Breast Cancer Classification

Contents

Introduction

According to the American Cancer Society (ACS), breast cancer is the most common cancer in American women, except for skin cancers. Today, a woman's chance of getting breast cancer is 1 in 8 chances. The chance that a woman will die from breast cancer is about 2.6%, or a 1 in 38 chance. Breast cancer still causes about 685.000 deaths annually in the world. Breast cancer research opens the door to finding better ways to prevent, detect, and treat breast cancer.

Early detection, often through screening (mammogram, breast ultrasound, MRI), can catch the disease when it is most treatable. Its detection ability depends on tumor size and breast tissue density.

Breast Cancer Wisconsin (Diagnostic) Dataset is a public dataset. The dataset contains a description and size of specific cells in the breast with diagnosis. The goal is to classify tumors as malignant (cancerous) or benign (noncancerous). https://www.kaggle.com/yasserh/breast-cancer-dataset

Our technology choice is open source: Python and its libraries are good supported and actively updated tools. We do not use ApacheSpark, PySpark because Our dataset is too small. Single computer is enough to process data. List of the libraries:

- pandas and numpy to explore and transform data
- matplotlib and seaborn for visualization
- sklearn, imblearn, keras, tensorflow to build and evaluate models

Data Exploring

During data exploring we investigated:

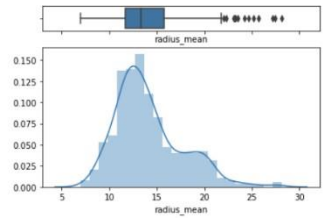
1. Data types of columns match their content. All columns have float data type and the datatype relates to the columns meaning (radius_mean, texture_mean, etc.)
2. All values are non-null in the source file.

3. The dataset does not have duplicates.
4. Column 'id' is meaningless for our study.
5. The value distribution of each feature makes sense.

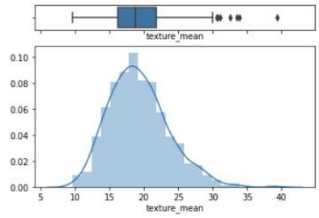
	count	mean	std	min	25%	50%	75%	max
radius_mean	569.0	14.127292	3.524049	6.981000	11.700000	13.370000	15.780000	28.11000
texture_mean	569.0	19.289649	4.301036	9.710000	16.170000	18.840000	21.800000	39.28000
perimeter_mean	569.0	91.969033	24.298981	43.790000	75.170000	86.240000	104.100000	188.50000
area_mean	569.0	654.889104	351.914129	143.500000	420.300000	551.100000	782.700000	2501.00000
smoothness_mean	569.0	0.096360	0.014064	0.052630	0.086370	0.095870	0.105300	0.16340
compactness_mean	569.0	0.104341	0.052813	0.019380	0.064920	0.092630	0.130400	0.34540
concavity_mean	569.0	0.088799	0.079720	0.000000	0.029560	0.061540	0.130700	0.42680
concave points_mean	569.0	0.048919	0.038803	0.000000	0.020310	0.033500	0.074000	0.20120
symmetry_mean	569.0	0.181162	0.027414	0.106000	0.161900	0.179200	0.195700	0.30400
fractal_dimension_mean	569.0	0.062798	0.007060	0.049960	0.057700	0.061540	0.066120	0.09744
radius_se	569.0	0.405172	0.277313	0.111500	0.232400	0.324200	0.478900	2.87300
texture_se	569.0	1.216853	0.551648	0.360200	0.833900	1.108000	1.474000	4.88500
perimeter_se	569.0	2.866059	2.021855	0.757000	1.606000	2.287000	3.357000	21.98000
area_se	569.0	40.337079	45.491006	6.802000	17.850000	24.530000	45.190000	542.20000
smoothness_se	569.0	0.007041	0.003003	0.001713	0.005169	0.006380	0.008146	0.03113
compactness_se	569.0	0.025478	0.017908	0.002252	0.013080	0.020450	0.032450	0.13540
concavity_se	569.0	0.031894	0.030186	0.000000	0.015090	0.025890	0.042050	0.39600
concave points_se	569.0	0.011796	0.006170	0.000000	0.007638	0.010930	0.014710	0.05279
symmetry_se	569.0	0.020542	0.008266	0.007882	0.015160	0.018730	0.023480	0.07895
fractal_dimension_se	569.0	0.003795	0.002646	0.000895	0.002248	0.003187	0.004558	0.02984
radius_worst	569.0	16.269190	4.833242	7.930000	13.010000	14.970000	18.790000	36.04000
texture_worst	569.0	25.677223	6.146258	12.020000	21.080000	25.410000	29.720000	49.54000
perimeter_worst	569.0	107.261213	33.602542	50.410000	84.110000	97.660000	125.400000	251.20000
area_worst	569.0	880.583128	569.356993	185.200000	515.300000	686.500000	1084.000000	4254.00000
smoothness_worst	569.0	0.132369	0.022832	0.071170	0.116600	0.131300	0.146000	0.22260
compactness_worst	569.0	0.254265	0.157336	0.027290	0.147200	0.211900	0.339100	1.05800
concavity_worst	569.0	0.272188	0.208624	0.000000	0.114500	0.226700	0.382900	1.25200
concave points_worst	569.0	0.114606	0.065732	0.000000	0.064930	0.099930	0.161400	0.29100
symmetry_worst	569.0	0.290076	0.061867	0.156500	0.250400	0.282200	0.317900	0.66380
fractal_dimension_worst	569.0	0.083946	0.018061	0.055040	0.071460	0.080040	0.092080	0.20750

6. Outliers exist and some features have skewed distribution.

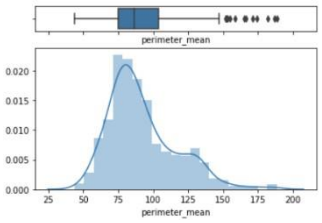
Name: radius_mean, dtype: float64



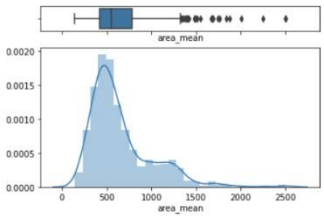
Name: texture_mean, dtype: float64



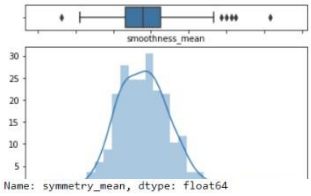
Name: perimeter_mean, dtype: float64



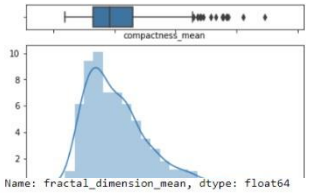
Name: area_mean, dtype: float64



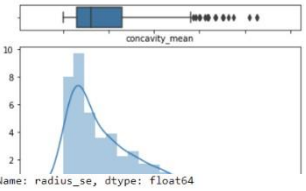
Name: smoothness_mean, dtype: float64



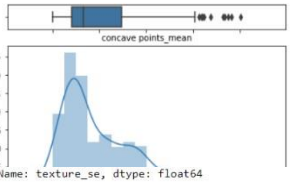
Name: compactness_mean, dtype: float64



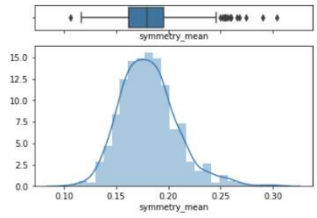
Name: concavity_mean, dtype: float64



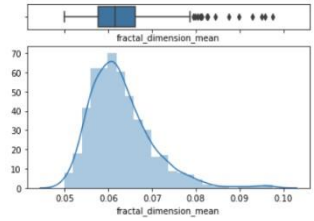
Name: concave points_mean, dtype: float64



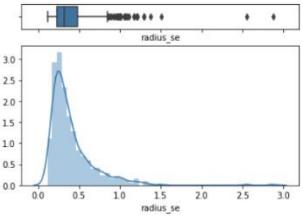
Name: symmetry_mean, dtype: float64



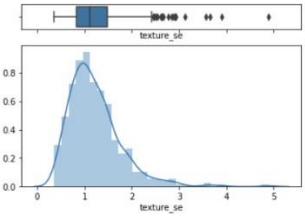
Name: fractal_dimension_mean, dtype: float64

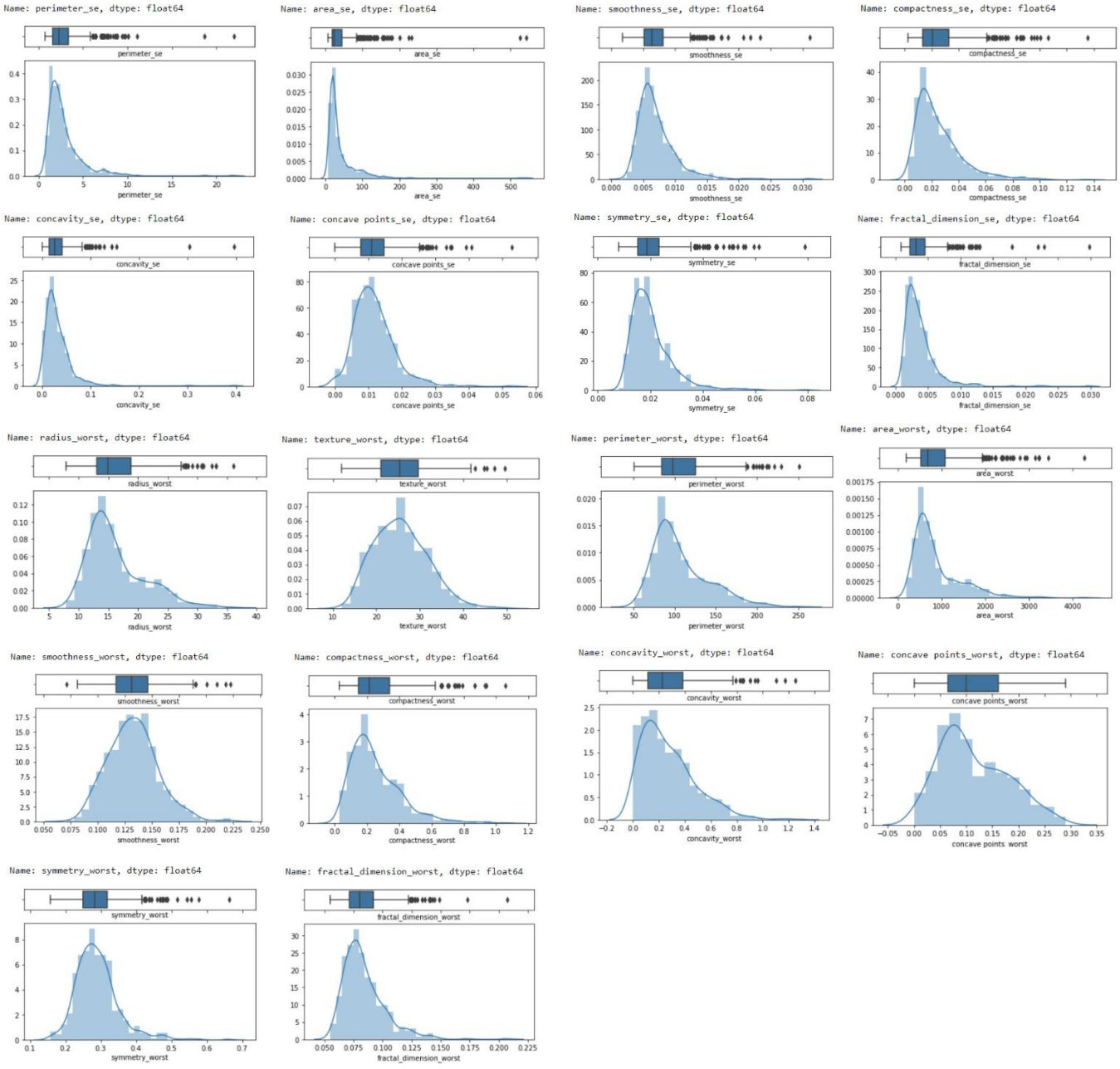


Name: radius_se, dtype: float64

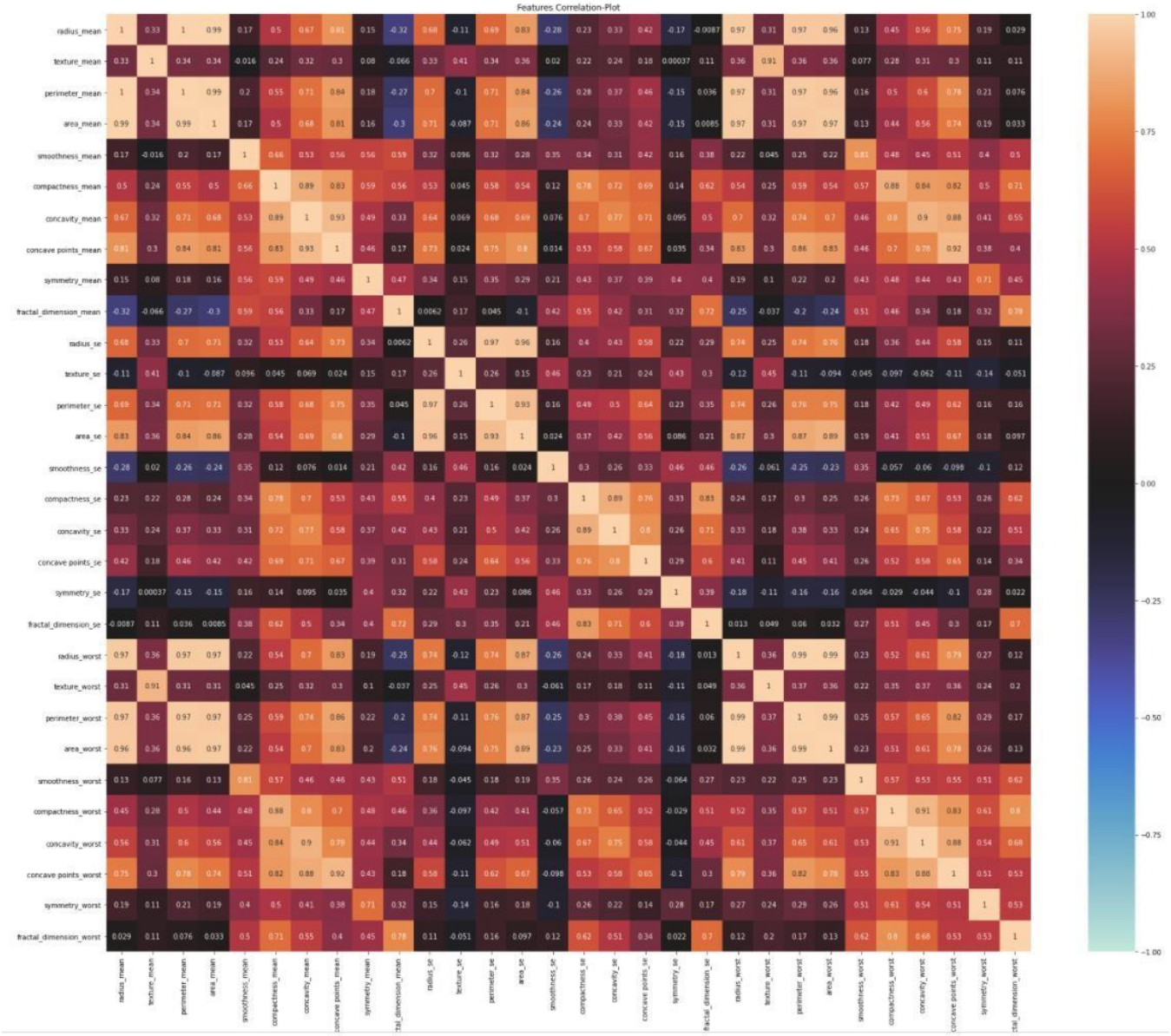


Name: texture_se, dtype: float64

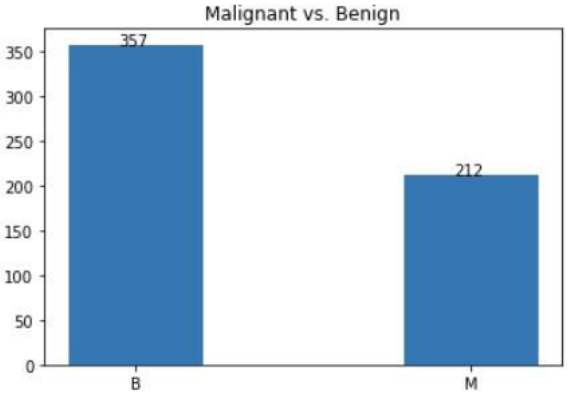




7. Highly correlated features exist. These features contribute very less in predicting the output but can increase the computational cost.



8. The dataset is slightly imbalanced.



Malignant – 37%
Benign – 63%

ETL

Data Clean up

During the cleaning we found and fix incorrect data (remove duplicates and irrelevant data, handle missing data, fix structural errors, filter outliers, etc.). As a result, we get more performed, manageable data and finally we can understand if our data make sense and helps to answer to our question.

1. All values are non-null in the source file, so we do not need to handle missing values.
2. The dataset does not have duplicates.
3. Column 'id' was removed from the dataset because it is meaningless for our study.
4. Outliers were updated using Inter Quantile Range technique.

```
def replace_outliers(col):
    percentile_25=col.quantile(0.25)
    percentile_75=col.quantile(0.75)

    IQR = percentile_75-percentile_25
    upper_limit=percentile_75+1.5*IQR
    lower_limit=percentile_25-1.5*IQR
    print('upper_limit: {0}, lower_limit {1}'.format(upper_limit, lower_limit))
    col = np.where(col < lower_limit, lower_limit, col)
    col = np.where(col > upper_limit, upper_limit, col)

    return col
```

List of features after cleaning up:
'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'

Feature engineering

Feature creation includes feature engineering (splitting or combining features, creating new features, normalization/standardization, etc.) and feature selection (removing redundant or irrelevant features, reduce dimensionality). Feature creation significantly affects model train process because the dataset becomes optimal. We can reduce training time and computation cost, use less memory, avoid overfitting. Also correct features help to interpret model results easily.

1. The features with correlation more than 95% are removed.
   Highly correlated features: 'perimeter_mean', 'area_mean', 'perimeter_se', 'area_se', 'radius_worst', 'perimeter_worst', 'area_worst' Final

list of features:
'radius_mean', 'texture_mean', 'smoothness_mean', 'compactness_mean', concavity_mean', 'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean', 'radius_se', texture_se', 'smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se', ‘symmetry_se', 'fractal_dimension_se', 'texture_worst', ‘smoothness_worst', 'compactness_worst', 'concavity_worst', 'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst'

2. MinMaxScaller was applied for features normalization.
3. For categorical target we used Binary encoding: Malignant – 1; Benign - 0

4. For creating a balanced train dataset was used SMOTE algorithm.
   SMOTE (synthetic minority oversampling technique) is one of the most commonly used oversampling methods to solve the imbalance problem. It aims to balance class distribution by randomly increasing minority class examples by replicating them.
   SMOTE synthesises new minority instances between existing minority instances. It generates the virtual training records by linear interpolation for the minority class. These synthetic training records are generated by randomly selecting one or more of the k-nearest neighbors for each example in the minority class. After the oversampling process, the data is reconstructed and several classification models can be applied for the processed data.

Model

Model Definition

The task is binary classification (Breast Cancer Classification). A target is 1 as 'Malignant' and 0 as 'Benign'.
4 models were chosen:
Logistic Regression - the best and commonly used model for binary classification. The output of Logistic Regression is a binary value (0 or 1).
Decision Tree - Each node is test of an attribute/feature, each branch presents full test. At the end (final leaf) we have class (0 or 1). So, this algorithm can be easily interpreted as set of rules.
Gradient Boosting - ensembled algorithm which uses decision tree models. Each model learns from the output of previous model - corrects the prediction errors made by prior models. This method usually has a good accuracy but can be easily overfitted.
Neural Network - the output layer with sigmoid activation function returns estimated probability which can be easily converted to 0/1 output using threshold. ANN model:

```
model = Sequential()
model.add(Input(shape=(23,)))
model.add(Dense(120,input_shape=(23, ), activation='relu')) # 23 features
model.add(Dropout(0.4))
model.add(Dense(30, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
print(model.summary())

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Model Training

For training we split the dataset on train and test datasets with proportion: 70% of the data – the train set and 30% -test set.
We trained all models based on the initial dataset, a scaled dataset, and a balanced dataset.

Initial dataset:

```
Source: (569, 24)
X_train: (398, 23), y_train: (398,)
X_test: (171, 23), y_test: (171,)
```

Balanced:

```
Before OverSampling, counts of label '1': 149
Before OverSampling, counts of label '0': 249
After OverSampling, the shape of train_X: (498, 23)
After OverSampling, the shape of train_y: (498,)

After OverSampling, counts of label '1': 249
After OverSampling, counts of label '0': 249
```

Model Evaluation

For evaluation of Binary Classification was used Accuracy, Precision and F1 score.
Accuracy does not always correctly demonstrate if the model is good or not for classification task (especially for unbalanced dataset) but it's always great to have good Accuracy.
Recall and Precision could work better but we should decide on what question we want to ask. What is better for us to have some False 'Malignant ' or False 'Benign'. Obviously to have False 'Malignant' is better because it's better to check than to miss. From other side we do not want to miss real Malignant cases, so Recall metric is also important for us. We should decrease False Benign.
F1 is harmonic mean of Precision and Recall and can help us to choose what classifier is better in common.

	Benign	Malignant
Benign	True Benign	False Malignant
Malignant	False Benign	True Malignant

Predicted (vertical) / *Actual* (horizontal)

$$Malignant\ Precision = \frac{True\ Malignant}{Predicted\ Malignant}\ OR\ \frac{True\ Malignant}{True\ Malignant + False\ Malignant}$$

$$Malignant\ Recall = \frac{True\ Malignant}{Actual\ Malignant}\ OR\ \frac{True\ Malignant}{True\ Malignant + False\ Benign}$$

Logistic Regression

	Accuracy	Precision(Malignant)	Recall (Malignant)	F1(avg)
Initial dataset	97%	98%	94%	97%
Scaled dataset	98%	98%	95%	97%
Balanced dataset	96%	97%	94%	96%

Decision Tree

	Accuracy	Precision(Malignant)	Recall (Malignant)	F1(avg)
Initial dataset	91%	84%	92%	90%

Scaled dataset	94%	88%	95%	93%
Balanced dataset	91%	82%	95%	91%

Gradient Boosting

	Accuracy	Precision(Malignant)	Recall (Malignant)	F1(avg)
Initial dataset	95%	94%	92%	95%
Scaled dataset	94%	92%	92%	94%
Balanced dataset	95%	92%	94%	94%

ANN

	Accuracy	Precision(Malignant)	Recall (Malignant)	F1(avg)
Initial dataset	95%	94%	92%	94%
Scaled dataset	98%	97%	98%	98%
Balanced dataset	95%	91%	95%	94%

All model work correctly, but ANN model with scaled data is the best. We applied more tuning techniques for this model:

- Different optimizers

Optimizer	Activation	Activation(output)	Accuracy	Precision(M)	Recall (M)	F1(avg)
adam	relu	sigmoid	98%	97%	98%	98%
adagrad	relu	sigmoid	92%	95%	84%	92%
sgd	relu	sigmoid	96%	97%	92%	96%

- Decrease number of units of hidden layers

```
cur_model = Sequential()
cur_model.add(Dense(16,input_shape=(23, ), activation='relu')) # 23 features, units = input*2/3+1
cur_model.add(Dense(16, activation='relu'))
cur_model.add(Dense(1, activation='sigmoid'))

cur_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Optimizer	Activation	Activation(output)	Accuracy	Precision(M)	Recall (M)	F1(avg)
adam	relu	sigmoid	98%	97%	98%	98%

Conclusion

The source dataset did not need a lot of transformation.

All models work good for solving Breast Cancer Classification. With small margin a winner is ANN model with adam optimizer, relu/sigmoid activation functions and loss function - binary-crossentropy.

It's interesting that a smaller number of units gave us the same results (16 units on hidden layers vs 30-120 units). So, we can simplify the ANN model without losing accuracy.

The best result:

Accuracy	98%	
Precision (Malignant)	97%	Only 2 of 108 Benign cases were predicted as Malignant. But as we remember it's better to check than to miss!
Recall (Malignant)	98%	Only 1 of 63 Malignant cases were predicted as Benign. Not bad at all!
F1 (avg)	97%	

Also, we see that Logistic regression gave us results pretty similar to ANN model. So, we have a choice based on what we prefer more.

Source code and reports are available https://github.com/omatveyuk/ibm_advanced_ds