Breast Cancer Prediction

1 Background

1.1 Overview

Breast cancer occurs when abnormal cells in the breast begin to grow and divide in an uncontrolled way and eventually form a growth (tumour).

Breast cancer starts in the breast tissue, most commonly in the cells that line the milk ducts of the breast. It is the most common cancer in the UK. It mainly affects women, but men can get it too [1].

Sometimes a small sample of breast cells or breast tissue may be taken to help make a cancer diagnosis. This will usually be done using either [2]

1. a core biopsy.
2. a fine needle aspiration (FNA) or another procedure, such as
3. a punch biopsy

FNA uses a fine needle and syringe to take a sample of cells. The samples can then be examined under a microscope.

1.2 Problem Statement

Features are computed from a digitized image of a fine needle aspirate (FNA) on a breast mass. They describe characteristics of the cell nuclei present in the image in the 3-dimensional space, full details can be found described in [3].

These features are then used to predict whether the breast mass is malignant or benign.

1.3 Dataset Detail & Origin

The dataset is available from Kaggle [4] and is described as the Breast Cancer Wisconsin (Diagnostic) Data Set.

This database is also available through the UW CS ftp server [5] and on the UCI Machine Learning Repository [6].

Kaggle [4] gives the following Attribute Information:

1) ID number
2) Diagnosis (M = malignant, B = benign)
3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from centre to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

1.4 Solution Strategy

After examination of the dataset for missing values and basic characteristics, variable reduction has been investigated using Principal Component Analysis.

The reduced variable dataset is then separated into build and test datasets.

Machine Learning techniques available within Amazon Sagemaker are deployed on the reduced variable datasets in order to initially assess accuracy. The following methods have been considered

1. Linear Learner, as this closely follows the benchmark solution methodology of logistic regression
2. XG Boost, due to the reputed model performance [7]
3. Neural Network in order to cover a deep learning approach

A further examination of the Linear Learner and XG Boost models focusing on the Model recall as the dominant metric is conducted, reflecting the importance of avoiding a False Negative result

1.3 Metrics

The following metrics have been utilised

1. False Positive Rates
2. False Negative Rates
3. True Positive Rates
4. True Negative Rates

This additionally allowed

5. Precision
6. Recall
7. Accuracy

to then be measured

Accuracy forms the comparison with the benchmark model however Recall has also been explored.

2 Data Exploration

2.1 Data Pre-processing

The data exploration begins by reading in the raw data stored in csv format.

Initial examination of the dataset is used to uncover the presence of problem variables or values.

This visual inspection uncovered the presence of a redundant blank variable which was removed.

The next stage was to check for and remove records with missing data. It should be noted that no such records were found, indicated by an unchanged record count at this stage

As the analysis stage of the Data Exploration would be examining whether it would be appropriate to use Standardisation on the variables a high-level view of descriptive Statistics is extracted for each variable. This is illustrated for the radius variables in Table 1 below

	radius_mean	radius_se	radius_worst
count	569.000000	569.000000	569.000000
mean	14.127292	0.405172	16.269190
std	3.524049	0.277313	4.833242
min	6.981000	0.111500	7.930000
25%	11.700000	0.232400	13.010000
50%	13.370000	0.324200	14.970000
75%	15.780000	0.478900	18.790000
max	28.110000	2.873000	36.040000

Table 1 – Descriptive Statistics

The balance of the dataset with respect to the outcome variable is then examined, it can be seen from table 2 that 37% of the FNA results are malignant tumors and 63% benign. This does not present any particular balance issues where the target outcome is an extremely low percentage

diagnosis	nId	pctID
B	357	0.627417
M	212	0.372583

Table 2 – Outcome Balance

As the data will ultimately be used in a predictive model, the final pre-processing step is to convert the diagnosis variable to integer with M=1 and B=0

Implementation Notes

1. The above pre-processing was implemented using basic methods available to the pandas DataFrame class. These methods include
   .read_csv
   .head()
   .drop()
   .describe()

2. As well as the shape property.

2.2 Analysis & Visualisation

Histograms of each variable have been used to assess the distribution against a normal distribution equivalent. The charts below show this for the radius and smoothness variables

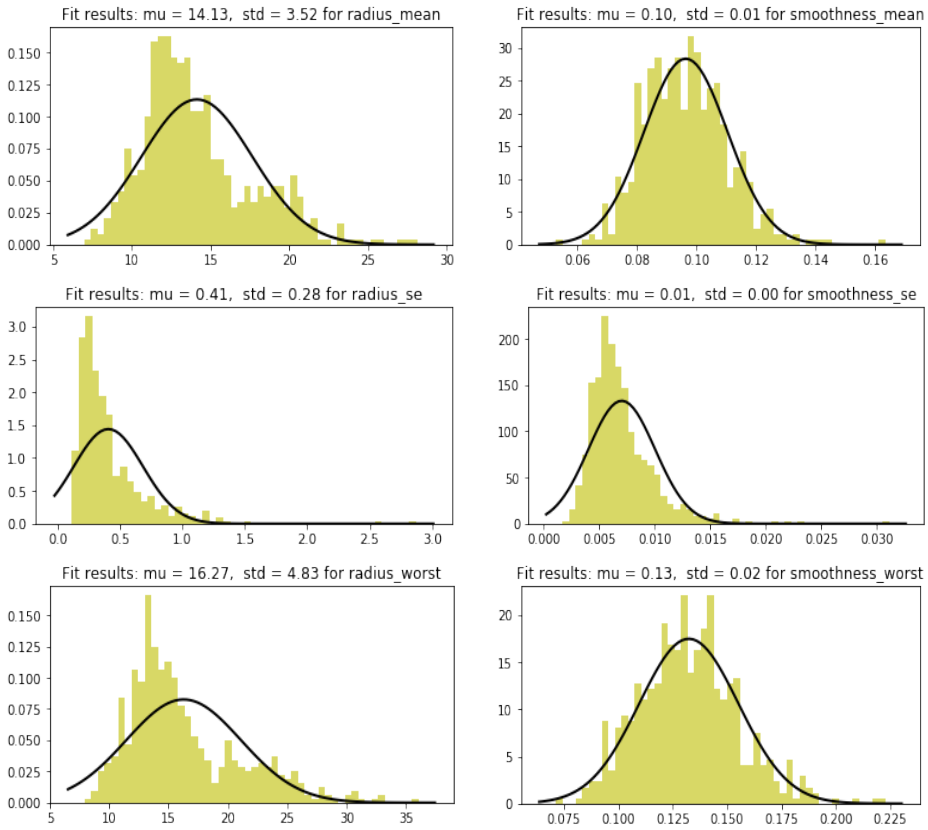


Figure 1 Radius and Smoothness Distributions

Standardisation can be applied when the data is approximately normally distributed. Examining the histograms for all variables, it can be seen that, whilst not perfect in every case, a Standardisation can be justified.

The feature variables are therefore standardised based on their mean and standard deviation.

Implementation Notes

1. Dataframes containing three features are extracted and looped through to plot histograms using the matplotlib package. This was repeated until all features had been examined.
2. The normal distribution curves are constructed using the scipy.stats package
3. Finally the standardisation is performed using the StandardScalar class from the sklearn.preprocessing package and the scaled dataset exported locally in csv format for further use.

2.3 Refinement & Alternative Discussion

A potential alternative to the use of Standardisation would be to Normalise using the MinMax scalar [9] also available in sklearn.preprocessing.

This would transform the features on to a [0,1] scale but would not guarantee unit variance – this has not been explored here.

2.4 Benchmark

The benchmark solution [8] uses a Normalisation approach to the features. This is the first point of difference with the solutions presented here.

3 Principal Component Analysis

3.1 Data Pre-processing

The data exploration section details the conversion to Standardised variables. These form the starting point of the data preparation required to run the Principal Component Analysis (PCA)

In order to take advantage of the PCA modelling available within Sagemaker [10] some further data pre-processing is required. This involves converting the dataframe of features into a Recordset object required by the PCA Model.

Implementation Notes

1. A dataframe of Standardised features is created from the cvs file output of the Data Exploration phase
2. The Sagemaker environment is set up with a Session and Role established
3. A location on S3 is created to store the model attributes
4. The session and role are then used to create a PCA estimator with the number of components one less than the number of features.
5. This estimator is initially used to create the Record set object from the features dataframe having first converted this to a numpy array

3.2 Analysis & Visualisation

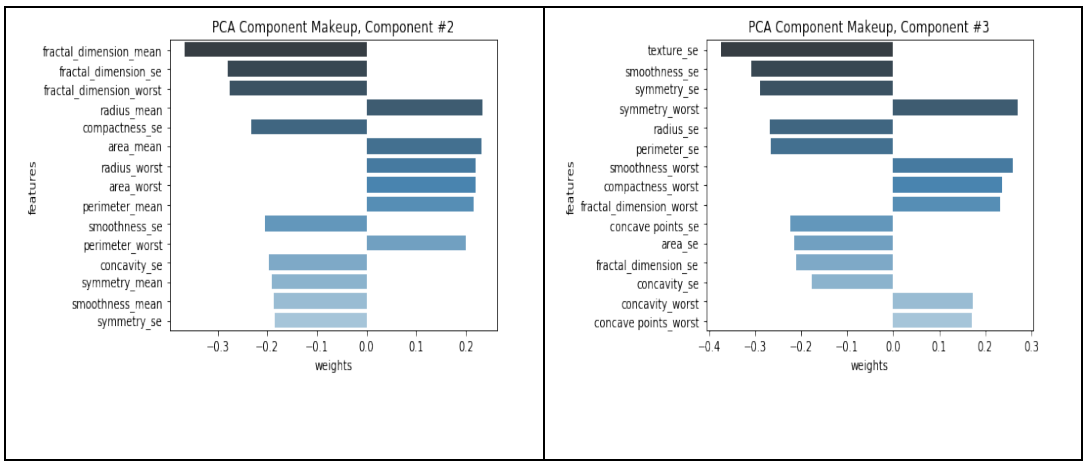
Principal components are projections of the Standardised features, mutually uncorrelated and ordered in variance [11]. The “principal components”, which are defined as weighted, linear combinations of existing features, essentially combine similar or redundant features to form a new, smaller feature set.

The Sagemaker estimator produces 29 components, however 95.2% of the variance within the data is explained by the top 10 components. The variance for each component is an output from the trained estimator, the explained variance is then calculated as:

- calculate s squared for each of the top N components,
- add those up and
- normalize by the sum of all squared s values, according to this formula:

$$\frac{\sum_N s_n^2}{\sum s^2}$$

Some of the components are illustrated below



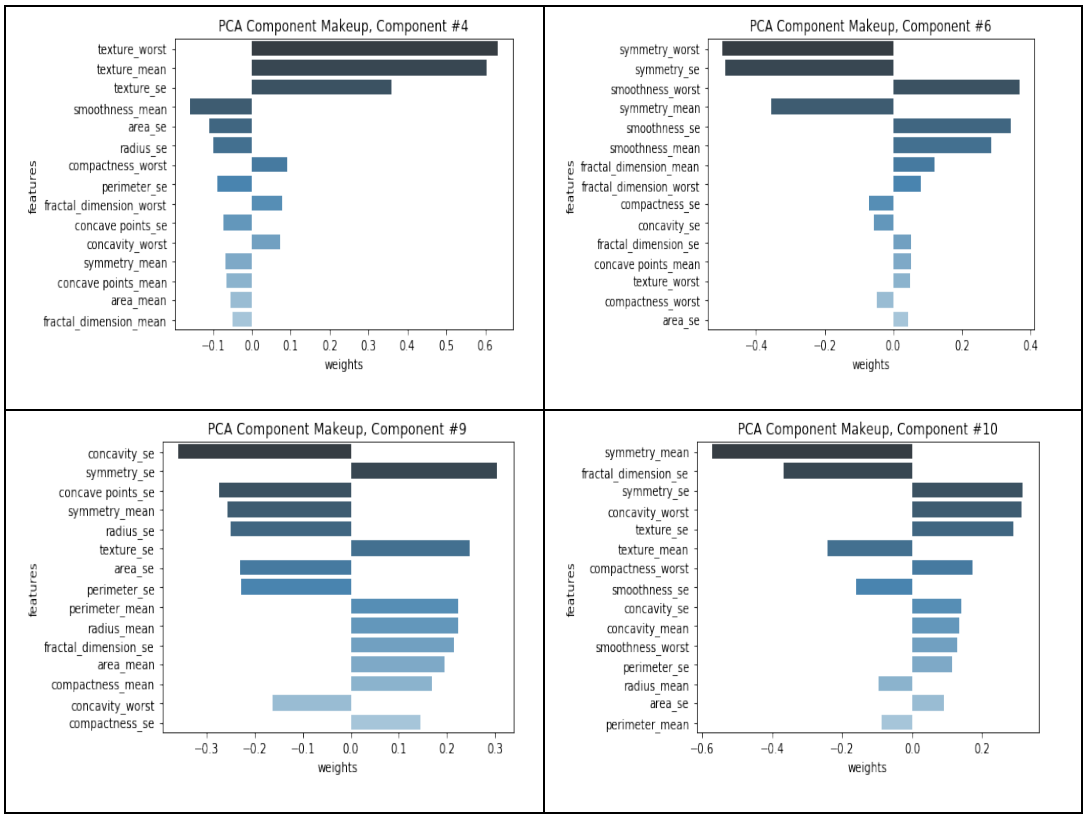


Figure 2 Some of the top 10 Principal Components

Generally it is difficult to describe the components but the salient features of the 6 components illustrated above could be considered as
1a Jaggedness to Size
1b Smoothness and Symmetry - variance to worst case
2a Texture
2b Symmetry to Smoothness
3a Symmetry and Concavity to Size
3b Symmetry, concavity and texture - mean to variance

Implementation Notes

1. The PCA estimators fit method is called with the Recordset created in the Data pre-processing stage
2. Model attributes are unzipped
3. Dataframes of the component variance and weightings are created
4. A function that loops through all components is used to calculate the explained variance
5. A function that uses the seaborn package then plots bar charts of the component weightings for the original variables
6. The PCA model is then deployed in Sagemaker to create a predictor endpoint.
7. This predictor is used on the features training data so that all principal components are calculated for each record
8. A function then extracts the top 10 principal component values for each record
9. Finally the diagnosis value is added to the principal component dataframe
10. The pca dataframe is exported to a csv file for further use in the model building phases

3.3 Benchmark

The benchmark solution does not make use of PCA

4 Linear Learner

4.1 Data Pre-processing

The PCA section details the conversion to 10 principal components. These form the starting point of the data preparation required for the Linear Learner model within Sagemaker.

One approach to model robustness is to randomly select train and test datasets, build the model using a training dataset and assess the model performance on the independent test dataset. If, however, the same metrics for the training dataset are similar, then we can derive some comfort that overfitting is not an issue.

With this in mind further data pre-processing of the principal component dataset has been performed.

Firstly, the PCA data is randomly split into a training (70%) and test (30%) datasets

These are then both transformed into a Recordset format required by the Linear Learner estimator within Sagemaker

Implementation Notes

1. A dataframe of Principal Component features is created from the cvs file output of the PCA phase
2. The Sagemaker environment is set up with a Session and Role established
3. A function is created that uses the seed and shuffle methods of the numpy.random class to generate the train and test arrays
4. A location on S3 is created to store the model attributes
5. The session and role are then used to create a Linear Learner [12] estimator

This estimator is initially used to create the Record set objects from the features train and test numpy arrays.

4.2 Analysis & Visualisation

The baseline LinearLearner estimator is created using the default settings for all parameters. The binary_classifier is used as the predictor type, this runs a Logistic Regression for 25 epochs.

Evaluation of the model performance is conducted using the Test dataset of principal components and is shown below.

Test

Linear Learner - Default	Predicted		
Actual	0	1	Total
0	107	1	108
1	2	61	63
Total	109	62	171

Recall	0.968
Precision	0.984
Accuracy	0.982

Train

Linear Learner - Default	Predicted		
Actual	0	1	Total
0	246	3	249
1	6	143	149
Total	252	146	398

Recall	0.96
Precision	0.979
Accuracy	0.977

Table 3 Test Performance Metrics for the Default Linear Learner Model

As a check on model robustness these metrics are repeated for the training data

With both precision (0.98 : 0.98) and recall (0.96 : 0.97) similar across the test and train datasets we can have some comfort that the model is not being overfitted.

It should be noted that across all records there are 8 False Negative results, so 8 patients with a malignant cancer were missed. This is a 1.4% False Negative Rate

Implementation Notes

1. The Linear Learner estimators fit method is called with the Recordset created in the Data pre-processing stage
2. The trained Linear Learner estimator is then deployed in Sagemaker to create a predictor endpoint.
3. An evaluation function is then constructed that
    a. Splits a features array into batches
    b. Applies the predictor end point and extracts the predictions
    c. Compares the predictions to the labels array associated with the features being predicted, this uses the logical_and method from numpy
    d. Calculates the performance metrics and confusion matrix
4. The evaluation function is applied to the test data to determine model performance and to the training data to assess robustness

4.3 Refinement & Alternative Discussion

A false negative in this context would represent interpreting the FNA outcomes as benign when in fact it was malignant. This is possibly the least desirable outcome from a patient perspective.

From the formula below we can see that reducing false negatives has the effect of increasing recall:

$$Recall = \frac{TP}{TP + FN}$$

Retraining the Linear Learner model with a target recall of 98.5% produced the following performance results

Test

Linear Learner - Recall	Predicted		
Actual	0	1	Total
0	78	30	108
1	1	62	63
Total	79	92	171

Recall	0.984
Precision	0.674
Accuracy	0.819

Train

Linear Learner - Recall	Predicted		
Actual	0	1	Total
0	154	95	249
1	2	147	149
Total	156	242	398

Recall	0.987
Precision	0.607
Accuracy	0.756

Table 4 Test Performance Metrics for the Recall Linear Learner Model

It can be seen that across all records there are now only 3 False Negative results, this is a 0.5% False Negative Rate – almost 1/3 the previous level from the Default model.

There is however a noticeable precision hit as a consequence of more false positives.

4.4 Evaluation Against Benchmark

The logistic regression model run in the benchmark solution has a test volume of only 86 records. The model performance on this test data is only measured in terms of accuracy, with a value of 97.7%.

As the default linear learner model also runs logistic regression, we are able to assess the impact of using the principal components approach based on Standardised variables.

The Default Linear Learner model using the principal components approach based on Standardised variables has an accuracy from the test data of 98.2% and therefore, for this test, has out-performed the benchmark. It should be noted however that the difference in accuracy produces a z score of 0.262 and so at these low volumes this is not a statistically significant increase.

5 XG Boost Model

5.3 Data Pre-processing

For the same reasons explained in section 4.1, further data pre-processing of the principal component dataset has been performed.

Firstly, the PCA data is randomly split into a training (70%) and test (30%) datasets.

The training dataset is then further split into training (80%) and validation (20%) datasets to be used as a deterrent to overfitting within the XG Boost estimator training,

Implementation Notes

1. A dataframe of Principal Component features is created from the cvs file output of the PCA phase
2. The Sagemaker environment is set up with a Session and Role established
3. A function is created that uses the seed and shuffle methods of the numpy.random class to generate the train and test arrays
4. This is reapplied to the training dataset to generate a validation dataset
5. The train, validate and test data is saved locally as csv files
6. A location on S3 is created to store the model attributes
7. The train and validate csv file are uploaded to S3 using the upload_data method of the Sagemaker.session class

5.4 Analysis & Visualisation

The baseline XGBoost estimator is created using binary:logistic as the objective and uses the default evaluation metric [13] for this objective. This focusses on the number of wrong classifications as a fraction of all.

The model produces a probability and after assigning p >= 0.5 = 1.0, evaluation of the model performance is conducted using the Test dataset of principal components and is shown below.

Test

XGBoost - Default	Predicted		
Actual	0	1	Total
0	107	1	108
1	6	57	63
Total	113	58	171

Recall	0.905
Precision	0.983
Accuracy	0.959

Train

XGBoost - Default	Predicted		
Actual	0	1	Total
0	197	7	204
1	7	107	114
Total	204	114	318

Recall	0.939
Precision	0.939
Accuracy	0.956

Table 5 Test Performance Metrics for the Default XGBoost Model

As a check on model robustness these metrics are repeated for the training data

With both precision (0.98 : 0.94) and recall (0.91 : 0.94) similar across the test and train datasets we can have some comfort that the model is not being overfitted.

It should be noted that across the test and train data there are 13 False Negative results. This is a 2.6% False Negative Rate

Implementation Notes

1. An xgboost training container is created
2. The estimator object is instantiated using this container, role and session details
3. The estimator hyperparameters [14] are set with a more conservative boosting in mind.
    a. Initially the early_stopping_rounds was set. to 20 but this constrained the epochs or number of rounds to 16 with a validation error around 0.062. Increasing this to 50 generated 68 rounds and dropped the validation error to 0.05
4. The estimator is trained using it's fit method
    a. As the eval_metric hyperprameter was not set the default error value was used here
5. The estimator is deployed to create a predictor end point
6. The test features are used in the predictor to generate test predictions as are the training features to generate the train predictions
7. The evaluation function described in 4.2 is used with test predictions to assess model performance and with the train predictions as a robustness check

5.5 Refinement & Alternative Discussion

Changing the evaluation metric for the xgboost estimator training to explicitly consider recall has also been considered. The aucpr uses the Area Under the Precision Recall cure. With all other parameters from the default xgboost model from 5.4 retained we obtain the performance below

Test

XGBoost - AUCPR	Predicted		
Actual	0	1	Total
0	107	1	108
1	5	58	63
Total	112	59	171

Recall	0.921
Precision	0.983
Accuracy	0.965

Train

XGBoost - AUCPR	Predicted		
Actual	0	1	Total
0	200	4	204
1	7	107	114
Total	207	111	318

Recall	0.939
Precision	0.964
Accuracy	0.965

Table 6 Test Performance Metrics for the AUCPR XGBoost Model

It can be seen that across all records there are now 12 False Negative results, this is a 2.5% False Negative Rate and so represents negligible improvement over the Default model performance.

It should be noted however that both precision and accuracy have improved slightly through fewer false positives

5.6 Evaluation Against Benchmark

Both the default and AUCPR versions of the XGBoost model fail to meet the 97.7% accuracy achieved by the benchmark logistic regression model. This however is not a statistically significant result given the low volumes

6 Neural Network Model

6.1 Data Pre-processing

For the same reasons explained in section 4.1, further data pre-processing of the principal component dataset has been performed.

The PCA data is randomly split into a training (70%) and test (30%) datasets.

Implementation Notes

1. A dataframe of Principal Component features is created from the cvs file output of the PCA phase
2. The Sagemaker environment is set up with a Session and Role established
3. As before a function is created that uses the seed and shuffle methods of the numpy.random class to generate the train and test arrays
4. The train and test data is saved locally as csv files
5. A location on S3 is created to store the model attributes
6. The train and validate csv file are uploaded to S3 using the upload_data method of the Sagemaker.session class

6.2 Analysis & Visualisation

A fully connected Neural Network was constructed with the following architecture

1. Input Layer with 10 nodes – one for each principal component feature
2. A Rectified Linear Unit (ReLU) activation function
3. 25% drop out applied to help void overfitting
4. A hidden layer with 15 nodes
5. A sigmoid activation function for 1 output dimension

Running too many epochs can result in model overfitting. To guard against this both the train and test datasets were initially run through 100 epochs (gradient recalculation halted for the test data) with an Adam Optimiser examining Binary Cross Entropy Loss after each epoch. It can be seen from Figure 3 below that the test loss bottoms out around 50 epochs

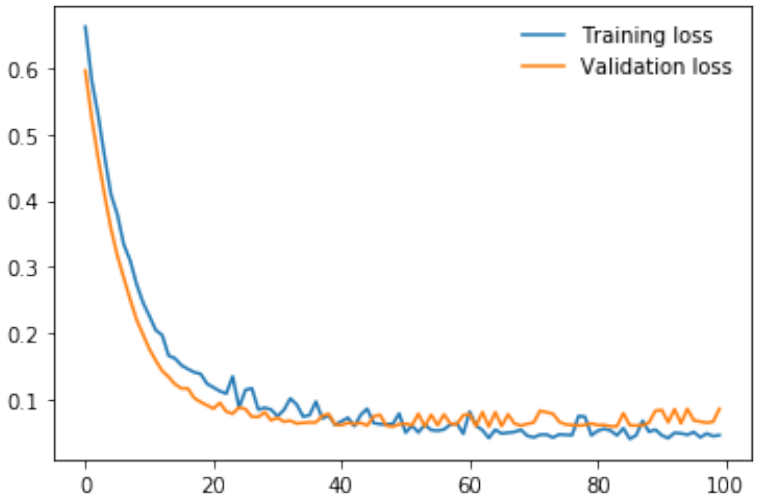


Figure 3 Validation Loss by Epoch

Training the Pytorch Neural Network estimator described above resulted in the performance described in Table 7.

Test

Neural Network	Predicted		
Actual	0	1	Total
0	108	0	108
1	2	61	63
Total	110	61	171

Recall	0.968
Precision	1
Accuracy	0.988

Train

Neural Network	Predicted		
Actual	0	1	Total
0	247	2	249
1	4	145	149
Total	251	147	398

Recall	0.968
Precision	0.986
Accuracy	0.985

Table 7 Test Performance Metrics for the Neural Network Model

As a check on model robustness these metrics are repeated for the training data

With both precision (1.0 : 0.99) and recall (0.97 : 0.97) similar across the test and train datasets we can have some comfort that the model is not being overfitted.

It should be noted that across the test and train data there are 6 False Negative results. This is a 1.1% False Negative Rate

Implementation Notes

1. The torch.nn classes are used to build our own classifier class with the structure defined above
2. The train and test data are transformed to Tenors
3. A DataLoader method is used to construct batches for both train and test tensors
4. The Neural Network model is created along with an Adam optimiser and BCELoss criterion
5. The train and test tensor are both looped through 100 epochs to determine the point at which the test loss bottoms out
6. A location on S3 is defined for model output storage
7. A PyTorch estimator is instantiated using the session and role from Sagemaker
    a. This requires the entry_point to be defined for training the model, this is given in a separate train.py code
8. The estimator is trained using it's fit method
9. The estimator is deployed to create a predictor end point
    a. The entry_point is defined in a separate predict.py code
10. The test features are used in the predictor to generate test predictions as are the training features to generate the train predictions
11. The evaluation function described in 4.2 is used with test predictions to assess model performance and with the train predictions as a robustness check

6.3 Evaluation Against Benchmark

The Neural Network model produces an accuracy of 98.8% and as such out-performs the benchmark model accuracy of 97.7%. Again this is not a statistically significant result given the low volumes.

7 Conclusions

7.1 Accuracy

Five models have been evaluated against the benchmark logistic regression model [8] of Nisa Soylu. This evaluation has been based purely upon the accuracy for the test dataset in each case. The chart in Figure 4 below shows that the Neural Network and Default Linear Learner models are capable of out-performing the benchmark model.

Given that the Default Linear Learner employs the same modelling technique as the benchmark model, there is some evidence that the PCA approach to feature preparation has contributed to the over performance here.

These results, however, are not statistically significant given the low volumes.

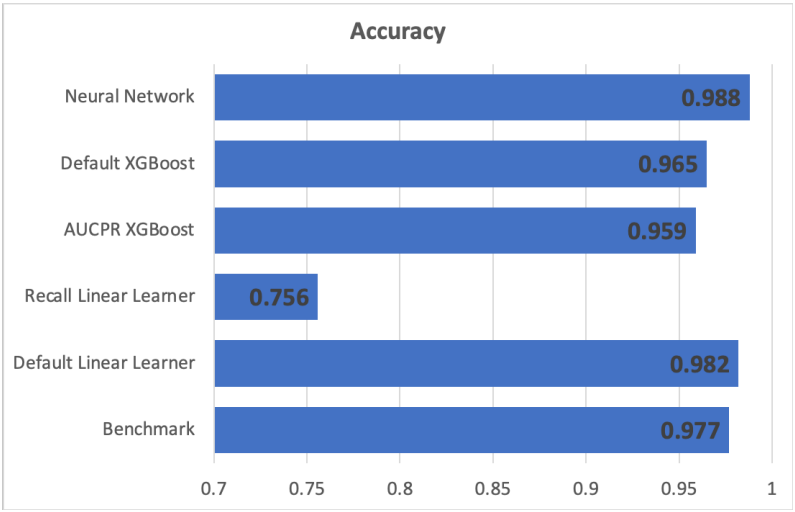


Figure 4 Accuracy Evaluation

It can also be seen that in the search to eliminate False Negatives the Recall Linear Learner model suffers a statistically significant drop in accuracy.

7.2 False Negative Rates

As a false negative is likely to be the least desirable outcome from a patient perspective, a further aim of this investigation has been to try to reduce these outcomes.

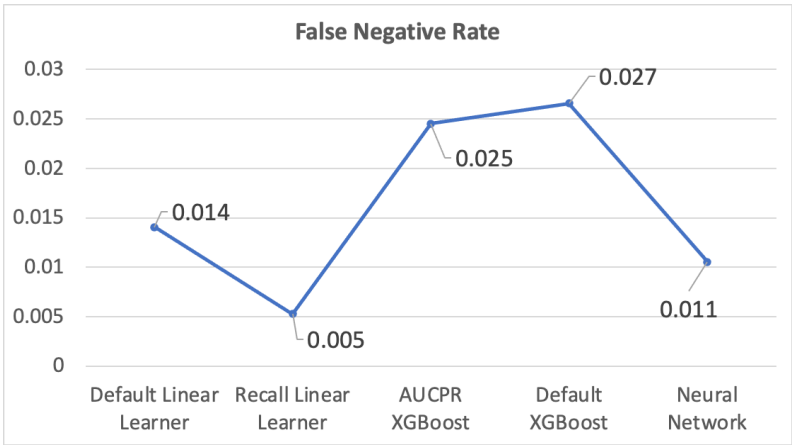


Figure 5 False Negative Rates by Model

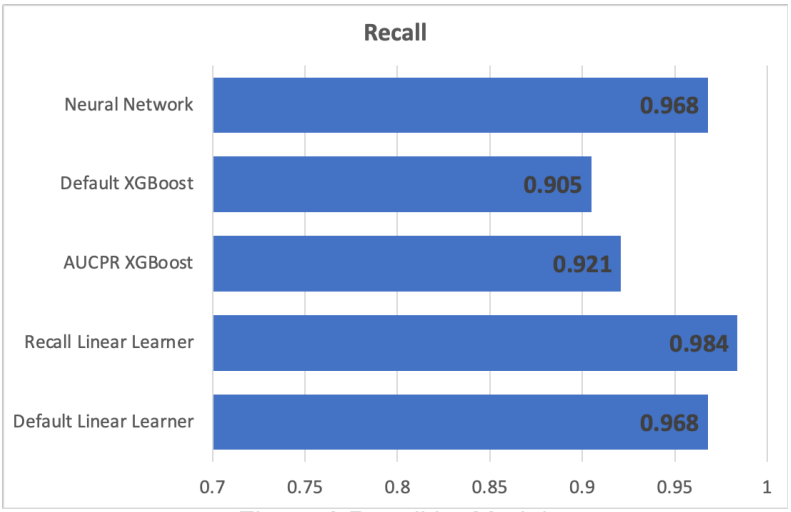


Figure 6 Recall by Model

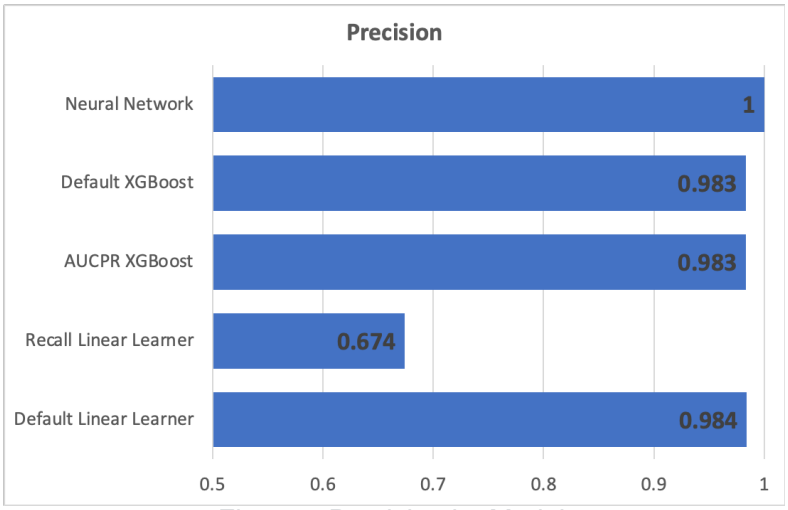


Figure 7 Precision by Model

The Recall Linear Learner was trained to a specified Recall level and has achieved this resulting in the lowest False Negative rate across both test and train datasets. However, as Figure 7 indicates, this improved Recall is at a significant expense to precision

The Neural Network produces a relatively low False Negative rate whilst avoiding unacceptably low precision.

The XG Boost model has the highest False Negative rates and although this can be improved by using an evaluation metric with greater focus on Recall it is still out performed on this metric by the other models

Refereneces

[1] https://www.cancerresearchuk.org/about-cancer/

[2] https://breastcancernow.org/information-support/have-i-got-breast-cancer/core-biopsy-fine-needle-aspiration-fna?gclid=EAIaIQobChMI54rj9-KO7AIVSebtCh2eVQctEAAYASAAEgI0D_D_BwE

[3] K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34

[4] https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

[5] ftp ftp.cs.wisc.edu cd math-prog/cpo-dataset/machine-learn/WDBC/

[6] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

[7] https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

[8] https://www.kaggle.com/nisasoylu/machine-learning-implementation-on-cancer-dataset

[9] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[10] https://sagemaker.readthedocs.io/en/latest/pca.html

[11] T. Hastie, R Tibshirani and J Friedman; “Elements of Statistical Learning”, Springer Series in Statistics, 2008

[12] https://sagemaker.readthedocs.io/en/stable/algorithms/linear_learner.html

[13] https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost_hyperparameters.html

[14] https://github.com/dmlc/xgboost/blob/master/doc/parameter.rst#learning-task-parameters

[15] https://pytorch.org/docs/stable/nn.html#loss-functions