**

**Introduction:**

The objective of this document is to introduce Sulvo predictor system to both technical and non-technical readers. The system is deep learning based, able to train the model with real-time data and also high performance, assured to send a response to ad server within 300ms. This document will provide a high level description of the system from an analytical and architectural perspective and along with operational instruction for the end user.

**System Overview:**

The system interacts with only one external actor, and that is Sulvo ad server. The system has three major components – collector, real-time trainer, and predictor. It listens to the programmatic ad selling information for each impression through a collector and sends the predicted floor price for each impression to ad server through predictor. Real-time trainer builds the model on recently collected data and saves it, and predictor uses that model for prediction. Sulvo ad server is hosted in AWS cloud, and real-time trainer and predictor are hosted in Goggle cloud platform. The collector which is hosted in AWS and exposes a REST API to collect data receives information from the ad server and push it to a Redis message queue. On de-queue message from Redis server; the message is pushed to a Google Big Query instance. Real-time trainer which is hosted in a Google Compute Engine fetches the latest information from Big Query and builds a multi-layer neural network model using tensor-flow and save the model in Google Data Store. Predictor server which is a Falcon-based REST API is hosted on Google App Engine, receives prediction requests from Sulvo ad server and response back floor value calculated on basis stored data in the Data Store.Below component diagram will describe the infrastructural component of the whole system..

Compute Engine

Collector Server (AWS )

Google Big Query

Messaging Queue

App Engine

Google Big Data Store

Visitor to site in internet

**Predicting the Floor:**

Sulvo predictor system realizes the floor value as a linear combination of features. Features have two types – user related features and context related features.

User related features are

1. Frequency – number of time user came on that site. (1-5). A frequency greater than five is treated as 5. And it also treated as a categorical feature.
2. Geo – Geographical location of the user with country level granularity
3. OS – Operating System of User's device
4. Device – Device type
5. Client Time – User local time

Context related features are

1. page URL – URL of the page user visited
2. ATF/BTF(visibility) – 1 if user can see the ad slot without scrolling otherwise 0
3. AVT – average view time of that ad unit, numerical feature
4. AVV – Average viewability, numerical feature
5. CTR – Click through rate, numerical feature

We convert categorical feature values to numerical score by the average floor value of successful impressions where that feature value presents. For example, numerical score for country =US is the average floor value of successful impressions where country = US. By successful impression we mean for that floor value, demand side partner (Google in our case) sends an Ad.

After converting all features to a numerical score, we calculate the correlation of floor price with that feature. If the correlation is less than Probable Error, we drop that feature from the list.

Next, we build a three-layer neural network. The input layer is the features and output layer has single node predicted floor value and the intermediate layer has two nodes. All user related features sum up in one node and context related feature is sum up in another node. These two nodes are feed to the output node. For example, if frequency and OS in user feature and page URL and CTR in context features come as important at some time point our neural network is looked like below. (Please keep in mind there is a weight associated with each arrow of below example figure).

We define error function as a sum of the square of the difference between predicted and actual floor value.

We use Adam Optimization algorithm which starts with some random value of the weights of each edge of the network and finds the optimum weights which minimize the error function.

Once we got the final weight, we multiply numerical score by the weights of the edges which connect the node to output node and store it. For example in above graph for OS = Windows we store the numerical score of Windows multiplied by weights of the edges between OS and User and User and Floor.

The real-time trainer does these all. It fetches data from Bigquery and stores the scores for each feature value in Big Table. When predictor gets the request to predict the floor for an impression, it fetches the numerical score for each feature value of that impression and sum them up and get the predicted floor value.

**Feedback Mechanism -1: Success Probability:**

One drawback of above mention system is that it is looking at successful impression only and once it starts to predict low that low floor value is collected by the collector so in next training iteration model will predict less. So we need a feedback mechanism.

We incorporate feedback by calculating the probability of prediction becoming successful or not. We are calculating this probability using the same algorithm used for predicting floor. We just replaced the floor vector by success vector which is 1 for successful impression otherwise 0 and we are converting the categorical value by the probability of an impression will be successful when that feature value is present there. So, for example, numerical score for country = US will be the probability of an impression be successful when impression has feature value country = US.

Now in predictor, it is calculating the probability of success and predicted floor value of an impression. There is a multiplier range- a pair of highest value and lowest value of multiplier (high, low) which is configurable for each domain. According to calculated probability predictor maps the multiplier between high and low by below formula and multiply it with predicted floor value.

multiplier = low + (high –low)\*probability

floor = floor\*multiplier

So if the probability of prediction of the floor for an impression being successful is high (> 0.5), then predictor return higher floor value than the predicted floor value. Similarly, if the probability is low, then predictor return the lower value.

**Feedback Mechanism -2: Dancing Floor:**

At the end of the day, things which matter is the impact of our predictor system on overall revenue. So it is always good to have direct feedback of overall revenue to our predictor system. So we are calculating average floor price and fill rate (success ratio) for each domain in each training iteration.

Now if the average floor price in current training iteration is greater than previous training iteration we do nothing. ("*We do not fix it until it is broken or about to break*")

Now if average floor price is going down then we look at the fill rate.

If fill rate is going up, that means we are under predicting. We make our multiplier range little aggressive, i.e., high = 1.05\*high and low = 1.05\*low.

Similarly, if fill rate is going down then we make our multiplier range little conservative, i.e., high = .95\*high and low = .95\*low

We give a name to this algorithmic strategy as *Dancing Floor.*

**High Value impression -Is Big:**

Some publishers have wide range distribution of floor value, and limited high-value impressions play critical role to their revenue. For example, in gestopolis.com 20% of impression brings 80% of their revenue. So fitting a single model for all impression is a bad idea for this kind of domain. So for this kind of domains, we separate the high-value impressions as a special bucket and build two different models, one for high-value impression and other for general impression. We also build a model which calculates the probability of an impression being high value using the same algorithm we used for calculating probability for prediction being successful by just filling the success column by 1 if an impression is a high value otherwise 0.

Now if predictor receives a request for an impression from big sites first it calculates the probability of impression being high value. If it is found impression seems to be high value, then it calls high value model for prediction otherwise low value.

**Overfitting – Make it simple:**

There is a concept of over-fitting in machine learning which says if we fit a too complicated model to a simple system it will give good accuracy in training data set but perform awfully in real data. For this reason, we also have a simple linear regression model which is nothing but a single layer neural network which may give a better result for some publisher.

**Predict Fast – Predict cheap – Predict in scale:**

Our predictor API is hosted in Google App Engine by default which is auto-scaled. All libraries, we are using are open sourced. Our Ad servers are located in US-East, Australia, and Europe. Our app and storage engine instance will be co-located with them. Our REST API is Falcon-based which is proven most high-performance framework to build Python REST API.

To make the prediction as fast as possible we are exploiting parallel computing in highest granularity level.

Below sequence diagram is illustrating how predictor works for a site with a high-value impression.

Send the response

Get the multiplier range

If big use the big values otherwise use general values

Send the response

Processing the features & get score

Request for is\_big, floor, floor\_big, probability, probability\_big

Request for prediction

Thread for feature N

Thread for feature 2

Google Data Store

Thread for feature 1

Ad Server

Predictor Main thread

1. Ad server sends request for prediction for an impression

2. On receiving request predictor, main thread creates separate thread for every feature

3. Each thread calculates is\_big, floor, floor\_big, probability, probability\_big for each feature by sending the concurrent request to the data store. If feature or value is not present in data store, it returns 0.

4. If is\_big indicates impression is high-value, it uses big scores otherwise general scores.

5. Predicted score = ∑ score for each feature

6. Get multiplier range from data store and calculate final predicted floor

7. The response backs the Ad server the predicted floor value.

**Operational Foot Note – Strategy Manager:**

There are two configuration files in predictor system.

One is multipler.txt. Each row represents one domain and consists of three tab-separated columns – domain, low, high. It specifies the multiplier range for this domain. For example below line indicates domain extraimage.net has multiplier range low = 0.8 and high = 1.2

extraimage.net 0.8 1.2

If a domain is declared as big (having distinct high-value impression ) it is required to add another row specifying multiplier range for the high-value impression.

For example below lines indicate estu.tv has two multiplier ranges -0.8 to 1.2 for general impression and .75 to 1.5 for a high-value impression.

estu.tv 0.8 1.2

estu.tv\_big .75 1.5

Another configuration file is publisher.list. Each row indicates a domain. The first column is the domain name. Second column 1 means it is big has high-value impression otherwise 0. Third column 1 means dancing floor strategy is on for that publisher otherwise 0. Fourth column 1 means model is multilayer and 0 means single layer neural network. If big is one then other columns have two comae separated values one for high-value impression and the second one for general impression.

Domain Big Dancing Multilayer

extraimage.net 0 1 0

estu.tv 1 0,1 1,0

If you want to turn on any publisher, then you have to make a proper entry for that publisher in these two files and to disable, just comment the line for that publisher by “#” sign. Changes in configuration files automatically reflected in the trainer, but for the predictor, you need to deploy in app engine, a new version

**Operational Scenario 1: Checking Jobs in Compute Engine:**

1. Check fetch\_ctr.py script is running or not.
   1. This is a daemon process which fetches AVT, AVV and CTR information of each ad unit from Sulvo Surge database and stores into Google Data Store and sleeps for 30 min and repeats the steps again.
   2. Make sure only one instance of this script should run.
2. Check real\_time\_trainining.py is running or not.
   1. This is a daemon process which fetches data from the Google Big Query, trains the model and stores it in the Google Data Store.
   2. Make sure only one instance of this script should run.

**Operational Scenario 2: Checking predictor web service is proper or not:**

curl -w "@curl-format.txt" -i -H "Content-Type: application/json" -X POST https://**sulvo-east**.appspot.com/predict/**extraimage.net** -d '{"root":"extraimage.net\_728x90\_listings","size":"336,280","device":"d","client\_time":"1482394186433","ad\_position":"0,0","ip":"201.170.89.208", "domain": "extraimage.net","client\_size":"2,3"}'

Above curl command will check predictor service for publisher extraimage.net in US East app engine is working or not. In JSON object, keys are necessary, for the value, you can send any value for any publisher.

For another publisher, you have to replace the extraimage.net in curl URL by publisher domain.

For another app engine, you have to change the project id in curl URL. Our app engines are deployed with three project ids. For US East region it is ‘sulvo-east’, for Europe region it is ‘sulvo-europe’, and for Asia region, it is ‘slvo-asia.'

**Operational Scenario 3: Deploying a new version of predictor to app engine**

1. Make an empty directory in compute engine
2. Download the version from git
3. Run command - gcloud app deploy --prject <prject id> --version <version no>
4. To check the last version number you can see command history in linux

**Operational Scenario 4: Check the time out in prediction**

In computing engine run the below command.

bq query "select count(\*),damus\_status from adlog.day9 group by damus\_status"

adlog.day9 is the table name in Google Big Query. To check the current table name see real time trainer code.

**Operational Scenario 5: Check the average latency of prediction**

bq query "select avg(damus\_latency) from adlog.day8 where damus\_status=1"

**Operational Scenario 6: Check the average floor for prediction**

bq query "select avg(floor) from adlog.day5”

**Operational Scenario 7: See the structure of big query table**

bq show adlog.day9

**Sulvo Predictor with Classification**

**Introduction:**

The objective of this document is to introduce the new version of Sulvo Predictor System. We are requesting the reader to before starting this document; please read the paper “A Birds Eye View on Sulvo Predictor System.” A major change in new version of Sulvo Predictor is that previously we were predicting the exact floor value for impression but in the new version, we are predicting the floor level of the impression. In Sulvo every impression is labeled by an ad unit and every ad unit has multiple levels of floor value associated with it. To display an ad for an impression, Sulvo sends request to ad exchange (Google) with the highest level floor value, if the response for that request comes empty then Sulvo ad server sends another request with second level floor value and the process continues iteratively until ad server receives a response to display ad from ad exchange.

In the new version of the predictor, we are predicting the level of the floor for an impression. To make the problem simpler, we are predicting only the impression belongs to level one (high floor value) or not. Mathematically we frame the problem as a problem of classification where every impression is classified into two classes – level one impression or not level one impression. We are using neural network based logistic regression algorithm to classify the impression. Except for the algorithm, everything else is same as the previous version so in the next part of the document we are explaining the algorithm in more detail.

**Classification Algorithm:**

As we described in our previous document, Sulvo Predictor System has two components - real time trainer and predictor API. Real time trainer fetches the training data from Big Query Table and trains the model and stores it to Google Data Store. Predictor API, on receiving prediction request from ad server for a particular impression, it fetches the model from Google Data Store and predict the level of floor for that impression. As there is no change in interface in ad server side for predictor, so it still expects the floor price for the impression in response. For that reason after predicting the level predictor API sends the corresponding the floor value for that level of that ad unit.

**Code Walkthrough:**

*Real Time Trainer:*

1. Real time trainer after fetching the data from Google Big Query converts each feature value to a numerical score. For this, we are using the same scoring mechanism described in the version one document.
2. We are using soft-max classification model of tensor-flow in this algorithm. For binary classification target variable y need to have two columns - one for level one and one for not level one class. Column for level one class has value one if the corresponding row is a level one impression otherwise 0. Similarly, the column for not level one class has value one if the corresponding row is a no level one impression otherwise 0.
3. After training, score of each feature value is populated in data frame df2 and from their pushed to Cloud data store.

*Predictor API:*

Predictor API has following major changes from version one.

1. Each thread for each feature value is fetching the score for level one, and on level one impression and the scores are aggregated.
2. Final scores are passed throw a soft-max function which generating the probability of the impression is in level one and not level one.
3. If the probability of level one is greater than some predefined threshold, then it is returning level one floor value otherwise level two floor value.

*Threshold Finder:*

Threshold Finder is a new component which is added in version two.

1. It is calling the Predictor class with the threshold as an argument.
2. Then fetches fresh 1000 level one and 500 no level one impressions from Big Query.
3. After that calculate the optimization measure ( #Exact Prediction - #Under Prediction + 0.5\*#Over Prediction) for 1500 impressions.
4. Repeat the last step for a predefined threshold range (0.3 to 0.65 with .025 precision) and find the threshold where optimization measure is maximum.
5. Push that optimized threshold to Google Data Store, and Predictor API uses that for prediction.