Name: Nishit Shaileshbhai Rathod
Student No: N01586439

Direct Marketing with Amazon SageMaker XGBoost and Hyperparameter Tuning (SageMaker API)

Supervised Learning with Gradient Boosted Trees: A Binary Prediction Problem With Unbalanced Classes

---

Kernel `Python 3 (Data Science)` works well with this notebook.

Contents

Background

Direct marketing, either through mail, email, phone, etc., is a common tactic to acquire customers. Because resources and a customer's attention is limited, the goal is to only target the subset of prospects who are likely to engage with a specific offer. Predicting those potential customers based on readily available information like demographics, past interactions, and environmental factors is a common machine learning problem.

This notebook will train a model which can be used to predict if a customer will enroll for a term deposit at a bank, after one or more phone calls. Hyperparameter tuning will be used in order to try multiple hyperparameter settings and produce the best model.

Preparation

Let's start by specifying:

- The S3 bucket and prefix that you want to use for training and model data. This should be within the same region as SageMaker training.
- The IAM role used to give training access to your data. See SageMaker documentation for how to create these.

In [1]:
```
import sagemaker
import boto3

import numpy as np  # For matrix operations and numerical processing
import pandas as pd  # For munging tabular data
from time import gmtime, strftime
import os

region = boto3.Session().region_name
smclient = boto3.Session().client("sagemaker")

role = sagemaker.get_execution_role()

bucket = sagemaker.Session().default_bucket()
prefix = "sagemaker/DEMO-hpo-xgboost-dm"
```

Data_Downloading

Let's start by downloading the direct marketing dataset (https://archive.ics.uci.edu/ml/datasets/bank+marketing) from UCI's ML Repository.

In [2]:
```
!wget -N https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-additional.zip
!unzip -o bank-additional.zip
```

```
--2023-03-22 00:13:55--  https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-additional.zip (http
s://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-additional.zip)
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 444572 (434K) [application/x-httpd-php]
Saving to: ‘bank-additional.zip’

100%[=====================================>] 444,572      820KB/s   in 0.5s

2023-03-22 00:13:56 (820 KB/s) - ‘bank-additional.zip’ saved [444572/444572]

Archive:  bank-additional.zip
   creating: bank-additional/
  inflating: bank-additional/.DS_Store
   creating: __MACOSX/
   creating: __MACOSX/bank-additional/
  inflating: __MACOSX/bank-additional/._.DS_Store
  inflating: bank-additional/.Rhistory
  inflating: bank-additional/bank-additional-full.csv
  inflating: bank-additional/bank-additional-names.txt
  inflating: bank-additional/bank-additional.csv
  inflating: __MACOSX/._bank-additional
```

Now lets read this into a Pandas data frame and take a look.

In [3]:
```
data = pd.read_csv("./bank-additional/bank-additional-full.csv", sep=";")
pd.set_option("display.max_columns", 500)  # Make sure we can see all of the columns
pd.set_option("display.max_rows", 50)  # Keep the output on one page
data
```

Out[3]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1	999	0
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	149	1	999	0
2	37	services	married	high.school	no	yes	no	telephone	may	mon	226	1	999	0
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1	999	0
4	56	services	married	high.school	no	no	yes	telephone	may	mon	307	1	999	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	fri	334	1	999	0
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	383	1	999	0
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	189	2	999	0
41186	44	technician	married	professional.course	no	no	no	cellular	nov	fri	442	1	999	0
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	fri	239	3	999	1

41188 rows × 21 columns

Let's talk about the data. At a high level, we can see:

- We have a little over 40K customer records, and 20 features for each customer
- The features are mixed; some numeric, some categorical
- The data appears to be sorted, at least by `time` and `contact`, maybe more

Specifics on each of the features:

Demographics:

- `age` : Customer's age (numeric)
- `job` : Type of job (categorical: 'admin.', 'services', ...)
- `marital` : Marital status (categorical: 'married', 'single', ...)
- `education` : Level of education (categorical: 'basic.4y', 'high.school', ...)

Past customer events:

- `default` : Has credit in default? (categorical: 'no', 'unknown', ...)
- `housing` : Has housing loan? (categorical: 'no', 'yes', ...)
- `loan` : Has personal loan? (categorical: 'no', 'yes', ...)

Past direct marketing contacts:

- `contact` : Contact communication type (categorical: 'cellular', 'telephone', ...)
- `month` : Last contact month of year (categorical: 'may', 'nov', ...)
- `day_of_week` : Last contact day of the week (categorical: 'mon', 'fri', ...)

- `duration` : Last contact duration, in seconds (numeric). Important note: If duration = 0 then `y` = 'no'.

Campaign information:

- `campaign` : Number of contacts performed during this campaign and for this client (numeric, includes last contact)
- `pdays` : Number of days that passed by after the client was last contacted from a previous campaign (numeric)
- `previous` : Number of contacts performed before this campaign and for this client (numeric)
- `poutcome` : Outcome of the previous marketing campaign (categorical: 'nonexistent','success', ...)

External environment factors:

- `emp.var.rate` : Employment variation rate - quarterly indicator (numeric)
- `cons.price.idx` : Consumer price index - monthly indicator (numeric)
- `cons.conf.idx` : Consumer confidence index - monthly indicator (numeric)
- `euribor3m` : Euribor 3 month rate - daily indicator (numeric)
- `nr.employed` : Number of employees - quarterly indicator (numeric)

Target variable:

- `y` : Has the client subscribed a term deposit? (binary: 'yes','no')

Data_Transformation

Cleaning up data is part of nearly every machine learning project. It arguably presents the biggest risk if done incorrectly and is one of the more subjective aspects in the process. Several common techniques include:

- Handling missing values: Some machine learning algorithms are capable of handling missing values, but most would rather not. Options include:
  - Removing observations with missing values: This works well if only a very small fraction of observations have incomplete information.
  - Removing features with missing values: This works well if there are a small number of features which have a large number of missing values.
  - Imputing missing values: Entire books (https://www.amazon.com/Flexible-Imputation-Missing-Interdisciplinary-Statistics/dp/1439868247) have been written on this topic, but common choices are replacing the missing value with the mode or mean of that column's non-missing values.
- Converting categorical to numeric: The most common method is one hot encoding, which for each feature maps every distinct value of that column to its own feature which takes a value of 1 when the categorical feature is equal to that value, and 0 otherwise.
- Oddly distributed data: Although for non-linear models like Gradient Boosted Trees, this has very limited implications, parametric models like regression can produce wildly inaccurate estimates when fed highly skewed data. In some cases, simply taking the natural log of the features is sufficient to produce more normally distributed data. In others, bucketing values into discrete ranges is helpful. These buckets can then be treated as categorical variables and included in the model when one hot encoded.
- Handling more complicated data types: Mainpulating images, text, or data at varying grains.

Luckily, some of these aspects have already been handled for us, and the algorithm we are showcasing tends to do well at handling sparse or oddly distributed data. Therefore, let's keep pre-processing simple.

First of all, Many records have the value of "999" for pdays, number of days that passed by after a client was last contacted. It is very likely to be a magic number to represent that no contact was made before. Considering that, we create a new column called "no_previous_contact", then grant it value of "1" when pdays is 999 and "0" otherwise.

In the "job" column, there are categories that mean the customer is not working, e.g., "student", "retire", and "unemployed". Since it is very likely whether or not a customer is working will affect his/her decision to enroll in the term deposit, we generate a new column to show whether the customer is working based on "job" column.

Last but not the least, we convert categorical to numeric, as is suggested above.

In [4]:
```
data["no_previous_contact"] = np.where(
    data["pdays"] == 999, 1, 0
)  # Indicator variable to capture when pdays takes a value of 999
data["not_working"] = np.where(
    np.in1d(data["job"], ["student", "retired", "unemployed"]), 1, 0
)  # Indicator for individuals not actively employed
model_data = pd.get_dummies(data)  # Convert categorical variables to sets of indicators
model_data
```

Out[4]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	no_previous_contact	not_w
0	56	261	1	999	0	1.1	93.994	-36.4	4.857	5191.0	1	
1	57	149	1	999	0	1.1	93.994	-36.4	4.857	5191.0	1	
2	37	226	1	999	0	1.1	93.994	-36.4	4.857	5191.0	1	
3	40	151	1	999	0	1.1	93.994	-36.4	4.857	5191.0	1	
4	56	307	1	999	0	1.1	93.994	-36.4	4.857	5191.0	1	
...	...	...	...	...	...	...	...	...	...	...	...	
41183	73	334	1	999	0	-1.1	94.767	-50.8	1.028	4963.6	1	
41184	46	383	1	999	0	-1.1	94.767	-50.8	1.028	4963.6	1	
41185	56	189	2	999	0	-1.1	94.767	-50.8	1.028	4963.6	1	
41186	44	442	1	999	0	-1.1	94.767	-50.8	1.028	4963.6	1	
41187	74	239	3	999	1	-1.1	94.767	-50.8	1.028	4963.6	1	

41188 rows × 67 columns

Another question to ask yourself before building a model is whether certain features will add value in your final use case. For example, if your goal is to deliver the best prediction, then will you have access to that data at the moment of prediction? Knowing it's raining is highly predictive for umbrella sales, but forecasting weather far enough out to plan inventory on umbrellas is probably just as difficult as forecasting umbrella sales without knowledge of the weather. So, including this in your model may give you a false sense of precision.

Following this logic, let's remove the economic features and `duration` from our data as they would need to be forecasted with high precision to use as inputs in future predictions.

Even if we were to use values of the economic indicators from the previous quarter, this value is likely not as relevant for prospects contacted early in the next quarter as those contacted later on.

In [5]:
```
model_data = model_data.drop(
    ["duration", "emp.var.rate", "cons.price.idx", "cons.conf.idx", "euribor3m", "nr.employed"],
    axis=1,
)
```

We'll then split the dataset into training (70%), validation (20%), and test (10%) datasets and convert the datasets to the right format the algorithm expects. We will use training and validation datasets during training. Test dataset will be used to evaluate model performance after it is deployed to an endpoint.

Amazon SageMaker's XGBoost algorithm expects data in the libSVM or CSV data format. For this example, we'll stick to CSV. Note that the first column must be the target variable and the CSV should not include headers. Also, notice that although repetitive it's easiest to do this after the train|validation|test split rather than before. This avoids any misalignment issues due to random reordering.

In [6]:
```
train_data, validation_data, test_data = np.split(
    model_data.sample(frac=1, random_state=1729),
    [int(0.7 * len(model_data)), int(0.9 * len(model_data))],
)

pd.concat([train_data["y_yes"], train_data.drop(["y_no", "y_yes"], axis=1)], axis=1).to_csv(
    "train.csv", index=False, header=False
)
pd.concat(
    [validation_data["y_yes"], validation_data.drop(["y_no", "y_yes"], axis=1)], axis=1
).to_csv("validation.csv", index=False, header=False)
pd.concat([test_data["y_yes"], test_data.drop(["y_no", "y_yes"], axis=1)], axis=1).to_csv(
    "test.csv", index=False, header=False
)
```

Now we'll copy the file to S3 for Amazon SageMaker training to pickup.

In [7]:
```
boto3.Session().resource("s3").Bucket(bucket).Object(
    os.path.join(prefix, "train/train.csv")
).upload_file("train.csv")
boto3.Session().resource("s3").Bucket(bucket).Object(
    os.path.join(prefix, "validation/validation.csv")
).upload_file("validation.csv")
```

Setup_Hyperparameter_Tuning

Note, with the default setting below, the hyperparameter tuning job can take about 30 minutes to complete.

Now that we have prepared the dataset, we are ready to train models. Before we do that, one thing to note is there are algorithm settings which are called "hyperparameters" that can dramtically affect the performance of the trained models. For example, XGBoost algorithm has dozens of hyperparameters and we need to pick the right values for those hyperparameters in order to achieve the desired model training results. Since which hyperparameter setting can lead to the best result depends on the dataset as well, it is almost impossible to pick the best hyperparameter setting without searching for it, and a good search algorithm can search for the best hyperparameter setting in an automated and effective way.

We will use SageMaker hyperparameter tuning to automate the searching process effectively. Specifically, we specify a range, or a list of possible values in the case of categorical hyperparameters, for each of the hyperparameter that we plan to tune. SageMaker hyperparameter tuning will automatically launch multiple training jobs with different hyperparameter settings, evaluate results of those training jobs based on a predefined "objective metric", and select the hyperparameter settings for future attempts based on previous results. For each hyperparameter tuning job, we will give it a budget (max number of training jobs) and it will complete once that many training jobs have been executed.

Now we configure the hyperparameter tuning job by defining a JSON object that specifies following information:

- The ranges of hyperparameters we want to tune
- Number of training jobs to run in total and how many training jobs should be run simultaneously. More parallel jobs will finish tuning sooner, but may sacrifice accuracy. We recommend you set the parallel jobs value to less than 10% of the total number of training jobs (we'll set it higher just for this example to keep it short).
- The objective metric that will be used to evaluate training results, in this example, we select *validation:auc* to be the objective metric and the goal is to maximize the value throughout the hyperparameter tuning process. One thing to note is the objective metric has to be among the metrics that are emitted by the algorithm during training. In this example, the built-in XGBoost algorithm emits a bunch of metrics and *validation:auc* is one of them. If you bring your own algorithm to SageMaker, then you need to make sure whatever objective metric you select, your algorithm actually emits it.

We will tune four hyperparameters in this examples:

- *eta*: Step size shrinkage used in updates to prevent overfitting. After each boosting step, you can directly get the weights of new features. The eta parameter actually shrinks the feature weights to make the boosting process more conservative.
- *alpha*: L1 regularization term on weights. Increasing this value makes models more conservative.
- *min_child_weight*: Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, the building process gives up further partitioning. In linear regression models, this simply corresponds to a minimum number of instances needed in each node. The larger the algorithm, the more conservative it is.
- *max_depth*: Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfitted.

Bayesian Search

In [8]:
```
from time import gmtime, strftime, sleep

tuning_job_name = "xgboost-tuningjob-" + strftime("%d-%H-%M-%S", gmtime())

print(tuning_job_name)

tuning_job_config = {
    "ParameterRanges": {
        "CategoricalParameterRanges": [],
        "ContinuousParameterRanges": [
            {
                "MaxValue": "1",
                "MinValue": "0",
                "Name": "eta",
            },
            {
                "MaxValue": "10",
                "MinValue": "1",
                "Name": "min_child_weight",
            },
            {
                "MaxValue": "2",
                "MinValue": "0",
                "Name": "alpha",
            },
        ],
        "IntegerParameterRanges": [
            {
                "MaxValue": "10",
                "MinValue": "1",
                "Name": "max_depth",
            }
        ],
    },
    "ResourceLimits": {"MaxNumberOfTrainingJobs": 20, "MaxParallelTrainingJobs": 3},
    "Strategy": "Bayesian",
    "HyperParameterTuningJobObjective": {"MetricName": "validation:auc", "Type": "Maximize"},
}
```

```
xgboost-tuningjob-22-00-14-38
```

Then we configure the training jobs the hyperparameter tuning job will launch by defining a JSON object that specifies following information:

- The container image for the algorithm (XGBoost)
- The input configuration for the training and validation data
- Configuration for the output of the algorithm
- The values of any algorithm hyperparameters that are not tuned in the tuning job (StaticHyperparameters)
- The type and number of instances to use for the training jobs
- The stopping condition for the training jobs

Again, since we are using built-in XGBoost algorithm here, it emits two predefined metrics: *validation:auc* and *train:auc*, and we elected to monitor *validation_auc* as you can see above. One thing to note is if you bring your own algorithm, your algorithm emits metrics by itself. In that case, you'll need to add a MetricDefinition object here to define the format of those metrics through regex, so that SageMaker knows how to extract those metrics.

In [9]:
```
from sagemaker.image_uris import retrieve

training_image = retrieve(framework="xgboost", region=region, version="1.5-1")

s3_input_train = "s3://{}/{}/train".format(bucket, prefix)
s3_input_validation = "s3://{}/{}/validation/".format(bucket, prefix)

training_job_definition = {
    "AlgorithmSpecification": {"TrainingImage": training_image, "TrainingInputMode": "File"},
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_train,
                }
            },
        },
        {
            "ChannelName": "validation",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_validation,
                }
            },
        },
    ],
    "OutputDataConfig": {"S3OutputPath": "s3://{}/{}/output".format(bucket, prefix)},
    "ResourceConfig": {"InstanceCount": 1, "InstanceType": "ml.m4.xlarge", "VolumeSizeInGB": 10},
    "RoleArn": role,
    "StaticHyperParameters": {
        "eval_metric": "auc",
        "num_round": "100",
        "objective": "binary:logistic",
        "rate_drop": "0.3",
        "tweedie_variance_power": "1.4",
    },
    "StoppingCondition": {"MaxRuntimeInSeconds": 43200},
}
```

Launch_Hyperparameter_Tuning

Now we can launch a hyperparameter tuning job by calling create_hyper_parameter_tuning_job API. After the hyperparameter tuning job is created, we can go to SageMaker console to track the progress of the hyperparameter tuning job until it is completed.

In [10]:
```
smclient.create_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuning_job_name,
    HyperParameterTuningJobConfig=tuning_job_config,
    TrainingJobDefinition=training_job_definition,
)
```

Out[10]:
```
{'HyperParameterTuningJobArn': 'arn:aws:sagemaker:ca-central-1:224670572127:hyper-parameter-tuning-job/xgboost-tuningjob-22-00-14-38',
 'ResponseMetadata': {'RequestId': 'bb3e9b72-1280-4dfa-ac33-2660b874cf96',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'bb3e9b72-1280-4dfa-ac33-2660b874cf96',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '133',
   'date': 'Wed, 22 Mar 2023 00:22:39 GMT'},
  'RetryAttempts': 0}}
```

Let's just run a quick check of the hyperparameter tuning jobs status to make sure it started successfully.

In [13]:
```
smclient.describe_hyper_parameter_tuning_job(HyperParameterTuningJobName=tuning_job_name)[
    "HyperParameterTuningJobStatus"
]
```

Out[13]:
```
'Completed'
```

In [14]:
```
smclient.describe_hyper_parameter_tuning_job(HyperParameterTuningJobName=tuning_job_name)['HyperParameterTuningJobName
```

Out[14]:
```
'xgboost-tuningjob-22-00-14-38'
```

Important:

Wait until the hyperparameter job is done. You can go to console and see if 20 jobs are completed or not. Alternatovely you have to re-run the folling code until you see 20 training jobs have completed.

In [15]:
```
# run this cell to check current status of hyperparameter tuning job
tuning_job_result = smclient.describe_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuning_job_name
)

status = tuning_job_result["HyperParameterTuningJobStatus"]
if status != "Completed":
    print("Reminder: the tuning job has not been completed.")

job_count = tuning_job_result["TrainingJobStatusCounters"]["Completed"]
print("%d training jobs have completed" % job_count)

objective = tuning_job_result["HyperParameterTuningJobConfig"]["HyperParameterTuningJobObjective"]
is_minimize = objective["Type"] != "Maximize"
objective_name = objective["MetricName"]
```

```
20 training jobs have completed
```

Although you can find that in the console, you can also query the best training job programmatically:

In [16]:
```
from pprint import pprint

if tuning_job_result.get("BestTrainingJob", None):
    print("Best model found so far:")
    pprint(tuning_job_result["BestTrainingJob"])
else:
    print("No training jobs have reported results yet.")
```

```
Best model found so far:
{'CreationTime': datetime.datetime(2023, 3, 22, 0, 27, 31, tzinfo=tzlocal()),
 'FinalHyperParameterTuningJobObjectiveMetric': {'MetricName': 'validation:auc',
                                                 'Value': 0.7778800129890442},
 'ObjectiveStatus': 'Succeeded',
 'TrainingEndTime': datetime.datetime(2023, 3, 22, 0, 28, 11, tzinfo=tzlocal()),
 'TrainingJobArn': 'arn:aws:sagemaker:ca-central-1:224670572127:training-job/xgboost-tuningjob-22-00-14-38-007-15a91
b83',
 'TrainingJobName': 'xgboost-tuningjob-22-00-14-38-007-15a91b83',
 'TrainingJobStatus': 'Completed',
 'TrainingStartTime': datetime.datetime(2023, 3, 22, 0, 27, 34, tzinfo=tzlocal()),
 'TunedHyperParameters': {'alpha': '1.2069750310954586',
                          'eta': '0.19541118326398887',
                          'max_depth': '3',
                          'min_child_weight': '9.825319775841665'}}
```

Fetch all results as DataFrame

We can actully see all the training jobs and list hyperparameters and objective metrics and pick up the training job with the best objective metric.

In [17]:
```
import pandas as pd

tuner = sagemaker.HyperparameterTuningJobAnalytics(tuning_job_name)

full_df = tuner.dataframe()

if len(full_df) > 0:
    df = full_df[full_df["FinalObjectiveValue"] > -float("inf")]
    if len(df) > 0:
        df = df.sort_values("FinalObjectiveValue", ascending=is_minimize)
        print("Number of training jobs with valid objective: %d" % len(df))
        print({"lowest": min(df["FinalObjectiveValue"]), "highest": max(df["FinalObjectiveValue"])})
        pd.set_option("display.max_colwidth", None)  # Don't truncate TrainingJobName
    else:
        print("No training jobs have reported valid results yet.")

full_df
```

```
Number of training jobs with valid objective: 20
{'lowest': 0.5, 'highest': 0.7778800129890442}
```

Out[17]:

	alpha	eta	max_depth	min_child_weight	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTim
0	1.078681	0.185214	10.0	6.918996	xgboost-tuningjob-22-00-14-38-020-3748cc51	Completed	0.76170	2023-03-22 00:31:52+00:00	2023-03-2 00:32:34+00:0
1	0.332531	0.194637	4.0	7.300702	xgboost-tuningjob-22-00-14-38-019-2f77b859	Completed	0.77718	2023-03-22 00:31:36+00:00	2023-03-2 00:32:13+00:0
2	1.836996	0.059446	10.0	8.115853	xgboost-tuningjob-22-00-14-38-018-0c874f71	Completed	0.77426	2023-03-22 00:31:29+00:00	2023-03-2 00:32:06+00:0
3	0.858057	0.126578	8.0	3.007403	xgboost-tuningjob-22-00-14-38-017-18247547	Completed	0.76833	2023-03-22 00:30:58+00:00	2023-03-2 00:31:40+00:0
4	1.437245	0.262215	4.0	9.551085	xgboost-tuningjob-22-00-14-38-016-3310a03f	Completed	0.77595	2023-03-22 00:30:36+00:00	2023-03-2 00:31:12+00:0
5	0.236419	1.000000	6.0	8.293952	xgboost-tuningjob-22-00-14-38-015-dc23cbbd	Completed	0.72764	2023-03-22 00:29:55+00:00	2023-03-2 00:31:22+00:0
6	1.558915	0.000000	3.0	9.493724	xgboost-tuningjob-22-00-14-38-014-9658bc79	Completed	0.50000	2023-03-22 00:29:40+00:00	2023-03-2 00:30:17+00:0
7	1.040051	1.000000	3.0	1.420227	xgboost-tuningjob-22-00-14-38-013-9ab9224a	Completed	0.75798	2023-03-22 00:30:10+00:00	2023-03-2 00:30:46+00:0
8	1.480312	0.194139	6.0	10.000000	xgboost-tuningjob-22-00-14-38-012-fd768509	Completed	0.77182	2023-03-22 00:28:59+00:00	2023-03-2 00:29:36+00:0
9	0.777446	0.000000	1.0	10.000000	xgboost-tuningjob-22-00-14-38-011-b7e6386e	Completed	0.50000	2023-03-22 00:28:48+00:00	2023-03-2 00:29:19+00:0
10	1.158060	0.000000	1.0	2.460492	xgboost-tuningjob-22-00-14-38-010-2070232a	Completed	0.50000	2023-03-22 00:28:42+00:00	2023-03-2 00:29:18+00:0
11	0.753993	0.000410	5.0	2.224004	xgboost-tuningjob-22-00-14-38-009-547c3810	Completed	0.70913	2023-03-22 00:27:57+00:00	2023-03-2 00:28:39+00:0
12	1.876525	0.000922	7.0	2.801939	xgboost-tuningjob-22-00-14-38-008-2d724562	Completed	0.73037	2023-03-22 00:28:00+00:00	2023-03-2 00:28:32+00:0
13	1.206975	0.195411	3.0	9.825320	xgboost-tuningjob-22-00-14-38-007-15a91b83	Completed	0.77788	2023-03-22 00:27:34+00:00	2023-03-2 00:28:11+00:0
14	0.326636	0.999315	8.0	4.042166	xgboost-tuningjob-22-00-14-38-006-80d89f72	Completed	0.71631	2023-03-22 00:27:00+00:00	2023-03-2 00:27:42+00:0
15	0.363174	0.244277	8.0	3.120908	xgboost-tuningjob-22-00-14-38-005-a3e563c2	Completed	0.75577	2023-03-22 00:26:37+00:00	2023-03-2 00:27:19+00:0
16	1.598259	0.039904	8.0	7.905557	xgboost-tuningjob-22-00-14-38-004-2ea9cb53	Completed	0.77440	2023-03-22 00:26:35+00:00	2023-03-2 00:27:41+00:0
17	0.339486	0.688410	9.0	4.088785	xgboost-tuningjob-22-00-14-38-003-f4548ca3	Completed	0.72423	2023-03-22 00:24:18+00:00	2023-03-2 00:26:10+00:0
18	1.879760	0.766921	9.0	3.757508	xgboost-tuningjob-22-00-14-38-002-c6fdbd71	Completed	0.71747	2023-03-22 00:24:53+00:00	2023-03-2 00:26:45+00:0
19	1.203004	0.898737	8.0	8.175674	xgboost-tuningjob-22-00-14-38-001-151ff5c5	Completed	0.71161	2023-03-22 00:24:13+00:00	2023-03-2 00:25:59+00:0

It came to notice that the highest accuracy achieved via. Bayesian search is of 0.7778 and the lowest accuracy achieved via. Bayesian search is of 0.5 after running 20 hyper tuning jobs.

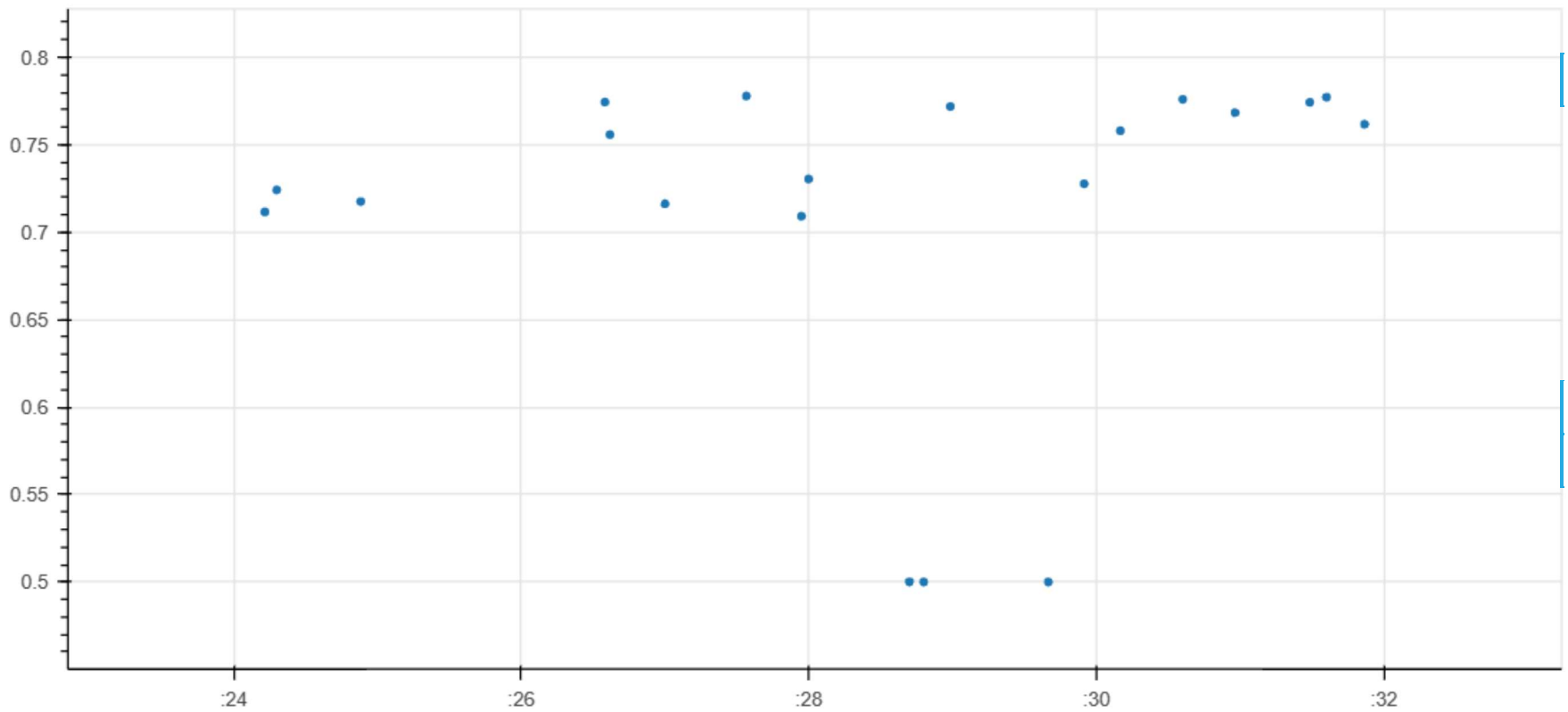
See TuningJob results vs time

Next we will show how the objective metric changes over time, as the tuning job progresses. For Bayesian strategy, you should expect to see a general trend towards better results, but this progress will not be steady as the algorithm needs to balance *exploration* of new areas of parameter space against *exploitation* of known good areas. This can give you a sense of whether or not the number of training jobs is sufficient for the complexity of your search space.

In [18]:
```
import bokeh
import bokeh.io

bokeh.io.output_notebook()
from bokeh.plotting import figure, show
from bokeh.models import HoverTool


class HoverHelper:
    def __init__(self, tuning_analytics):
        self.tuner = tuning_analytics

    def hovertool(self):
        tooltips = [
            ("FinalObjectiveValue", "@FinalObjectiveValue"),
            ("TrainingJobName", "@TrainingJobName"),
        ]
        for k in self.tuner.tuning_ranges.keys():
            tooltips.append((k, "@{%s}" % k))

        ht = HoverTool(tooltips=tooltips)
        return ht

    def tools(self, standard_tools="pan,crosshair,wheel_zoom,zoom_in,zoom_out,undo,reset"):
        return [self.hovertool(), standard_tools]


hover = HoverHelper(tuner)

p = figure(width=900, height=400, tools=hover.tools(), x_axis_type="datetime")
p.circle(source=df, x="TrainingStartTime", y="FinalObjectiveValue")
show(p)
```

(https://bokeh.org)

Loading BokehJS ...



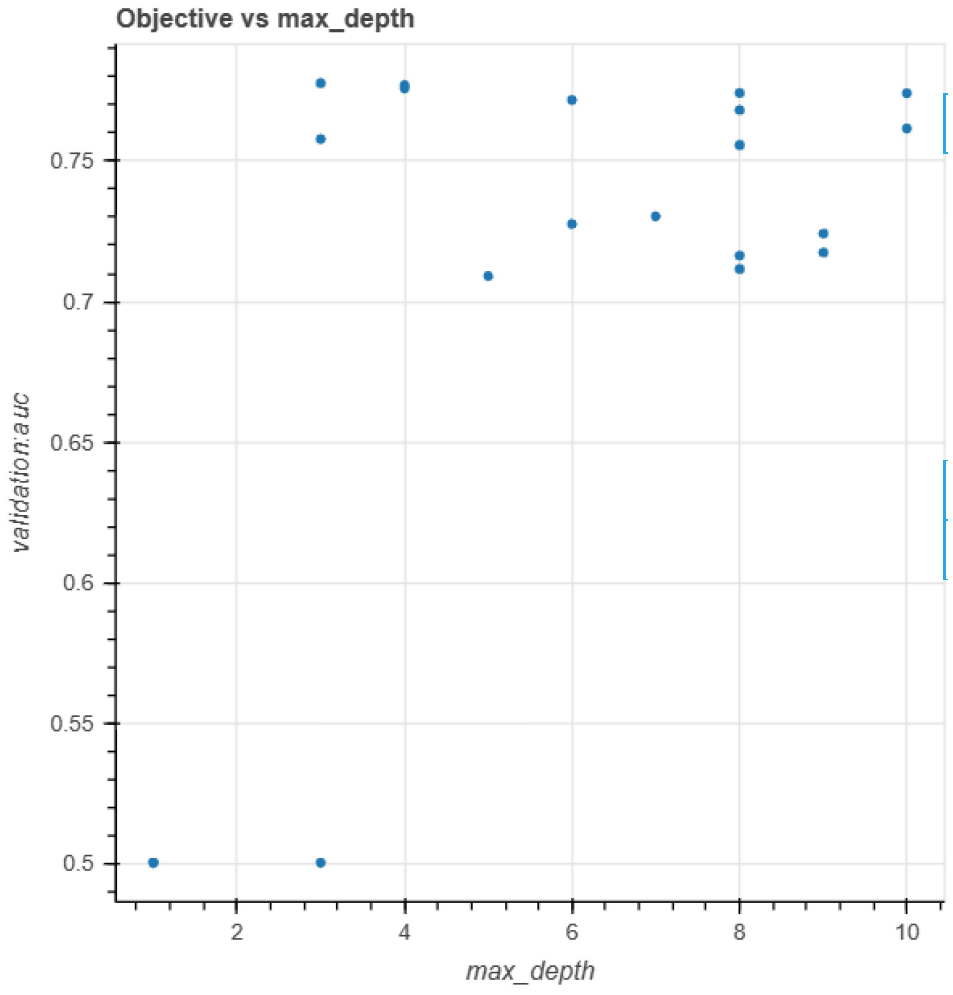
From the above graph it can be inferred that as the time passes the accuracy rremains to be in range of 0.7 to 0.8 but after .28 there is a sudden drop in the accuracy which continues till .30. The highest accuracy achieved is in the initial time phase where max_depth=3, eta=0.195, min_child_weight=9.825 and alpha=1.207 which makes best suited parameters.

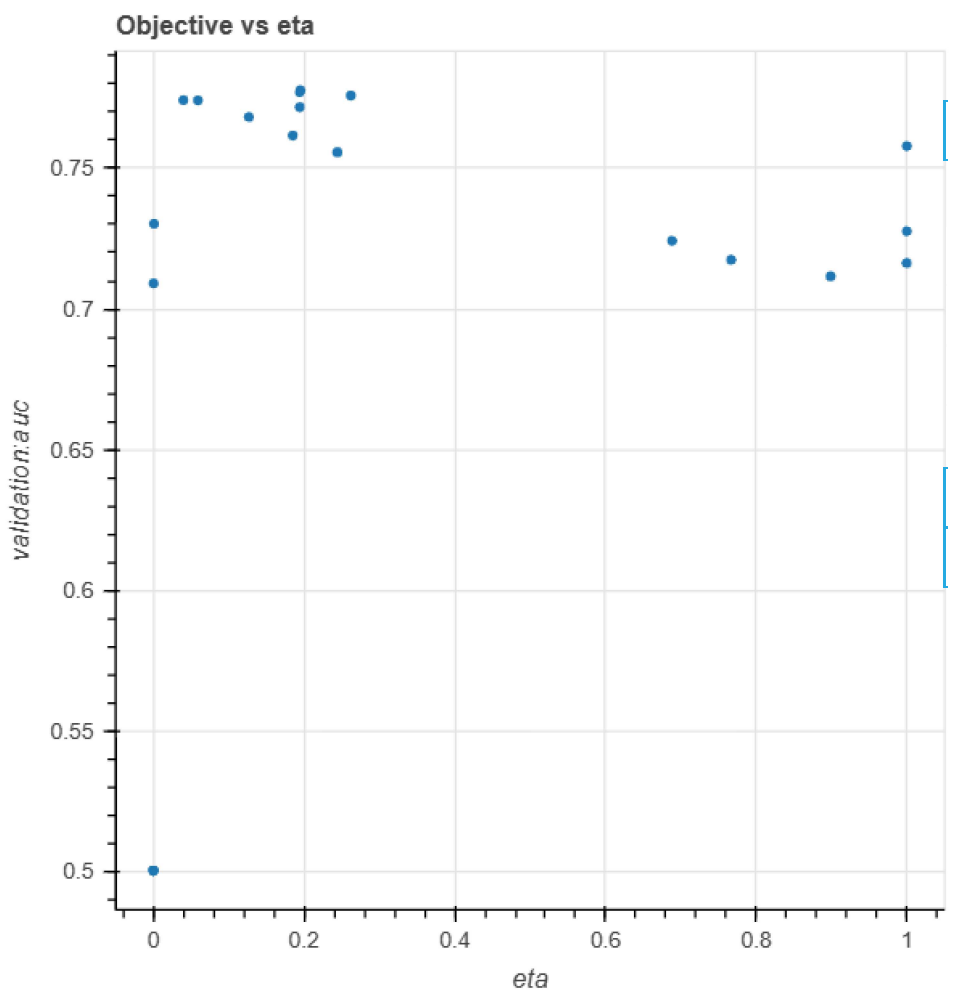
Analyze the correlation between objective metric and individual hyperparameters

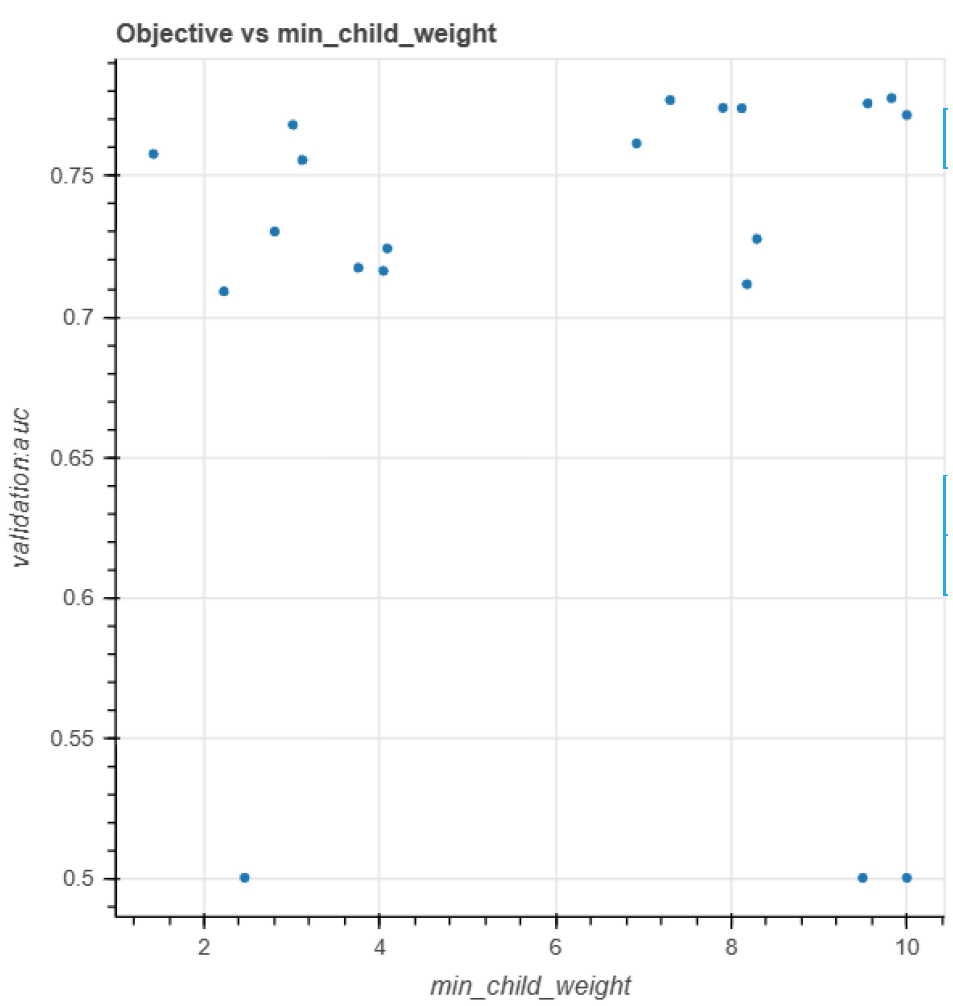
Now you have finished a tuning job, you may want to know the correlation between your objective metric and individual hyperparameters you've selected to tune. Having that insight will help you decide whether it makes sense to adjust search ranges for certain hyperparameters and start another tuning job. For example, if you see a positive trend between objective metric and a numerical hyperparameter, you probably want to set a higher tuning range for that hyperparameter in your next tuning job.

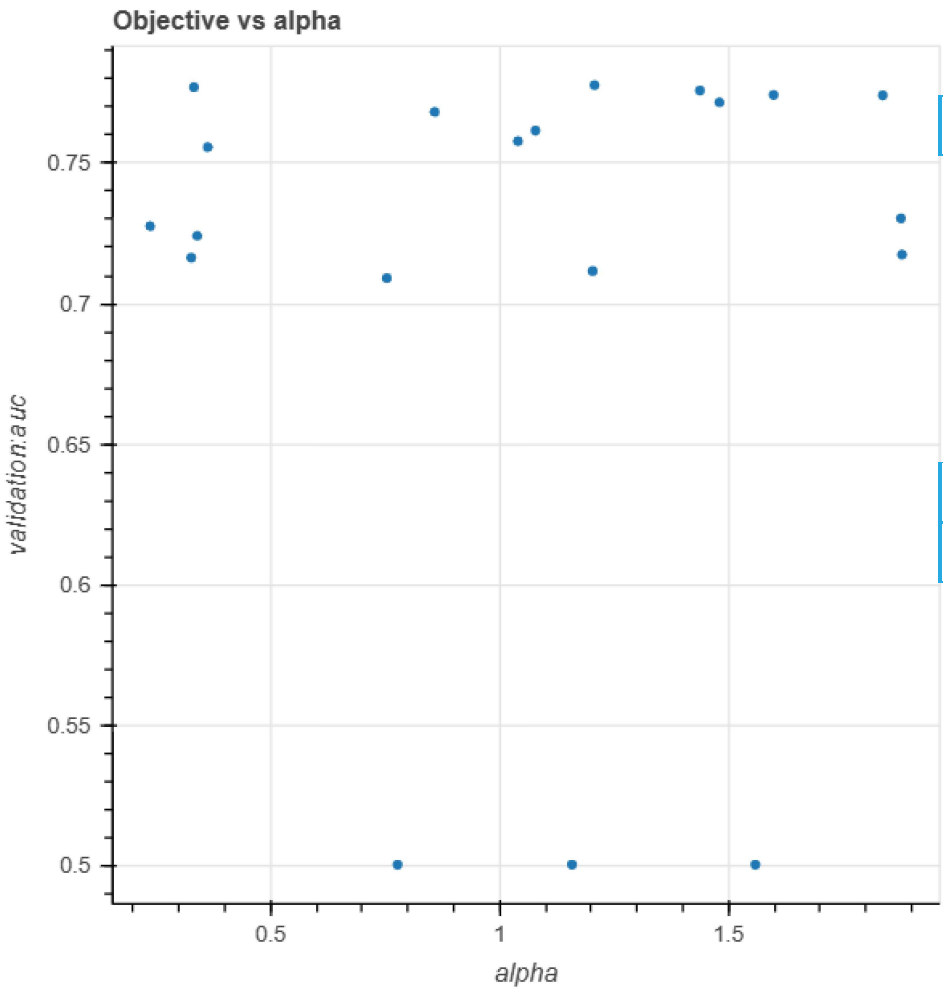
The following cell draws a graph for each hyperparameter to show its correlation with your objective metric.

In [19]:
```
ranges = tuner.tuning_ranges
figures = []
for hp_name, hp_range in ranges.items():
    categorical_args = {}
    if hp_range.get("Values"):
        # This is marked as categorical.  Check if all options are actually numbers.
        def is_num(x):
            try:
                float(x)
                return 1
            except:
                return 0

        vals = hp_range["Values"]
        if sum([is_num(x) for x in vals]) == len(vals):
            # Bokeh has issues plotting a "categorical" range that's actually numeric, so plot as numeric
            print("Hyperparameter %s is tuned as categorical, but all values are numeric" % hp_name)
        else:
            # Set up extra options for plotting categoricals.  A bit tricky when they're actually numbers.
            categorical_args["x_range"] = vals

    # Now plot it
    p = figure(
        width=500,
        height=500,
        title="Objective vs %s" % hp_name,
        tools=hover.tools(),
        x_axis_label=hp_name,
        y_axis_label=objective_name,
        **categorical_args,
    )
    p.circle(source=df, x=hp_name, y="FinalObjectiveValue")
    figures.append(p)
show(bokeh.layouts.Column(*figures))
```


Objective vs max_depth
validation:auc
0.75
0.7
0.65
0.6
0.55
0.5
2
4
6
8
10
max_depth


Objective vs eta
validation:auc
0.75
0.7
0.65
0.6
0.55
0.5
0
0.2
0.4
0.6
0.8
1
eta


Objective vs min_child_weight
validation:auc
0.75
0.7
0.65
0.6
0.55
0.5
2
4
6
8
10
min_child_weight



Random Search

Description of Random Search Code

- tuning_job_name = "xgboost-tuningjob-rs-" + strftime("%d-%H-%M-%S", gmtime()): This line creates a unique name for the tuning job, which is composed of a prefix "xgboost-tuningjob-rs-" and a timestamp generated using the strftime function. The timestamp is in the format dd-HH-MM-SS, where dd is the day of the month, HH is the hour (in 24-hour format), MM is the minute, and SS is the second. This ensures that each tuning job has a unique name.
- print(tuning_job_name): This line simply prints the name of the tuning job to the console.
- tuning_job_config = {...}: This line creates a dictionary object that contains the configuration for the tuning job. This dictionary has several keys:
- "ParameterRanges": This key contains a dictionary that specifies the ranges for the hyperparameters that will be tuned. There are three types of hyperparameters:
- "CategoricalParameterRanges": This list is empty, indicating that there are no categorical hyperparameters to tune.
- "ContinuousParameterRanges": This list contains dictionaries that define the range for each continuous hyperparameter. Each dictionary has four keys:
- "MaxValue": The maximum value for the hyperparameter.
- "MinValue": The minimum value for the hyperparameter.
- "Name": The name of the hyperparameter.
- "ScalingType": The scaling type for the hyperparameter. In this case, it is set to "Auto", which means that SageMaker will automatically scale the hyperparameters based on the range specified.
- "IntegerParameterRanges": This list contains dictionaries that define the range for each integer hyperparameter. Each dictionary has the same four keys as in the continuous parameter range list.
- "ResourceLimits": This key contains a dictionary that specifies the maximum number of training jobs and parallel training jobs that can be run during the tuning job.
- "MaxNumberOfTrainingJobs": The maximum number of training jobs to run during the tuning job. In this case, it is set to 20.
- "MaxParallelTrainingJobs": The maximum number of training jobs to run in parallel during the tuning job. In this case, it is set to 3.
- "Strategy": This key specifies the tuning strategy to use. In this case, it is set to "Random", which means that SageMaker will randomly sample hyperparameter values from the specified ranges.
- "HyperParameterTuningJobObjective": This key contains a dictionary that specifies the objective metric to optimize during the tuning job.
- "MetricName": The name of the objective metric. In this case, it is set to "validation:auc", which means that SageMaker will optimize the area under the ROC curve (AUC) on the validation set.
- "Type": The type of objective metric. In this case, it is set to "Maximize", which means that SageMaker will maximize the AUC on the validation set.

In [22]:
```
tuning_job_name = "xgboost-tuningjob-rs-" + strftime("%d-%H-%M-%S", gmtime())

print(tuning_job_name)

tuning_job_config = {
    "ParameterRanges": {
        "CategoricalParameterRanges": [],
        "ContinuousParameterRanges": [
            {
                "MaxValue": "1",
                "MinValue": "0",
                "Name": "eta",
                "ScalingType": "Auto"
            },
            {
                "MaxValue": "10",
                "MinValue": "1",
                "Name": "min_child_weight",
                "ScalingType": "Auto"
            },
            {
                "MaxValue": "2",
                "MinValue": "0",
                "Name": "alpha",
                "ScalingType": "Auto"
            },
        ],
        "IntegerParameterRanges": [
            {
                "MaxValue": "10",
                "MinValue": "1",
                "Name": "max_depth",
                "ScalingType": "Auto"
            }
        ],
    },
    "ResourceLimits": {"MaxNumberOfTrainingJobs": 20, "MaxParallelTrainingJobs": 3},
    "Strategy": "Random",
    "HyperParameterTuningJobObjective": {"MetricName": "validation:auc", "Type": "Maximize"},
}
```

```
xgboost-tuningjob-rs-22-01-21-18
```

In [23]:
```
from sagemaker.image_uris import retrieve

training_image = retrieve(framework="xgboost", region=region, version="1.5-1")

s3_input_train = "s3://{}/{}/train".format(bucket, prefix)
s3_input_validation = "s3://{}/{}/validation/".format(bucket, prefix)

training_job_definition = {
    "AlgorithmSpecification": {"TrainingImage": training_image, "TrainingInputMode": "File"},
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_train,
                }
            },
        },
        {
            "ChannelName": "validation",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_validation,
                }
            },
        },
    ],
    "OutputDataConfig": {"S3OutputPath": "s3://{}/{}/output".format(bucket, prefix)},
    "ResourceConfig": {"InstanceCount": 1, "InstanceType": "ml.m4.xlarge", "VolumeSizeInGB": 10},
    "RoleArn": role,
    "StaticHyperParameters": {
        "eval_metric": "auc",
        "num_round": "100",
        "objective": "binary:logistic",
        "rate_drop": "0.3",
        "tweedie_variance_power": "1.4",
    },
    "StoppingCondition": {"MaxRuntimeInSeconds": 43200},
}
```

In [24]:
```
smclient.create_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuning_job_name,
    HyperParameterTuningJobConfig=tuning_job_config,
    TrainingJobDefinition=training_job_definition,
)
```

Out[24]:
```
{'HyperParameterTuningJobArn': 'arn:aws:sagemaker:ca-central-1:224670572127:hyper-parameter-tuning-job/xgboost-tuningjob-rs-22-01-21-18',
 'ResponseMetadata': {'RequestId': 'a40dbfb8-e1d3-4d16-92f3-13609d2a44ed',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'a40dbfb8-e1d3-4d16-92f3-13609d2a44ed',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '136',
   'date': 'Wed, 22 Mar 2023 01:22:50 GMT'},
  'RetryAttempts': 0}}
```

In [26]:
```
smclient.describe_hyper_parameter_tuning_job(HyperParameterTuningJobName=tuning_job_name)[
    "HyperParameterTuningJobStatus"
]
```

Out[26]:
```
'Completed'
```

In [27]:
```
smclient.describe_hyper_parameter_tuning_job(HyperParameterTuningJobName=tuning_job_name)['HyperParameterTuningJobName
```

Out[27]:
```
'xgboost-tuningjob-rs-22-01-21-18'
```

In [28]:
```
# run this cell to check current status of hyperparameter tuning job
tuning_job_result = smclient.describe_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuning_job_name
)

status = tuning_job_result["HyperParameterTuningJobStatus"]
if status != "Completed":
    print("Reminder: the tuning job has not been completed.")

job_count = tuning_job_result["TrainingJobStatusCounters"]["Completed"]
print("%d training jobs have completed" % job_count)

objective = tuning_job_result["HyperParameterTuningJobConfig"]["HyperParameterTuningJobObjective"]
is_minimize = objective["Type"] != "Maximize"
objective_name = objective["MetricName"]
```

```
20 training jobs have completed
```

In [29]:
```
from pprint import pprint

if tuning_job_result.get("BestTrainingJob", None):
    print("Best model found so far:")
    pprint(tuning_job_result["BestTrainingJob"])
else:
    print("No training jobs have reported results yet.")
```

```
Best model found so far:
{'CreationTime': datetime.datetime(2023, 3, 22, 1, 22, 56, tzinfo=tzlocal()),
 'FinalHyperParameterTuningJobObjectiveMetric': {'MetricName': 'validation:auc',
                                                 'Value': 0.7781999707221985},
 'ObjectiveStatus': 'Succeeded',
 'TrainingEndTime': datetime.datetime(2023, 3, 22, 1, 26, 5, tzinfo=tzlocal()),
 'TrainingJobArn': 'arn:aws:sagemaker:ca-central-1:224670572127:training-job/xgboost-tuningjob-rs-22-01-21-18-002-6a
a26985',
 'TrainingJobName': 'xgboost-tuningjob-rs-22-01-21-18-002-6aa26985',
 'TrainingJobStatus': 'Completed',
 'TrainingStartTime': datetime.datetime(2023, 3, 22, 1, 24, 13, tzinfo=tzlocal()),
 'TunedHyperParameters': {'alpha': '1.815984937295079',
                          'eta': '0.1667413849313858',
                          'max_depth': '3',
                          'min_child_weight': '2.8502616265791945'}}
```

In [30]:
```
import pandas as pd

tuner = sagemaker.HyperparameterTuningJobAnalytics(tuning_job_name)

full_df1 = tuner.dataframe()

if len(full_df1) > 0:
    df1 = full_df1[full_df1["FinalObjectiveValue"] > -float("inf")]
    if len(df1) > 0:
        df1 = df1.sort_values("FinalObjectiveValue", ascending=is_minimize)
        print("Number of training jobs with valid objective: %d" % len(df1))
        print({"lowest": min(df1["FinalObjectiveValue"]), "highest": max(df1["FinalObjectiveValue"])})
        pd.set_option("display.max_colwidth", None)  # Don't truncate TrainingJobName
    else:
        print("No training jobs have reported valid results yet.")

full_df1
```

```
Number of training jobs with valid objective: 20
{'lowest': 0.7134100198745728, 'highest': 0.7781999707221985}
```

Out[30]:

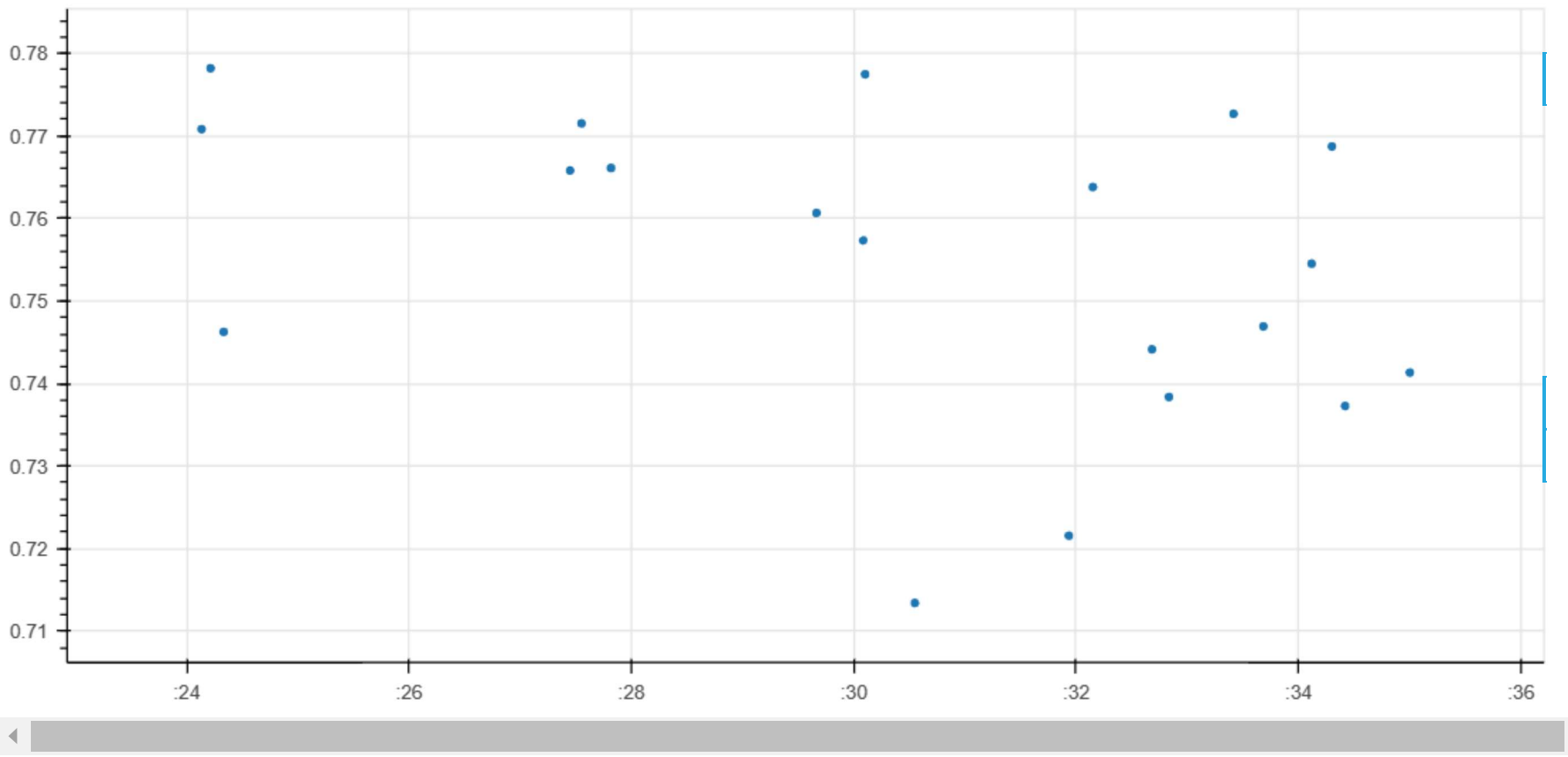
	alpha	eta	max_depth	min_child_weight	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTim
0	1.313310	0.543080	7.0	7.624844	xgboost-tuningjob-rs-22-01-21-18-020-26ff63bb	Completed	0.74134	2023-03-22 01:35:00+00:00	2023-03-2 01:35:37+00:0
1	1.232995	0.686836	8.0	9.918198	xgboost-tuningjob-rs-22-01-21-18-019-d30ede52	Completed	0.73728	2023-03-22 01:34:25+00:00	2023-03-2 01:35:02+00:0
2	0.831776	0.401977	6.0	3.818208	xgboost-tuningjob-rs-22-01-21-18-018-8570735d	Completed	0.75452	2023-03-22 01:34:07+00:00	2023-03-2 01:34:44+00:0
3	1.556484	0.053366	1.0	8.817449	xgboost-tuningjob-rs-22-01-21-18-017-feae49a1	Completed	0.74690	2023-03-22 01:33:41+00:00	2023-03-2 01:34:13+00:0
4	0.408923	0.179380	2.0	4.527277	xgboost-tuningjob-rs-22-01-21-18-016-39c6946b	Completed	0.77268	2023-03-22 01:33:25+00:00	2023-03-2 01:33:57+00:0
5	1.818933	0.634865	6.0	2.272718	xgboost-tuningjob-rs-22-01-21-18-015-1a0bbae4	Completed	0.73837	2023-03-22 01:32:50+00:00	2023-03-2 01:33:31+00:0
6	0.555698	0.717729	5.0	5.215219	xgboost-tuningjob-rs-22-01-21-18-014-0753e5de	Completed	0.74415	2023-03-22 01:32:41+00:00	2023-03-2 01:33:17+00:0
7	0.132047	0.818713	2.0	7.244067	xgboost-tuningjob-rs-22-01-21-18-013-ad1c430e	Completed	0.76870	2023-03-22 01:34:18+00:00	2023-03-2 01:36:06+00:0
8	0.226612	0.483060	5.0	8.744770	xgboost-tuningjob-rs-22-01-21-18-012-03ec2cf8	Completed	0.76383	2023-03-22 01:32:09+00:00	2023-03-2 01:32:41+00:0
9	0.150895	0.731217	7.0	4.107317	xgboost-tuningjob-rs-22-01-21-18-011-52598127	Completed	0.72156	2023-03-22 01:31:56+00:00	2023-03-2 01:32:29+00:0
10	0.573610	0.827721	8.0	5.488797	xgboost-tuningjob-rs-22-01-21-18-010-e9e92d91	Completed	0.71341	2023-03-22 01:30:33+00:00	2023-03-2 01:31:41+00:0
11	1.567118	0.091282	6.0	8.473839	xgboost-tuningjob-rs-22-01-21-18-009-fd8dd89c	Completed	0.77748	2023-03-22 01:30:06+00:00	2023-03-2 01:32:02+00:0
12	0.746660	0.709451	1.0	8.111485	xgboost-tuningjob-rs-22-01-21-18-008-e2d495c2	Completed	0.76067	2023-03-22 01:29:40+00:00	2023-03-2 01:30:17+00:0
13	0.326287	0.767829	3.0	6.483328	xgboost-tuningjob-rs-22-01-21-18-007-53980aa6	Completed	0.75733	2023-03-22 01:30:05+00:00	2023-03-2 01:31:59+00:0
14	0.665373	0.606057	3.0	3.687661	xgboost-tuningjob-rs-22-01-21-18-006-5b7fcfd9	Completed	0.76612	2023-03-22 01:27:49+00:00	2023-03-2 01:29:36+00:0
15	1.607904	0.610290	3.0	5.781670	xgboost-tuningjob-rs-22-01-21-18-005-03efb5dd	Completed	0.77152	2023-03-22 01:27:33+00:00	2023-03-2 01:29:20+00:0
16	1.186171	0.256099	6.0	3.329939	xgboost-tuningjob-rs-22-01-21-18-004-b47ba522	Completed	0.76582	2023-03-22 01:27:27+00:00	2023-03-2 01:29:14+00:0
17	0.089898	0.435765	3.0	1.146703	xgboost-tuningjob-rs-22-01-21-18-003-f649cc6c	Completed	0.77082	2023-03-22 01:24:08+00:00	2023-03-2 01:25:55+00:0
18	1.815985	0.166741	3.0	2.850262	xgboost-tuningjob-rs-22-01-21-18-002-6aa26985	Completed	0.77820	2023-03-22 01:24:13+00:00	2023-03-2 01:26:05+00:0
19	1.687559	0.488393	9.0	8.681093	xgboost-tuningjob-rs-22-01-21-18-001-0afa1636	Completed	0.74627	2023-03-22 01:24:20+00:00	2023-03-2 01:26:17+00:0

It came to notice that the highest accuracy achieved via. Random search is of 0.778 and the lowest accuracy achieved via. Random search is of 0.713 after running 20 hyper tuning jobs.

In [32]:
```
import bokeh
import bokeh.io

bokeh.io.output_notebook()
from bokeh.plotting import figure, show
from bokeh.models import HoverTool


class HoverHelper:
    def __init__(self, tuning_analytics):
        self.tuner = tuning_analytics

    def hovertool(self):
        tooltips = [
            ("FinalObjectiveValue", "@FinalObjectiveValue"),
            ("TrainingJobName", "@TrainingJobName"),
        ]
        for k in self.tuner.tuning_ranges.keys():
            tooltips.append((k, "@{%s}" % k))

        ht = HoverTool(tooltips=tooltips)
        return ht

    def tools(self, standard_tools="pan,crosshair,wheel_zoom,zoom_in,zoom_out,undo,reset"):
        return [self.hovertool(), standard_tools]


hover = HoverHelper(tuner)

p = figure(width=900, height=400, tools=hover.tools(), x_axis_type="datetime")
p.circle(source=df1, x="TrainingStartTime", y="FinalObjectiveValue")
show(p)
```

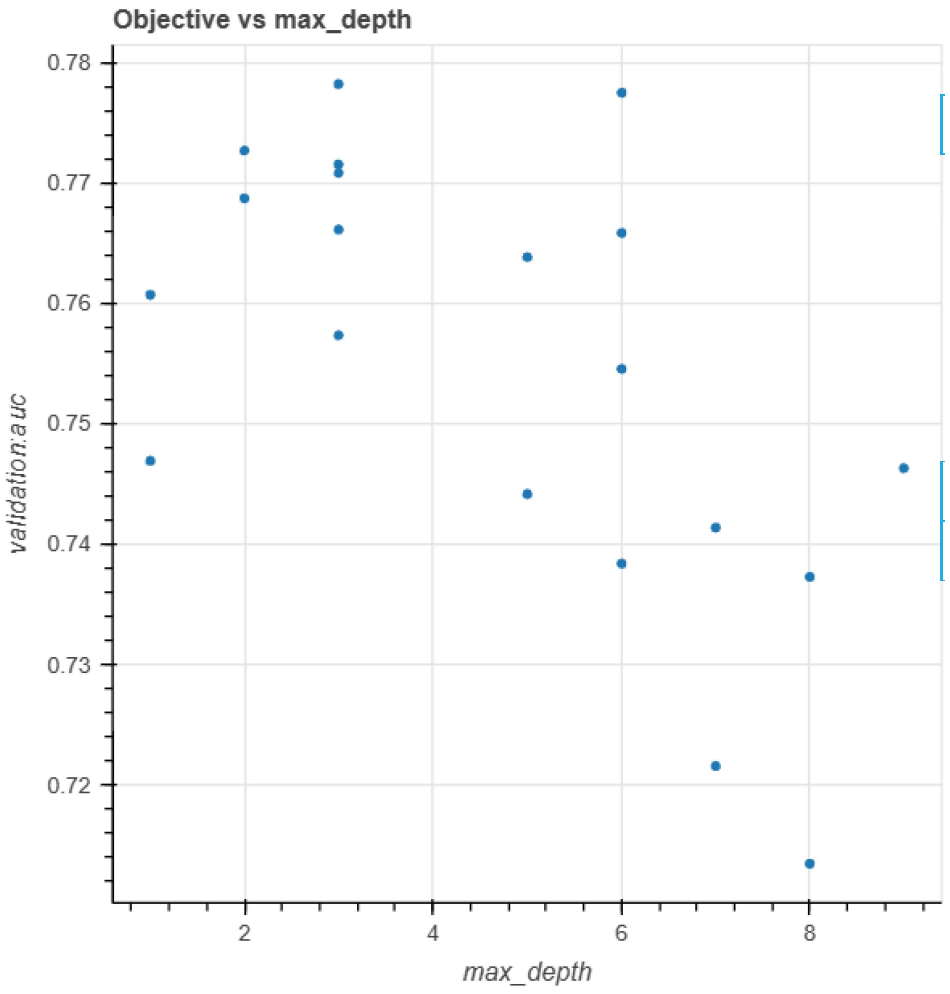
(https://bokeh.org)

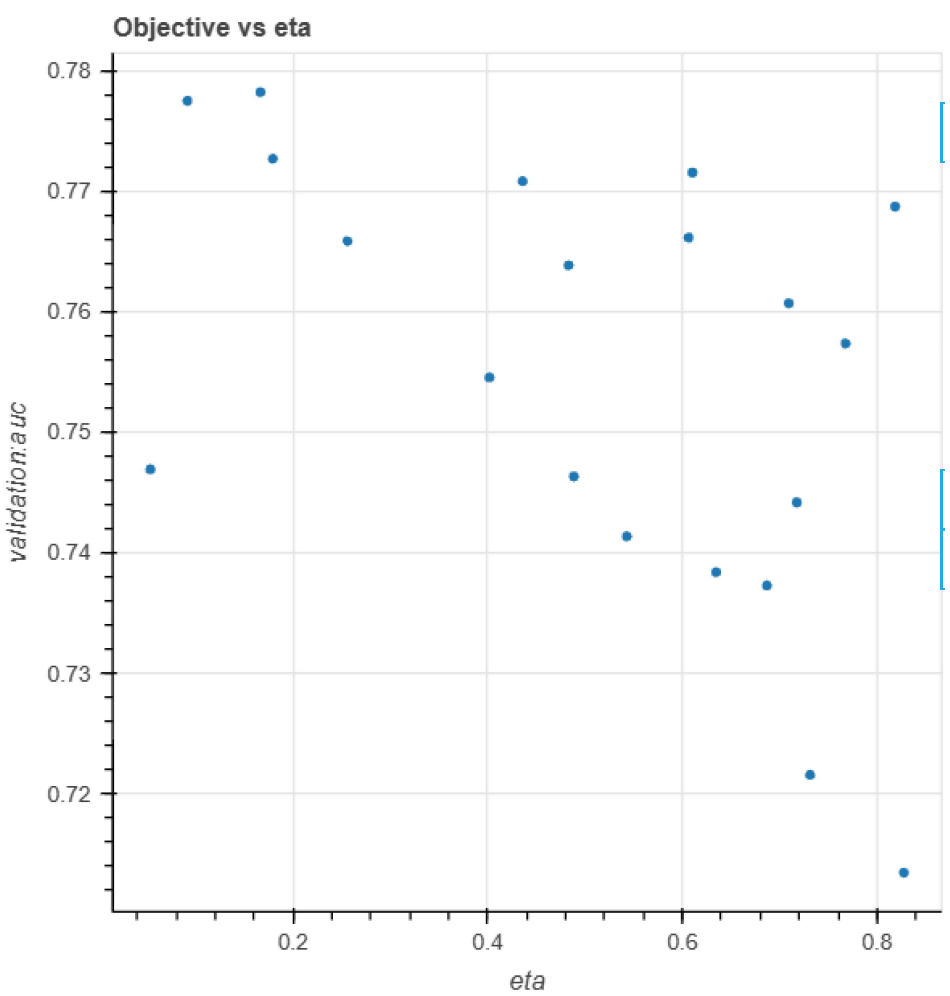
Loading BokehJS ...

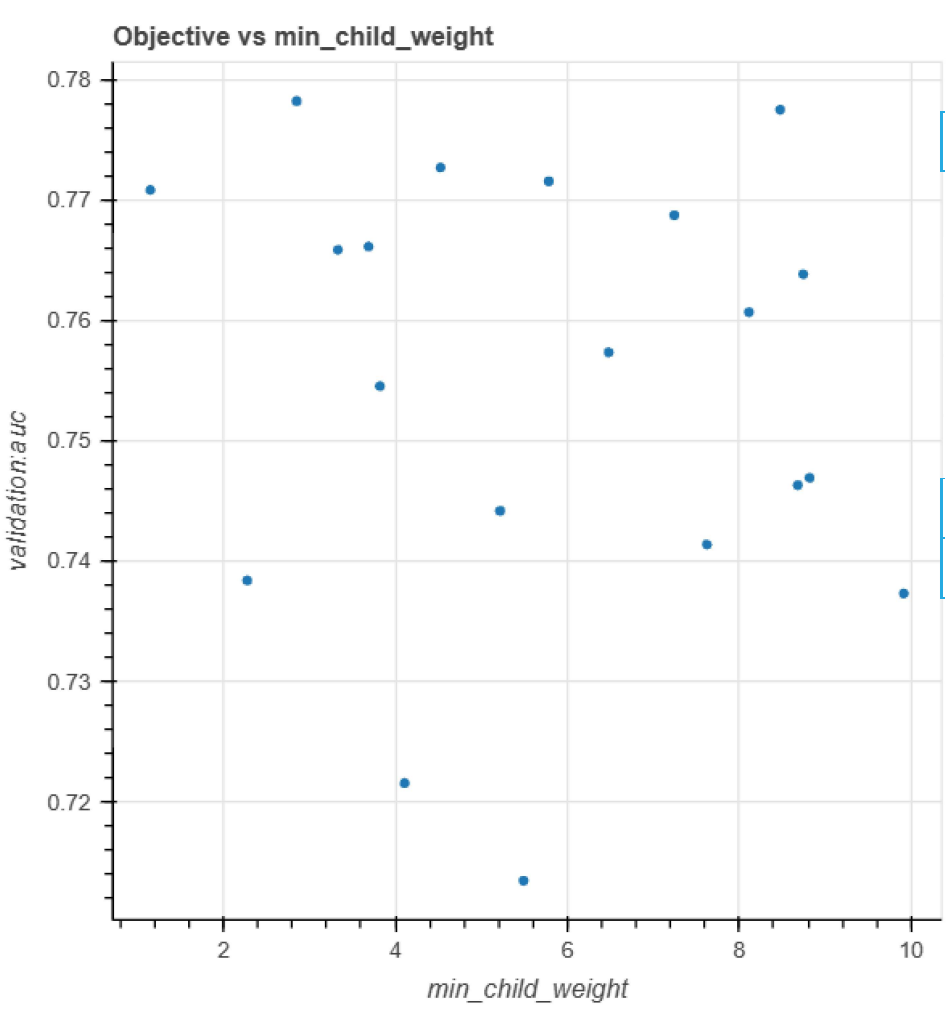


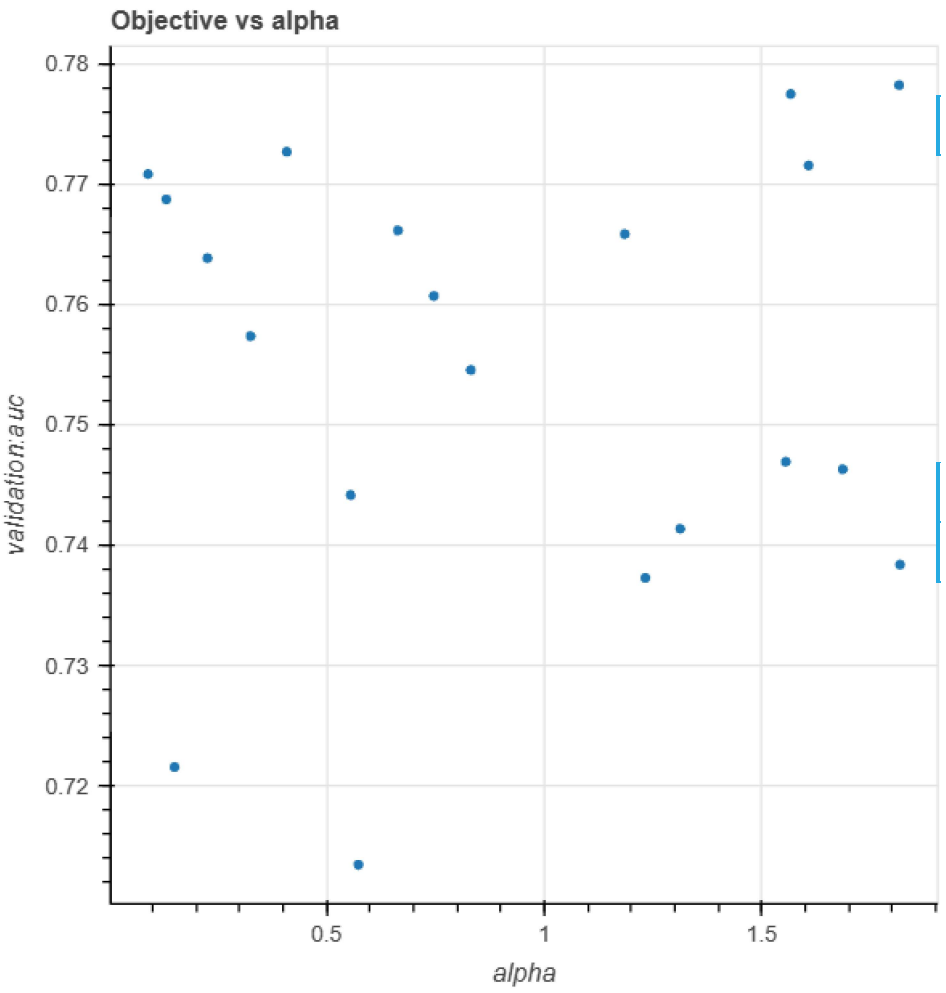
From the above graph it can be inferred that as the time passes the accuracy decreases. The highest accuracy achieved is in the initial time phase where max_depth=3, eta=0.167, min_child_weight=2.850 and alpha=1.816 which makes best suited parameters.

In [33]:
```
ranges = tuner.tuning_ranges
figures = []
for hp_name, hp_range in ranges.items():
    categorical_args = {}
    if hp_range.get("Values"):
        # This is marked as categorical.  Check if all options are actually numbers.
        def is_num(x):
            try:
                float(x)
                return 1
            except:
                return 0

        vals = hp_range["Values"]
        if sum([is_num(x) for x in vals]) == len(vals):
            # Bokeh has issues plotting a "categorical" range that's actually numeric, so plot as numeric
            print("Hyperparameter %s is tuned as categorical, but all values are numeric" % hp_name)
        else:
            # Set up extra options for plotting categoricals.  A bit tricky when they're actually numbers.
            categorical_args["x_range"] = vals

    # Now plot it
    p = figure(
        width=500,
        height=500,
        title="Objective vs %s" % hp_name,
        tools=hover.tools(),
        x_axis_label=hp_name,
        y_axis_label=objective_name,
        **categorical_args,
    )
    p.circle(source=df1, x=hp_name, y="FinalObjectiveValue")
    figures.append(p)
show(bokeh.layouts.Column(*figures))
```


Objective vs max_depth
validation:auc
0.78
0.77
0.76
0.75
0.74
0.73
0.72
2
4
6
8
max_depth


Objective vs eta
validation:auc
0.78
0.77
0.76
0.75
0.74
0.73
0.72
0.2
0.4
0.6
0.8
eta


Objective vs min_child_weight
validation:auc
0.78
0.77
0.76
0.75
0.74
0.73
0.72
2
4
6
8
10
min_child_weight



Grid Search

Description of Grid Search Code

- tuning_job_name = "xgboost-tuningjob-gs-" + strftime("%d-%H-%M-%S", gmtime()): This line creates a unique name for the tuning job, which is composed of a prefix "xgboost-tuningjob-gs-" and a timestamp generated using the strftime function. The timestamp is in the format dd-HH-MM-SS, where dd is the day of the month, HH is the hour (in 24-hour format), MM is the minute, and SS is the second. This ensures that each tuning job has a unique name.
- print(tuning_job_name): This line simply prints the name of the tuning job to the console.
- tuning_job_config = {...}: This line creates a dictionary object that contains the configuration for the tuning job. This dictionary has several keys:
- "ParameterRanges": This key contains a dictionary that specifies the ranges for the hyperparameters that will be tuned. In this case, there is only one type of hyperparameter:
- "CategoricalParameterRanges": This list contains dictionaries that define the values for each categorical hyperparameter. Each dictionary has two keys:
- "Name": The name of the hyperparameter.
- "Values": The list of values to search over for the hyperparameter.
- In this case, there are three categorical hyperparameters: "subsample", "colsample_bytree", and "max_depth". For each hyperparameter, a list of values to search over is provided.
- "ResourceLimits": This key contains a dictionary that specifies the maximum number of training jobs and parallel training jobs that can be run during the tuning job.
- "MaxNumberOfTrainingJobs": The maximum number of training jobs to run during the tuning job. In this case, it is set to 3x3x3=27, as there are 3 possible values for each of the 3 hyperparameters being tuned.
- "MaxParallelTrainingJobs": The maximum number of training jobs to run in parallel during the tuning job. In this case, it is set to 3, meaning that up to 3 training jobs can be run in parallel.
- "Strategy": This key specifies the tuning strategy to use. In this case, it is set to "Grid", which means that SageMaker will search over all combinations of hyperparameter values specified in the ParameterRanges dictionary.
- "HyperParameterTuningJobObjective": This key contains a dictionary that specifies the objective metric to optimize during the tuning job.
- "MetricName": The name of the objective metric. In this case, it is set to "validation:auc", which means that SageMaker will optimize the area under the ROC curve (AUC) on the validation set.
- "Type": The type of objective metric. In this case, it is set to "Maximize", which means that SageMaker will maximize the AUC on the validation set.

In [37]:
```
tuning_job_name = "xgboost-tuningjob-gs-" + strftime("%d-%H-%M-%S", gmtime())

print(tuning_job_name)

tuning_job_config = {
    "ParameterRanges": {
        "CategoricalParameterRanges": [
            {
                "Name": "subsample",
                "Values": ["0.3", "0.5", "0.7"]
            },
            {
                "Name": "colsample_bytree",
                "Values": ["0.3", "0.5", "0.7"]
            },
            {
                "Name": "max_depth",
                "Values": ["3", "5", "7"]
            }
        ],
    },
    "ResourceLimits": {"MaxNumberOfTrainingJobs": 3*3*3, "MaxParallelTrainingJobs": 3},
    "Strategy": "Grid",
    "HyperParameterTuningJobObjective": {"MetricName": "validation:auc", "Type": "Maximize"}
}
```

```
xgboost-tuningjob-gs-22-01-56-21
```

In [38]:
```
from sagemaker.image_uris import retrieve

training_image = retrieve(framework="xgboost", region=region, version="1.5-1")

s3_input_train = "s3://{}/{}/train".format(bucket, prefix)
s3_input_validation = "s3://{}/{}/validation/".format(bucket, prefix)

training_job_definition = {
    "AlgorithmSpecification": {"TrainingImage": training_image, "TrainingInputMode": "File"},
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_train,
                }
            },
        },
        {
            "ChannelName": "validation",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_validation,
                }
            },
        },
    ],
    "OutputDataConfig": {"S3OutputPath": "s3://{}/{}/output".format(bucket, prefix)},
    "ResourceConfig": {"InstanceCount": 1, "InstanceType": "ml.m4.xlarge", "VolumeSizeInGB": 10},
    "RoleArn": role,
    "StaticHyperParameters": {
        "eval_metric": "auc",
        "num_round": "100",
        "objective": "binary:logistic",
        "rate_drop": "0.3",
        "tweedie_variance_power": "1.4",
    },
    "StoppingCondition": {"MaxRuntimeInSeconds": 43200},
}
```

In [39]:
```
smclient.create_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuning_job_name,
    HyperParameterTuningJobConfig=tuning_job_config,
    TrainingJobDefinition=training_job_definition,
)
```

Out[39]:
```
{'HyperParameterTuningJobArn': 'arn:aws:sagemaker:ca-central-1:224670572127:hyper-parameter-tuning-job/xgboost-tuninjob-gs-22-01-56-21',
 'ResponseMetadata': {'RequestId': '129c5286-4391-4fb0-b2f2-439e652d2dd5',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '129c5286-4391-4fb0-b2f2-439e652d2dd5',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '136',
   'date': 'Wed, 22 Mar 2023 01:56:27 GMT'},
  'RetryAttempts': 0}}
```

In [41]:
```
smclient.describe_hyper_parameter_tuning_job(HyperParameterTuningJobName=tuning_job_name)[
    "HyperParameterTuningJobStatus"
]
```

Out[41]:
```
'Completed'
```

In [42]:
```
smclient.describe_hyper_parameter_tuning_job(HyperParameterTuningJobName=tuning_job_name)['HyperParameterTuningJobName
```

Out[42]:
```
'xgboost-tuningjob-gs-22-01-56-21'
```

In [43]:
```
# run this cell to check current status of hyperparameter tuning job
tuning_job_result = smclient.describe_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuning_job_name
)

status = tuning_job_result["HyperParameterTuningJobStatus"]
if status != "Completed":
    print("Reminder: the tuning job has not been completed.")

job_count = tuning_job_result["TrainingJobStatusCounters"]["Completed"]
print("%d training jobs have completed" % job_count)

objective = tuning_job_result["HyperParameterTuningJobConfig"]["HyperParameterTuningJobObjective"]
is_minimize = objective["Type"] != "Maximize"
objective_name = objective["MetricName"]
```

```
27 training jobs have completed
```

In [44]:
```
from pprint import pprint

if tuning_job_result.get("BestTrainingJob", None):
    print("Best model found so far:")
    pprint(tuning_job_result["BestTrainingJob"])
else:
    print("No training jobs have reported results yet.")
```

```
Best model found so far:
{'CreationTime': datetime.datetime(2023, 3, 22, 2, 0, 51, tzinfo=tzlocal()),
 'FinalHyperParameterTuningJobObjectiveMetric': {'MetricName': 'validation:auc',
                                                 'Value': 0.7745000123977661},
 'ObjectiveStatus': 'Succeeded',
 'TrainingEndTime': datetime.datetime(2023, 3, 22, 2, 1, 31, tzinfo=tzlocal()),
 'TrainingJobArn': 'arn:aws:sagemaker:ca-central-1:224670572127:training-job/xgboost-tuningjob-gs-22-01-56-21-009-d901384c',
 'TrainingJobName': 'xgboost-tuningjob-gs-22-01-56-21-009-d901384c',
 'TrainingJobStatus': 'Completed',
 'TrainingStartTime': datetime.datetime(2023, 3, 22, 2, 0, 54, tzinfo=tzlocal()),
 'TunedHyperParameters': {'colsample_bytree': '0.7',
                          'max_depth': '3',
                          'subsample': '0.7'}}
```

In [45]:
```
import pandas as pd

tuner = sagemaker.HyperparameterTuningJobAnalytics(tuning_job_name)

full_df2 = tuner.dataframe()

if len(full_df2) > 0:
    df2 = full_df2[full_df2["FinalObjectiveValue"] > -float("inf")]
    if len(df2) > 0:
        df2 = df2.sort_values("FinalObjectiveValue", ascending=is_minimize)
        print("Number of training jobs with valid objective: %d" % len(df2))
        print({"lowest": min(df2["FinalObjectiveValue"]), "highest": max(df2["FinalObjectiveValue"])})
        pd.set_option("display.max_colwidth", None)  # Don't truncate TrainingJobName
    else:
        print("No training jobs have reported valid results yet.")

full_df2
```

```
Number of training jobs with valid objective: 27
{'lowest': 0.7267000079154968, 'highest': 0.7745000123977661}
```

Out[45]:

	colsample_bytree	max_depth	subsample	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	Traini
0	0.7	7.0	0.7	xgboost-tuningjob-gs-22-01-56-21-027-0f3859a2	Completed	0.74586	2023-03-22 02:05:56+00:00	2023-03-22 02:06:32+00:00	
1	0.7	7.0	0.5	xgboost-tuningjob-gs-22-01-56-21-026-c78b31c3	Completed	0.73673	2023-03-22 02:05:52+00:00	2023-03-22 02:06:28+00:00	
2	0.7	7.0	0.3	xgboost-tuningjob-gs-22-01-56-21-025-385c353a	Completed	0.72670	2023-03-22 02:05:51+00:00	2023-03-22 02:06:27+00:00	
3	0.5	7.0	0.7	xgboost-tuningjob-gs-22-01-56-21-024-edfe4f70	Completed	0.74422	2023-03-22 02:05:04+00:00	2023-03-22 02:05:37+00:00	
4	0.5	7.0	0.5	xgboost-tuningjob-gs-22-01-56-21-023-8cc94c4c	Completed	0.74302	2023-03-22 02:05:07+00:00	2023-03-22 02:05:39+00:00	
5	0.5	7.0	0.3	xgboost-tuningjob-gs-22-01-56-21-022-d0d79a46	Completed	0.73768	2023-03-22 02:05:01+00:00	2023-03-22 02:05:37+00:00	
6	0.3	7.0	0.7	xgboost-tuningjob-gs-22-01-56-21-021-c1115d0c	Completed	0.74921	2023-03-22 02:04:01+00:00	2023-03-22 02:04:38+00:00	
7	0.3	7.0	0.5	xgboost-tuningjob-gs-22-01-56-21-020-4fd57ddc	Completed	0.74718	2023-03-22 02:04:10+00:00	2023-03-22 02:04:47+00:00	
8	0.3	7.0	0.3	xgboost-tuningjob-gs-22-01-56-21-019-fba19ad3	Completed	0.74596	2023-03-22 02:03:56+00:00	2023-03-22 02:04:27+00:00	
9	0.7	5.0	0.7	xgboost-tuningjob-gs-22-01-56-21-018-a973f4d9	Completed	0.76387	2023-03-22 02:03:12+00:00	2023-03-22 02:03:49+00:00	
10	0.7	5.0	0.5	xgboost-tuningjob-gs-22-01-56-21-017-64fda631	Completed	0.75205	2023-03-22 02:03:15+00:00	2023-03-22 02:03:52+00:00	
11	0.7	5.0	0.3	xgboost-tuningjob-gs-22-01-56-21-016-95bb64da	Completed	0.74765	2023-03-22 02:03:09+00:00	2023-03-22 02:03:41+00:00	
12	0.5	5.0	0.7	xgboost-tuningjob-gs-22-01-56-21-015-13629bc0	Completed	0.76152	2023-03-22 02:02:28+00:00	2023-03-22 02:03:05+00:00	
13	0.5	5.0	0.5	xgboost-tuningjob-gs-22-01-56-21-014-6fdd3670	Completed	0.76086	2023-03-22 02:02:21+00:00	2023-03-22 02:02:58+00:00	
14	0.5	5.0	0.3	xgboost-tuningjob-gs-22-01-56-21-013-d5307327	Completed	0.74891	2023-03-22 02:02:10+00:00	2023-03-22 02:02:42+00:00	
15	0.3	5.0	0.7	xgboost-tuningjob-gs-22-01-56-21-012-896e23d7	Completed	0.76481	2023-03-22 02:01:39+00:00	2023-03-22 02:02:21+00:00	
16	0.3	5.0	0.5	xgboost-tuningjob-gs-22-01-56-21-011-afeae1c3	Completed	0.76123	2023-03-22 02:01:36+00:00	2023-03-22 02:02:13+00:00	
17	0.3	5.0	0.3	xgboost-tuningjob-gs-22-01-56-21-010-6260681e	Completed	0.75717	2023-03-22 02:01:26+00:00	2023-03-22 02:02:03+00:00	
18	0.7	3.0	0.7	xgboost-tuningjob-gs-22-01-56-21-009-d901384c	Completed	0.77450	2023-03-22 02:00:54+00:00	2023-03-22 02:01:31+00:00	
19	0.7	3.0	0.5	xgboost-tuningjob-gs-22-01-56-21-008-beafc5f8	Completed	0.76628	2023-03-22 02:00:50+00:00	2023-03-22 02:01:27+00:00	
20	0.7	3.0	0.3	xgboost-tuningjob-gs-22-01-56-21-007-06102222	Completed	0.77016	2023-03-22 02:00:35+00:00	2023-03-22 02:01:12+00:00	
21	0.5	3.0	0.7	xgboost-tuningjob-gs-22-01-56-21-006-15fc9090	Completed	0.76985	2023-03-22 02:00:09+00:00	2023-03-22 02:00:46+00:00	
22	0.5	3.0	0.5	xgboost-tuningjob-gs-22-01-56-21-005-b6f1b738	Completed	0.77267	2023-03-22 02:00:00+00:00	2023-03-22 02:00:42+00:00	
23	0.5	3.0	0.3	xgboost-tuningjob-gs-22-01-56-21-004-6b4028d4	Completed	0.76590	2023-03-22 01:59:48+00:00	2023-03-22 02:00:20+00:00	
24	0.3	3.0	0.7	xgboost-tuningjob-gs-22-01-56-21-003-c5b53008	Completed	0.77103	2023-03-22 01:57:53+00:00	2023-03-22 01:59:46+00:00	
25	0.3	3.0	0.5	xgboost-tuningjob-gs-22-01-56-21-002-5d003923	Completed	0.77111	2023-03-22 01:57:51+00:00	2023-03-22 01:59:58+00:00	

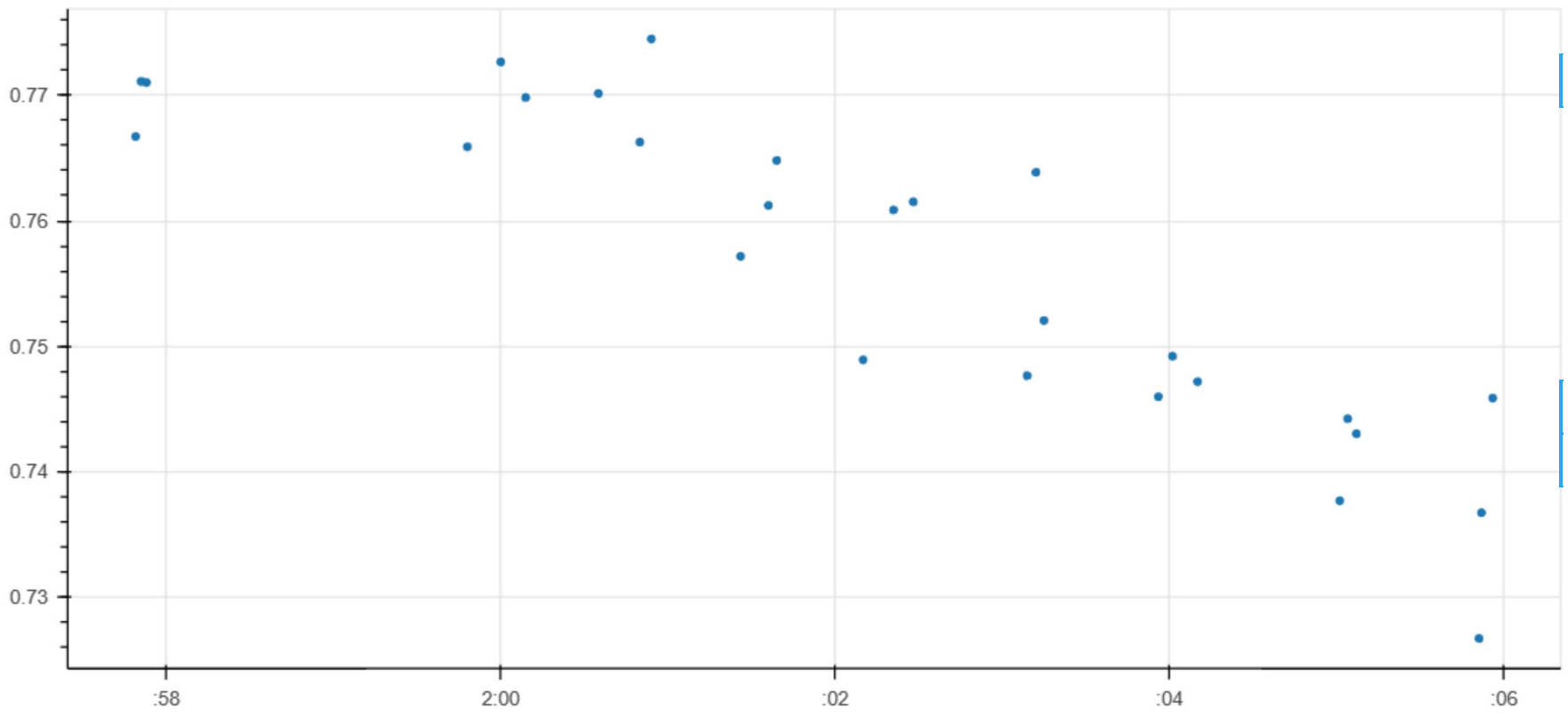
	colsample_bytree	max_depth	subsample	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	Traini
26	0.3	3.0	0.3	xgboost-tuningjob-gs-22-01-56-21-001-4f047d59	Completed	0.76672	2023-03-22 01:57:49+00:00	2023-03-22 01:59:36+00:00	

It came to notice that the highest accuracy achieved via. Grid search is of 0.7745 and the lowest accuracy achieved via. Grid search is of 0.7267 after running 27 hyper tuning jobs.

In [46]:
```
import bokeh
import bokeh.io

bokeh.io.output_notebook()
from bokeh.plotting import figure, show
from bokeh.models import HoverTool


class HoverHelper:
    def __init__(self, tuning_analytics):
        self.tuner = tuning_analytics

    def hovertool(self):
        tooltips = [
            ("FinalObjectiveValue", "@FinalObjectiveValue"),
            ("TrainingJobName", "@TrainingJobName"),
        ]
        for k in self.tuner.tuning_ranges.keys():
            tooltips.append((k, "@{%s}" % k))

        ht = HoverTool(tooltips=tooltips)
        return ht

    def tools(self, standard_tools="pan,crosshair,wheel_zoom,zoom_in,zoom_out,undo,reset"):
        return [self.hovertool(), standard_tools]


hover = HoverHelper(tuner)

p = figure(width=900, height=400, tools=hover.tools(), x_axis_type="datetime")
p.circle(source=df2, x="TrainingStartTime", y="FinalObjectiveValue")
show(p)
```

(https://bokeh.org)

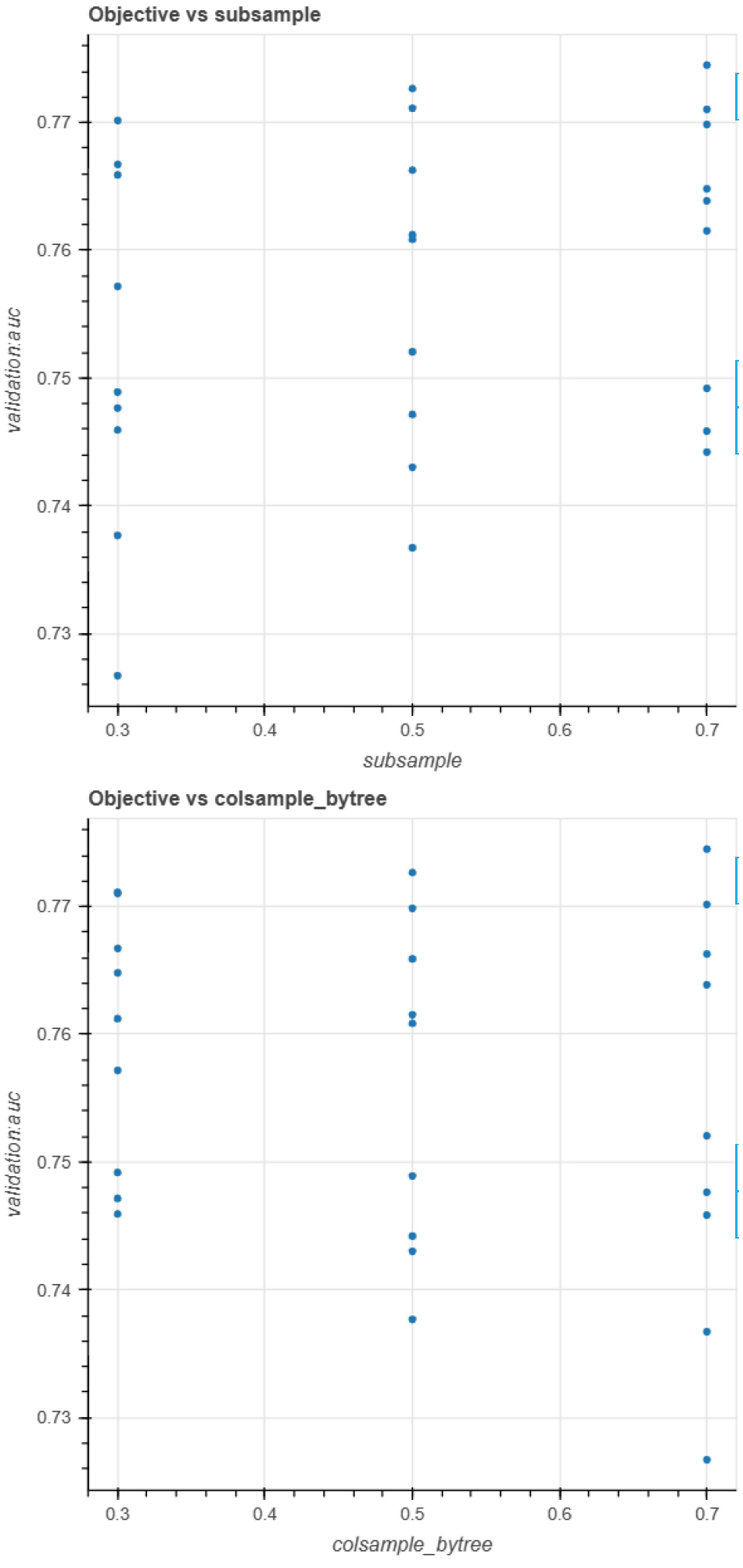
BokehJS 3.0.3 successfully loaded.

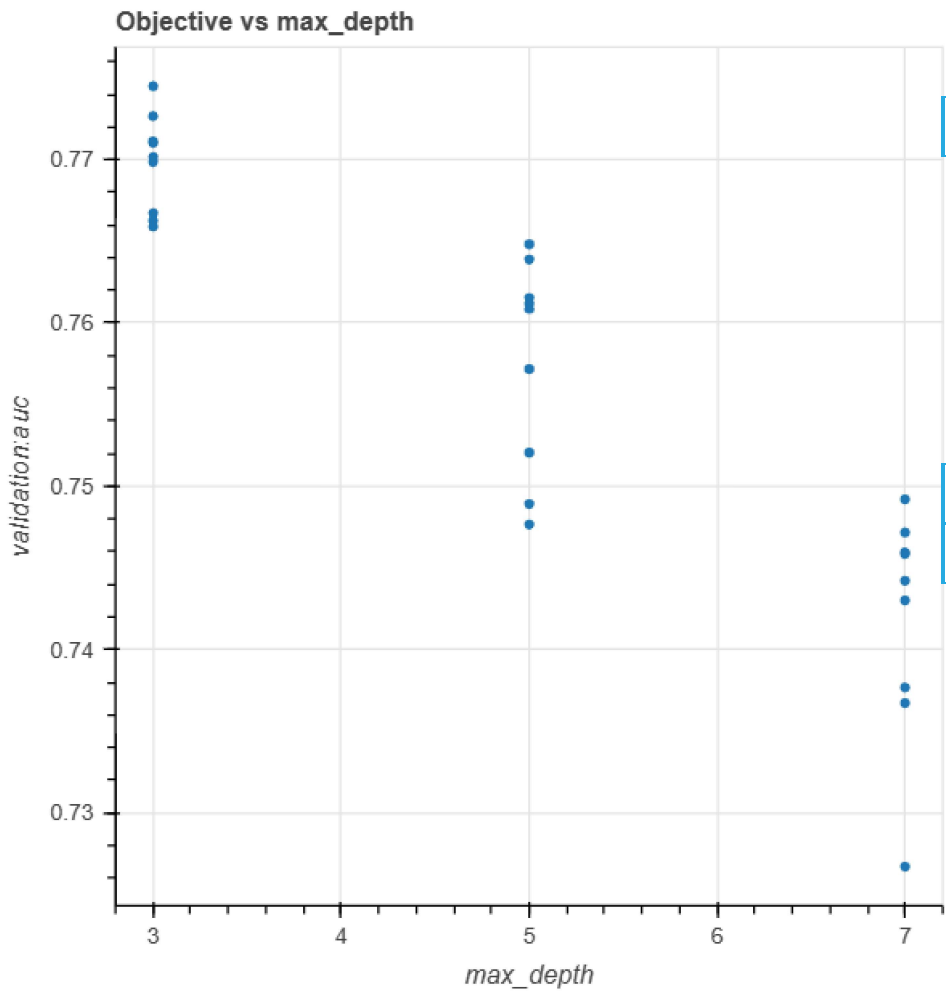


From the above graph it can be inferred that as the time passes the accuracy decreases. The highest accuracy achieved is in the initial time phase where subsample = 0.7, colsample_bytree = 0.7, and max_depth=3 which makes best suited parameters.

In [47]:
```
ranges = tuner.tuning_ranges
figures = []
for hp_name, hp_range in ranges.items():
    categorical_args = {}
    if hp_range.get("Values"):
        # This is marked as categorical.  Check if all options are actually numbers.
        def is_num(x):
            try:
                float(x)
                return 1
            except:
                return 0

        vals = hp_range["Values"]
        if sum([is_num(x) for x in vals]) == len(vals):
            # Bokeh has issues plotting a "categorical" range that's actually numeric, so plot as numeric
            print("Hyperparameter %s is tuned as categorical, but all values are numeric" % hp_name)
        else:
            # Set up extra options for plotting categoricals.  A bit tricky when they're actually numbers.
            categorical_args["x_range"] = vals

    # Now plot it
    p = figure(
        width=500,
        height=500,
        title="Objective vs %s" % hp_name,
        tools=hover.tools(),
        x_axis_label=hp_name,
        y_axis_label=objective_name,
        **categorical_args,
    )
    p.circle(source=df2, x=hp_name, y="FinalObjectiveValue")
    figures.append(p)
show(bokeh.layouts.Column(*figures))
```

```
Hyperparameter subsample is tuned as categorical, but all values are numeric
Hyperparameter colsample_bytree is tuned as categorical, but all values are numeric
Hyperparameter max_depth is tuned as categorical, but all values are numeric
```


Objective vs subsample
validation:auc
0.77
0.76
0.75
0.74
0.73
0.3
0.4
0.5
0.6
0.7
subsample
Objective vs colsample_bytree
validation:auc
0.77
0.76
0.75
0.74
0.73
0.3
0.4
0.5
0.6
0.7
colsample_bytree



Comparison

Bayesian Search vs Random Search vs Grid Search

- In terms of Bayesian search the lowest and highest accuracy achieved was 0.5 and 0.7778 respectively.
- In terms of Random search the lowest and highest accuracy achieved was 0.713 and 0.7781 respectively.
- In terms of Grid search the lowest and highest accuracy achieved was 0.7267 and 0.7745 respectively.
- Suprisingly it was noticed for grid search and random search that as the time passed the accuracy decreased gradually, but this was not the case for Bayesian search. For bayesian search it was noticed that after .28 time there was sudden drop in accuracy to 0.5 from 0.77.

Among the three startegies, I found random searche to be more relevant in terms of accuracy when compared to Grid and Bayesian Search.

Deploy the best model (Optional)

Now that we have got the best model, we can deploy it to an endpoint. Please refer to other SageMaker sample notebooks or SageMaker documentation to see how to deploy a model.