CSCI 4723 Machine Learning
Course Project

Predicting the Accuracy of Cancer with Adaptive Boosting (AdaBoostClassifier)

Sachin Karki
Texas Woman's University
Denton, Texas

Author Note
Sachin Karki, Texas Woman's University, Mathematics & Computer Science

Abstract. Machine learning has become a reliable tool for problem-solving. Machine learning as a unique subfield of computer science entails data and raw information to generate valuable insights from a different set of learnings. There are various methods and technics of machine learning algorithms to build a model and every model has a benefit of its own. The paper will explore an instance of such method. Adaptive Boosting is one of the methods that is quite characterized for retaining minimal bias and better results. This paper is an exploration of *AdaBoostClassifier* on the breast cancer Wisconsin dataset to check the accuracy rate of cancer. With AdaBoostClassifier it was found that it is possible to retain over 95% of the accuracy from the model.

1. Background/Introduction

The sklearn dataset about breast cancer is an ideal dataset for any beginners or new learners of the machine learning arena. The breast cancer dataset is easy to comprehend and does not require a lot of understanding or even solve new instances. From the dataset, we can ask, analyze, and interpret a wide range of information and insights. New knowledge and evidence can be explored by training and testing the breast cancer dataset. Out of the many instances, some popular ones are the appearance of a lump in the breast area or armpits, thickening of cancer-fused organs, redness in parts of the body, etc. To predict various scenarios and instances the dataset was utilized specifically for this project. In addition to this, adaptive boosting is known quite for retaining maximum accuracy. This paper will analyze the dataset with the described method. Lastly, the main motivation to choose this dataset was to learn and unlock new ways of machine learning models.

1.1 Intro to Adaptive Boosting

Adaptive Boosting or *AdaBoost* for short is a machine learning meta-algorithm that can be used in conjunction with many other types of learning algorithms to improve performance. The output generated from the other learning algorithms “weak learners” is combined into a weighted sum that represents the final output of the boosted classifier. *AdaBoost* subsequently tweaks weak learners in favor of those instances misclassified by previous classifiers hence the adaptive process makes sense. In addition to this, *AdaBoost* is sensitive to noisy data and outliers meaning chances of biases are minimal and accuracy is somewhat better than other methods.

1.2 Brief Description of Dataset

For this project, the Breast cancer Wisconsin dataset was utilized from the scikit learn datasets. i.e. *load_breast_cancer* from *sklearn.datasets.* The breast cancer dataset is a collection of 569 samples of cancer incidences. The dataset is based on binary classification and has 30 dimensionality features on attributes like radius, perimeter, smoothness, and so on related to cancer. The diagnostic cancer dataset has been used in ML to identify its nature and diagnose cancer scans as either it’s benign or malignant. This paper will inspect and analyze the breast cancer dataset from the method of Adaptive Boosting to see the accuracy rate of cancer.

2. Problem Statement

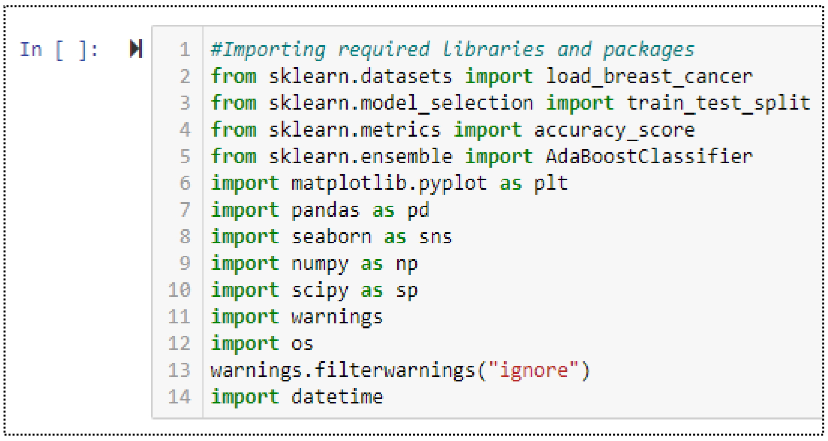
The main goal of this paper also aligns with the very problem it is asking. From the breast cancer dataset, we are tasked to find the accuracy rate and state of cancer, whether it's benign or malignant. Benign means that it has no chance of spreading to other body parts and malignant means that it has the chance of spreading to other parts of the body. With the adaptive boosting technique, we will find out the accuracy level and interpret what does it mean for our analysis.

3. Methodology

We will take a deep dive into the overall procedure of the analysis, how data were organized, prepared, and viewed, and how testing and training models were implemented. The complete breakdown of the whole process is described below.

3.1 Importing Required Libraries

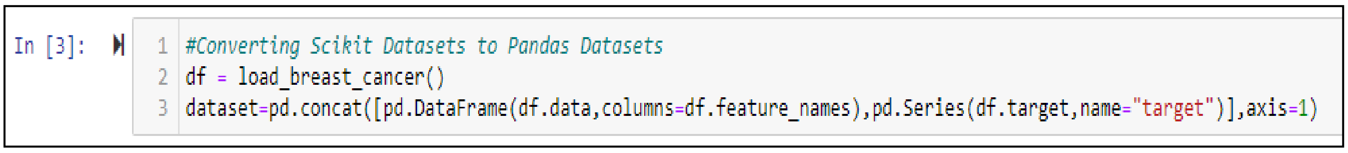
To begin with, several required libraries were imported to make our model operate. For example, NumPy, pandas, train_test_split, matplot, etc. These libraries are essential to validate and make our model functional. A screenshot is presented below.



```
In [ ]:
#Importing required libraries and packages
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import AdaBoostClassifier
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
import scipy as sp
import warnings
import os
warnings.filterwarnings("ignore")
import datetime
```

3.2 Conversion of Dataset (sklearn to pandas)

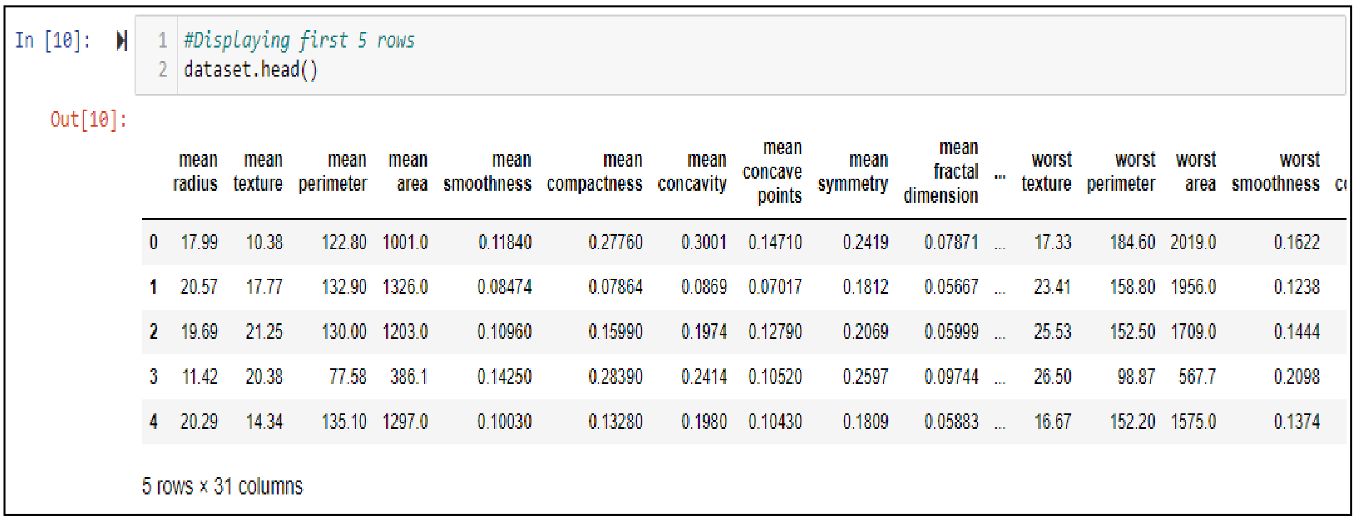
Another step that was involved in the process was a conversion of the *sklearn* dataset into pandas dataset. Knowing that it was a little tricky to get the scikit datasets working in a way like I was used to, conversion was done for a minor but crucial purpose. Converting the dataset helped in relaying the correct data and variables to be passed into a definite data frame. A screenshot is pictured below.



```
In [3]:
#Converting Scikit Datasets to Pandas Datasets
df = load_breast_cancer()
dataset=pd.concat([pd.DataFrame(df.data,columns=df.feature_names),pd.Series(df.target,name="target")],axis=1)
```

3.3 Description and Display of Dataset

To check if our dataset was correctly imported and converted by the shown method above, I checked the dataset and display the first 5 heads from the dataset. The result is presented in the screenshot below.



```
In [10]:
#Displaying first 5 rows
dataset.head()
```

Out[10]:

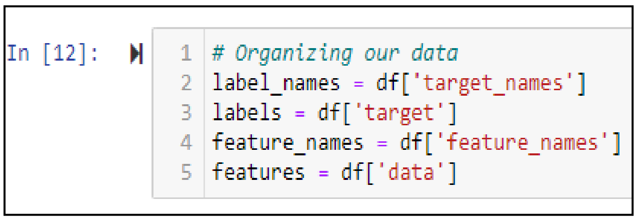
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374

5 rows × 31 columns

As it can be seen, the code returned 5 rows and 31 columns from the dataset.

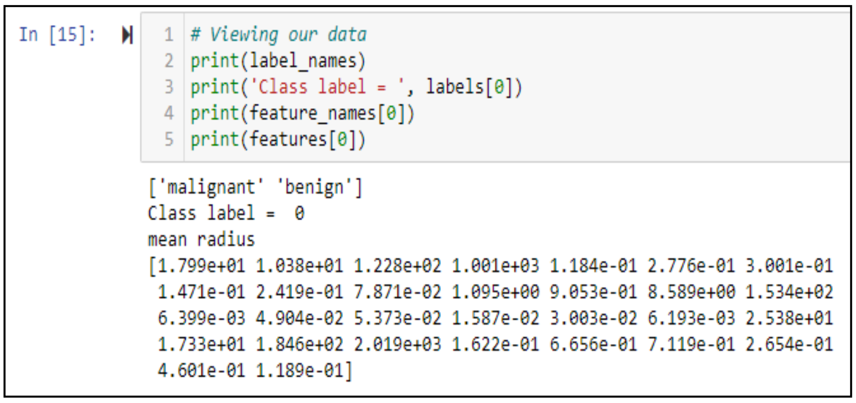
3.4 Organizing the Data

A crucial step in our entire process was to organize our data to work perfectly with the classifier and the model we are going to create. To do so we need to create a new variable for each important set of information that appears useful to us. Then we need to assign the attributes in the dataset respective of those variables. This was done with the following line of code.



```
In [12]:
# Organizing our data
label_names = df['target_names']
labels = df['target']
feature_names = df['feature_names']
features = df['data']
```

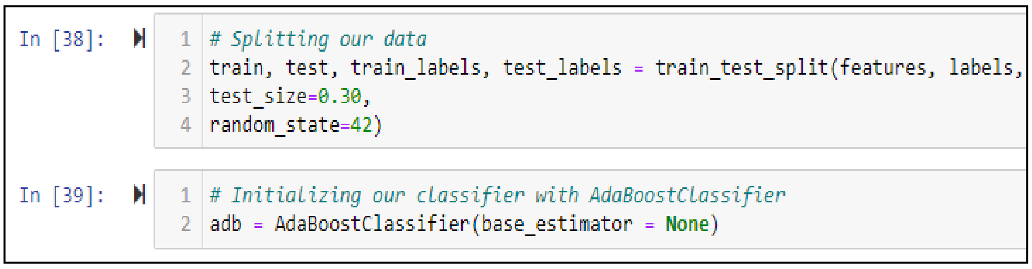
After organizing our data, we can view our data by printing class labels, feature names, feature values, etc. to get a better understanding of our dataset. The following line of code highlights the code and its result.



```
In [15]:
# Viewing our data
print(label_names)
print('Class label = ', labels[0])
print(feature_names[0])
print(features[0])

['malignant' 'benign']
Class label =  0
mean radius
[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
 1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
 6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
 1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
 4.601e-01 1.189e-01]
```

3.5 Splitting the Model and Initializing the Classifier

To check the performance of a classifier, one should always test the model on invisible or unseen data. Building a model is all about splitting data into two sets: a training set and a test set. A training set is used for training and evaluating a model during its developmental stage and then that trained model is what we use to make predictions on the unseen or invisible test set. In the depicted picture below, I divided the training set into 70% and the test set into 30%. The resulting output can be seen below.

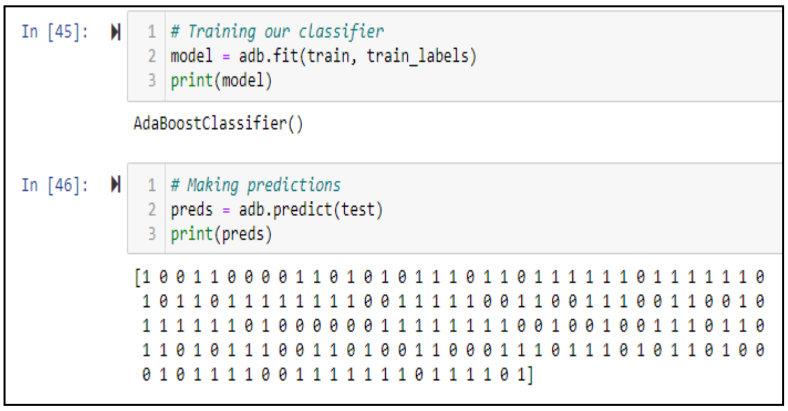


```
In [38]:
# Splitting our data
train, test, train_labels, test_labels = train_test_split(features, labels,
test_size=0.30,
random_state=42)

In [39]:
# Initializing our classifier with AdaBoostClassifier
adb = AdaBoostClassifier(base_estimator = None)
```

Additionally, we can see that *AdaBoostClassifier* was initialized on the variable *adb*. This variable is further utilized with functions like training the classifier and making predictions.

3.6 Training the Classifier and Making Predictions

After initializing our classifier, we have to train the model. Using the *adb* variable that we defined for *AdaBoostClassifier*, we fit the model into the training set. The process can be seen in the picture below.

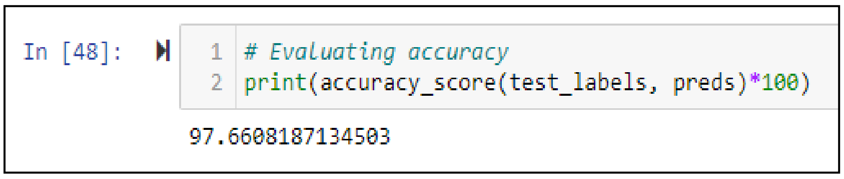


```
In [45]:
# Training our classifier
model = adb.fit(train, train_labels)
print(model)

AdaBoostClassifier()

In [46]:
# Making predictions
preds = adb.predict(test)
print(preds)

[1 0 0 1 1 0 0 0 0 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0
 1 1 1 1 1 1 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0
 1 1 0 1 0 1 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 0 0
 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1]
```

In the above picture, we can also see that we used the same *adb* variable to make predictions on the *preds* variable and printed its output on the bottom. The returned arrays of *0s* and *1s* represent our predicted values for the state of cancer (i.e. malignant or benign)

3.7 Evaluating Accuracy

Lastly, the accuracy check was performed on the model. To do so, two arrays i.e. *test_labesl and preds* of our model were compared with the *accuracy_score* function of *sklearn*. Then the resulting output was multiplied by 100 to compute the proportion of the accuracy. The code is pictured below.



```
In [48]:  # Evaluating accuracy
          print(accuracy_score(test_labels, preds)*100)

          97.6608187134503
```

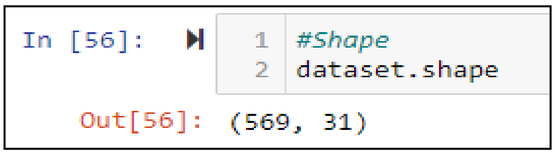
The depicted result above shows that our model with *AdaBoostClassifier* is able to retain 97.66% of accuracy. This means that our breast cancer prediction model is 97.66% accurate. In other words, about 97.66% of the time our classifier is able to make a correct prediction and determine if the tumor is malignant or benign.

4. Experimental Results

To accompany our entire learning process, there were a couple of other factors that helped us generate some useful insights from the model. They contributed in a major way to tell some untold narrative of our learning process. The two key experimental results for our machine learning model came from visualization and a table of correlation plots. Although no new groundbreaking patterns or evidence were found, it was still helpful to see what was on the surface level with our dataset and how attributes of data were interrelated with each other. Here are the elemental breakdowns of the experimental results generated throughout the process.

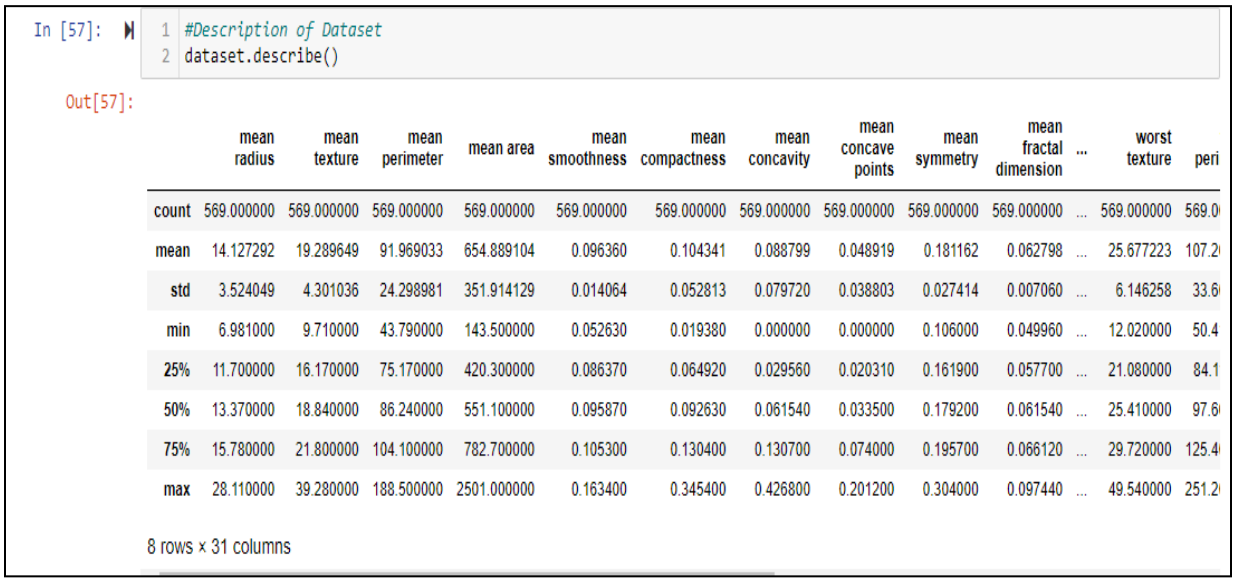
4.1 Shape and Description of Dataset

One of the minor insights that were generated early on in the process was from the shape and description function for the dataset. The shape helped in identifying the sample size and the description helped in viewing the contents of the dataset. Although these are not new or major insights it was the quickest way to summarize and see our dataset as a whole. The *dataset.shape* code allowed us to know that there were 569 samples sizes and 31 attributes (column) in the dataset. The code and result are depicted below.



```
In [56]:  #Shape
          dataset.shape

Out[56]:  (569, 31)
```

Similarly, *dataset.describe* helped to see the contents of the dataset. The resulting code and output are pictured below.



In [57]:
```
#Description of Dataset
dataset.describe()
```

Out[57]:

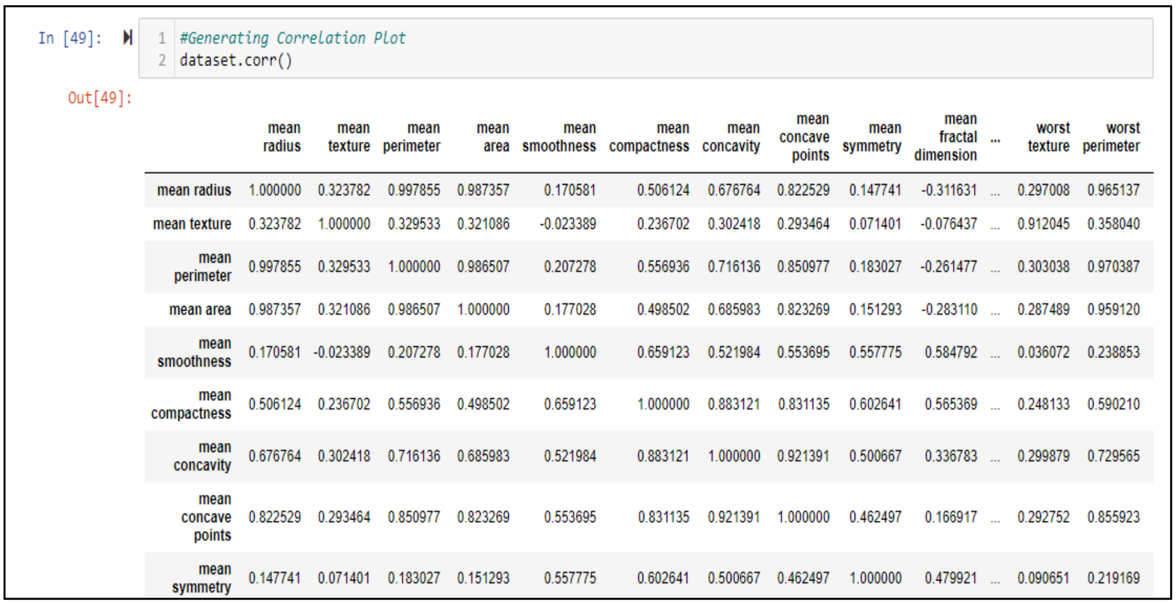
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	peri
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.0
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	25.677223	107.2
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	6.146258	33.6
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	12.020000	50.4
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	21.080000	84.1
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	25.410000	97.6
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	29.720000	125.4
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	49.540000	251.2

8 rows × 31 columns

As we can see, we are presented with 8 rows and 31 columns representing the instance and attributes of the dataset. This helps us quickly see the full content of the dataset.

4.2 Correlation Plot

The correlation plot table helped us see the variables or say attributes that posed a correlation with others. Suffice to say that not many patterns or evidence were found since the relevant correlation turned out to be as expected. The table is pictured below.



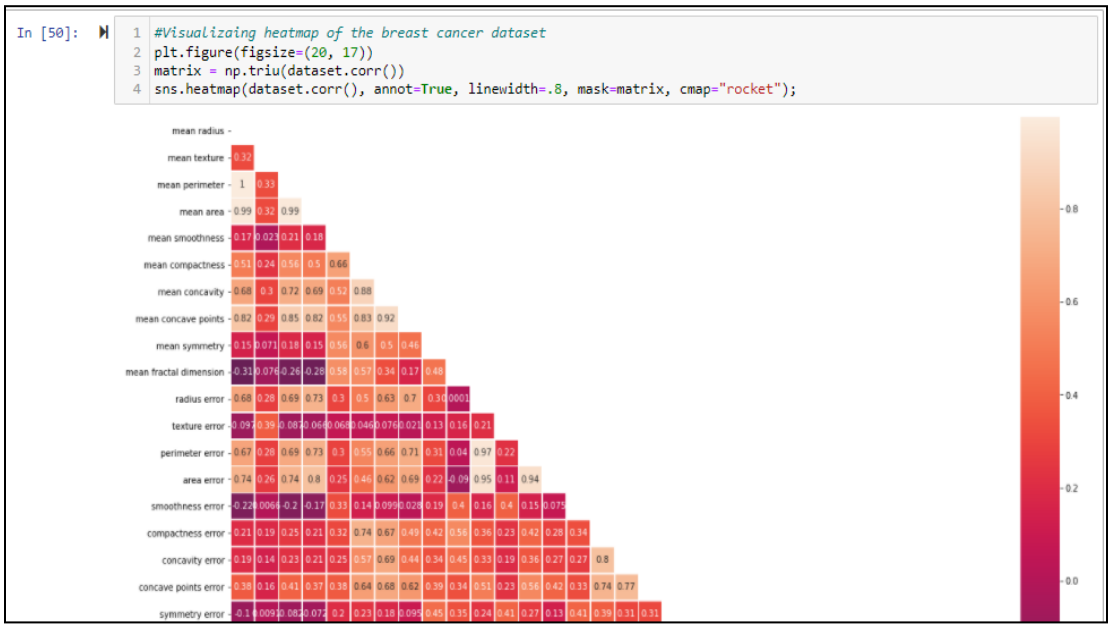
In [49]:
```
#Generating Correlation Plot
dataset.corr()
```

Out[49]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter
mean radius	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.822529	0.147741	-0.311631	...	0.297008	0.965137
mean texture	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.293464	0.071401	-0.076437	...	0.912045	0.358040
mean perimeter	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.850977	0.183027	-0.261477	...	0.303038	0.970387
mean area	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.823269	0.151293	-0.283110	...	0.287489	0.959120
mean smoothness	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.553695	0.557775	0.584792	...	0.036072	0.238853
mean compactness	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.831135	0.602641	0.565369	...	0.248133	0.590210
mean concavity	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.921391	0.500667	0.336783	...	0.299879	0.729565
mean concave points	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.000000	0.462497	0.166917	...	0.292752	0.855923
mean symmetry	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.462497	1.000000	0.479921	...	0.090651	0.219169

4.3 Heatmap of the Attributes

In addition to the correlation plot, a heatmap style matrix was generated to visualize the correlation plot. This gave us a visual summary of what was happening with the correlated data within the dataset. The result of the heatmap style matrix is pictured below.



In [50]:
```python
#Visualizaing heatmap of the breast cancer dataset
plt.figure(figsize=(20, 17))
matrix = np.triu(dataset.corr())
sns.heatmap(dataset.corr(), annot=True, linewidth=.8, mask=matrix, cmap="rocket");
```

5. Conclusions

From my comprehensive learning process and sophisticated model making experience, I was able to learn a lot. First of all, not all techniques or methods of machine learning algorithms retains the same level of result, accuracy, and attention to detail. Some of the most sophisticated methods generate the most interesting results. Out of all *AdaBoostClassifier* is an excellent algorithm or model that is well able to generate precise and accurate results. Adaptive boosting is ideal for optimization and improving performance and weaker learners can take advantage of it. It is an ideal machine learning algorithm for a dataset like breast cancer since it retains over 95% of accuracy.

From the model and classifier that was built in this paper, it was concluded that with adaptive boosting (*AdaBoostClassifier)* the percentage of accuracy came out to be 97.66. It is ideally a better level of accuracy compared to other methods like logistic regression or SVC. Again, to sum up, about 97.66% of the time our classifier is able to correctly predict if a cancer tumor is malignant or benign.

5. References

1. Géron, Aurélien. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* 1st Edition. O'Reilly Media.

2. Breast Cancer Dataset. Retrieved from:
https://archive-beta.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+diagnostic

3. Sklearn Datasets: sklearn.datasets.load_breast_cancer. Retrieved From:
https://docs.w3cub.com/scikit_learn/modules/generated/sklearn.datasets.load_breast_cancer

4. Discussion on Breast Cancer Dataset: Retrieved from:
https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/discussion