Springboard Data Science Capstone Project II

Breast Cancer Prediction from Digitized Image of Fine Needle Aspirate (FNA) of a Breast Mass Using Deep Neural Network (TensorFlow 2.0)

Frew Berhe

December 13, 2019

Table of Contents

1. Introduction

Breast cancer is a type of cancer with high mortality rates among women, and it is one of the most common causes of death in women. According to National Cancer Institute statistics in America, one out of eight women suffers from breast cancer and 6% of all deaths worldwide are caused by this type of cancer. Despite major advances in genetics and modern imaging, the diagnosis catches most breast cancer patients by surprise. For some, it comes too late. Later diagnosis means aggressive treatments, uncertain outcomes, and more medical expenses. Early diagnosis of breast cancer (maximum 5 years after the first cancer cell division) will increase survival chances from 56% to 86%. Besides, accurate diagnosis of breast cancer is of prime importance. Thus, a precise and reliable system is essential for the timely diagnosis of benign or malignant breast tumors.

In order to detect breast cancer, radiologists conduct Fine Needle Aspirate (FNA) procedure of breast tumor. FNA is a simple, inexpensive, noninvasive and accurate technique for detecting breast cancer. This procedure reveals features such as tumor radius, area, perimeter, concavity, texture and fractal dimensions. These features are further studied by medical experts to classify tumor as benign or malignant. Pathologists require a lot of expertise and skill to perform the analysis on the FNA sample. Applying the suitable features of the FNA results is the most important diagnostic problem in early stages of breast cancer. Hence, development of algorithms which rely on digitized image analysis, is of great interest. The key objective of this project was to predict breast cancer as benign or malignant using data set from the digitized image of FNA sample. We proposed to use data and build a Deep Neural Network (DNN) model using TensorFlow

2.0 that can help doctors find the cancer cells and ultimately save human lives. Doctors and pathologists in different Hospitals can use such a model to identify breast cancer from digitized images of FNA sample, which as a result would help them make early clinical decisions with greater accuracy.

2. Data Acquisition and Cleaning

The data set was acquired from the Kaggle Competition. This data (Breast Cancer Wisconsin Diagnostic Data Set) contains information computed from a digitized image of FNA of a breast mass along with labels (B=Benign, & M=Malignant). It describes the characteristics of the cell nuclei present in the image. We used this digitized image data in order to fit a better model. Since the data is already labeled, DNN using TensorFlow 2.0 is a perfect choice to build a predictive model with greater accuracy.

The dataset has 569 observations and 33 columns. Out of the 33 total columns, ten were real-valued features that are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter)

b) texture (standard deviation of gray-scale values)

c) perimeter

d) area

e) smoothness (was quantified by measuring the difference between the length of each radius and the mean length of adjacent radii.)

f) compactness (perimeter^2 / area - 1.0)

g) concavity (was determined by measuring the size of any indentations in the nuclear border.)

h) concave points (counted the number of points on the nuclear border that lie on an indentation.)

i) symmetry (was measured by finding the relative difference in length between line segments perpendicular to and on either side of the major axis.)

j) fractal dimension (was approximated using the “coastline approximation” described by Mandelbrot that measured nuclear border irregularity)

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. The column ‘diagnosis’ is the target variable that has two classes labeled as ‘B’ and ‘M’ which stands for Benign and Malignant respectively. The rest two columns; ‘id’ and ‘Unnamed: 32’ are dropped from the dataset as they are not relevant for our analysis. Our dataset didn’t have any missing value. Moreover, all the features are of data type 'float64' except the variable 'diagnosis' which is 'object'. In order to make it suitable for our Deep Learning algorithm, we replaced the ‘B’ and ‘M’ classes of the target variable by 0 and 1 respectively. As a result, the data type of our target variable became ‘int64’. More details on acquiring, cleaning, merging, parsing these datasets can be found in IPython notebook.

3. Data Exploration

3.1 Introduction to the cleaned data

Our cleaned data has 569 observations and 31 columns. Our target variable is ‘diagnosis’, which is a binary variable with 0 representing Benign (not cancerous) and 1 representing Malignant (cancerous). We will go through most of the features of the dataset to explore their relationship with Malignant Diagnosis.

3.2 Understanding the target variable

The total number of samples that are benign are 357 (62.7%), while that of malignant/cancerous is 212 (37.3%). The proportion of malignant in our dataset is unusually large as compared to the real-world proportion of malignancy in which most masses are benign. However, this larger proportion of malignant cases is so important for our DNN as it learns well when data has a balanced proportion of classes.

3.3 Summary Statistics

From the descriptive statistics, we can understand that the mean of 'smoothness_mean', 'compactness_mean', 'symmetry_mean', 'texture_mean', 'symmetry_mean', 'symmetry_worst', 'smoothness_worst', 'texture_worst' is almost same to the median represented by the 50%. However, the mean of the rest features is different from the median. Besides, there is remarkably large difference between the maximum value and the 75 percentile (75%) of the feature 'perimeter_mean', 'area_mean', 'area_se', 'perimeter_se', 'perimeter_worst', and 'area_worst'. In addition, there is also a big difference between 95th percentile and maximum value for 'perimeter_mean','area_mean', 'area_se', 'perimeter_se', 'perimeter_worst', and 'area_worst'. Hence, we can conclude that these features ('perimeter_mean','area_mean', 'area_se', 'perimeter_se', 'perimeter_worst', and 'area_worst') have outliers. There is no big difference between the 5th percentile and minimum value for all the variables. More details on descriptive statistics of these dataset can be found in this IPython notebook.

3.4 Visual exploratory data analysis

Outliers are data values that are far away from other data values. It is important to check for the presence of such values in all the features as they are prone to affect our prediction. A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution. You can see IPython notebook for further information on the box plot of the 30 features. The black dot above or below the whiskers indicate the presence of outlier. All the five features our data set have outliers. Specifically, while all features have only extremely large values (one direction), the 'smoothness_mean' feature has both extremely large and extremely small values as indicated by the dots in both directions. With that in mind, we also plotted a '*distplot*' to check the skewness of the distribution of all the thirty features. Fig. 1 shows the distribution of all the thirty features.

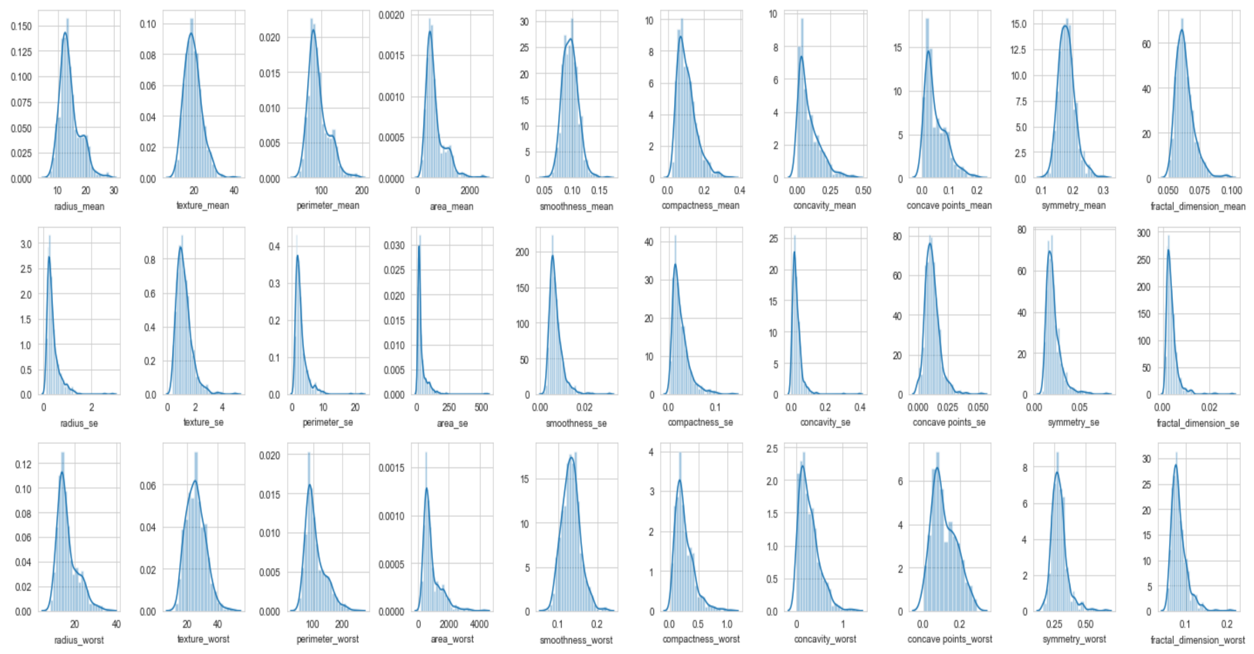

Figure 1: Distribution of all features.

The above figure depicts that 'texture_mean', 'smoothness_mean', 'symmetry_mean' 'texture_worst', 'smoothness_worst', 'symmetry_worst', 'concave points_se' appear to be normally distributed. It is also witnessed by the box plot displayed in IPython notebook that the median (middle line) of the box lies at the center between q1 and q3. On the other hand, the rest features are slightly skewed to the right.

Nuclear morphometry is a useful objective tool in differentiating benign and malignant breast lesions. One of which is Mean Radius, which is the mean of distances from center to points on the perimeter of cell nuclei. The mean radius for the malignant is much larger than that of the benign one. It is 17.5 for malignant and 12.2 for benign. This is also witnessed by the swarm plot and the density plot. The data points for the benign are compact, but for the malignant the data points are scattered as can be seen from the swarm plot and the density plot.

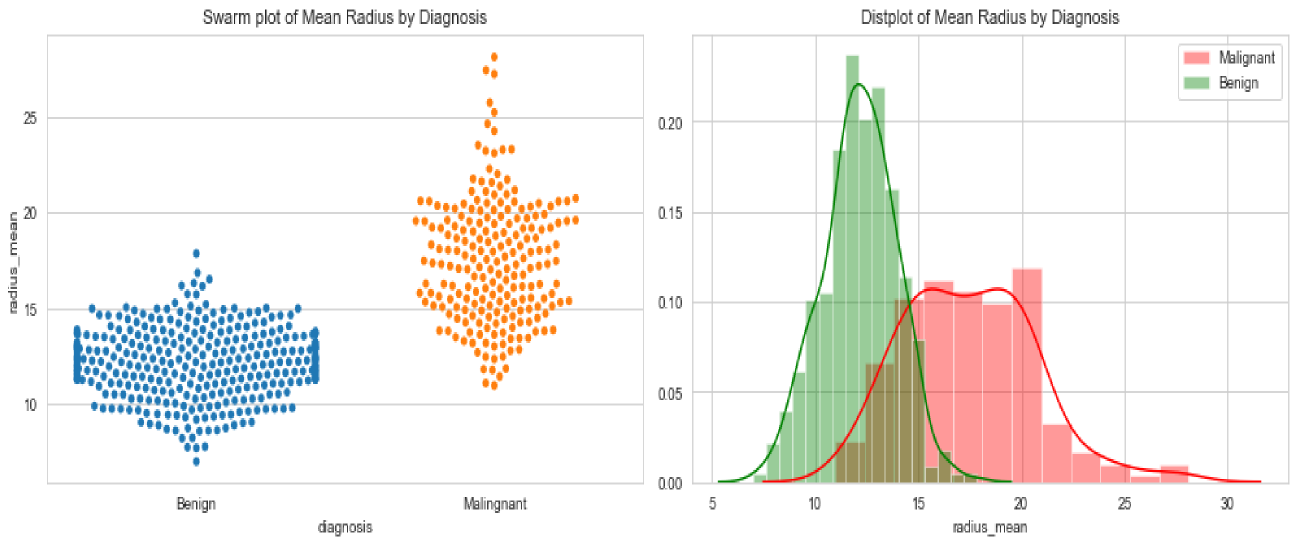

Figure 2. radius_mean by diagnosis

The second feature, mean texture, is the mean of the standard deviation of gray-scale values of cell nuclei. There is some difference in the proportion of mean texture between the malignant and benign. The mean texture for the malignant is 21.6 while it is 17.9 for the benign. Hence, the Malignant nuclei has larger mean texture that the benign. We can see from the swarm plot drawn on top of violin plot that both groups data points are compact.

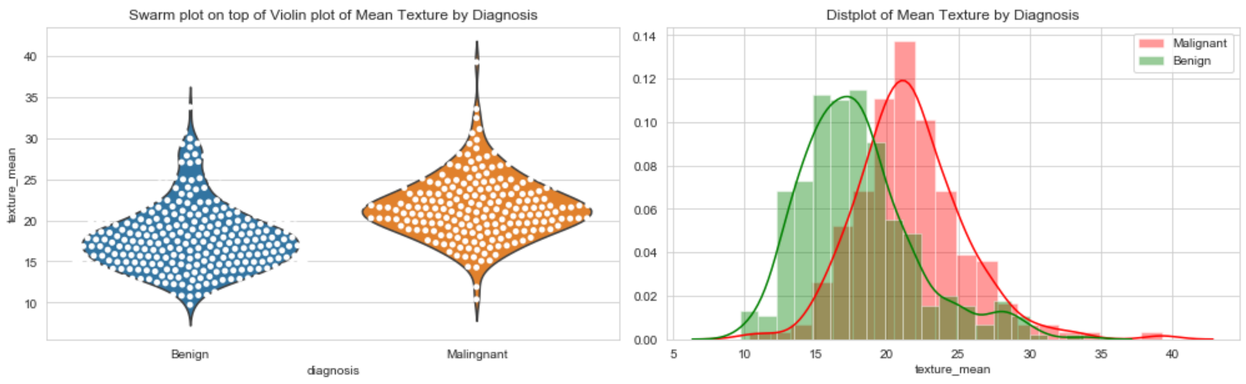

Figure 3. texture_mean by diagnosis

Mean perimeter is another nuclear morphometric that indicates the mean of the distance around the nuclear border. Like the mean radius, the malignant group has notably larger mean of nuclei perimeter as compared to the benign group. The mean of mean perimeter is 115.4 for the malignant and 78.1 for the benign. In addition, the malignant group has a dispersed data point with some outliers, while the benign has compact data points.

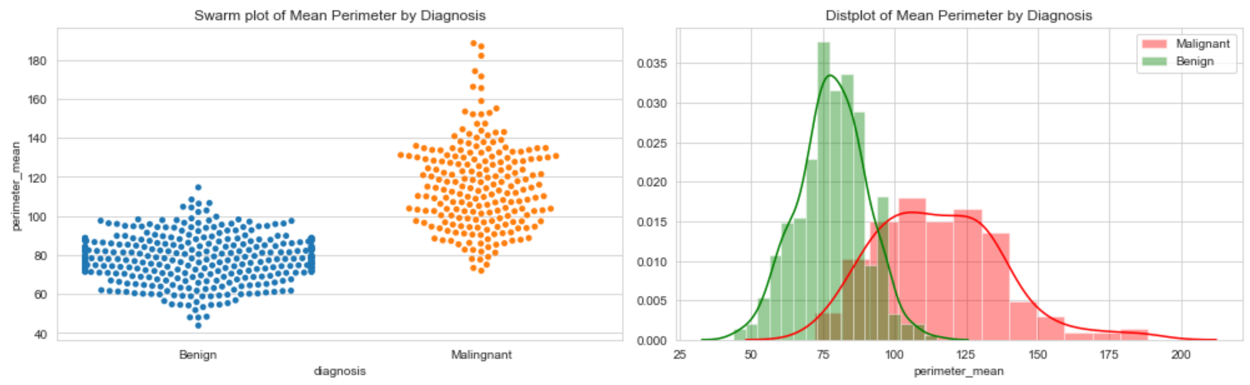

Figure 4. perimeter_mean by diagnosis

Mean area is defined as the mean of the area within the outlined nuclear perimeter. In line with the findings of mean radius and mean perimeter, the malignant group has more than double the mean of area mean of those of benign group. The mean of area mean for the malignant and benign group is 978.4 and 462.8 respectively. The malignant group has scattered data points with outliers, whereas the benign one has compact data points.

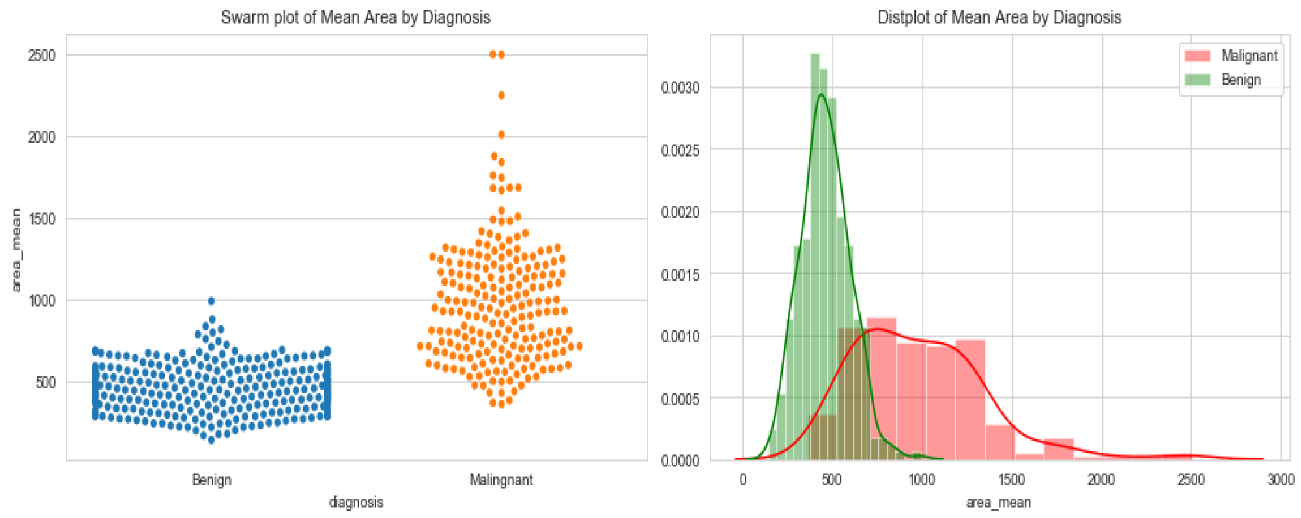

Figure 5. area_mean by diagnosis

Mean Smoothness is the local variation in radius lengths of cell nuclei. There is a slight difference in the mean of smoothness mean between both groups. The mean of smoothness means for the malignant group (0.10) is slightly higher than the benign group (0.09). Both groups have compact data points.

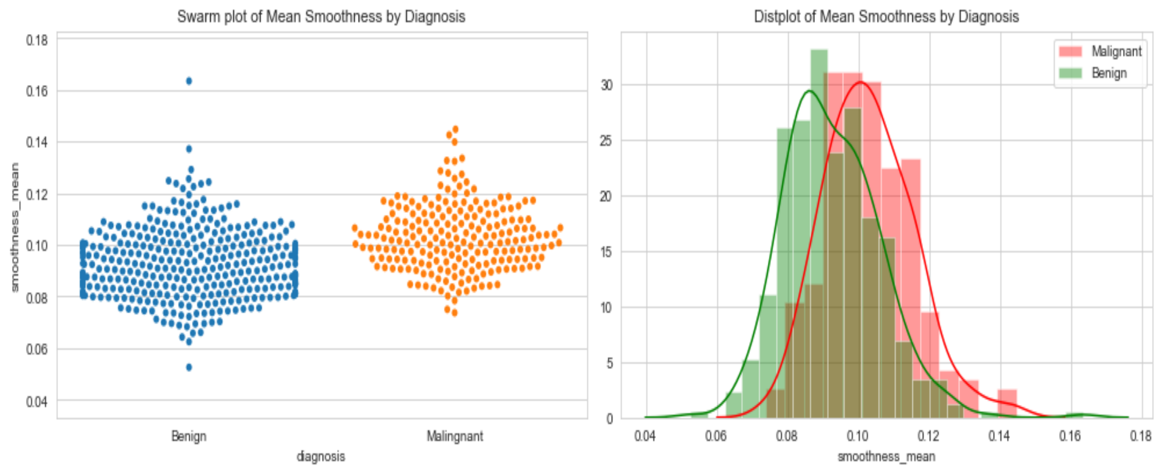

Figure 6. smoothness_mean by diagnosis

Compactness is a feature derived from two features mentioned above (perimeter, and area). Its formula is (perimeter^2 / area - 1.0). Like both mean area and mean perimeter, the mean of compactness mean of the malignant group has larger (0.15) mean than the benign group (0.08). This is not surprising since it is derived from these two features. The malignant group has scattered data points with outliers, whereas the benign one has compact data points.

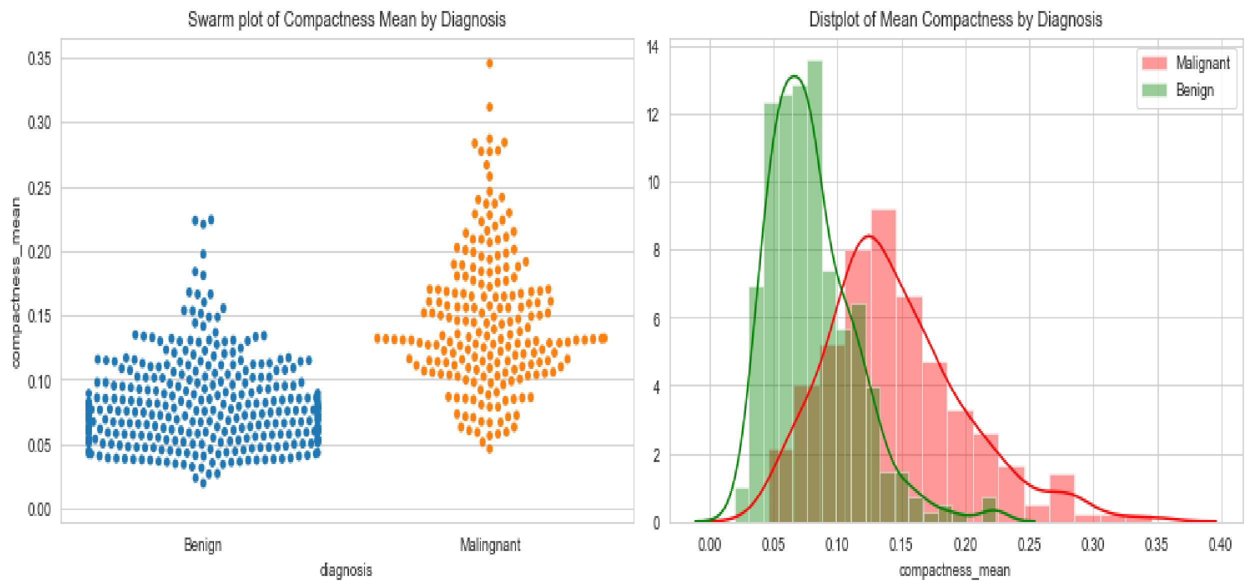

Figure 7. compactness_mean by diagnosis

Concavity mean is the mean severity of concave portions of the contour. The mean of concavity mean is 0.16 for malignant while it is 0.04 for the benign one, indicating a large difference between the two. The malignant group has scattered data points with some outliers, whereas the benign group has compact data points with few outliers.

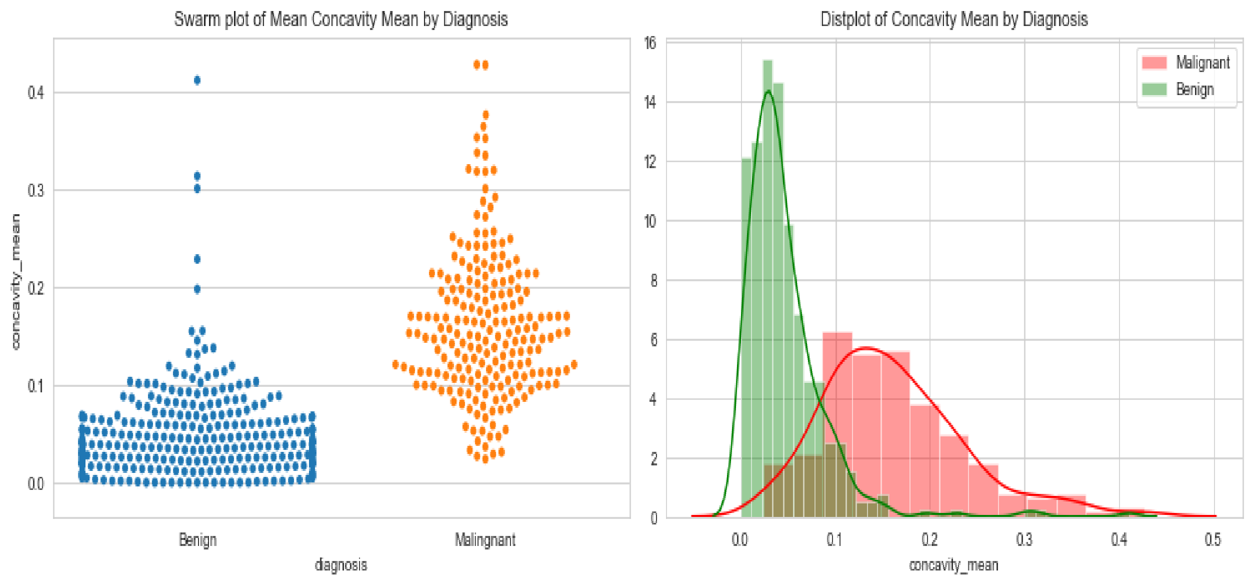

Figure 8. concavity_mean by diagnosis

Concave points mean is the mean of the number of concave portions of the contour. From the above chart we can see that the difference in mean and the distribution is concave points mean is like that of concavity mean. The malignant nuclei image has larger mean (0.09) of concavity mean than the benign one (0.03). The malignant group has scattered data points with some outliers, whereas the benign group has compact data points with few outliers.

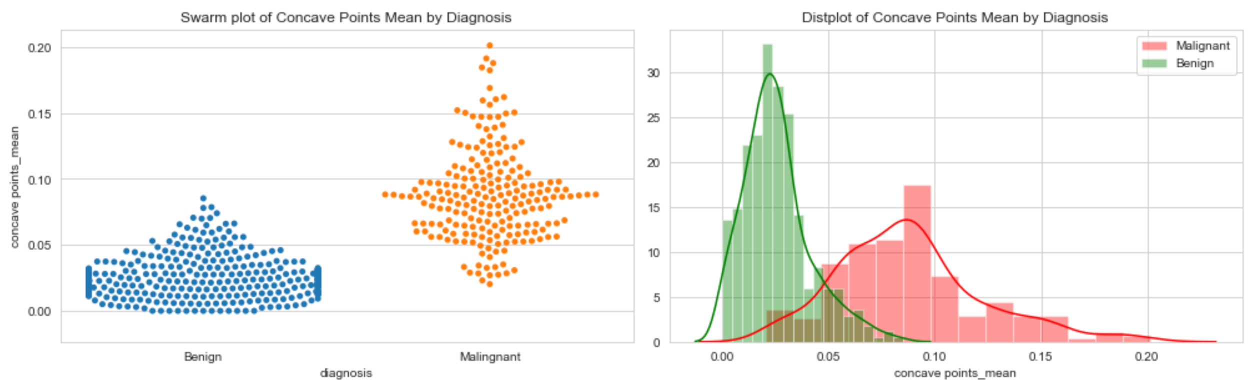

Figure 9. concave points_mean by diagnosis

There is no big difference in the mean symmetry between the two groups. The mean of symmetry means of the malignant is 0.19 and that of benign is 0.17. The malignant group has slightly higher mean than the benign group. Both groups have compact bell-shaped distribution with few outliers.

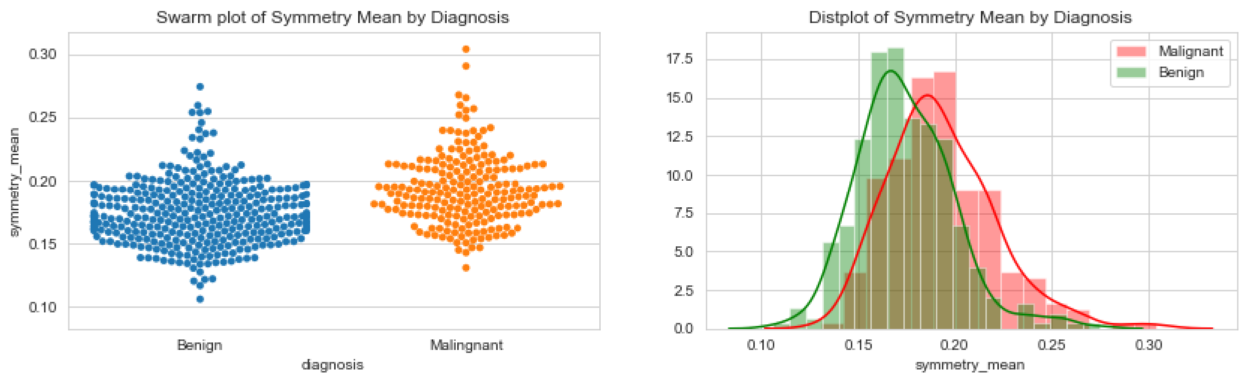

Figure 10. symmetry_mean by diagnosis

The fractal dimension is approximated using the “coastline approximation” described by Mandelbrot that measured nuclear border irregularity. The mean of fractal dimension mean is the same for both groups. Both groups have also similar distribution of data points.

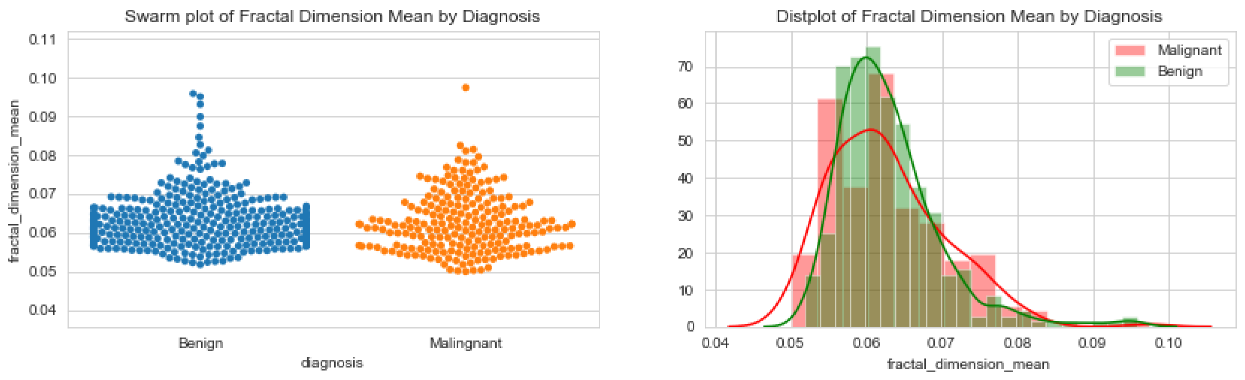

Figure 11. fractal_dimension_mean by diagnosis

Generally, 'radius_mean', 'perimeter_mean', 'concavity_mean', 'concave points_mean' are the four features that seem most relevant features for our prediction, followed by 'area_mean', 'compactness_mean', and 'texture_mean'. We observed Higher mean values in malignant group than benign group for all these six features. Contrarily, the fractal dimension mean, is the one that didn’t show any difference, while symmetry mean, and smoothness mean only show slight difference between the two groups. On the other hand, in most of the features the malignant groups have scattered data points as compared to the benign groups data points.

The ten features displayed below are the standard errors for the for the ten real valued features. Features like 'radius_se', 'perimeter_se', 'area_se', 'compactness_se', 'concavity_se', 'concave points_se' showed a slight difference in standard error between malignant and benign groups. The rest didn't show any difference in standard error. Fig 12 shows distplot of all the standard error related features.

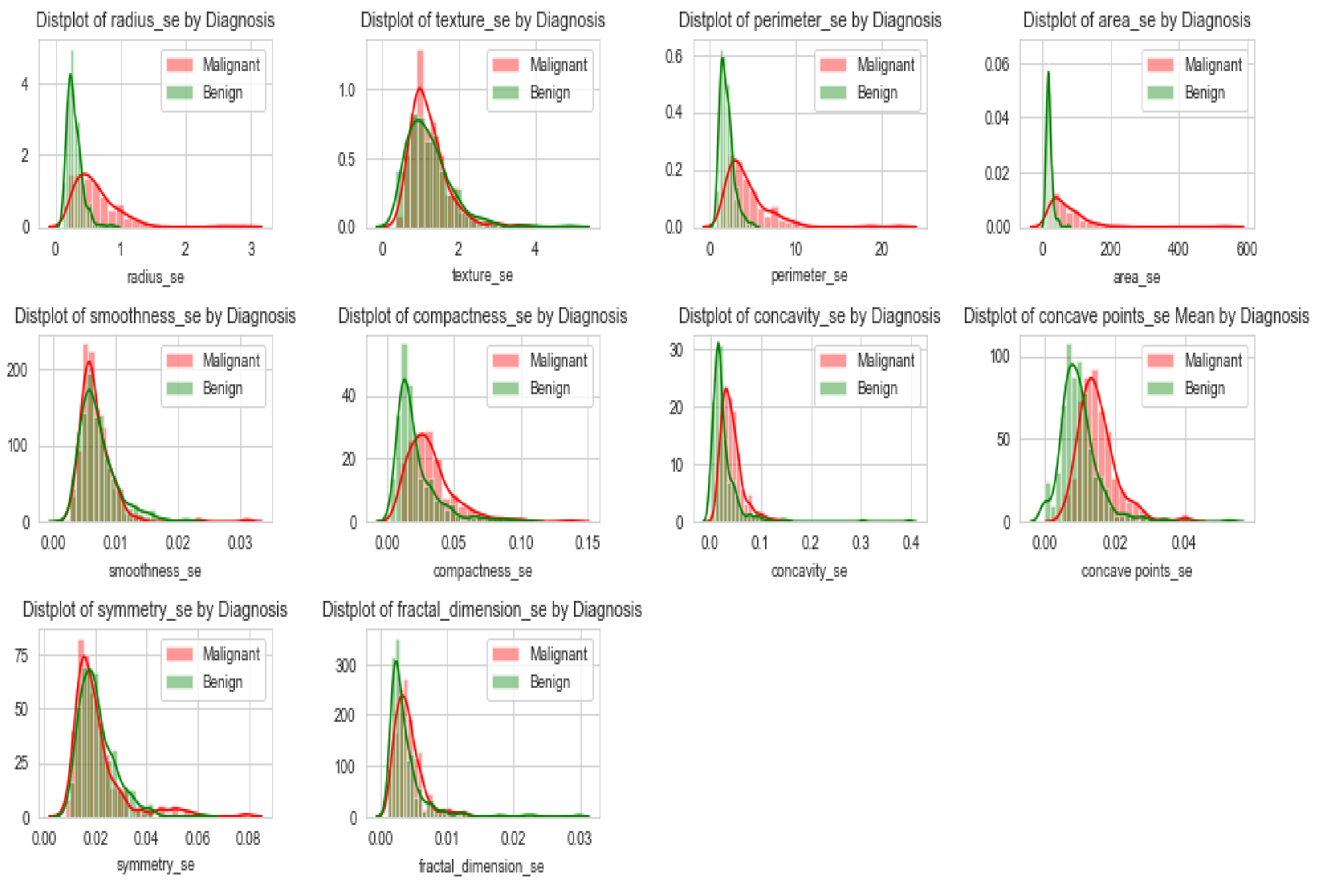

Figure 12. Standard error of all the ten real valued features by diagnosis

In our case, the suffix 'worst' indicates mean of the three largest values of each of the ten real valued features. The above chart indicated that the malignant group has a higher mean than the benign group for all the ten features. Features like 'radius_worst', 'perimeter_worst', 'area_worst', 'compactness_worst', 'concavity_worst', 'concave points_worst' showed bigger difference in mean between the malignant and benign group. Fig. 13 shows distplot of all the worst mean related features.

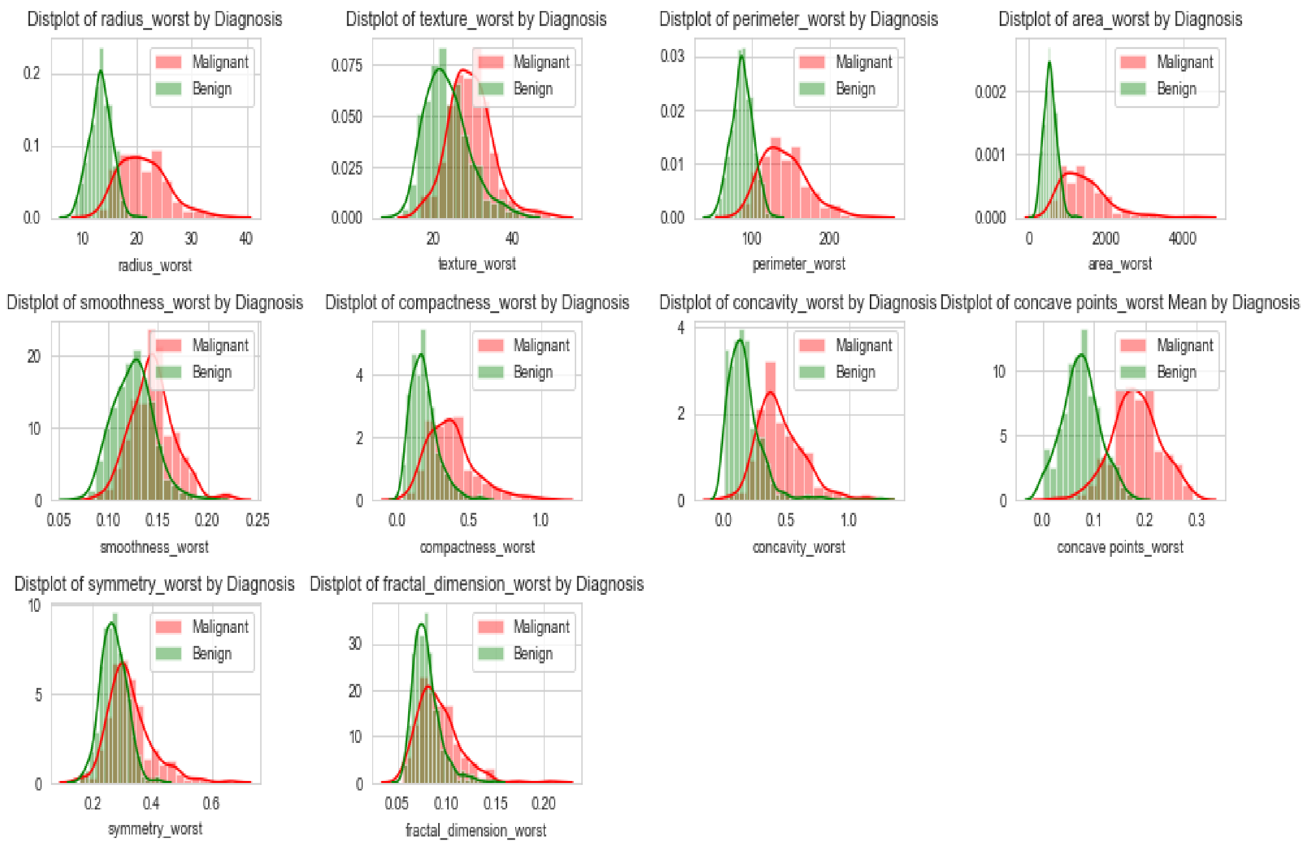

Figure 13. Worst mean of all the ten real valued features by diagnosis

The relevance of the six features stated above in classifying the target variable is witnessed by the scatter plot below. On the other hand, the scatter plot showed 'radius_mean', 'perimeter_mean', 'area_mean' are highly correlated for each other indicated by the steep data point among each other. For more information about correlation, we will do correlation matrix below. Fig. 14 shows pair plot of the six mean of real valued features highly correlated with the target variable.

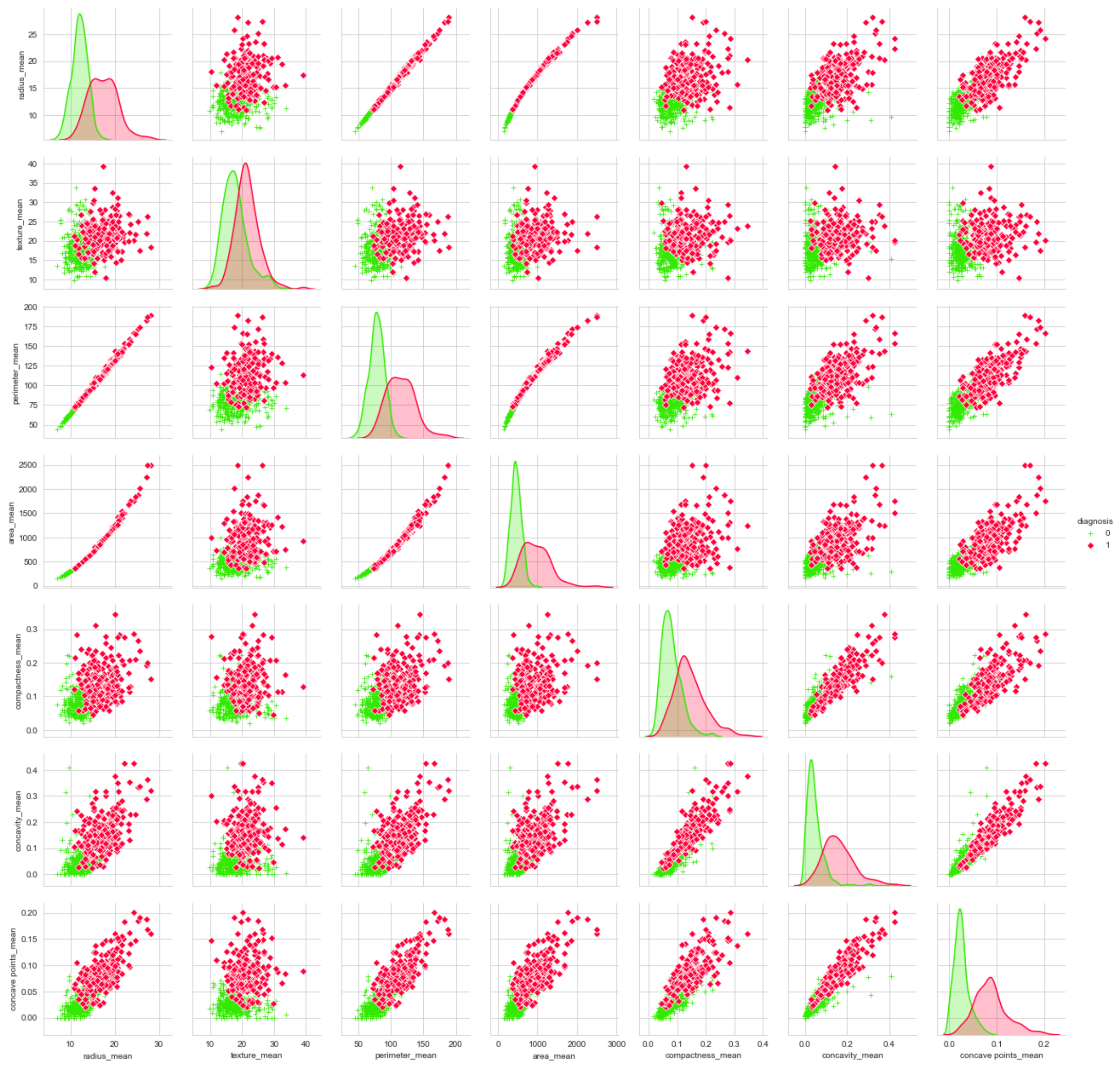

Figure 14. Pair plot of the six features by diagnosis

3.4.1 Correlation

The correlation matrix below shows the correlation of features among the top 20 highly correlated variables with the target variable.

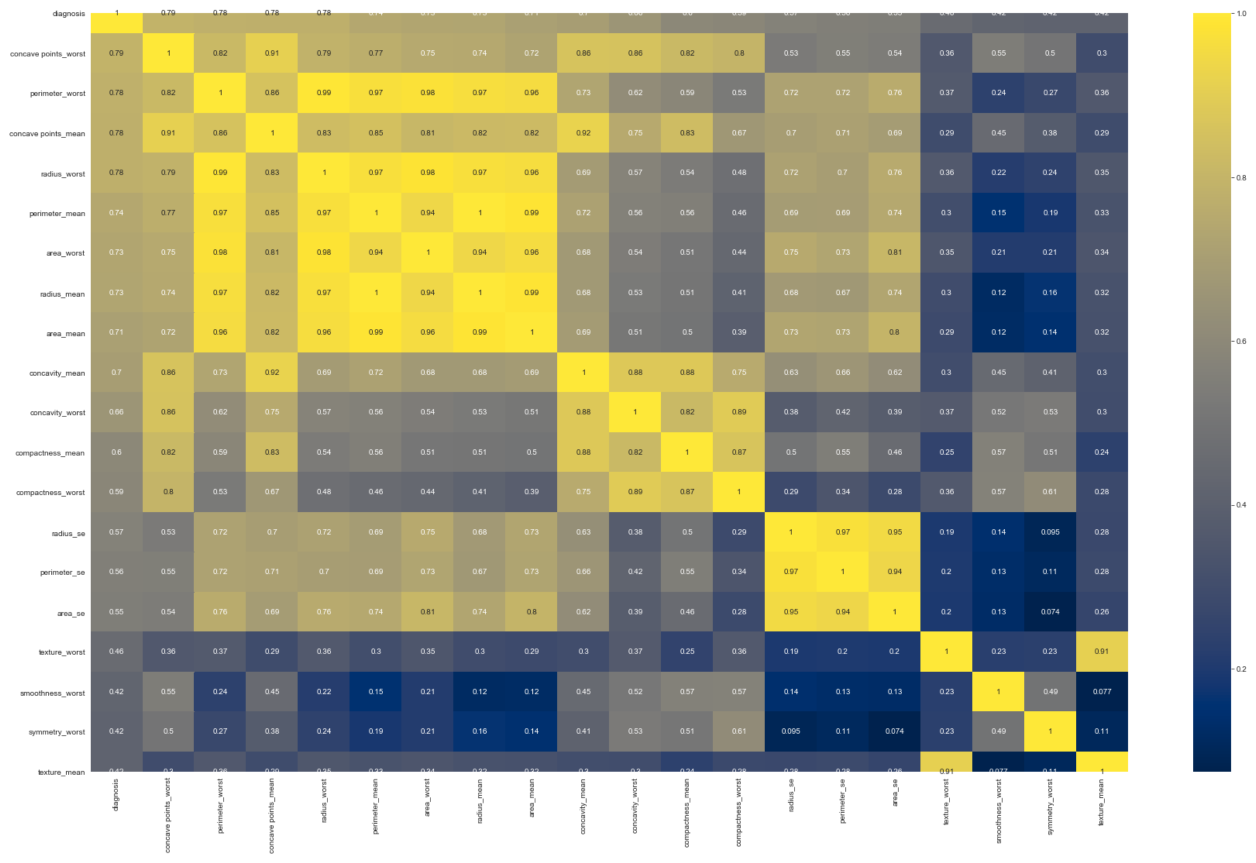

Figure 15. Correlation matrix of the top 20 correlated features to the variable diagnosis

Regarding the correlation with the target variable, concave points_worst, perimeter_worst, concave points_mean, radius_worst are the four top features that have the highest correlation with the target variable. On the other hand, the following features are the most correlated features with each other:

- radius_mean, perimeter_mean,area_mean,perimeter_worst, radius_worst, and area_worst
- texture_mean , and texture_worst
- area_se, perimeter_se, and radius_se

- compactness_mean, compactness_worst, concavity_mean, concavity_worst, and concave points_worst.

3.5 Outliers Handling

Data values that are either less than the 2nd percentile or above the 98th percentile are removed from the dataset. We removed four observations that lies in the mentioned range. Out of the four outliers that we removed, two were malignant and the other two were benign.

4. Modeling

Since the data is already labeled (0 for Benign, and 1 for malignant) a supervised machine learning algorithm is a perfect choice to build a predictive model. Moreover, since there are only two classes in the target data (0 and 1), we used binary classification algorithms. Therefore, in order to successfully classify our data into either malignant or benign, we applied DNN by using TensorFlow 2.0. DNN is one type of Deep Learning. Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks**.** It is gaining much popularity due to its supremacy in terms of accuracy as compared to traditional machine learning. Neural networks are very good at classifying data points into different regions, even in cases when the data are not linearly separable.

However, Deep Learning requires high-end machines contrary to traditional Machine Learning algorithms. GPU has become an integral part now to execute any Deep Learning algorithm. TensorFlow allows for the execution of code on either CPU or GPU, which is a useful feature especially when you're working with a massive dataset. For our project we used latest CORE i7 laptop with ***nvidia*** graphics (GPU) in order to use TensorFlow 2.0.

4.1 Data Pre-processing

In binary classification scenarios, it's good to have a balanced proportion of data from both classes. We have a 63%-37% distribution**,** which is good enough. Thus, we don't need to balance our dataset. Before feeding the data into any DNN algorithms, there are some preprocessing steps that must be performed on the data. We outline these steps below.

4.1.1 Handling Categorical variable: To run the different machine learning algorithms on this data, we would have to convert all non-numeric features into numeric ones. Luckily, we don't have any categorical feature that needs to be converted to numeric.

4.1.2 Scaling: Feature scaling in machine learning is one of the most important steps during preprocessing of data before creating machine learning model. DNN models learn a mapping from input variables to an output variable. As such, the scale and distribution of the data drawn from the domain may be different for each variable. Differences in the scales across input variables may increase the difficulty of the problem being modeled. An example of this is that large input values (e.g. a spread of hundreds or thousands of units) can result in a model that learns large weight values. A model with large weight values is often unstable, meaning that it may suffer from poor performance during learning and sensitivity to input values resulting in higher generalization error. This can make a difference between a weak machine learning model and a strong one. Since our data set has values of features with different scale of measurement ranging from fractions in features like ‘smoothness_mean’ to thousands in features such as ‘area_mean’; scaling would be paramount. We needed to bring all features to the same level of magnitudes. Subsequently, we achieved this by scaling, that is by standardizing our data into a scale 0 to 1.

4.1.3 Shuffling: For datasets that are sorted in a specified order, If the order of data within each epoch is the same, then the model may use this as a way of reducing the training error, which is a sort of overfitting. Hence, shuffling data serves the purpose of reducing variance and making sure that models remain general and overfit less. However, our dataset is not sorted in a specified order, and thus we don’t need to shuffle our data.

4.1.4 Split Data into Training, Validation, and Test set: We split the scaled data set into 80% training, 10% validation and 10% test set. As a result, we got 452 observations for training, 56 observations for test and 57 observations for test set. The training dataset is the sample of data used to fit the model. It is the actual dataset that we use to train the model. The model sees and learns from this data. Validation dataset is the sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. Test dataset is the sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. It provides the gold standard used to evaluate the model. It is only used once a model is completely trained (using the train and validation sets).

4.1.5 Saving the Split datasets in *.npz format: After we split the scaled dataset, we saved it in “*.npz” format, which is TensorFlow friendly format. The “.savez” method of *numpy* combine and saves several arrays into a single file in *uncompressed* .npz format. Now, our dataset is ready to be loaded and modeled using the DNN algorithm.

4.2 Define Model

One of the first steps in building a neural network is finding the appropriate activation function. We used the *rectified linear unit activation function* referred to as *ReLU* on the three hidden layers. The input size is the total number of features we have in our dataset, which is 30. However, in TensorFlow version of Keras we don't need to specify this input size as it infers it from the argument 'x' of 'model.fit' and only then it builds the whole model. In addition, after repeated trials and errors we selected 40 nodes for the size of the hidden layers. In our case, we wish to predict if the digitized image of nuclei is malignant or benign. Therefore, this can be framed as a binary classification problem. Ideally, we would have a function that outputs 1 for a malignant, and 0 otherwise. Thus, we used the sigmoid function as an activation function for the output layer of the DNN to ensure our network output is between 0 and 1 and easy to map to either a probability of class 1. As part of hyperparameter tuning we tried *softmax* as an activation function for the output layer, but it didn't yield a good result. As a result, we decided to stick with *sigmoid* activation function. Now that the model is defined, *we can compile it*.

4.3 Compile Model

When compiling, we must specify some additional properties required when training the network. Remember training a network means finding the best set of weights to map inputs to outputs in our dataset. We must specify the loss function to use to evaluate a set of weights. The optimizer is used to search through different weights for the network and any optional metrics we would like to collect and report during training. In this case, we used "**binary_crossentropy**" as the **loss** argument which is a loss used for a binary classification problem. Initially we tired " sparse_categorical_crossentropy" as the loss but it didn't work well with the model. On the other hand, we defined the **optimizer** as the efficient stochastic gradient descent algorithm "**adam**"(Adaptive Moment Estimation). This is a popular version of gradient descent because it automatically tunes itself and gives good results in a wide range of problems. Finally, because it is a classification problem, we used two metrics namely accuracy and AUC, defined via the **metrics** argument.

4.4 Fit Model

We have defined our model and compiled it ready for efficient computation. Now it is time to execute the model on some data. We can train or fit our model on our loaded data by calling the **fit()** function on the model. Hence, we fit the model to the training data and validation data in order to train and validate the model respectively. Here, the purpose of using validation data is to validate the data so that we can prevent overfitting the model parameters

Training occurs over epochs and each epoch is split into batches. One epoch is comprised of one or more batches, based on the chosen batch size the model is fit for many epochs. We must also set the number of samples that are going to be propagated through the network before the model weights are updated within each epoch, called the batch size. The batch size and number of epochs have been set by trial and error. For our case, we used 200 epochs and used a relatively moderate batch size of 64 using the epochs argument and the batch_size argument respectively. We want to train the model enough so that it learns a good enough mapping of rows of input data to the output classification. The model will always have some error, but the amount of error will level out after some point for a given model configuration. This is called model convergence. Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on the validation dataset. Thus, we set early stopping along with 2 as a value for the option ***patience*** using the ***callbacks*** argument. "***patience"*** is defined as the number of epochs with no improvement after which training will be stopped. Furthermore, we set the shuffle argument to 'False' so that we can get reproducible results. However, only setting the shuffle argument to false doesn't stop TensorFlow from producing different result every time you run the model. It must be accompanied by around five lines of codes that forces TensorFlow to use a single thread (For further detail you can refer to the second cell of this IPython notebook for DNN.

After we trained the model, we fiddled with the hyperparameters such as depth and width, batch size, activation functions for each layer. Lastly, we got an accuracy of 0.9956 and AUC of 0.9998 for the training data and accuracy of 0.9821 and AUC of 1.00 for the validation data. Therefore, this indicates that we have an excellent model as there is no significant difference between the training data and validation data.

Table 1. Evaluation metrics for training and validation data.

	Training Data	**Validation Data**
Loss	0.0203	0.0510
AUC	0.9998	1.0000
Accuracy	0.9956	0.9821

Here, we usually keep an eye on the validation loss of the model or set ***Early Stopping*** in order to prevent overfitting. In our case, we have no danger of overfitting as we already set ***Early Stopping*** mechanisms. The validation accuracy on the other hand is the true accuracy of the model for the epoch. This is because the training accuracy is the average accuracy across batches while the validation accuracy is that of the whole validation set. Nevertheless, this is not our end result rather it is just a step that helps us make sure the weights and biases don't overfit. From the figure below we see a plot of the training process. We can observe that there is no big difference between the training and validation error indicating that the model didn't overfit.

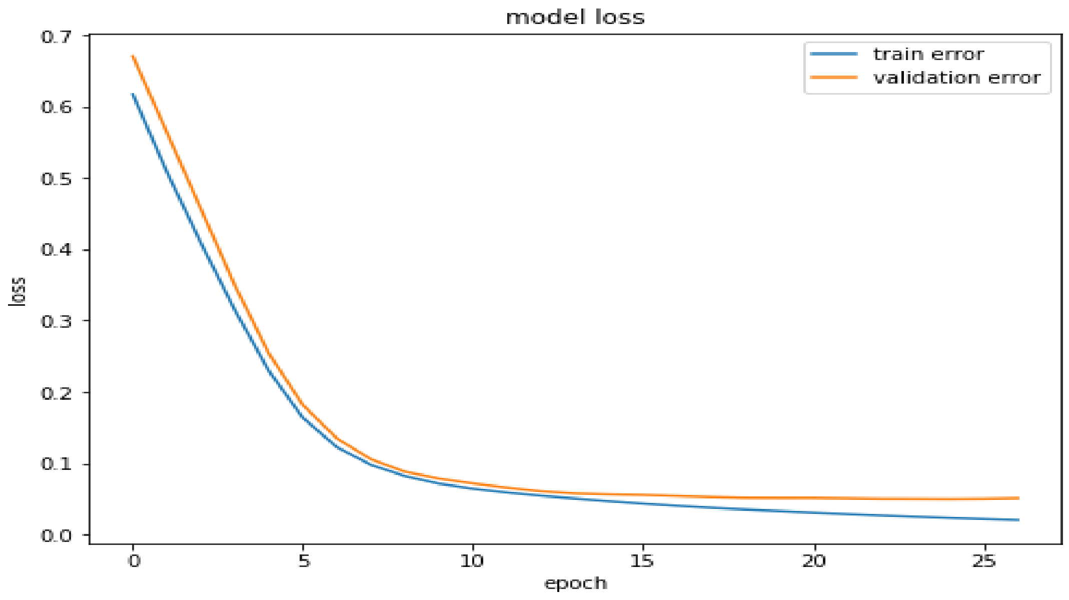

Figure 16. Plot of the training process

4.5 Evaluate Model

We evaluated our model on our testing dataset using the **evaluate()** function on our model. This will generate a prediction for each input and output pair and collect scores, including the average loss and any metrics we have configured, such as accuracy and AUC. The test dataset is our reality check that prevents us from overfitting the hyperparameters. It is the dataset that the model has truly never seen.

Table 2. Evaluation metrics for testing dataset.

Metrics	**Test Data**
AUC	0.9967
Accuracy	0.9825
Loss	0.0979

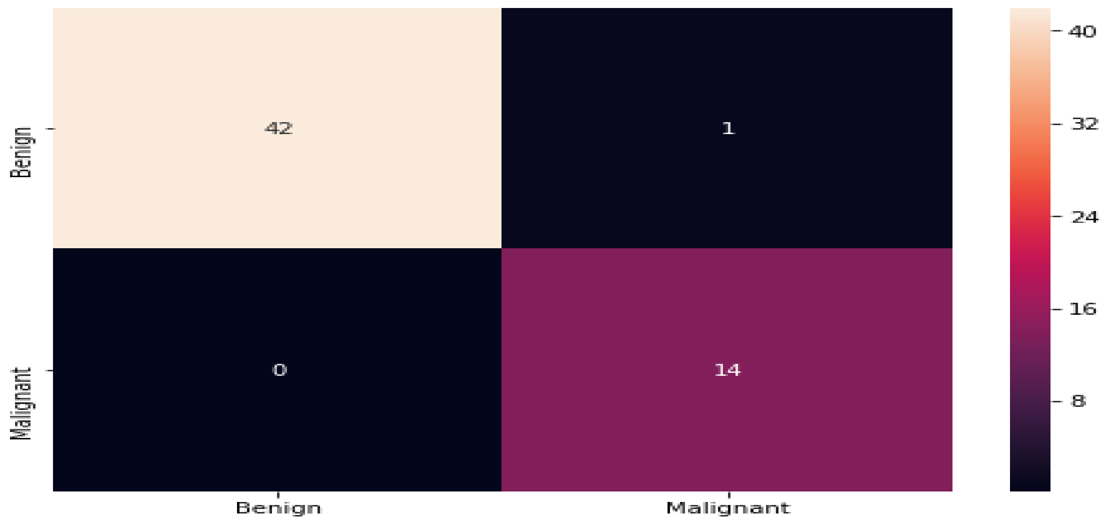

Figure 17. Heatmap for the confusion matrix

Finally, we have got an outstanding result from the test data which is in line with validation data results we got earlier. We got an AUC score of 0.9967, and accuracy of 0.9825 with extremely small loss (Table 2). Besides, we got a test AUC and accuracy very close to the validation AUC and accuracy indicating that we have not overfit. The heat map of the confusion matrix for the test data shows that out of the total 57 test observations 56 are correctly predicted (42 true benign, and 14 true malignant) and only 1 incorrectly labeled as benign (false negative). The AUC/ROC curve below is a visual witness of how outstanding our prediction model is.

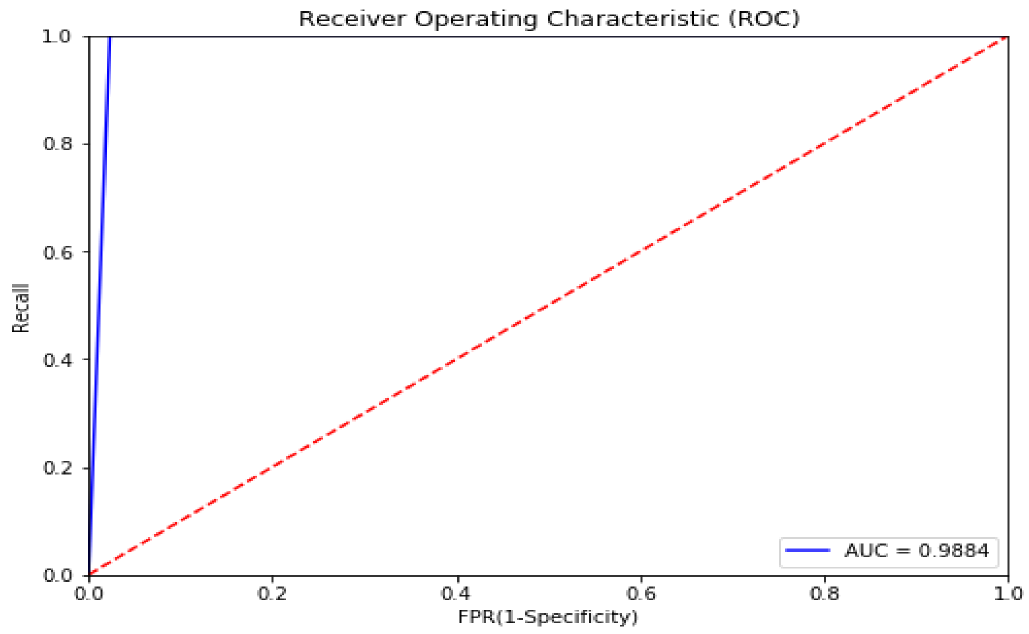

Figure 18. AUC/ROC Curve

4.6 Using SHAP Values to Explain our TensorFlow Model

Unlike model Scikit-learn, model in TensorFlow doesn’t have any attribute or method to calculate feature importance. Instead, we will use SHAP to understand how our TensorFlow model is making predictions. SHAP, or SHapley Additive exPlanations, is a Python library created by Scott Lundberg that can explain the output of many machine learning frameworks. It can help explain an individual prediction or summarize predictions across a larger population. The summary plot below displays feature importance using a distribution of Shapley values for the top twenty out of the total thirty features. The color of each point is on a spectrum where highest values for that feature are red, and lowest values are blue. The features are ranked by the sum of the absolute values of the Shapley values.

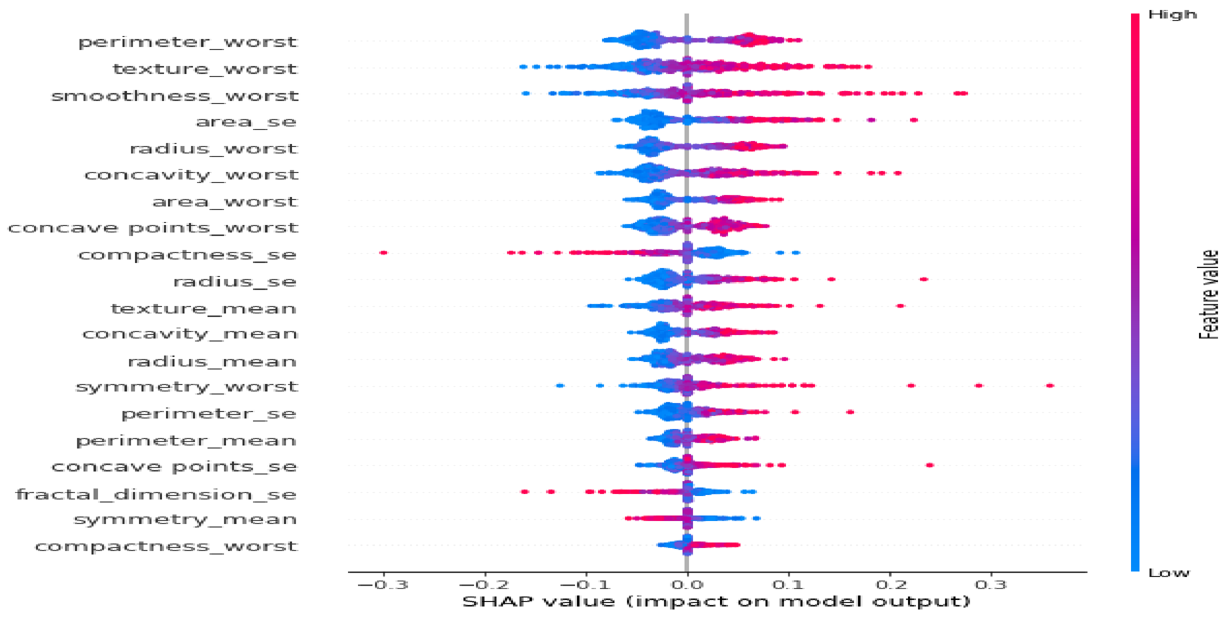

Figure 19. Summary Plot of SHAP values of features

The first three features with the highest contribution are the **'perimeter_worst', 'texture_worst', 'smoothness_worst'**; followed by **'area_se', 'radius_worst', 'concavity_worst', 'concave points_worst', 'area_worst'**. Note that each of these features have predominantly red dots (high feature values) on the right-side where there are positive SHAP values. This tells us that high values for these features lead our model to predict a malignant diagnosis. On the contrary, **'compactness_se',**

'fractal_dimension_se', and **'symmetry_mean'** has an opposite relationship, where a lower value is associated with a malignant diagnosis.

Alternatively, as shown in fig. 20, we can also just take the mean absolute value of the SHAP values for each feature to get a standard bar plot

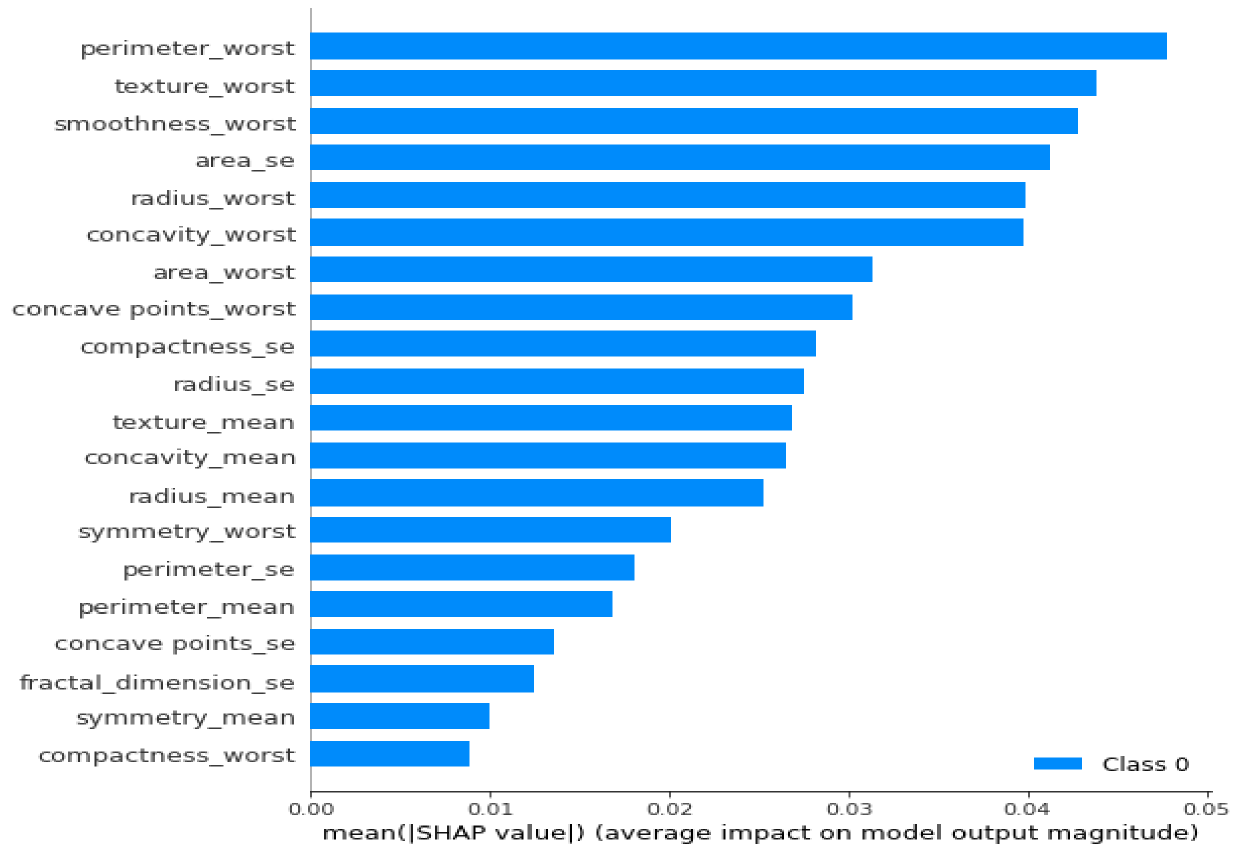

Figure 20. Standard Bar Plot of SHAP values for each feature

5. Using Model and Recommendations

In order to make predictions, one would need to select the 20 Important features, scale the input data, and take it into the model. As a result, one can achieve about 98% accuracy in predicting whether a tumor is malignant or benign. Therefore, Doctors and pathologists in different Hospitals can apply this outstanding model to identify breast cancer from digitized images of FNA sample, in order to make early clinical decisions with greater confidence. This would be a major breakthrough in the healthcare industry as it can play a substantial role for the early and accurate detection of the disease. Early detection can give patients more treatment options, which in turn would increase their survival.

6. Assumptions and Limitations

In order to make a prediction on new data, the prediction model is dependent on how good the morphologic measurement of the nuclei from the FNAC sample. This means that the model will predict better if the morphologic measurement of the nuclei FNAC is better. Furthermore, a sample size of 569 is relatively small for DNN as more sample for training can ensure better performance of model.

7. Conclusions

We explored the Wisconsin breast cancer diagnostic data set to classify the diagnosis of FNAC sample. The proportion of Benign to malignant tumors was 62.7% and 37.3% respectively. After doing exploratory data analysis, and pre-processing we applied DNN using TensorFlow version of Keras to train on 80%, validate on 10% and test on the remaining 10% of the dataset. Finally, we got an outstanding model with AUC score of 0.9967, and accuracy of 0.9825. This would mean that, if we are given 10,000 samples of digitized images of nuclei of breast mass and their morphologic measurement, we can be able to identify 9825 of them correctly. Regarding the feature importance, **'perimeter_worst', 'texture_worst', 'smoothness_worst'** are the top three features with the highest **contribution** to a malignant diagnosis**.** In sum, we strongly believe that this data can be utilized by the healthcare industry for the early diagnosis of patients with breast mass.