Breast Cancer Prediction

Predict Breast Cancer using SageMaker's Linear-Learner with features derived from images of Breast Mass

---

Contents

1. Background
2. Setup
3. Data
4. Train
5. Host
6. Predict
7. Extensions

Background

This notebook illustrates how one can use SageMaker's algorithms for solving applications which require `linear models` for prediction. For this illustration, we have taken an example for breast cancer prediction using UCI'S breast cancer diagnostic data set. The purpose here is to use this data set to build a predictve model of whether a breast mass image indicates benign or malignant tumor. The data set will be used to illustrate:

- Basic setup for using SageMaker.
- Converting datasets to protobuf format used by the Amazon SageMaker algorithms and uploading to S3.
- Training SageMaker's linear learner on the data set.
- Hosting the trained model.
- Scoring using the trained model.

Setup

Let's start by specifying:

- The SageMaker role arn used to give learning and hosting access to your data. The snippet below will use the same role used by your SageMaker notebook instance, if you're using other. Otherwise, specify the full ARN of a role with the SageMakerFullAccess policy attached.
- The S3 bucket that you want to use for training and storing model objects.

In [9]:
```
import os
import boto3
import re
import sagemaker

role = sagemaker.get_execution_role()
region = boto3.Session().region_name

# S3 bucket for saving code and model artifacts.
# Feel free to specify a different bucket and prefix
bucket = "day-07-nr"

prefix = ("sagemaker/DEMO-breast-cancer-prediction")  # place to upload training files within the bucket
```

In [10]:
```
role
```

Out[10]:
```
'arn:aws:iam::224670572127:role/fast-ai-academic-12-Student-Azure'
```

Now we'll import the Python libraries we'll need.

In [11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import io
import time
import json
import sagemaker.amazon.common as smac
```

Data sources

> Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

> Breast Cancer Wisconsin (Diagnostic) Data Set [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)].

> *Also see:* Breast Cancer Wisconsin (Diagnostic) Data Set [https://www.kaggle.com/uciml/breast-cancer-wisconsin-data].

Let's download the data and save it in the local folder with the name data.csv and take a look at it.

In [12]:
```
s3 = boto3.client("s3")

filename = "wdbc.csv"
s3.download_file("sagemaker-sample-files", "datasets/tabular/breast_cancer/wdbc.csv", filename)
data = pd.read_csv(filename, header=None)

# specify columns extracted from wbdc.names
data.columns = [
    "id",
    "diagnosis",
    "radius_mean",
    "texture_mean",
    "perimeter_mean",
    "area_mean",
    "smoothness_mean",
    "compactness_mean",
    "concavity_mean",
    "concave points_mean",
    "symmetry_mean",
    "fractal_dimension_mean",
    "radius_se",
    "texture_se",
    "perimeter_se",
    "area_se",
    "smoothness_se",
    "compactness_se",
    "concavity_se",
    "concave points_se",
    "symmetry_se",
    "fractal_dimension_se",
    "radius_worst",
    "texture_worst",
    "perimeter_worst",
    "area_worst",
    "smoothness_worst",
    "compactness_worst",
    "concavity_worst",
    "concave points_worst",
    "symmetry_worst",
    "fractal_dimension_worst",
]

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

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

# show the top few rows
display(data.head())

# describe the data object
display(data.describe())

# we will also summarize the categorical field diganosis
display(data.diagnosis.value_counts())
```

```
(569, 32)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concav points_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.1471
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.0701
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.1279
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.1052
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.1043

5 rows × 32 columns

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200

8 rows × 31 columns

```
B    357
M    212
```

Key observations:

- Data has 569 observations and 32 columns.
- First field is 'id'.
- Second field, 'diagnosis', is an indicator of the actual diagnosis ('M' = Malignant; 'B' = Benign).
- There are 30 other numeric features available for prediction.

Create Features and Labels

Split the data into 80% training, 10% validation and 10% testing.

In [13]:
```
rand_split = np.random.rand(len(data))
train_list = rand_split < 0.8
val_list = (rand_split >= 0.8) & (rand_split < 0.9)
test_list = rand_split >= 0.9

data_train = data[train_list]
data_val = data[val_list]
data_test = data[test_list]

train_y = ((data_train.iloc[:, 1] == "M") + 0).to_numpy()
train_X = data_train.iloc[:, 2:].to_numpy()

val_y = ((data_val.iloc[:, 1] == "M") + 0).to_numpy()
val_X = data_val.iloc[:, 2:].to_numpy()

test_y = ((data_test.iloc[:, 1] == "M") + 0).to_numpy()
test_X = data_test.iloc[:, 2:].to_numpy();
```

Now, we'll convert the datasets to the recordIO-wrapped protobuf format used by the Amazon SageMaker algorithms, and then upload this data to S3. See this page for the data format that this algorithm supports: https://docs.aws.amazon.com/sagemaker/latest/dg/linear-learner.html

We'll start with training data.

In [14]:
```
train_file = "linear_train.data"

f = io.BytesIO()
smac.write_numpy_to_dense_tensor(f, train_X.astype("float32"), train_y.astype("float32"))
f.seek(0)

boto3.Session().resource("s3").Bucket(bucket).Object(os.path.join(prefix, "train", train_file)).upload_fileobj(f)
```

Next we'll convert and upload the validation dataset.

In [15]:
```
validation_file = "linear_validation.data"

f = io.BytesIO()
smac.write_numpy_to_dense_tensor(f, val_X.astype("float32"), val_y.astype("float32"))
f.seek(0)

boto3.Session().resource("s3").Bucket(bucket).Object(
    os.path.join(prefix, "validation", validation_file)
).upload_fileobj(f)
```

Check S3 bucket to see if they have been uploaded to S3.

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.

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

In [16]:
```
from sagemaker import image_uris

container = image_uris.retrieve(region=boto3.Session().region_name, framework="linear-learner")
```

In [17]:
```
linear_job = "DEMO-linear-" + time.strftime("%Y-%m-%d-%H-%M-%S", time.gmtime())
print("Job name is:", linear_job)

linear_training_params = {
    "RoleArn": role,
    "TrainingJobName": linear_job,
    "AlgorithmSpecification": {"TrainingImage": container, "TrainingInputMode": "File"},
    "ResourceConfig": {"InstanceCount": 1, "InstanceType": "ml.c4.2xlarge", "VolumeSizeInGB": 10},
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://{}/{}/train/".format(bucket, prefix),
                    "S3DataDistributionType": "ShardedByS3Key",
                }
            },
            "CompressionType": "None",
            "RecordWrapperType": "None",
        },
        {
            "ChannelName": "validation",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://{}/{}/validation/".format(bucket, prefix),
                    "S3DataDistributionType": "FullyReplicated",
                }
            },
            "CompressionType": "None",
            "RecordWrapperType": "None",
        },
    ],
    "OutputDataConfig": {"S3OutputPath": "s3://{}/{}/".format(bucket, prefix)},
    "HyperParameters": {
        "feature_dim": "30",
        "mini_batch_size": "100",
        "predictor_type": "regressor",
        "epochs": "10",
        "num_models": "32",
        "loss": "absolute_loss",
    },
    "StoppingCondition": {"MaxRuntimeInSeconds": 60 * 60},
}
```

```
Job name is: DEMO-linear-2023-03-22-21-28-13
```

Now let's kick off our training job in SageMaker's distributed, managed training, using the parameters we just created. Because training is managed, we don't have to wait for our job to finish to continue, but for this case, let's use boto3's 'training_job_completed_or_stopped' waiter so we can ensure that the job has been started.

In [18]:
```
%%time

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

sm.create_training_job(**linear_training_params)
```

```
status = sm.describe_training_job(TrainingJobName=linear_job)["TrainingJobStatus"]
print(status)
sm.get_waiter("training_job_completed_or_stopped").wait(TrainingJobName=linear_job)
if status == "Failed":
    message = sm.describe_training_job(TrainingJobName=linear_job)["FailureReason"]
    print("Training failed with the following error: {}".format(message))
    raise Exception("Training job failed")
```

```
InProgress
CPU times: user 306 ms, sys: 8.05 ms, total: 314 ms
Wall time: 10min 1s
```

Host

Now that we've trained the linear algorithm on our data, let's setup a model which can later be hosted. We will:

1. Point to the scoring container
2. Point to the model.tar.gz that came from training
3. Create the hosting model

In [19]:
```
linear_hosting_container = {
    "Image": container,
    "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:ca-central-1:224670572127:model/demo-linear-2023-03-22-21-28-13
```

Once we've setup a model, we can configure what our hosting endpoints should be. Here we specify:

1. EC2 instance type to use for hosting
2. Initial number of instances
3. Our hosting model name

In [20]:
```
linear_endpoint_config = "DEMO-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.m4.xlarge",
            "InitialInstanceCount": 1,
            "ModelName": linear_job,
            "VariantName": "AllTraffic",
        }
    ],
)

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

```
DEMO-linear-endpoint-config-2023-03-22-21-40-46
Endpoint Config Arn: arn:aws:sagemaker:ca-central-1:224670572127:endpoint-config/demo-linear-endpoint-config-2023-03-22-21-40
-46
```

Now that we've specified how our endpoint should be configured, we can create them. This can be done in the background, but for now let's run a loop that updates us on the status of the endpoints so that we know when they are ready for use.

In [22]:
```
%%time

linear_endpoint = "DEMO-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_in_service").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")
```

```
DEMO-linear-endpoint-202303222144
arn:aws:sagemaker:ca-central-1:224670572127:endpoint/demo-linear-endpoint-202303222144
Status: Creating
Arn: arn:aws:sagemaker:ca-central-1:224670572127:endpoint/demo-linear-endpoint-202303222144
Status: InService
CPU times: user 376 ms, sys: 0 ns, total: 376 ms
Wall time: 4min 2s
```

Predict

Predict on Test Data

Now that we have our hosted endpoint, we can generate statistical predictions from it. Let's predict on our test dataset to understand how accurate our model is.

There are many metrics to measure classification accuracy. Common examples include include:

- Precision
- Recall
- F1 measure
- Area under the ROC curve - AUC
- Total Classification Accuracy
- Mean Absolute Error

For our example, we'll keep things simple and use total classification accuracy as our metric of choice. We will also evaluate Mean Absolute Error (MAE) as the linear-learner has been optimized using this metric, not necessarily because it is a relevant metric from an application point of view. We'll compare the performance of the linear-learner against a naive benchmark prediction which uses majority class observed in the training data set for prediction on the test data.

Function to convert an array to a csv

In [23]:
```
def np2csv(arr):
    csv = io.BytesIO()
    np.savetxt(csv, arr, delimiter=",", fmt="%g")
    return csv.getvalue().decode().rstrip()
```

Next, we'll invoke the endpoint to get predictions.

In [24]:
```
runtime = boto3.client("runtime.sagemaker")

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"]])
```

Let's compare linear learner based mean absolute prediction errors from a baseline prediction which uses majority class to predict every instance.

In [25]:
```
test_mae_linear = np.mean(np.abs(test_y - test_pred))
test_mae_baseline = np.mean(
    np.abs(test_y - np.median(train_y))
)  ## training median as baseline predictor

print("Test MAE Baseline :", round(test_mae_baseline, 3))
print("Test MAE Linear:", round(test_mae_linear, 3))
```

```
Test MAE Baseline : 0.34900000000000003
Test MAE Linear: 0.198
```

Let's compare predictive accuracy using a classification threshold of 0.5 for the predicted and compare against the majority class prediction from training data set.

In [26]:
```
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: 88.4 %
Baseline Accuracy: 65.10000000000001 %
```

Questions

1. Our linear model does a good job of predicting breast cancer and has an overall accuracy of close to 92%. We can re-run the model with different values of the hyper-parameters, loss functions etc and see if we get improved prediction. Re-running the model with further tweaks to these hyperparameters may provide more accurate out-of-sample predictions.

2. We also did not do much feature engineering. We can create additional features by considering cross-product/interaction of multiple features, squaring or raising higher powers of the features to induce non-linear effects, etc. If we expand the features using non-linear terms and interactions, we can then tweak the regulaization parameter to optimize the expanded model and hence generate improved forecasts.

3. You can use non-linear models available through SageMaker such as XGBoost, MXNet etc.

Answer 1

Our linear model does a good job of predicting breast cancer and has an overall accuracy of close to 92%. We can re-run the model with different values of the hyper-parameters, loss functions etc and see if we get improved prediction. Re-running the model with further tweaks to these hyperparameters may provide more accurate out-of-sample predictions.

In [36]:
```
linear_job = "DEMO-linear-" + time.strftime("%Y-%m-%d-%H-%M-%S", time.gmtime())

print("Job name is:", linear_job)

linear_training_params = {
    "RoleArn": role,
    "TrainingJobName": linear_job,
    "AlgorithmSpecification": {"TrainingImage": container, "TrainingInputMode": "File"},
    "ResourceConfig": {"InstanceCount": 1, "InstanceType": "ml.c4.2xlarge", "VolumeSizeInGB": 10},
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://{}/{}/train/".format(bucket, prefix),
                    "S3DataDistributionType": "ShardedByS3Key",
                }
            },
            "CompressionType": "None",
            "RecordWrapperType": "None",
        },
        {
            "ChannelName": "validation",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://{}/{}/validation/".format(bucket, prefix),
                    "S3DataDistributionType": "FullyReplicated",
                }
            },
            "CompressionType": "None",
            "RecordWrapperType": "None",
        },
    ],
    "OutputDataConfig": {"S3OutputPath": "s3://{}/{}/".format(bucket, prefix)},
    "HyperParameters": {
        "feature_dim": "auto",
        "mini_batch_size": "50",
        "predictor_type": "regressor",
        "epochs": "20",
        "num_models": "25",
        "loss": "squared_loss",
    },
    "StoppingCondition": {"MaxRuntimeInSeconds": 60 * 60},
}
```

```
Job name is: DEMO-linear-2023-03-22-22-24-58
```

In [37]:
```
%%time

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

sm.create_training_job(**linear_training_params)
```

```
status = sm.describe_training_job(TrainingJobName=linear_job)["TrainingJobStatus"]
print(status)
sm.get_waiter("training_job_completed_or_stopped").wait(TrainingJobName=linear_job)
if status == "Failed":
    message = sm.describe_training_job(TrainingJobName=linear_job)["FailureReason"]
    print("Training failed with the following error: {}".format(message))
    raise Exception("Training job failed")
```

```
InProgress
CPU times: user 155 ms, sys: 5.85 ms, total: 161 ms
Wall time: 4min
```

In [38]:
```
linear_hosting_container = {
    "Image": container,
    "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:ca-central-1:224670572127:model/demo-linear-2023-03-22-22-24-58
```

In [39]:
```
linear_endpoint_config = "DEMO-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.m4.xlarge",
            "InitialInstanceCount": 1,
            "ModelName": linear_job,
            "VariantName": "AllTraffic",
        }
    ],
)

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

```
DEMO-linear-endpoint-config-2023-03-22-22-31-47
Endpoint Config Arn: arn:aws:sagemaker:ca-central-1:224670572127:endpoint-config/demo-linear-endpoint-config-2023-03-22-22-31
-47
```

In [40]:
```
%%time

linear_endpoint = "DEMO-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_in_service").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")
```

```
DEMO-linear-endpoint-202303222231
arn:aws:sagemaker:ca-central-1:224670572127:endpoint/demo-linear-endpoint-202303222231
Status: Creating
Arn: arn:aws:sagemaker:ca-central-1:224670572127:endpoint/demo-linear-endpoint-202303222231
Status: InService
CPU times: user 330 ms, sys: 5.58 ms, total: 335 ms
Wall time: 4min 1s
```

In [41]:
```
def np2csv(arr):
    csv = io.BytesIO()
    np.savetxt(csv, arr, delimiter=",", fmt="%g")
    return csv.getvalue().decode().rstrip()
```

In [42]:
```
runtime = boto3.client("runtime.sagemaker")

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 [43]:
```
test_mae_linear = np.mean(np.abs(test_y - test_pred))
test_mae_baseline = np.mean(
    np.abs(test_y - np.median(train_y))
)  ## training median as baseline predictor

print("Test MAE Baseline :", round(test_mae_baseline, 3))
print("Test MAE Linear:", round(test_mae_linear, 3))
```

```
Test MAE Baseline : 0.34900000000000003
Test MAE Linear: 0.213
```

In [44]:
```
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: 93.0 %
Baseline Accuracy: 65.10000000000001 %
```

Answer 2

We also did not do much feature engineering. We can create additional features by considering cross-product/interaction of multiple features, squaring or raising higher powers of the features to induce non-linear effects, etc. If we expand the features using non-linear terms and interactions, we can then tweak the regulaization parameter to optimize the expanded model and hence generate improved forecasts.

In [64]:
```
data.head()
```

Out[64]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concav points_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.1471
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.0701
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.1279
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.1052
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.1043

5 rows × 32 columns

In [69]:
```
data.dtypes
```

Out[69]:
```
id                         int64
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
radius_se                float64
texture_se               float64
perimeter_se             float64
area_se                  float64
smoothness_se            float64
compactness_se           float64
concavity_se             float64
concave points_se        float64
symmetry_se              float64
fractal_dimension_se     float64
radius_worst             float64
texture_worst            float64
perimeter_worst          float64
area_worst               float64
smoothness_worst         float64
compactness_worst        float64
concavity_worst          float64
concave points_worst     float64
symmetry_worst           float64
fractal_dimension_worst  float64
dtype: object
```

In [65]:
```
data.isnull().sum()
```

Out[65]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

In [67]:
```
data.columns
```

Out[67]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

In [72]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
```

```
data["diagnosis"] = le.fit_transform(data["diagnosis"])
```

In [73]:
```
data.corr()
```

Out[73]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concav
id	1.000000	0.039769	0.074626	0.099770	0.073159	0.096893	-0.012968	0.000096	
diagnosis	0.039769	1.000000	0.730029	0.415185	0.742636	0.708984	0.358560	0.596534	
radius_mean	0.074626	0.730029	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	
texture_mean	0.099770	0.415185	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	
perimeter_mean	0.073159	0.742636	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	
area_mean	0.096893	0.708984	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	
smoothness_mean	-0.012968	0.358560	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	
compactness_mean	0.000096	0.596534	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	
concavity_mean	0.050080	0.696360	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	
concave points_mean	0.044158	0.776614	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	
symmetry_mean	-0.022114	0.330499	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	
fractal_dimension_mean	-0.052511	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	
radius_se	0.143048	0.567134	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	
texture_se	-0.007526	-0.008303	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	
perimeter_se	0.137331	0.556141	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	
area_se	0.177742	0.548236	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	
smoothness_se	0.096781	-0.067016	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	
compactness_se	0.033961	0.292999	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	
concavity_se	0.055239	0.253730	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	
concave points_se	0.078768	0.408042	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	
symmetry_se	-0.017306	-0.006522	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	
fractal_dimension_se	0.025725	0.077972	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	
radius_worst	0.082405	0.776454	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	
texture_worst	0.064720	0.456903	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	
perimeter_worst	0.079986	0.782914	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	
area_worst	0.107187	0.733825	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	
smoothness_worst	0.010338	0.421465	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	
compactness_worst	-0.002968	0.590998	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	
concavity_worst	0.023203	0.659610	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	
concave points_worst	0.035174	0.793566	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	
symmetry_worst	-0.044224	0.416294	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	
fractal_dimension_worst	-0.029866	0.323872	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	

32 rows × 32 columns

In [76]:
```
X = data[['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']]
y = data["diagnosis"]
```

In [77]:
```
from sklearn.preprocessing import StandardScaler
# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [78]:
```
from sklearn.feature_selection import SelectKBest, f_classif
# Feature selection
selector = SelectKBest(f_classif, k = 10)
X_selected = selector.fit_transform(X_scaled, y)
```

In [79]:
```
# Print the selected feature names
feature_names = X.columns[selector.get_support()]
print(feature_names)
```

```
Index(['radius_mean', 'perimeter_mean', 'area_mean', 'concavity_mean',
       'concave points_mean', 'radius_worst', 'perimeter_worst', 'area_worst',
       'concavity_worst', 'concave points_worst'],
      dtype='object')
```

Answer 3

You can use non-linear models available through SageMaker such as XGBoost, MXNet etc.

In [52]:
```
train_file = "XGBoost_train.data"

f = io.BytesIO()
smac.write_numpy_to_dense_tensor(f, train_X.astype("float32"), train_y.astype("float32"))
f.seek(0)

boto3.Session().resource("s3").Bucket(bucket).Object(
    os.path.join(prefix, "train", train_file)
).upload_fileobj(f)
```

In [53]:
```
validation_file = "XGBoost_validation.data"

f = io.BytesIO()
smac.write_numpy_to_dense_tensor(f, val_X.astype("float32"), val_y.astype("float32"))
f.seek(0)

boto3.Session().resource("s3").Bucket(bucket).Object(
    os.path.join(prefix, "validation", validation_file)
).upload_fileobj(f)
```

In [54]:
```
from sagemaker import image_uris
container_ = image_uris.retrieve(region=boto3.Session().region_name, framework="xgboost", version="1.5-1")
```

In [55]:
```
xgs_job = "xgs-job" + time.strftime("%Y-%m-%d-%H-%M-%S", time.gmtime())

print("Job name is:", xgs_job)

xgs_training_params = {
    "RoleArn": role,
    "TrainingJobName": xgs_job,
    "AlgorithmSpecification": {"TrainingImage": container, "TrainingInputMode": "File"},
    "ResourceConfig": {"InstanceCount": 1, "InstanceType": "ml.c4.2xlarge", "VolumeSizeInGB": 10},
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://{}/{}/train/".format(bucket, prefix),
                    "S3DataDistributionType": "ShardedByS3Key",
                }
            },
            "CompressionType": "None",
            "RecordWrapperType": "None",
        },
        {
            "ChannelName": "validation",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://{}/{}/validation/".format(bucket, prefix),
                    "S3DataDistributionType": "FullyReplicated",
                }
            },
            "CompressionType": "None",
            "RecordWrapperType": "None",
        },
    ],
    "OutputDataConfig": {"S3OutputPath": "s3://{}/{}/".format(bucket, prefix)},
    "HyperParameters": {
        "feature_dim": "auto",
        "mini_batch_size": "90",
        "predictor_type": "regressor",
        "epochs": "10",
        "num_models": "25",
        "loss": "eps_insensitive_squared_loss",
    },
    "StoppingCondition": {"MaxRuntimeInSeconds": 60 * 60},
}
```

```
Job name is: xgs-job2023-03-22-22-55-17
```

In [56]:
```
region = boto3.Session().region_name
sm = boto3.client("sagemaker")

sm.create_training_job(**xgs_training_params)

status = sm.describe_training_job(TrainingJobName=xgs_job)["TrainingJobStatus"]
print(status)
sm.get_waiter("training_job_completed_or_stopped").wait(TrainingJobName=xgs_job)
if status == "Failed":
    message = sm.describe_training_job(TrainingJobName=xgs_job)["FailureReason"]
    print("Training failed with the following error: {}".format(message))
    raise Exception("Training job failed")
```

```
InProgress
```

In [57]:
```
xgboost_hosting_container = {
    "Image": container,
    "ModelDataUrl": sm.describe_training_job(TrainingJobName=xgs_job)["ModelArtifacts"][
        "S3ModelArtifacts"
    ],
}

create_model_response = sm.create_model(
    ModelName=xgs_job, ExecutionRoleArn=role, PrimaryContainer=xgboost_hosting_container
)

print(create_model_response["ModelArn"])
```

```
arn:aws:sagemaker:ca-central-1:224670572127:model/xgs-job2023-03-22-22-55-17
```

In [58]:
```
xgboost_endpoint_config = "DEMO-xgboost-endpoint-config-" + time.strftime(
    "%Y-%m-%d-%H-%M-%S", time.gmtime()
)
print(xgboost_endpoint_config)
create_endpoint_config_response = sm.create_endpoint_config(
    EndpointConfigName=xgboost_endpoint_config,
    ProductionVariants=[
        {
            "InstanceType": "ml.m4.xlarge",
            "InitialInstanceCount": 1,
            "ModelName": xgs_job,
            "VariantName": "AllTraffic",
        }
    ],
)

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

```
DEMO-xgboost-endpoint-config-2023-03-22-22-59-52
Endpoint Config Arn: arn:aws:sagemaker:ca-central-1:224670572127:endpoint-config/demo-xgboost-endpoint-config-2023-03-22-22-59-52
```

In [59]:
```
%%time

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


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

sm.get_waiter("endpoint_in_service").wait(EndpointName=xgboost_endpoint)

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

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

```
DEMO-linear-endpoint-202303222231
arn:aws:sagemaker:ca-central-1:224670572127:endpoint/demo-xgboost-endpoint-202303222259
Status: Creating
Arn: arn:aws:sagemaker:ca-central-1:224670572127:endpoint/demo-xgboost-endpoint-202303222259
Status: InService
CPU times: user 335 ms, sys: 2.59 ms, total: 337 ms
Wall time: 4min 1s
```

In [60]:
```
def np2csv(arr):
    csv = io.BytesIO()
    np.savetxt(csv, arr, delimiter=",", fmt="%g")
    return csv.getvalue().decode().rstrip()
```

In [61]:
```
runtime = boto3.client("runtime.sagemaker")

payload = np2csv(test_X)

response = runtime.invoke_endpoint(
    EndpointName=xgboost_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 [62]:
```
test_mae_xgboost = np.mean(np.abs(test_y - test_pred))
test_mae_baseline = np.mean(
    np.abs(test_y - np.median(train_y))
)  ## training median as baseline predictor

print("Test MAE Baseline :", round(test_mae_baseline, 3))
print("Test MAE xgboost:", round(test_mae_linear, 3))
```

```
Test MAE Baseline : 0.34900000000000003
Test MAE xgboost: 0.213
```

In [63]:
```
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: 90.7 %
Baseline Accuracy: 65.10000000000001 %
```

Observation

- When I changed the hyperparameters and loss function value, the Prediction Accuracy which was 88.4 % and Baseline Accuracy was 65.1 % initially, changed to Prediction Accuracy being 93.0 % and Baseline Accuracy being as it is.
- After feature engineering on the data, I came to know top 10 features who has more significance on the predictions. The top 10 features are 'radius_mean', 'perimeter_mean', 'area_mean', 'concavity_mean','concave points_mean', 'radius_worst', 'perimeter_worst', 'area_worst','concavity_worst', 'concave points_worst'. This features were the effect of scalling which was done on the X Labels.
- After implementing XGBoost Algorithm instead of Linear Learner Algorithm, it was noticed that the accuracy via. XgBoost was greater by 3 percent "90.7%" w.r.t. Linear Learner.

Cleanup

NOTE: Do not run the following delete endpoint if you want to work on the 3 questions below

In [19]:
```
sm.delete_endpoint(EndpointName=linear_endpoint)
```

Out[19]:
```
{'ResponseMetadata': {'RequestId': 'bdf059ca-63e9-459c-8508-d32e4ab16c85',
  'HTTPStatusCode': 200,
 'HTTPHeaders': {'x-amzn-requestid': 'bdf059ca-63e9-459c-8508-d32e4ab16c85',
  'content-type': 'application/x-amz-json-1.1',
  'content-length': '0',
  'date': 'Wed, 08 Mar 2023 18:32:45 GMT'},
 'RetryAttempts': 0}}
```