Datagrokr Analytics Pvt. Ltd.

AWS Sagemaker

In a typical production environment, it is time consuming to deploy machine learning (ML) systems. And it is difficult to deploy and manage environments for ML systems. AWS Sagemaker makes it easier to deploy and manage your ML system environment.

AWS Sagemaker is an end-to-end machine learning platform, which allows to easily deploy production ready machine learning systems. It features auto scaling on deployed distributed ML models. It provides optimized algorithms and also allows for custom built algorithms for use. It provides flexibility in training models by providing support to other frameworks.
It supports deep learning frameworks like TensorFlow, Apache MXNet and Gluon. It also supports Apache spark to deploy ML models similar to Spark's MLLib. And custom built algorithms or models based on other frameworks like scikit-learn can be deployed using docker images. It also integrates jupyter notebook for easier writing up of ML model code to be hosted and deployed.

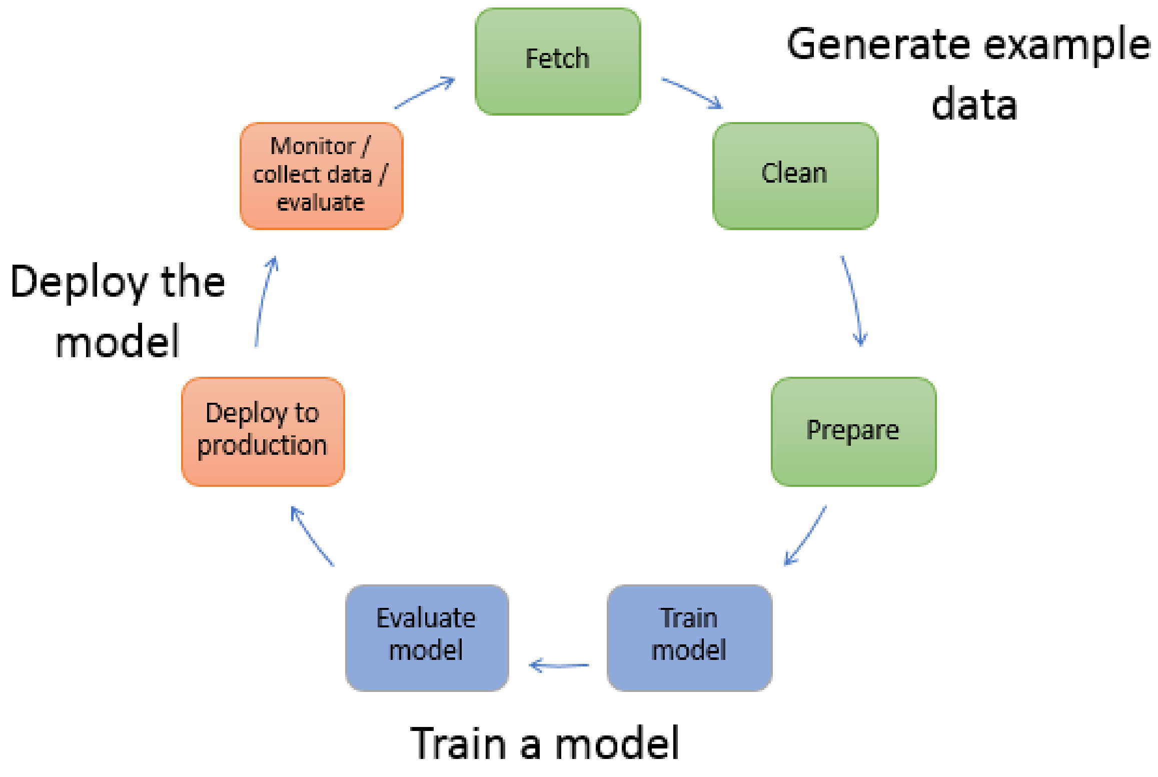
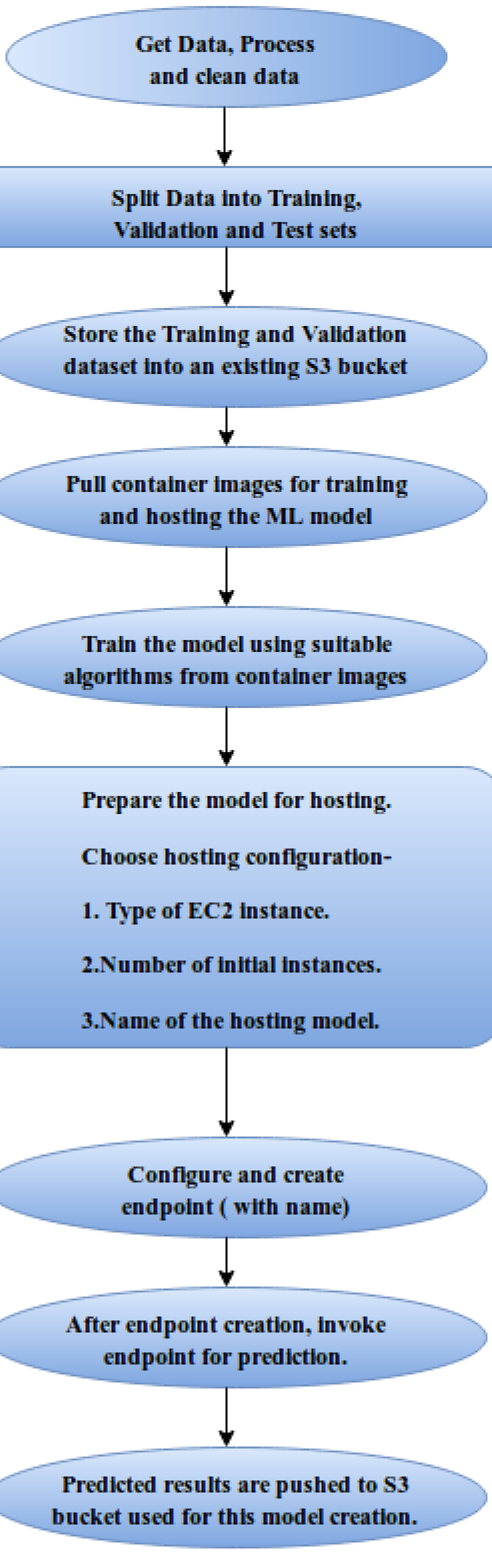


Fig 1. Machine Learning model deployment.

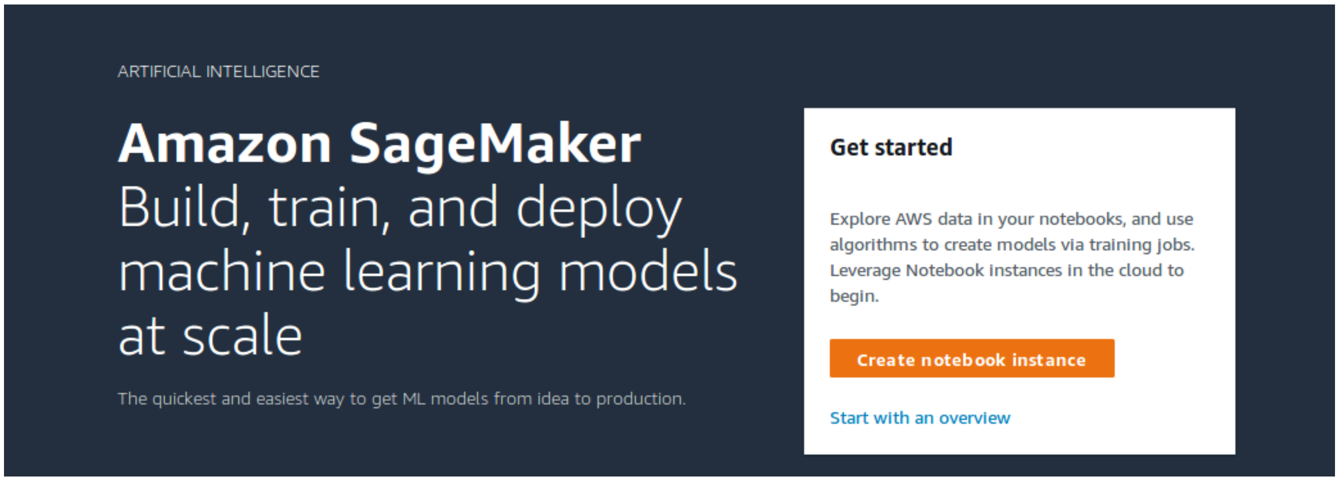
Below flow diagram explains the workflow of AWS Sagemaker.


Get Data, Process and clean data
Split Data into Training, Validation and Test sets
Store the Training and Validation dataset into an existing S3 bucket
Pull container images for training and hosting the ML model
Train the model using suitable algorithms from container images
Prepare the model for hosting.
Choose hosting configuration-
1. Type of EC2 instance.
2.Number of initial instances.
3.Name of the hosting model.
Configure and create endpoint (with name)
After endpoint creation, invoke endpoint for prediction.
Predicted results are pushed to S3 bucket used for this model creation.

AWS Sagemaker

Below screenshots are workflow of a sample Model creation on AWS Sagemaker.

1. Create notebook instance in Amazon Sagemaker.



ARTIFICIAL INTELLIGENCE

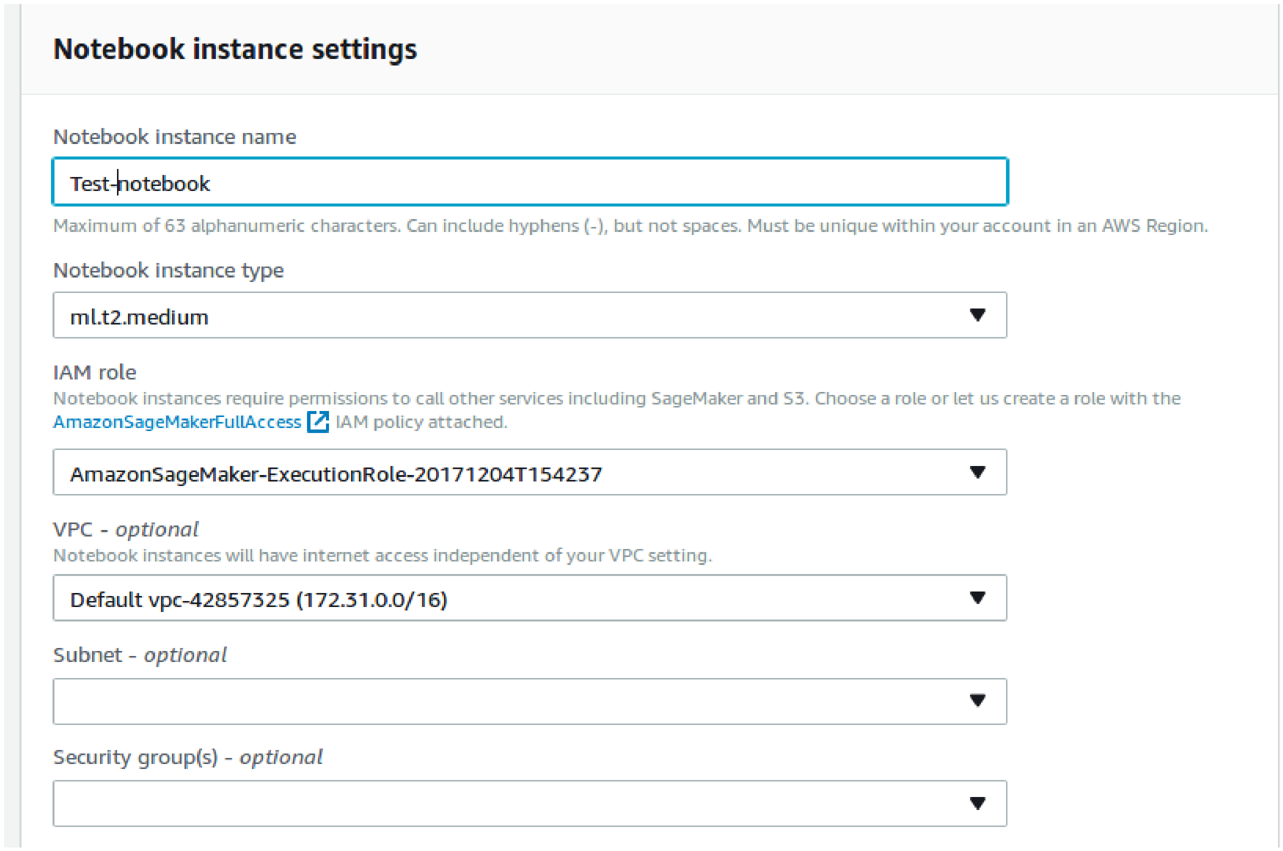
Amazon SageMaker
Build, train, and deploy machine learning models at scale

The quickest and easiest way to get ML models from idea to production.

Get started

Explore AWS data in your notebooks, and use algorithms to create models via training jobs. Leverage Notebook instances in the cloud to begin.

2. Notebook Instance settings.



Notebook instance settings

Notebook instance name: Test-notebook

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

Notebook instance type: ml.t2.medium

IAM role

Notebook instances require permissions to call other services including SageMaker and S3. Choose a role or let us create a role with the AmazonSageMakerFullAccess IAM policy attached.

AmazonSageMaker-ExecutionRole-20171204T154237

VPC - *optional*

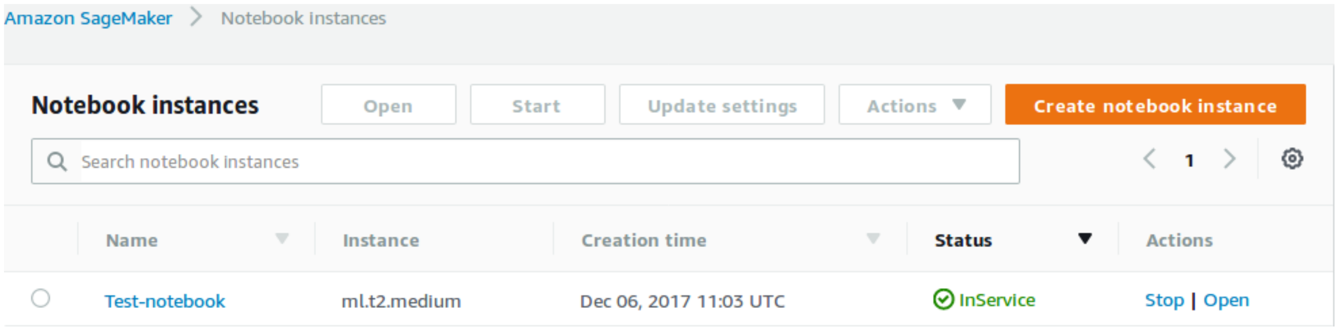
Notebook instances will have internet access independent of your VPC setting.

Default vpc-42857325 (172.31.0.0/16)

Subnet - *optional*

Security group(s) - *optional*

3. Notebook instance.



Amazon SageMaker > Notebook instances

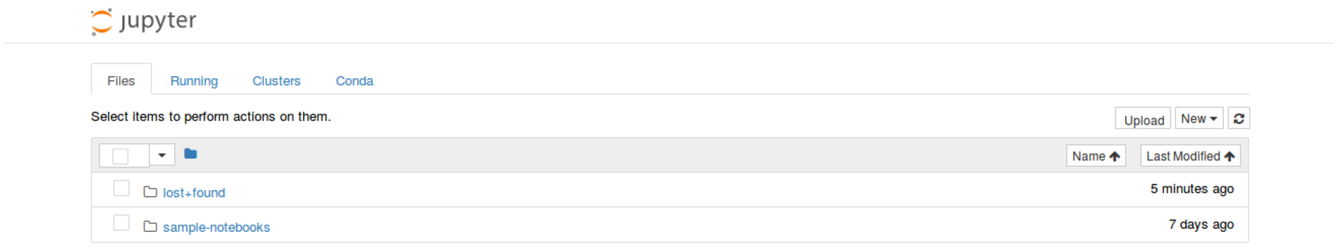
Notebook instances

Open | Start | Update settings | Actions | Create notebook instance

Search notebook instances

Name	Instance	Creation time	Status	Actions
Test-notebook	ml.t2.medium	Dec 06, 2017 11:03 UTC	InService	Stop | Open

4. Open notebook instance.

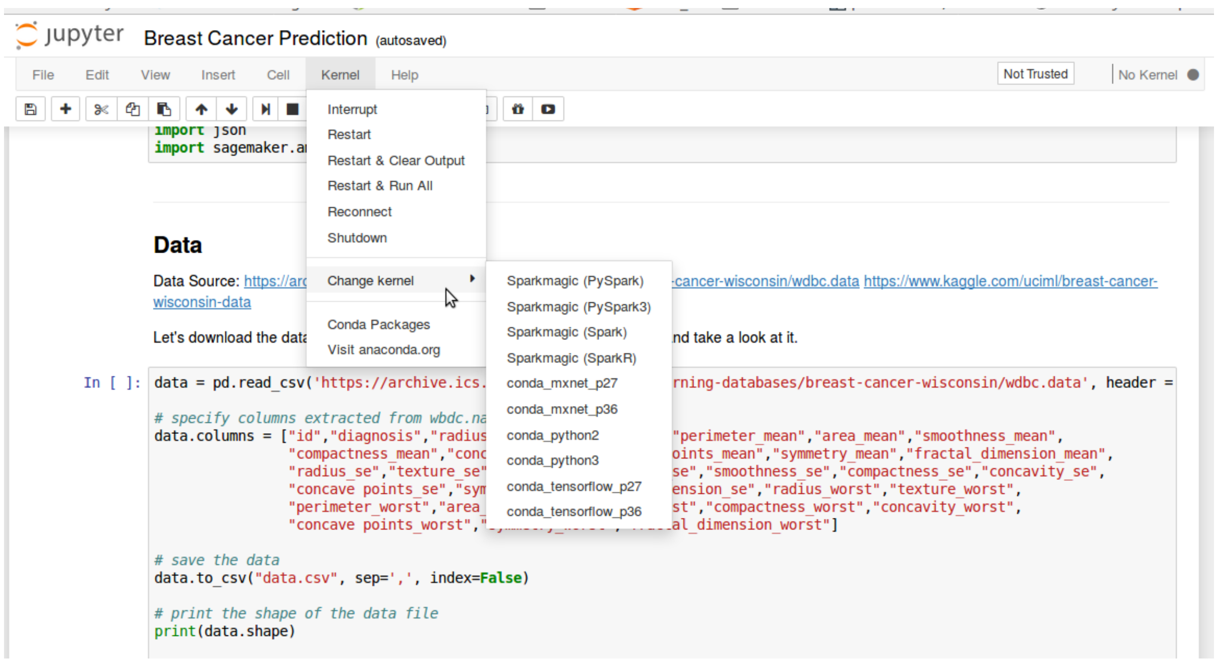


Select items to perform actions on them.

Name	Last Modified
lost+found	5 minutes ago
sample-notebooks	7 days ago

5. Running sample notebook (breast cancer prediction).

5.1. Checking available kernels to run notebook.



```
import json
import sagemaker.a
```

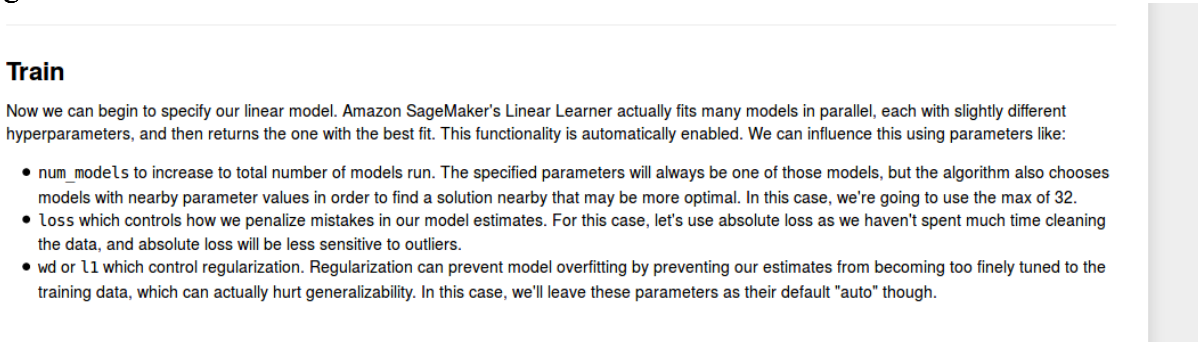
Data

Data Source: https://arc -cancer-wisconsin/wdbc.data https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

Let's download the data nd take a look at it.

```
In [ ]: data = pd.read_csv('https://archive.ics. rning-databases/breast-cancer-wisconsin/wdbc.data', header =

        # specify columns extracted from wbdc.na
        data.columns = ["id","diagnosis","radius "perimeter_mean","area_mean","smoothness_mean",
                        "compactness_mean","conc oints_mean","symmetry_mean","fractal_dimension_mean",
                        "radius_se","texture_se" se","smoothness_se","compactness_se","concavity_se",
                        "concave points_se","sym ension_se","radius_worst","texture_worst",
                        "perimeter_worst","area_ st","compactness_worst","concavity_worst",
                        "concave points_worst"," al_dimension_worst"]

        # save the data
        data.to_csv("data.csv", sep=',', index=False)

        # print the shape of the data file
        print(data.shape)
```

6. Training ML model.

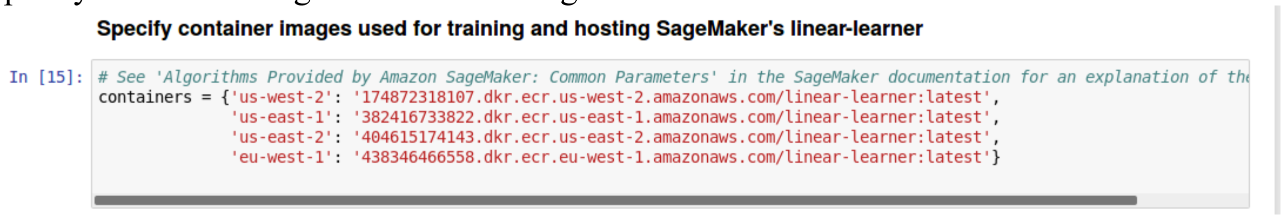


Train

Now we can begin to specify our linear model. Amazon SageMaker's Linear Learner actually fits many models in parallel, each with slightly different hyperparameters, and then returns the one with the best fit. This functionality is automatically enabled. We can influence this using parameters like:

- `num_models` to increase to total number of models run. The specified parameters will always be one of those models, but the algorithm also chooses models with nearby parameter values in order to find a solution nearby that may be more optimal. In this case, we're going to use the max of 32.
- `loss` which controls how we penalize mistakes in our model estimates. For this case, let's use absolute loss as we haven't spent much time cleaning the data, and absolute loss will be less sensitive to outliers.
- `wd` or `l1` which control regularization. Regularization can prevent model overfitting by preventing our estimates from becoming too finely tuned to the training data, which can actually hurt generalizability. In this case, we'll leave these parameters as their default "auto" though.

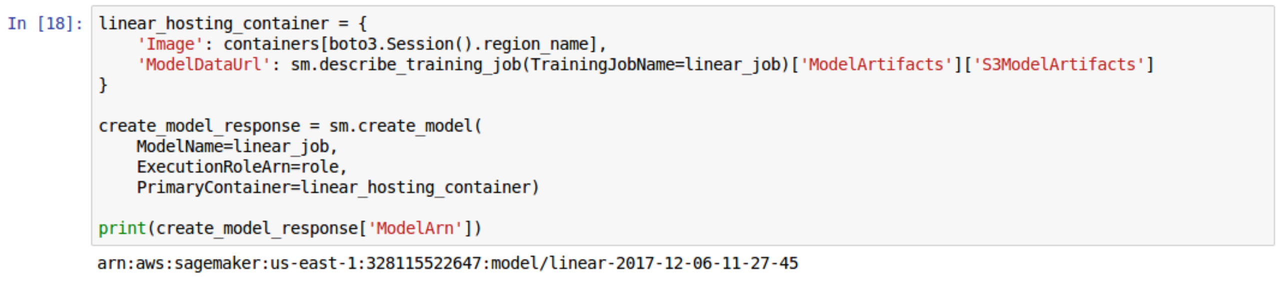
7. Specifycontainers images used for training.



Specify container images used for training and hosting SageMaker's linear-learner

```
In [15]: # See 'Algorithms Provided by Amazon SageMaker: Common Parameters' in the SageMaker documentation for an explanation of the
         containers = {'us-west-2': '174872318107.dkr.ecr.us-west-2.amazonaws.com/linear-learner:latest',
                       'us-east-1': '382416733822.dkr.ecr.us-east-1.amazonaws.com/linear-learner:latest',
                       'us-east-2': '404615174143.dkr.ecr.us-east-2.amazonaws.com/linear-learner:latest',
                       'eu-west-1': '438346466558.dkr.ecr.eu-west-1.amazonaws.com/linear-learner:latest'}
```

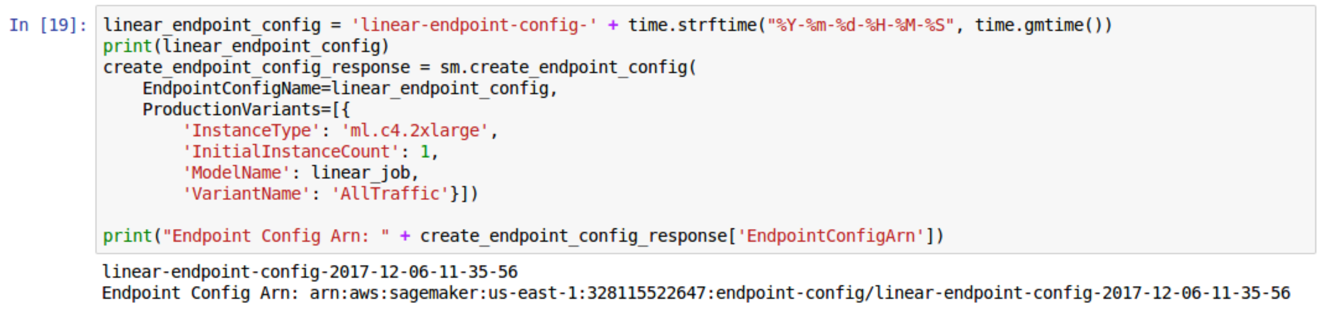
8. Creating hosting model for use.



```
In [18]: linear_hosting_container = {
             'Image': containers[boto3.Session().region_name],
             'ModelDataUrl': sm.describe_training_job(TrainingJobName=linear_job)['ModelArtifacts']['S3ModelArtifacts']
         }

         create_model_response = sm.create_model(
             ModelName=linear_job,
             ExecutionRoleArn=role,
             PrimaryContainer=linear_hosting_container)

         print(create_model_response['ModelArn'])

         arn:aws:sagemaker:us-east-1:328115522647:model/linear-2017-12-06-11-27-45
```

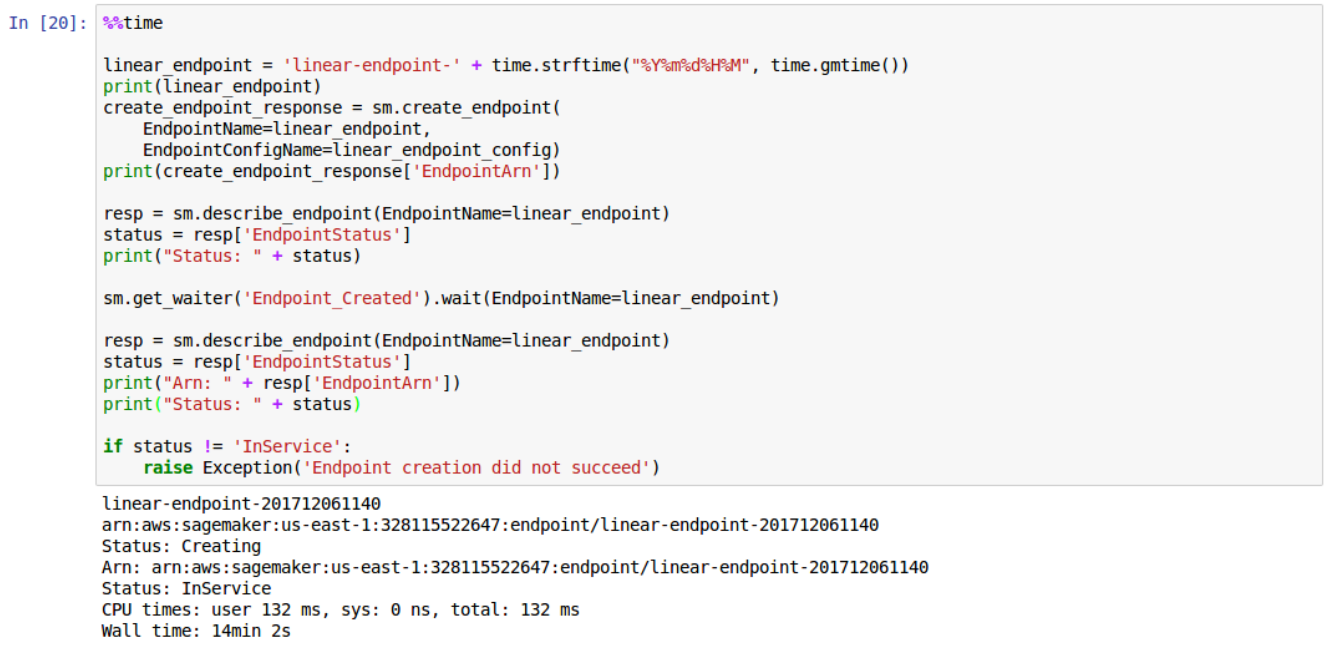
9. Creating endpoint for hosting model with initial number of instances.

9.1. Configure endpoint.



```
In [19]: linear_endpoint_config = 'linear-endpoint-config-' + time.strftime("%Y-%m-%d-%H-%M-%S", time.gmtime())
         print(linear_endpoint_config)
         create_endpoint_config_response = sm.create_endpoint_config(
             EndpointConfigName=linear_endpoint_config,
             ProductionVariants=[{
                 'InstanceType': 'ml.c4.2xlarge',
                 'InitialInstanceCount': 1,
                 'ModelName': linear_job,
                 'VariantName': 'AllTraffic'}])

         print("Endpoint Config Arn: " + create_endpoint_config_response['EndpointConfigArn'])

         linear-endpoint-config-2017-12-06-11-35-56
         Endpoint Config Arn: arn:aws:sagemaker:us-east-1:328115522647:endpoint-config/linear-endpoint-config-2017-12-06-11-35-56
```

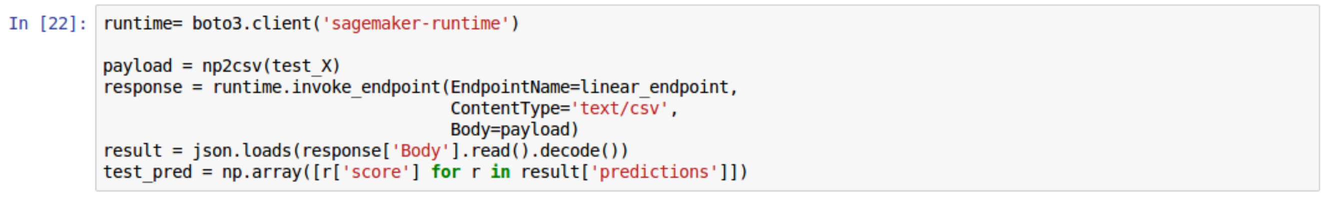
9.2. Creating Endpoint.



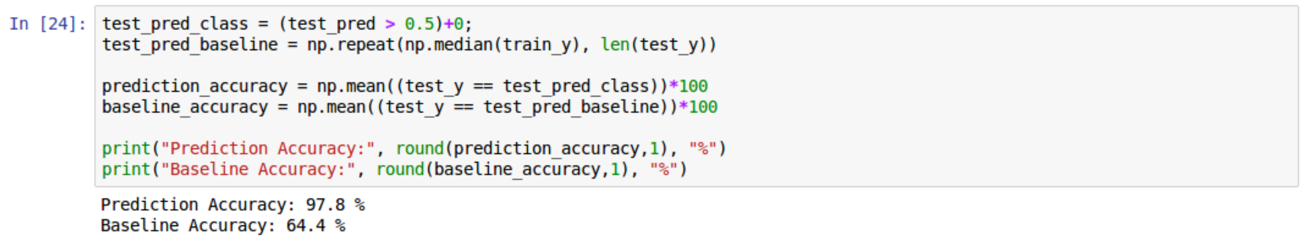
```
In [20]: %%time

         linear_endpoint = 'linear-endpoint-' + time.strftime("%Y%m%d%H%M", time.gmtime())
         print(linear_endpoint)
         create_endpoint_response = sm.create_endpoint(
             EndpointName=linear_endpoint,
             EndpointConfigName=linear_endpoint_config)
         print(create_endpoint_response['EndpointArn'])

         resp = sm.describe_endpoint(EndpointName=linear_endpoint)
         status = resp['EndpointStatus']
         print("Status: " + status)

         sm.get_waiter('Endpoint_Created').wait(EndpointName=linear_endpoint)

         resp = sm.describe_endpoint(EndpointName=linear_endpoint)
         status = resp['EndpointStatus']
         print("Arn: " + resp['EndpointArn'])
         print("Status: " + status)

         if status != 'InService':
             raise Exception('Endpoint creation did not succeed')
```

```
linear-endpoint-201712061140
arn:aws:sagemaker:us-east-1:328115522647:endpoint/linear-endpoint-201712061140
Status: Creating
Arn: arn:aws:sagemaker:us-east-1:328115522647:endpoint/linear-endpoint-201712061140
Status: InService
CPU times: user 132 ms, sys: 0 ns, total: 132 ms
Wall time: 14min 2s
```

10. Prediction invoking endpoint.



```
In [22]: runtime= boto3.client('sagemaker-runtime')

         payload = np2csv(test_X)
         response = runtime.invoke_endpoint(EndpointName=linear_endpoint,
                                            ContentType='text/csv',
                                            Body=payload)
         result = json.loads(response['Body'].read().decode())
         test_pred = np.array([r['score'] for r in result['predictions']])
```

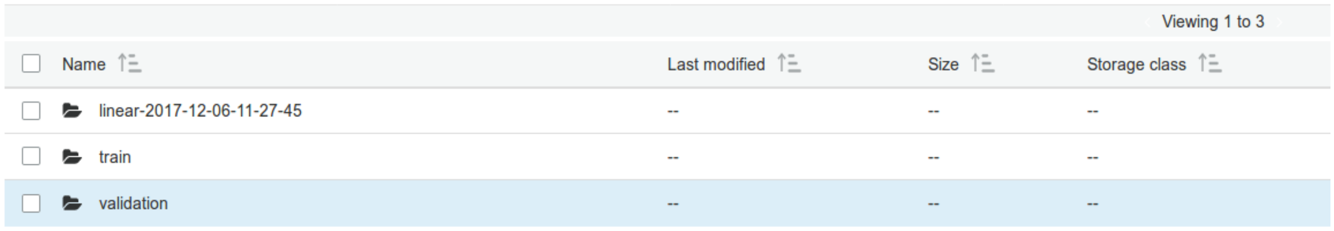


```
In [24]: test_pred_class = (test_pred > 0.5)+0;
         test_pred_baseline = np.repeat(np.median(train_y), len(test_y))

         prediction_accuracy = np.mean((test_y == test_pred_class))*100
         baseline_accuracy = np.mean((test_y == test_pred_baseline))*100

         print("Prediction Accuracy:", round(prediction_accuracy,1), "%")
         print("Baseline Accuracy:", round(baseline_accuracy,1), "%")
```

```
Prediction Accuracy: 97.8 %
Baseline Accuracy: 64.4 %
```

Results and training dataset are stored in S3 bucket. Custom built or other framework based algorithms can be hosted into container image and can be used like above. Sagemaker assumes S3 bucket already exists else you need to handle the exception of S3 bucket.

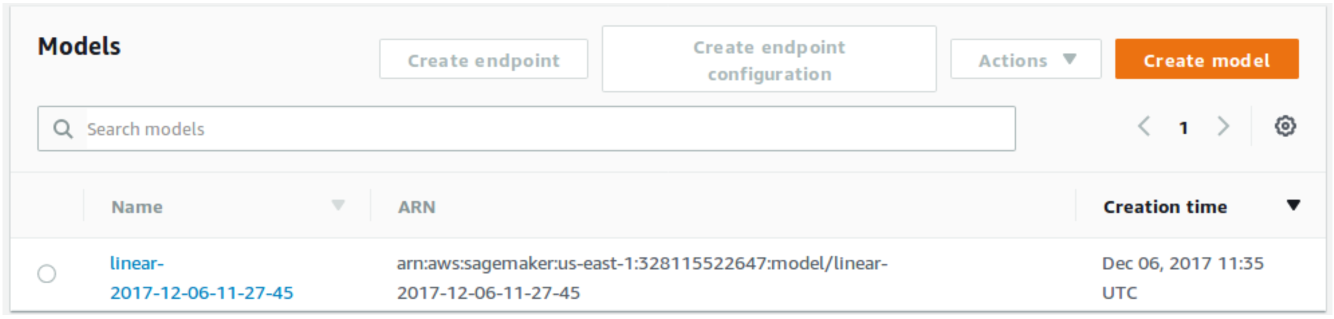
11. S3 Bucket output.



Viewing 1 to 3

Name	Last modified	Size	Storage class
linear-2017-12-06-11-27-45	--	--	--
train	--	--	--
validation	--	--	--

12. Hosted Model.



Models

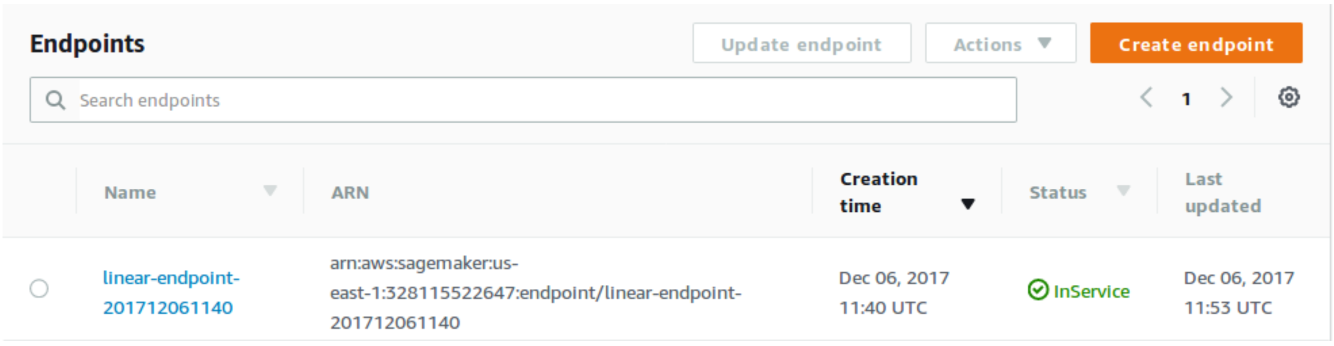
Create endpoint | Create endpoint configuration | Actions | Create model

Search models

Name	ARN	Creation time
linear-2017-12-06-11-27-45	arn:aws:sagemaker:us-east-1:328115522647:model/linear-2017-12-06-11-27-45	Dec 06, 2017 11:35 UTC

13. Endpoint.

Endpoint can invoked from the console instead of jupyter notebook.



Endpoints

Update endpoint | Actions | Create endpoint

Search endpoints

Name	ARN	Creation time	Status	Last updated
linear-endpoint-201712061140	arn:aws:sagemaker:us-east-1:328115522647:endpoint/linear-endpoint-201712061140	Dec 06, 2017 11:40 UTC	✓ InService	Dec 06, 2017 11:53 UTC

AWS Sagemaker endpoint can be embedded into your application by using or invoking the endpoint with help of boto3 and sagemaker SDK.

This document is simple take on AWS Sagemaker, it can be tuned accordingly to needs of the requirement at hand. It is essential to know docker to build docker files for your custom built or other frame work based algorithms.

References:

1. https://github.com/awslabs/amazon-sagemaker-examples
2. https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html
3. https://pypi.python.org/pypi/sagemaker