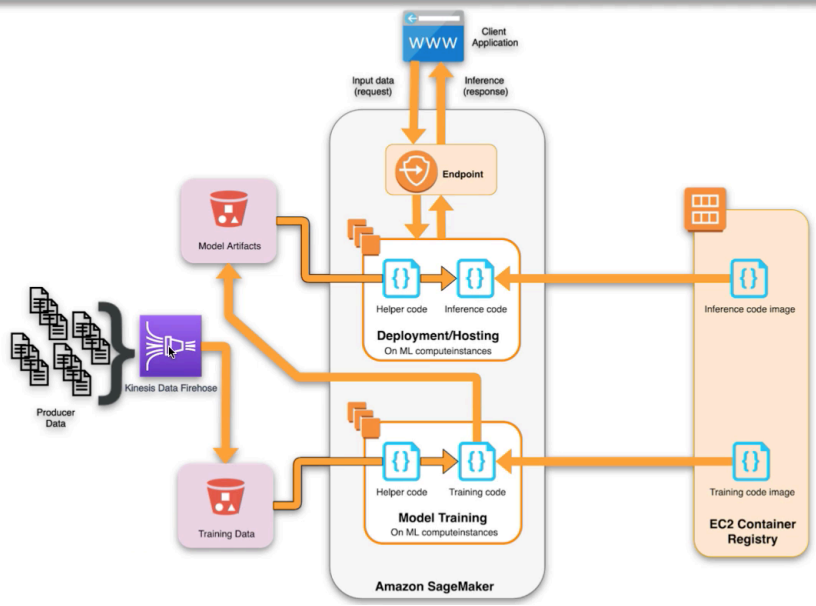
Whizlabs - ML Specialty Exam Course - Modeling - part 1

AWS Machine Learning - Modeling

The Machine Learning Cycle - Training and Evaluating the Model

- ❏ Machine learning with SageMaker
  - ❏ Generate example data to use in training
  - ❏ Train the model to make predictions, or inferences
    1. Use an algorithm and example data to train the model
    2. Evaluate the model for inference accuracy
    3. Integrate model into your application to generate inferences in real time and at scale

Train the Model



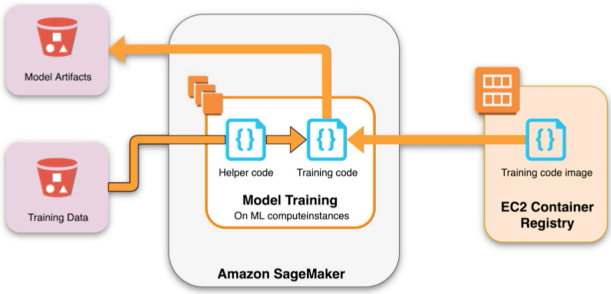
Produce data via different means ready in S3
We then produce our model by writing training code, storing training image in EC2 Container registry

The training produces inference code which is then stored in as a model artifact in S3.
With inference code, we can pull the serialized model and expose the inference as

end point (or use as Bach)

Training Algorithm Options

- Training algorithm options
  - Use a SageMaker built-in algorithm
  - Use the SageMaker debugger
  - Use Apache Spark with SageMaker
  - Use custom deep learning code
  - Use your own algorithms
  - Use an algorithm from the AWS Marketplace



2 deployment types for SageMaker

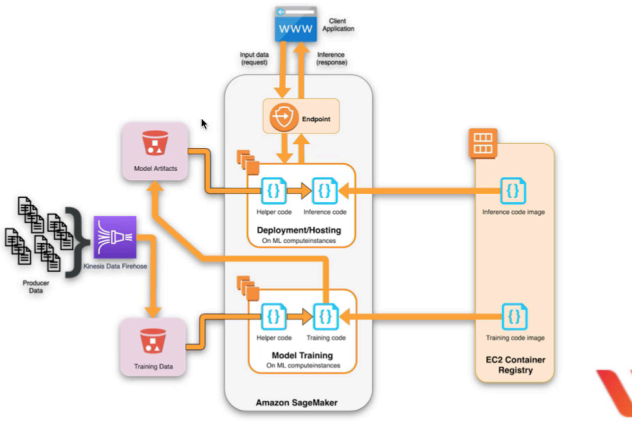
AWS Machine Learning - Modeling

Deploy the Model

- Deploy in one of two ways
  - Persistent endpoint to get one prediction at a time: SageMaker hosting services
  - Get predictions for an entire dataset: SageMaker batch transform

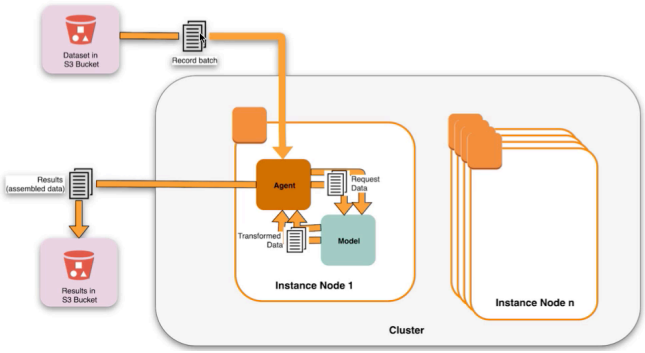
Deploy the Model - SageMaker Hosting Services

- SageMaker provides an HTTPS endpoint, making your model available for inference requests
  - Three steps
    1. Create model in SageMaker
    2. Create an endpoint configuration
    3. Create HTTPS endpoint



Deploy the Model - SageMaker Batch Transform

- SageMaker batch transform provides inferences for an entire dataset
  - Three steps
    1. Create batch transform job using trained model and dataset
    2. Run the batch transform job
    3. SageMaker saves in a results S3 bucket



Evaluate the Model

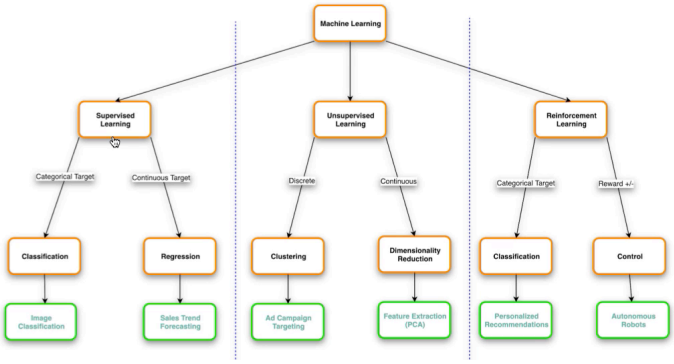
- After training and deploying the model, evaluate it to determine performance and accuracy
- Often generate multiple models with different algorithms/hyperparameters and evaluate each
- Two different validation approaches
  1. Offline testing: use historical data to send requests to the model for inferences
  2. Online testing with live data: use production variants
- Options for offline evaluation
  - Holdout set: set aside a subset of the data for evaluation after training
  - K-fold validation: split the example dataset into k parts, treat each as a holdout set for k training runs

Modeling Concepts

AWS Machine Learning - Modeling Concepts

The Three Types of Models

- Three types of models
  1. Supervised machine learning
  2. Unsupervised machine learning
  3. Reinforcement machine learning



Supervised Machine Learning

- Supervised machine learning
  - Target variable (dependent variable) to be predicted from independent variables
  - Two types
    - Classification: categorize vectors of independent variables into buckets represented by the dependent variable
    - Regression: identifying patterns in the vectors of independent variables and calculating the prediction of independent variable outcomes
  - Most widely used
    - Linear Regression
    - Logistical Regression
    - Random Forest
    - Gradient Boosted Trees
    - Nearest Neighbor
    - Support Vector Machines (SVM)
    - Neural Networks
    - Decision Trees
    - Naive Bayes

Unsupervised Machine Learning

- Unsupervised machine learning
  - No target or outcome to predict, used for clustering and dimensionality reduction
  - Two types
    - Clustering: segment the observations into meaningful groups or clusters based on patterns in the observations
    - Dimensionality reduction: distill the relevant information for the observations while reducing the number of features
  - Most widely used
    - k-means Clustering
    - t-Distributed Stochastic Neighbor Embedding
    - Principal Component Analysis (PCA)

Reinforcement Machine Learning

- Reinforcement machine learning
  - Trained to make specific decisions, trains itself continuously in an environment using trial and error
  - Reward signals generated when a task is performed, two types:
    - Positive reward signal: encourages continuing performance of the task
    - Negative reward signal: penalizes for performing the task
  - Most widely used
    - Q-Learning
    - Temporal Difference (TD)
    - Monte-Carlo Tree Search
    - Asynchronous Actor-Critic Agents (A3C)

How to Choose the correct Machine Learning approach?

Choosing the Correct Machine Learning Approach

- ❏ Discrete or Continuous?
- ❏ Target variable or not?

Data Type	Supervised Learning	Unsupervised Learning
Discrete	Classification	Clustering
Continuous	Regression	Dimensionality Reduction

- ❏ Optimizing an objective function through interacting with an environment: Reinforcement Learning

Hyperparameters

- ❏ A hyperparameter is a parameter whose value is set before the learning process
  - ❏ Two types
    - ❏ Model hyperparameters: influence the performance of the model
    - ❏ Algorithm hyperparameters: affect the speed and quality of the learning process
  - ❏ Each algorithm has its own set of hyperparameter settings
  - ❏ Automatic model tuning, a.k.a. Hyperparameter Tuning
    - ❏ Finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of specified hyperparameters
    - ❏ Then selects the hyperparameter values that result in a model that performs the best based on a selected metric
    - ❏ Two approaches
      - ❏ Random Search: chooses a random combination of values for each training job
      - ❏ Bayesian Search: performs hyperparameter tuning as a regression problem

Training and Evaluating

- ❏ Training is done via a training job that includes
  - ❏ URL of the S3 bucket that holds the training data
  - ❏ Compute resources used for model training (managed by SageMaker)
  - ❏ URL of the S3 bucket to store the output of the job (model artifacts)
  - ❏ ECS registry path where the training job code is stored
- ❏ After training job is created, SageMaker launches the ML instances and uses the training code and training data to train the model and stores the resulting artifacts in the artifact S3 bucket
- ❏ Use online or offline testing to evaluate performance and accuracy

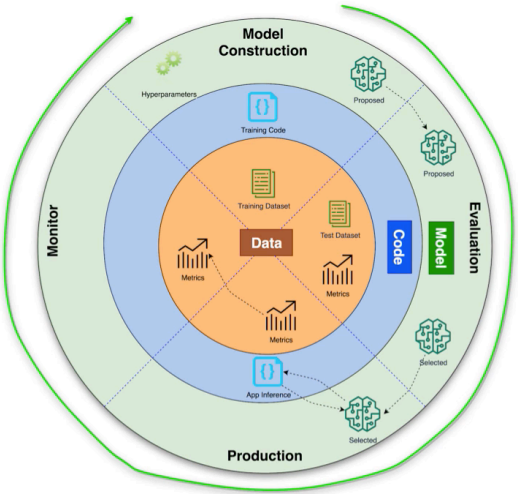
Deploying and Continuously Evaluating

- ❏ Use SageMaker Model Monitor to continuously monitor model quality
  - ❏ Set alerts for deviations in quality such as data drift
  - ❏ Steps of Model Monitoring
    - ❏ Capture data: enable endpoint to capture data from incoming requests and resulting predictions
    - ❏ Create a baseline: from the data used to train the model
    - ❏ Schedule monitoring jobs: create monitoring schedule specifying data to collect, how to interpret the data
    - ❏ Interpret results: compare latest data with baseline, watch for violations and metric notifications in CloudWatch

AWS Machine Learning - Modeling Concepts

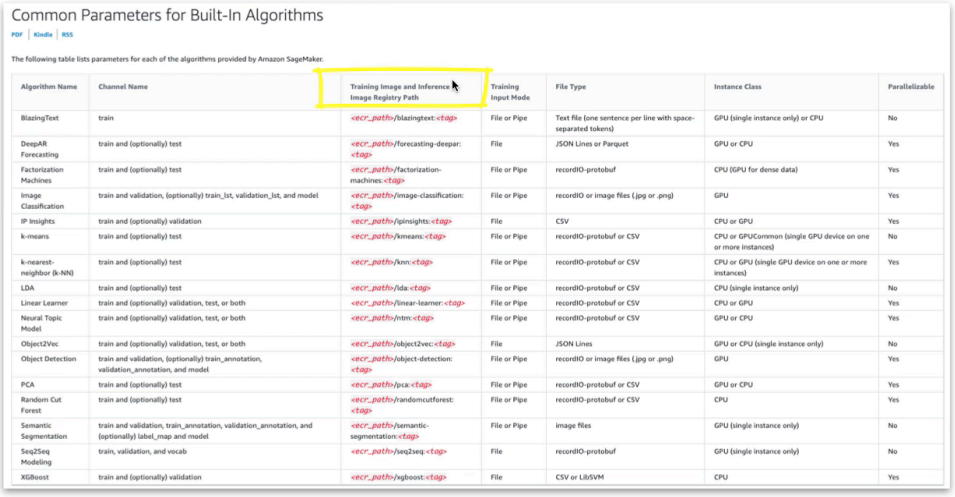
Deploying and Continuously Evaluating

- ❏ Use SageMaker Model Monitor to continuously monitor model quality



Training the Machine Learning Model - Training & Inference Instances

- Training and inference instances for SageMaker built-in algorithms



Common Parameters for Built-In Algorithms

PDF | Kindle | RSS

The following table lists parameters for each of the algorithms provided by Amazon SageMaker.

Algorithm Name	Channel Name	Training Image and Inference Image Registry Path	Training Input Mode	File Type	Instance Class	Parallelizable
BlazingText	train	<ecr_path>/blazingtext:<tag>	File or Pipe	Text file (one sentence per line with space-separated tokens)	GPU (single instance only) or CPU	No
DeepAR Forecasting	train and (optionally) test	<ecr_path>/forecasting-deepar:<tag>	File	JSON Lines or Parquet	GPU or CPU	Yes
Factorization Machines	train and (optionally) test	<ecr_path>/factorization-machines:<tag>	File or Pipe	recordIO-protobuf	CPU (GPU for dense data)	Yes
Image Classification	train and validation, (optionally) train_lst, validation_lst, and model	<ecr_path>/image-classification:<tag>	File or Pipe	recordIO or image files (.jpg or .png)	GPU	Yes
IP Insights	train and (optionally) validation	<ecr_path>/ipinsights:<tag>	File	CSV	CPU or GPU	Yes
k-means	train and (optionally) test	<ecr_path>/kmeans:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU or GPUCommon (single GPU device on one or more instances)	No
k-nearest-neighbor (k-NN)	train and (optionally) test	<ecr_path>/knn:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU or GPU (single GPU device on one or more instances)	Yes
LDA	train and (optionally) test	<ecr_path>/lda:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU (single instance only)	No
Linear Learner	train and (optionally) validation, test, or both	<ecr_path>/linear-learner:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU or GPU	Yes
Neural Topic Model	train and (optionally) validation, test, or both	<ecr_path>/ntm:<tag>	File or Pipe	recordIO-protobuf or CSV	GPU or CPU	Yes
Object2Vec	train and (optionally) validation, test, or both	<ecr_path>/object2vec:<tag>	File	JSON Lines	GPU or CPU (single instance only)	No
Object Detection	train and validation, (optionally) train_annotation, validation_annotation, and model	<ecr_path>/object-detection:<tag>	File or Pipe	recordIO or image files (.jpg or .png)	GPU	Yes
PCA	train and (optionally) test	<ecr_path>/pca:<tag>	File or Pipe	recordIO-protobuf or CSV	GPU or CPU	Yes
Random Cut Forest	train and (optionally) test	<ecr_path>/randomcutforest:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU	Yes
Semantic Segmentation	train and validation, train_annotation, validation_annotation, and (optionally) label_map and model	<ecr_path>/semantic-segmentation:<tag>	File or Pipe	image files	GPU (single instance only)	No
Seq2Seq Modeling	train, validation, and vocab	<ecr_path>/seq2seq:<tag>	File	recordIO-protobuf	GPU (single instance only)	No
XGBoost	train and (optionally) validation	<ecr_path>/xgboost:<tag>	File	CSV or LibSVM	CPU	Yes

9 steps of Training a ML Model

AWS Machine Learning - Train Machine Learning Models

The Steps of Training a Machine Learning Model

- Training model steps
  1. Gather/engineer data into your dataset
  2. Randomize the dataset
  3. Split the dataset into train and test datasets
  4. Choose best algorithm
  5. Load container for chosen model
  6. Manage compute capacity
  7. Create an instance of chosen model
  8. Define the model's hyperparameter values
  9. Train the model

The Steps of Training a Machine Learning Model - Gather/Engineer

1. Gather/engineer data into your dataset

```
bucket_name = 'your-s3-bucket-name' # <--- CHANGE THIS VARIABLE TO A UNIQUE NAME FOR YOUR BUCKET
s3 = boto3.resource('s3')
try:
    if  my_region == 'us-east-1':
      s3.create_bucket(Bucket=bucket_name)
    else:
      s3.create_bucket(Bucket=bucket_name, CreateBucketConfiguration={ 'LocationConstraint': my_region })
    print('S3 bucket created successfully')
except Exception as e:
    print('S3 error: ',e)
```

Retrieve training dataset (csv) from website
Read the csv into a Pandas data frame

```
try:
  urllib.request.urlretrieve ("https://d1.awsstatic.com/tmt/build-train-deploy-machine-learning-model-sagemaker/bank_clean.27f01fbbdf43271788427f3682996ae29ceca05d.csv", "bank_clean.csv")
  print('Success: downloaded bank_clean.csv.')
except Exception as e:
  print('Data load error: ',e)

try:
  model_data = pd.read_csv('./bank_clean.csv',index_col=0)
  print('Success: Data loaded into dataframe.')
except Exception as e:
    print('Data load error: ',e)
```

Split dataset into Train/Test dataset and randomize it

2. Randomize the dataset
3. Split the dataset into train and test datasets

```
train_data, test_data = np.split(model_data.sample(frac=1, random_state=1729), [int(0.7 * len(model_data))])
print(train_data.shape, test_data.shape)
```

Choose best algorithm
Reminder - Define first a role to access the containers that hold our algorithm

4. Choose best algorithm

```
# Define IAM role
role = get_execution_role()
prefix = 'sagemaker/DEMO-xgboost-dm'
containers = {'us-west-2': '433757028032.dkr.ecr.us-west-2.amazonaws.com/xgboost:latest',
              'us-east-1': '811284229777.dkr.ecr.us-east-1.amazonaws.com/xgboost:latest',
              'us-east-2': '825641698319.dkr.ecr.us-east-2.amazonaws.com/xgboost:latest',
              'eu-west-1': '685385470294.dkr.ecr.eu-west-1.amazonaws.com/xgboost:latest'} # each region has its XGBoost container
my_region = boto3.session.Session().region_name # set the region of the instance
print("Success - the MySageMakerInstance is in the " + my_region + " region. You will use the " + containers[my_region] + " container for your SageMaker endpoint.")
```

Load container for chosen model - in that case: XGBoost
And define the compute: m4.XL and specify instance count, using the Estimator()
Pass some hyper-parameters to the xib instance of our model with an objective of:

"binary:logistic" for a binary classification

5. Load container for chosen model
6. Manage compute capacity
7. Create an instance of chosen model
8. Define the model's hyperparameter values

```
sess = sagemaker.Session()
xgb = sagemaker.estimator.Estimator(containers[my_region],role, train_instance_count=1, train_instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(bucket_name, prefix),sagemaker_session=sess)
xgb.set_hyperparameters(max_depth=5,eta=0.2,gamma=4,min_child_weight=6,subsample=0.8,silent=0,objective='binary:logistic',num_round=100)
```

Last, we train the model

9. Train the model

```
xgb.fit({'train': s3_input_train})
```

Train Machine Learning Models - Lab

The notebook is available at:
https://www.whizlabs.com/learn/media/2020/03/23/files_train-deploy-evaluate-model.ipynb
Along with the dataset:

```
In [1]: # import libraries
        import boto3, re, sys, math, json, os, sagemaker, urllib.request
        from sagemaker import get_execution_role
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from IPython.display import Image
        from IPython.display import display
        from time import gmtime, strftime
        from sagemaker.predictor import csv_serializer

        # Define IAM role
        role = get_execution_role()
        prefix = 'sagemaker/DEMO-xgboost-dm'
        # each region has its XGBoost container
        containers = {'us-west-2': '433757028032.dkr.ecr.us-west-2.amazonaws.com/xgboost:latest',
                      'us-east-1': '811284229777.dkr.ecr.us-east-1.amazonaws.com/xgboost:latest',
                      'us-east-2': '825641698319.dkr.ecr.us-east-2.amazonaws.com/xgboost:latest',
                      'eu-west-1': '685385470294.dkr.ecr.eu-west-1.amazonaws.com/xgboost:latest'}
        my_region = boto3.session.Session().region_name # set the region of the instance
        print("Great! - your SageMaker Instance is in the " + my_region + " region. You will use the " + containers[my_region]

        Great! - your SageMaker Instance is in the us-east-1 region. You will use the 811284229777.dkr.ecr.us-east-1.amazonaw
        s.com/xgboost:latest container for your SageMaker endpoint to make inference requests.
```

In [3]:
```
# Download from your S3 bucket the census data CSV file based on the publically available census data from the ML repos
from io import StringIO
s3 = boto3.resource('s3')
bucket_name = 'train-census-earnings-xgboost' # place the adult_census_clean.csv file in a bucket in your account
object_key = 'adult_census_clean.csv'

# Load the data into a pandas dataframe

csv_obj = s3.Object(bucket_name, object_key)
csv_string = csv_obj.get()['Body'].read().decode('utf-8')

model_data = pd.read_csv(StringIO(csv_string))
model_data.head()
```

Out[3]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	y_no	y_yes
0	39	6	77516	1	13	3	9	4	1	2	2174	0	40	1	0	1
1	50	2	83311	1	13	1	5	3	1	2	0	0	13	1	0	1
2	38	1	215646	4	9	2	7	4	1	2	0	0	40	1	0	1
3	53	1	234721	3	7	1	7	3	5	2	0	0	40	1	0	1
4	28	1	338409	1	13	1	6	1	5	1	0	0	40	13	0	1

Split 70% train, 30% test data

In [4]:
```
# Randomize the data and split it between train and test datasets on a 70% 30% split respectively
train_data, test_data = np.split(model_data.sample(frac=1, random_state=1729), [int(0.7 * len(model_data))])
print(train_data.shape, test_data.shape)
```

```
(22792, 16) (9769, 16)
```

We can see the 2 datasets, each with 16 features

Now save the datasets to S3, under /train directory and load it back

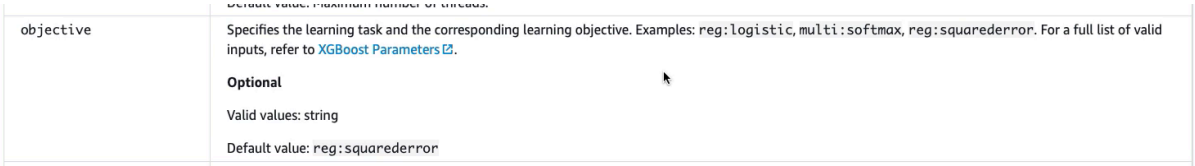
In [5]:
```
# Reformat the header and first column of the training data,
# save the new train dataset to your S3 bucket as train.csv and load the data from the S3 bucket
pd.concat([train_data['y_yes'], train_data.drop(['y_no', 'y_yes'], axis=1)], axis=1).to_csv('train.csv', index=False, h
boto3.Session().resource('s3').Bucket(bucket_name).Object(os.path.join(prefix, 'train/train.csv')).upload_file('train.c
s3_input_train = sagemaker.s3_input(s3_data='s3://{}/{}/train'.format(bucket_name, prefix), content_type='csv')
```

Now set up a SageMaker session and create XGboost model (estimator)

In [6]:
```
# Set up the SageMaker session, create an instance of the XGBoost model (an estimator),
# and define the model's hyperparameters
session_sm = sagemaker.Session()
xgb = sagemaker.estimator.Estimator(containers[my_region],
                                    role,
                                    train_instance_count=1,
                                    train_instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(bucket_name, prefix),
                                    sagemaker_session=session_sm)

xgb.set_hyperparameters(max_depth=5,eta=0.2,gamma=4,min_child_weight=6,subsample=0.8,silent=0,
                        objective='binary:logistic',
                        num_round=100)
```

There ar multiple hyper-parameters that we can set - refer to SM doc
For our objective, we use binary logistic



objective	Specifies the learning task and the corresponding learning objective. Examples: `reg:logistic`, `multi:softmax`, `reg:squarederror`. For a full list of valid inputs, refer to XGBoost Parameters. **Optional** Valid values: string Default value: `reg:squarederror`

We now train the model using Gradient optimization

```
In [7]: # After the data is loaded and the XGBoost estimator is set up,
        # train the model using gradient optimization on a ml.m4.xlarge instance
        xgb.fit({'train': s3_input_train})
```

```
2020-02-04 21:56:00 Starting - Starting the training job...
2020-02-04 21:56:02 Starting - Launching requested ML instances......
2020-02-04 21:57:09 Starting - Preparing the instances for training......
2020-02-04 21:58:23 Downloading - Downloading input data
2020-02-04 21:58:23 Training - Downloading the training image.....Arguments: train
[2020-02-04:21:59:10:INFO] Running standalone xgboost training.
[2020-02-04:21:59:10:INFO] Path /opt/ml/input/data/validation does not exist!
[2020-02-04:21:59:10:INFO] File size need to be processed in the node: 0.83mb. Available memory size in the node: 850
6.63mb
[2020-02-04:21:59:10:INFO] Determined delimiter of CSV input is ','
[21:59:10] S3DistributionType set as FullyReplicated
[21:59:10] 22792x14 matrix with 319088 entries loaded from /opt/ml/input/data/train?format=csv&label_column=0&delimit
er=,
[21:59:10] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 32 extra nodes, 4 pruned nodes, max_depth=5
[0]#011train-error:0.147683
[21:59:10] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 34 extra nodes, 4 pruned nodes, max_depth=5
[1]#011train-error:0.147683
[21:59:10] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 38 extra nodes, 4 pruned nodes, max_depth=5
```

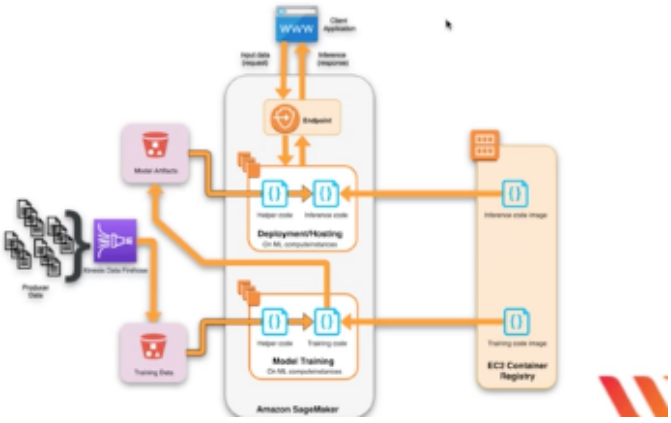
```
[21:59:11] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 16 extra nodes, 22 pruned nodes, max_depth=5
[95]#011train-error:0.117541
[21:59:11] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 14 extra nodes, 12 pruned nodes, max_depth=5
[96]#011train-error:0.117541
[21:59:11] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra nodes, 14 pruned nodes, max_depth=5
[97]#011train-error:0.117717
[21:59:11] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 8 extra nodes, 24 pruned nodes, max_depth=4
[98]#011train-error:0.117673
[21:59:11] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 8 extra nodes, 12 pruned nodes, max_depth=3
[99]#011train-error:0.117761


2020-02-04 21:59:30 Uploading - Uploading generated training model
2020-02-04 21:59:37 Completed - Training job completed
Training seconds: 94
Billable seconds: 94
```

Training job completed in 94 seconds

Now Deploy the model and evaluate its performance

- SageMaker provides an HTTPS endpoint, making your model available for inference requests
  - Three steps
    1. Create model in SageMaker
    2. Create an endpoint configuration
    3. Create HTTPS endpoint




```
In [8]: # Deploy your model and create an endpoint that you can access
        xgb_predictor = xgb.deploy(initial_instance_count=1,instance_type='ml.m4.xlarge')

        ----------------!
```

Now use our test data and run inferences on it, using the XGboost predict()

In [9]:
```
# Predict whether census participants in the test dataset earned more than 50K
test_data_array = test_data.drop(['y_no', 'y_yes'], axis=1).values #load the data into an array
xgb_predictor.content_type = 'text/csv' # set the data type for an inference
xgb_predictor.serializer = csv_serializer # set the serializer type
predictions = xgb_predictor.predict(test_data_array).decode('utf-8') # predict!
predictions_array = np.fromstring(predictions[1:], sep=',') # and turn the prediction into an array
print(predictions_array.shape)
```

```
(9769,)
```

Now evaluate the results on the 9,769 observations, building a confusion matrix

In [10]:
```
# Evaluate the performance and accuracy of the model
cm = pd.crosstab(index=test_data['y_yes'],
                 columns=np.round(predictions_array),
                 rownames=['Observed'],
                 colnames=['Predicted'])

tn = cm.iloc[0,0]; fn = cm.iloc[1,0];
tp = cm.iloc[1,1]; fp = cm.iloc[0,1];
p = (tp+tn)/(tp+tn+fp+fn)*100

print("\n{0:<20}{1:<4.1f}%\n".format("Overall Classification Rate: ", p))
print("{0:<15}{1:<15}{2:>8}".format("Predicted", "Under 50K", "Over 50K"))
print("Observed")
print("{0:<15}{1:<2.0f}% ({2:<}){3:>6.0f}% ({4:<})".format("Under 50K", tn/(tn+fn)*100,tn, fp/(tp+fp)*100, fp))
print("{0:<16}{1:<1.0f}% ({2:<}){3:>7.0f}% ({4:<}) \n".format("Over 50K", fn/(tn+fn)*100,fn, tp/(tp+fp)*100, tp))
```

```
Overall Classification Rate: 87.3%

Predicted      Under 50K      Over 50K
Observed
Under 50K      79% (1572)    10% (814)
Over 50K        21% (430)     90% (6953)
```

Our accuracy is 87.3%
And we can see the confusion matrix

Clean up the labs resources

In [11]:
```
# Terminate your SageMaker-related resources,
# delete the SageMaker endpoint and the objects in your S3 bucket
sagemaker.Session().delete_endpoint(xgb_predictor.endpoint)
# delete your bucket holding your training data and model artifacts
bucket_to_delete = boto3.resource('s3').Bucket(bucket_name)
bucket_to_delete.objects.all().delete()
```

Out[11]:
```
[{'ResponseMetadata': {'RequestId': '1293DC00AF9250DD',
   'HostId': 'toguxy0B2e7bvyZU4vxJaxMAFBMdKrNZxYPTuan2MlpItTMAITLgvYnlNDzDgxuBK8J/prtAW6I=',
   'HTTPStatusCode': 200,
   'HTTPHeaders': {'x-amz-id-2': 'toguxy0B2e7bvyZU4vxJaxMAFBMdKrNZxYPTuan2MlpItTMAITLgvYnlNDzDgxuBK8J/prtAW6I=',
    'x-amz-request-id': '1293DC00AF9250DD',
    'date': 'Wed, 05 Feb 2020 02:37:57 GMT',
    'connection': 'close',
    'content-type': 'application/xml',
    'transfer-encoding': 'chunked',
    'server': 'AmazonS3'},
   'RetryAttempts': 0},
  'Deleted': [{'Key': 'sagemaker/DEMO-xgboost-dm/output/xgboost-2020-02-04-21-56-00-808/output/model.tar.gz'},
   {'Key': 'sagemaker/DEMO-xgboost-dm/train/train.csv'},
   {'Key': 'adult_census_clean.csv'}]}]
```