International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

Breast Cancer Wisconsin (Diagnostic) Prediction

Prithwish Ghosh

Masters 1st year Student, Department of Statistics, Visva-Bharati University, Santiniketan, India
ghosh.prithwish1999[at]gmail.com

Abstract: ***Breast cancer continues to remain the most lethal malignancy in women across the world. In 2008, approximately 1.4 million women were diagnosed with breast cancer worldwide with corresponding 460000 deaths. Here we use a machine learning tool xgboost to diagnose patients and then we split the data to classify them into categories. Here we use xgboost because its time efficient and gives more precise results rather than many other Machine Learning algorithms.***

Keywords: Xgboost, Machine Learning, malignancy

1. Introduction

Breast cancer is a type of cancer that starts in the breast. Cancer starts when cells

Begin to grow out of control. (To learn more about how cancers start and spread, see What Is Cancer?1)

Breast cancer cells usually form a tumor that can often be seen on an x-ray or felt as a lump. Breast cancer occurs almost entirely in women, but men can get breast cancer2,

Too.

It's important to understand that most breast lumps are benign and not cancer (Malignant). Non-cancerous breast tumors are abnormal growths, but they do not Spread outside of the breast. They are not life threatening, but some types of benign.

Breast lumps can increase a woman's risk of getting breast cancer. Any breast lump or Change needs to be checked by a health care professional to determine if it is benign or Malignant (cancer) and if it might affect your future cancer risk. See Non-cancerous

Breast Conditions to learn more.Breast cancers can start from different parts of the breast.
1) Most breast cancers begin in the ducts that carry milk to the nipple (ductal cancers)
2) Some start in the glands that make breast milk (lobular cancers)
3) There are also other types of breast cancer that are less common like phyllodes tumor and angiosarcoma
4) A small number of cancers start in other tissues in the breast. These cancersare called sarcomas4 and lymphomas5 and are not really thought of as breast cancers.

Data set information:
Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei. Separating plane described above was obtained using multifurcate method –tree (MSM-T), a classification method which uses linear programming to construct a decision tree. Relevant feature was selected using an exhaustive search in the space of 1-4 features and 1-3 features.

Problem Statement
Our aim is to diagnose patients with breast cancer by analyzing the data of patients and classifying them into categories, having diagnosis results as:
1) Benign (B)
2) Malignant(M)

Libraries used:
1) **NumPy:** The fundamental package for scientific computing with python
2) **Pandas:** An open source, BSD- licensed library providing high performance, easy-to-use data structures and data analysis tools for the python programming language.
3) **Sklearn.model_selection:**
   - Split arrays or matrices into random train and test subsets
   - Selecting optimal Features
4) **Matplotlib.pyplot:** Provides a **MATLAB** like plotting framework
5) **Seaborn:**Seaborn is a python data visualization library based on matplotlib. It provides a high-levelinterface for drawing attractive and informative statistical graphics.
6) **XGBoost:**XGBoost is an optimized distributed gradient boosting library designed to be highly efficient , flexible and portable
7) **Sklearn.metrics**:
   - **Accuracy score:** In multilevel classification, this function computes subset accuracy, the set of labels predicted for a sample must exactly match the corresponding set of labels in true.
   - **Classification report**: Build a report showing the main classification metrics
   - **Confusion matrix**: Compute confusion matrix to evaluate the accuracy of classification.

Parameters:
Id number, Diagnosis (M=malignant, B = benign), Radius, Texture, Perimeter, Area, Smoothness, Compactness, Concavity, Concave Points, Symmetry, Fractal Dimension. 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, and field 23 is worst

Radius. All feature values are recorded with four significant digits.

Checking for missing values:

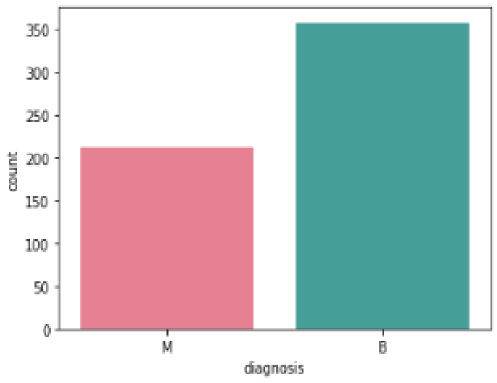
There are no missing values in the data set

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

Class distributions:

357 benign, 212 malignant

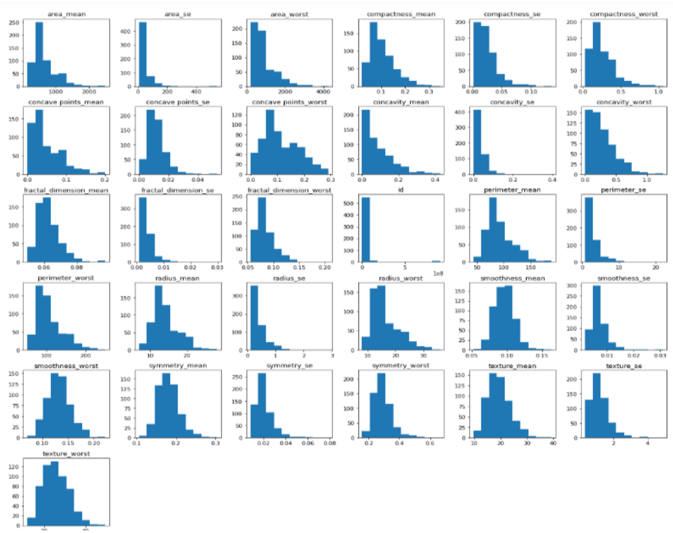
Understanding the frequency distribution of data in target variable "diagnosis"



Frequency of unique values of the target variable

```
: B    357
  M    212
  Name: diagnosis, dtype: int64
```

Bar plots:

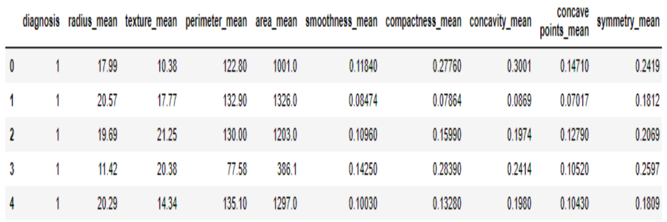


This will give us the visualizing information about the data with respect to the target value Diagnosis, which is defined as two parts which are Malignant(M) and Benign (B). We can also visualize the data using the box plot and

Preparing the data and mapping the string to numerical data:

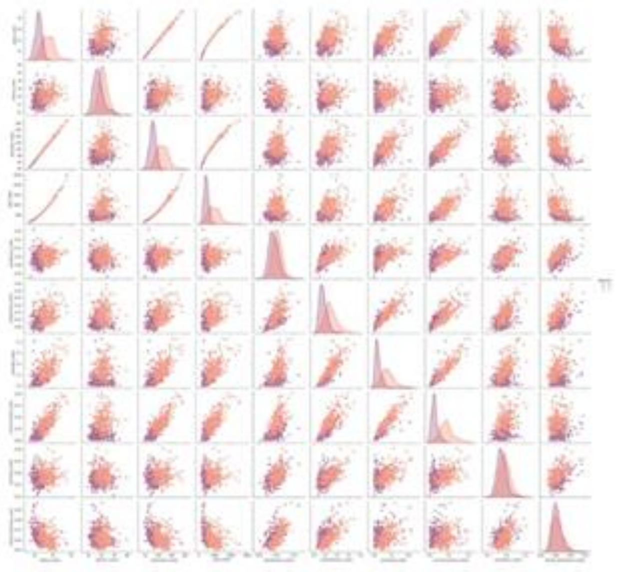
Here we can see that we don't need the id parmeter so we are removing it and after thet we can see that the diagnosis attribute is giving string('object') value which is not understandable by the programming algorythm so we are changing it, for our betterment

'B' -> 0 AND 'M' -> 1



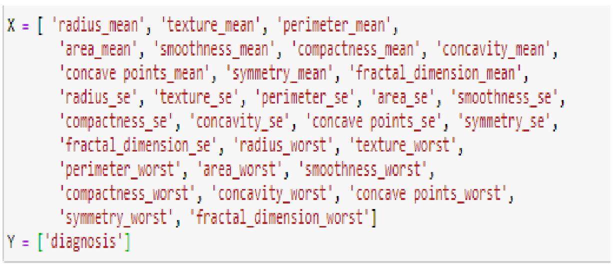
	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

Dimension reduction:



Almost it's perfectly linear patterns between the radius and area attributes which are hinting at the presence of multicollinearity between these variables. (They are highly linearly related) Another set of variables that possibly imply multicollinearity are the concavity. Concavepoints and compactness. Multicollinearity is a problem as it undermines the significance of independent variables and we fix it by removing the highly correlated predictors from the model. Use of partial least square or principal component analysis, regression methods that cut the number of predictors to a smaller number of uncorrelated components.

The features which are highly correlated (show dependency) with the target variable (Diagnosis) are highly relevant attributes for our classification problem. We can remove irrelevant attributes in order to reduce the size of data for easier computation. Here, t stores the target or class variable and x stores the non-class attributes.

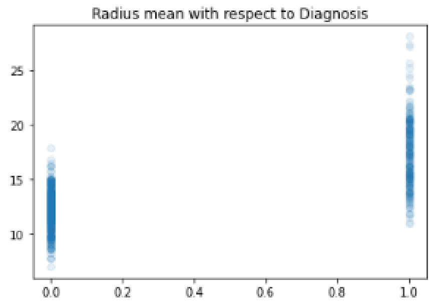


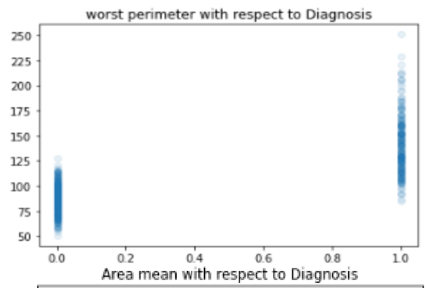
```
X = [ 'radius_mean', 'texture_mean', 'perimeter_mean',
      'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
      'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
      'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
      'compactness_se', '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 = ['diagnosis']
```

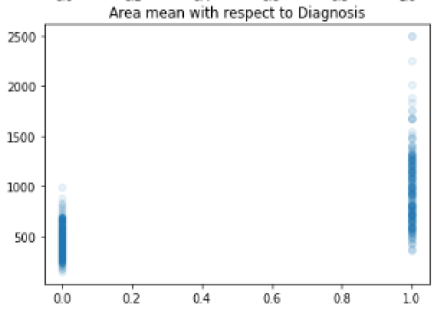
The irrelevant attributes can be found by computing the correlation among the non-class attributes, and then we can reduce a subset of highly co-relevant non-class attributes to a single or a smaller number of attributes which would reduce the size of the data. The heat plot visualized the correlation between each pair of attributes in the dataset. We can select those attributes which have high correlation with each other according to a threshold value, and then we will implement the classification algorithm with a reduced set of attributes by taking various smaller subsets of these highly correlated (dependent) attributes, and compare the results. Here we are taking threshold value: 0.90. From the heat plot we infer that the following features are the most related (corr>.9) to the other features:

1) Radius_mean
2) Perimeter_mean
3) Area_mean
4) Radius_worst
5) Perimeter_worst

Then we plot scatter plots for all the high correlation features with respect to the target feature (diagnosis):

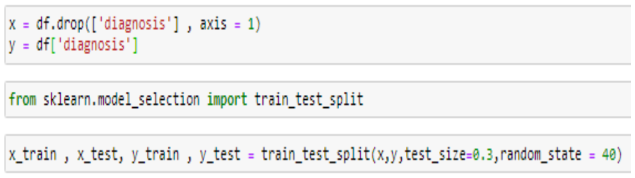






Splitting of data into two sets:

We now split the data into a test set and training set. The test set size is chosen to be 30% of the whole data.

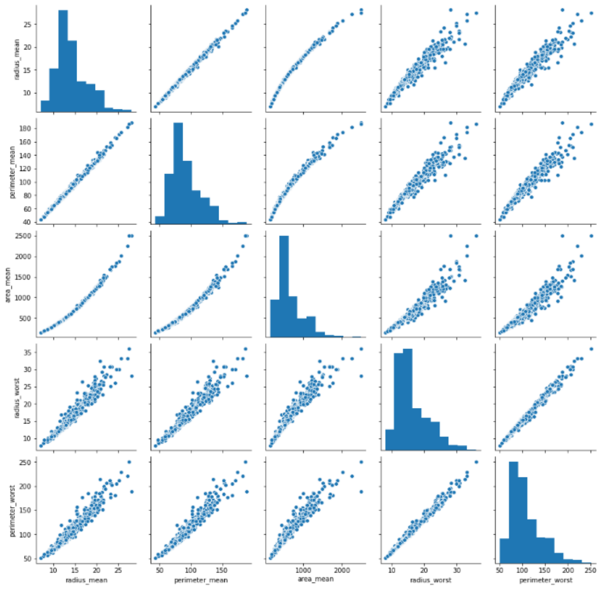


```
x = df.drop(['diagnosis'] , axis = 1)
y = df['diagnosis']
```

```
from sklearn.model_selection import train_test_split
```

```
x_train , x_test, y_train , y_test = train_test_split(x,y,test_size=0.3,random_state = 40)
```

We then make a pair plot between all the high correlation features:



Theory behind this algorithm (XGBoost):

XG Boost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data.XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

What is XG Boost?

XG Boost stands for e**X**treme **G**radient **B**oosting.*The name xgboost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree* *algorithms. which is the reason why many people use xgboost.* Tianqi Chen, in answer to the question "What is the difference between the R gbm (gradient boosting machine) and xgboost (extreme gradient boosting)?" on Quora. It is an implementation of gradient boosting machines created by Tianqi Chen, now with contributions from many developers. It belongs to a broader collection of tools under the umbrella of the Distributed Machine Learning Community or DMLC who are also the creators of the popular mxnet deep learning library. Tianqi Chen provides a brief and interesting back story on the creation of XGBoost in the post Story and Lessons Behind the Evolution of XGBoost. XGBoost is a software library that you can download and install on your machine, then access from a variety ofinterfaces. Specifically, XGBoost supports the following main interfaces:

- Command Line Interface (CLI).
- C++ (the language in which the library is written).
- Python interface as well as a model in scikit-learn.
- R interface as well as a model in the caret package.
- Julia.
- Java and JVM languages like Scala and platforms like Hadoop.

XGBoost Features

The library is laser focused on computational speed and model performance, as such there are few frills. Nevertheless, it does offer a number of advanced features.

Model Features

The implementation of the model supports the features of the scikit-learn and R implementations, with new additions like regularization. Three main forms of gradient boosting are supported:

- **Gradient Boosting** algorithm also called gradient boosting machine including the learning rate.
- **Stochastic Gradient Boosting** with sub-sampling at the row, column and column per split levels.
- **Regularized Gradient Boosting** with both L1 and L2 regularization.
- **Parallelization** of tree construction using all of your CPU cores during training.
- **Distributed Computing** for training very large models using a cluster of machines.
- **Out-of-Core Computing** for very large datasets that don't fit into memory.
- **Cache Optimization** of data structures and algorithm to make best use of hardware.

Algorithm Features

The implementation of the algorithm was engineered for efficiency of compute time and memory resources. A design goal was to make the best use of available resources to train the model. Some key algorithm implementation features include:

- **Sparse Aware** implementation with automatic handling of missing data values.
- **Block Structure** to support the parallelization of tree construction.
- **Continued Training** so that you can further boost an already fitted model on new data.

XGBoost is free open-source software available for use under the permissive Apache-2 license.

What Algorithm Does XGBoost Use?

The XGBoost library implements the algorithm. This algorithm goes by lots of different names such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines. Boosting is an ensemble technique where new models are added to correct the errors made existing models. Models are added sequentially until no further improvements can be made. A existing models. Models are added sequentially until no further improvements can be made. Anexisting model. Models are added sequentially until no further improvements can be made. A by popular example is the AdaBoost algorithm that weights data points that are hard to predict. Regression trees, stochastic gradient boosting or gradient boosting machines.Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. A popular example is the AdaBoost algorithm that weights data points that are hard to predict.Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.This approach supports both regression and classification predictive modeling problems.For more on boosting and gradient boosting, see Trevor Hastie's talk on Gradient Boosting Machine Learning.

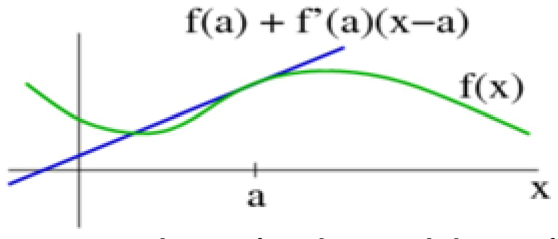
XGBoost objective function

The objective function (loss function and regularization) at iteration ***t*** that we need to minimize is the following:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(\overset{\text{Real value (label) known from the training data set}}{y_i}, \underbrace{\hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)}_{\text{Can be seen as } f(x+\Delta x) \text{ where } x = \hat{y}_i^{(t-1)}}) + \Omega(f_t)$$

It is easy to see that the XGBoost objective is a function of functions (i.e., ***l*** is a function of CART learners), and as the authors refer in the paper [2] "cannot be optimized using traditional optimization methods in Euclidean space". Taylor's Theorem and Gradient Boosted Trees. From the reference [1] we can see as an example that the best linear approximation for a function ***f(x)*** at point ***a*** is:

Why we need to use the Taylor approximation?



Because we need to transform the original objective function to a function in the Euclidean domain, in order to be able to use traditional optimization techniques.

Explanation of the above answer:
Let’s take the simplest linear approximation of a function ***f(x)*** from [1]:

The “trick” here is that we can transform a function ***f(x)*** to a simplest function of **Δx** around a specific point ***a*** by using Taylor’s theorem.

$$f(x) \approx f(a) + f'(a)\boxed{(x-a)}$$

$$\Delta x = f_t(\mathbf{x}_i)$$

Note that the objective function must be differentiable.

In our case ***f(x)*** is the loss function ***l***, while ***a*** is the previous step ***(t-1)*** predicted value and Δx is the new learner we need to add in step ***t***.Using the above in each iteration ***t*** we can write the objective (loss) function as a simple function of the new added learner and thus to apply Euclidean space optimization techniques.As we already said, ***a*** is the prediction at step ***(t-1)*** while ***(x-a)*** is the new learner that we need to add in step ***(t)*** in order to greedily minimize the objective.So, if we decide to take the second-order Taylor approximation, we have:

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2$$

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n}[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

Where:

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \text{ and } h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

Finally, if we remove the constant parts, we have the following simplified objective to minimize at step ***t***:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n}[g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

The above is a sum of simple quadratic functions of one variable and can be minimized by using known techniques, so our next goal is to find a learner that minimizes the loss function at iteration ***t***.

$$\operatorname{argmin}_x Gx + \frac{1}{2}Hx^2 = -\frac{G}{H},\ H > 0 \qquad \min_x Gx + \frac{1}{2}Hx^2 = -\frac{1}{2}\frac{G^2}{H}$$

How to build the next learner
Being at iteration ***t*** we need to build a learner that achieves the maximum possible reduction of loss, the following questions arrive:

Is it possible to find the optimal next learner?Is there any way to calculate the gain (loss reduction) after adding a specific learner?The good news is that there is a way to “measure the quality of a tree structure ***q***” as the authors refer, and the scoring function is the following (see the correspondence to the “simple quadratic function solution” above):

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{\overset{\text{instances mapped to leaf}_j}{\boxed{(\sum_{i\in I_j} g_i)^2}}}{\sum_{i\in I_j} h_i + \lambda} + \gamma T.$$

While the bad news is that it is impossible in terms of required calculations to “enumerate all the possible tree structures ***q***” and thus find the one with maximum loss reduction.

Note that the “quality scoring function” above returns the minimum loss value for a given tree structure, meaning that the original loss function is evaluated by using the optimal weight values. So, for any given tree structure we have a way to calculate the optimal weights in leaves.

In practice what we do in order to build the learner is to:

- Start with single root (contains all the training examples)
- Iterate over all features and values per feature, and evaluate each possible split loss reduction: ***gain*** = ***loss*** (father instances) - (***loss*** (left branch) +***loss*** (right branch))
- The gain for the best split must be positive (and > ***min_split_gain*** parameter), otherwise we must stop growing the branch.

The above algorithm is called the “*Exact Greedy Algorithm*” and its complexity is ***O(n*m)*** where ***n*** is the number of training samples and ***m*** is the features dimension.

Binary classification with log loss optimization
Let’s take the case of binary classification and log loss objective function:

$$y\ln(p) + (1-y)\ln(1-p) \text{ where } p = \frac{1}{(1+e^{-x})}$$

where ***y*** is the real label in ***{0,1}*** and ***p*** is the probability score.

Note that ***p*** (score or pseudo-probability) is calculated after applying the famous sigmoid function into the *output of the **GBT model x***.

The output *x* of the model is the sum across the the CART tree learners.

So, in order to minimize the log loss objective function we need to find its 1st and 2nd derivatives (gradient and hessian) with respect to ***x***.

In this stats.stackexchange post you can find that gradient = ***(p-y)*** and hessian = ***p*(1-p)***.

Summarizing the GBT model which — remember — is a sum of CART (tree) learners will try to minimize the log loss objective and the scores at leaves which are actually the weights have a meaning as a sum across all the trees of the model and are always adjusted in order to minimize the loss. This is why we need to apply the sigmoid function in the output of GBT models in case of binary classification probability scoring.

Showcase implementations
Below you can find an awesome pure Python implementation of the Gradient Boosted Trees algorithm that is using the “*Exact Greedy Algorithm*”:
https://github.com/lancifollia/tinygbt : L2 Regression
https://github.com/dimleve/tinygbt : Binary Classification.

SUMMARY
In this post you discovered the XGBoost algorithm forapplied machine learning.We learned:

- That XGBoost is a library for developing fast and high-performance gradient boosting tree models.

- That XGBoost is achieving the best performance on a range of difficult machine learning tasks.
- That you can use this library from the command line, Python and R and how to get started.

Training the model and prediction of the target variable:

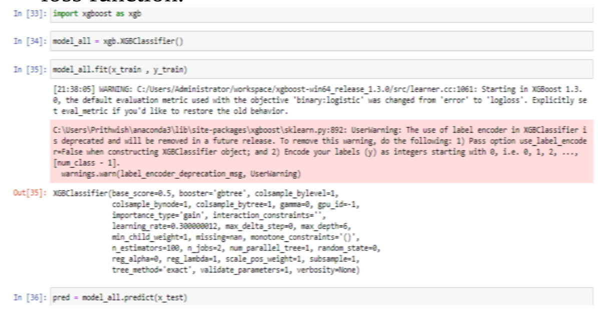
We are using a gradient boosting framework provided by XGBoost to train the model. XGBoost is short for eXtreme gradient boosting. It is library designed and optimized for boosted tree algorithms.

Its main goal is to push the extreme of the computation limits of machines to provide a scalable, portable and accurate for large scale tree boosting.

Features

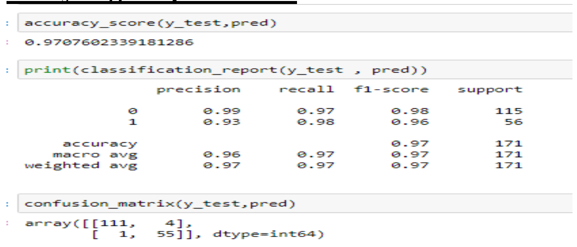
- Speed: it can automatically do parallel computation on Windows and Linux, with OpenMp. It is generally over 10 times faster than the classical gbm.
- Input Type: It takes several types of input data.
- Dense Matrix: R; s dense matrix, i.e., Matrix
- Sparse Matrix:R; s sparse matrix, i.e.,Matrix: dgCMatrix
- Data file: Local data files
- Xgb.DMatrix: its own class (recommended)
- Sparsity: It accepts sparse input for both tree booster and linear booster, and is optimized for sparse input.
- Customization: it supports customized objective function and evaluation functions.

Gradient boosting involves three elements:
1) A loss function to be optimized.
2) A weak learner to make predictions.
3) An additive model to add weak learners to minimize the loss function.



```
In [33]: import xgboost as xgb

In [34]: model_all = xgb.XGBClassifier()

In [35]: model_all.fit(x_train , y_train)

         [21:38:05] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.
         0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
         t eval_metric if you'd like to restore the old behavior.

         C:\Users\Prithwish\anaconda3\lib\site-packages\xgboost\sklearn.py:892: UserWarning: The use of label encoder in XGBClassifier i
         s deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encode
         r=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
         [num_class - 1].
           warnings.warn(label_encoder_deprecation_msg, UserWarning)

Out[35]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                       colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
                       importance_type='gain', interaction_constraints='',
                       learning_rate=0.300000012, max_delta_step=0, max_depth=6,
                       min_child_weight=1, missing=nan, monotone_constraints='()',
                       n_estimators=100, n_jobs=2, num_parallel_tree=1, random_state=0,
                       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
                       tree_method='exact', validate_parameters=1, verbosity=None)

In [36]: pred = model_all.predict(x_test)
```

Analyzing the prediction:

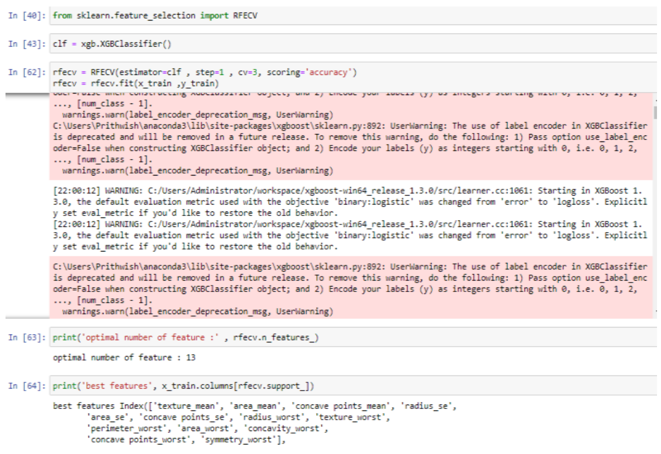


```
: accuracy_score(y_test,pred)

: 0.9707602339181286

: print(classification_report(y_test , pred))

               precision    recall  f1-score   support

           0       0.99      0.97      0.98       115
           1       0.93      0.98      0.96        56

    accuracy                           0.97       171
   macro avg       0.96      0.97      0.97       171
weighted avg       0.97      0.97      0.97       171

: confusion_matrix(y_test,pred)

: array([[111,   4],
         [  1,  55]], dtype=int64)
```

We get an accuracy of 97.07%

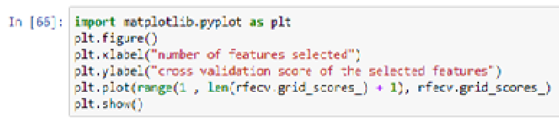
Finding the optimal features:

Now we find the optimal number of features and what those features are. We do this by RFECV.

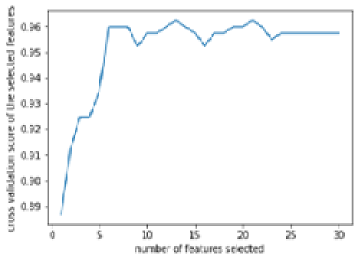


```
In [40]: from sklearn.feature_selection import RFECV

In [43]: clf = xgb.XGBClassifier()

In [62]: rfecv = RFECV(estimator=clf , step=1 , cv=3, scoring='accuracy')
         rfecv = rfecv.fit(x_train ,y_train)

         ..., [num_class - 1].
           warnings.warn(label_encoder_deprecation_msg, UserWarning)
         C:\Users\Prithwish\anaconda3\lib\site-packages\xgboost\sklearn.py:892: UserWarning: The use of label encoder in XGBClassifier
         is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_enc
         oder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2,
         ..., [num_class - 1].
           warnings.warn(label_encoder_deprecation_msg, UserWarning)

         [22:00:12] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.
         3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitl
         y set eval_metric if you'd like to restore the old behavior.
         [22:00:12] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.
         3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitl
         y set eval_metric if you'd like to restore the old behavior.

         C:\Users\Prithwish\anaconda3\lib\site-packages\xgboost\sklearn.py:892: UserWarning: The use of label encoder in XGBClassifier
         is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_enc
         oder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2,
         ..., [num_class - 1].
           warnings.warn(label_encoder_deprecation_msg, UserWarning)

In [63]: print('optimal number of feature :' , rfecv.n_features_)

         optimal number of feature : 13

In [64]: print('best features', x_train.columns[rfecv.support_])

         best features Index(['texture_mean', 'area_mean', 'concave points_mean', 'radius_se',
                'area_se', 'concave points_se', 'radius_worst', 'texture_worst',
                'perimeter_worst', 'area_worst', 'concavity_worst',
                'concave points_worst', 'symmetry_worst'],
```

13 optimal features are selected. We now plot the number of features VS. Cross-validation scores to see how it effects the prediction.

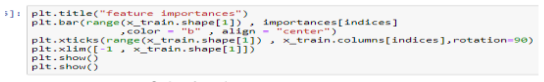


```
In [65]: import matplotlib.pyplot as plt
         plt.figure()
         plt.xlabel("number of features selected")
         plt.ylabel("cross validation score of the selected features")
         plt.plot(range(1 , len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
         plt.show()
```

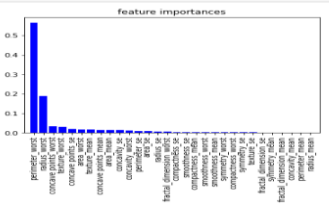


The graph shows how the scores vary with varying number of features. The score is highest when the number of features is 13. We can now rank the features by their importance.

Now we plot those important features:

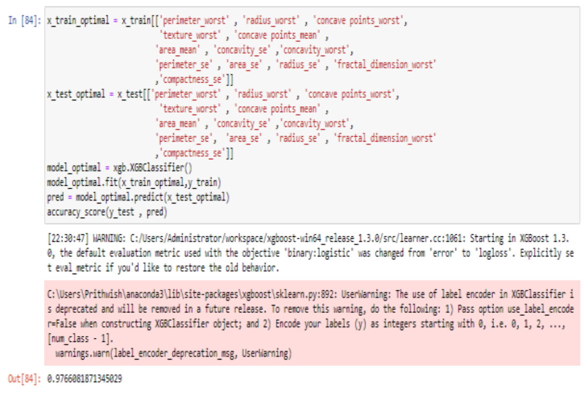


```
]: plt.title("feature importances")
   plt.bar(range(x_train.shape[1]) , importances[indices]
               ,color = "b" , align = "center")
   plt.xticks(range(x_train.shape[1]) , x_train.columns[indices],rotation=90)
   plt.xlim([-1 , x_train.shape[1]])
   plt.show()
   plt.show()
```



Training and analyzing the model with only the optimal features:

We train model and predict the target variable using only the selected optimal features (13) and get an accuracy of 97.66%



```
In [84]: x_train_optimal = x_train[['perimeter_worst' , 'radius_worst' , 'concave points_worst',
                          'texture_worst' , 'concave points_mean' ,
                         'area_mean' , 'concavity_se' ,'concavity_worst',
                         'perimeter_se' , 'area_se' , 'radius_se' , 'fractal_dimension_worst'
                         ,'compactness_se']]
         x_test_optimal = x_test[['perimeter_worst' , 'radius_worst' , 'concave points_worst',
                          'texture_worst' , 'concave points_mean' ,
                         'area_mean' , 'concavity_se' ,'concavity_worst',
                         'perimeter_se',  'area_se' , 'radius_se' , 'fractal_dimension_worst'
                         ,'compactness_se']]
         model_optimal = xgb.XGBClassifier()
         model_optimal.fit(x_train_optimal,y_train)
         pred = model_optimal.predict(x_test_optimal)
         accuracy_score(y_test , pred)

         [22:30:47] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.
         0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
         t eval_metric if you'd like to restore the old behavior.

         C:\Users\Prithwish\anaconda3\lib\site-packages\xgboost\sklearn.py:892: UserWarning: The use of label encoder in XGBClassifier i
         s deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encode
         r=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
         [num_class - 1].
           warnings.warn(label_encoder_deprecation_msg, UserWarning)

Out[84]: 0.9766081871345029
```

Why we choose this algorithm:

1) A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function that may want to be used, instead, it is a generic enough framework that any differentiable loss function can be used.
2) Regression trees are used that output real values for splits and whose output can be added together, allowing subsequent model outputs to be added and 'correct' the residual in the predictions. Trees are constructed in a greedy manner, choosing the best split points based on purity score like to minimize the loss.
3) Instead of parameters, we have weak learner sub-models or more specifically decision trees.
4) After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss (i.e., follow the gradient). We do this by parametrizing the tree, then modify the parameters of the tree and move in the right direction by (reducing the residual loss)
5) XGBoost manages only numerical vectors. What to do when you have categorical data? A simple method to convert categorical variable into numerical vector is One Hot Encoding.
6) XGBoost can scale with hundreds of workers (with each worker utilizing multiple processors) smoothly and solve machine learning problems involving Terabytes of real-world data.

Other four enhancements to basic gradient boosting:

1) Tree constraints
2) Shrinkage
3) Random Sampling
4) Penalized Learning

2. Conclusion

- We started by exploring the data with EDA (exploratory data analysis) methods. We analyzed the data types of the feature and converted the string data type to integer (using hot encoding).
- We then checked for missing values and duplicates and dealt with them.
- After performing preliminary analysis, we plotted the data to get more insights.
- Further we dropped certain features that upon analysis, we found to be irrelevant to the target variable.
- We found the correlation of all the features with the target variable as well as among each other. We then found the 13 most optimal features.

As we can observe, we have reduced the number of features from30 to 13, and have also increased the accuracy of our model from 97% to 97.7% during the process.

Sources

1) Introduction to Taylor's theorem: https://mathinsight.org/taylors_theorem_multivariable_introduction
2) *Tianqi Chen, Carlos Guestrin*: XGBoost: A Scalable Tree Boosting System https://arxiv.org/abs/1603.02754
3) *Tianqi Chen*: Introduction to Boosted Trees: https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf
4) How to calculate gradient and hessian of log loss objective function: https://stats.stackexchange.com/questions/231220/how-to-compute-the-gradient-and-hessian-of-logarithmic-loss-question-is-based
5) https://github.com/dmlc/xgboost
6) https://arxiv.org/pdf/1603.02754.pdf

References

[1] The Cancer Atlas- http://canceratlas.cancer.org/the-burden/breast-cancer/

[2] American Institute of Cancer Research Statistics - https://www.wcrf.org/dietandcancer/cancer-trends/breast-cancer-statistics

[3] World Health Organization Cancer Report: https://www.who.int/cancer/prevention/diagnosis-screening/breast-cancer/en/

[4] Kumar, V., Tiwari. P, Mishra. B.K., Kumar. S.: Implementation of n-gram methodology for rotten tomatoes review dataset sentiment analysis. International Journal of Knowledge Discovery in Bioinformatics (IJKDB), 7(1), pp. 30–41. DOI: https://doi.org/10.4018/ijkdb.2017010103 (2017).

[5] Kumar V., Verma A., Mittal N., Gromov S.V.: Anatomy of Preprocessing of Big Data for Monolingual Corpora Paraphrase Extraction: Source Language Sentence Selection. In: Emerging Technologies in Data Mining and Information Security. Advances in Intel-ligent Systems and Computing, Vol 814, pp. 495-505, Springer Nature, Singapore. DOI: https://doi.org/10.1007/978-981-13-1501-5_43 (2019).

[6] Kumar, V., Kalitin D., Tiwari., P.: Unsupervised Learning Dimensionality Reduction Algorithm PCA for Face Recognition, IEEE Xplore: International Conference on. Com-puting, Communication and Automation (ICCCA), pp. 32-37. DOI: 10.1109/CCAA.2017.8229826 (2017).

[7] Kumar., V., Zinovyev., R., Verma., Tiwari., P.: Performance Evaluation of Lazy and Decision Tree Classifier: A Data Mining Approach for Global Celebrity 's Death Analy-sis. IEEE Xplore: In International Conference on Research in Intelligent and Computing in Engineering (RICE), pp 1-6, DOI: 10.1109/RICE.2018.8509082 (2018).

[8] Kumar V., Mazzara M., Messina., A., Lee., J.Y.:A Conjoint Application of Data Mining Techniques for Analysis of Global Terrorist Attacks, Prevention and Prediction for Com-bating Terrorism. Proceedings of 6th International Conference in Software Engineering for Defense Applications- SEDA 2018. Advances in Intelligent Systems and Computing.

Author Profile

Prithwish Ghosh, Department of Statistics, Visva Bharati University, Post Graduate 1st year Student