Machine Learning Engineer Nanodegree

Capstone Project

Suresh Paruchuri
February 2nd, 2019

I. Definition

Project Overview

Cancer occurs as a result of mutations, or abnormal changes, in the genes responsible for regulating the growth of cells and keeping them healthy. The genes are in each cell's nucleus, which acts as the "control room" of each cell. Normally, the cells in our bodies replace themselves through an orderly process of cell growth: healthy new cells take over as old ones die out. But over time, mutations can "turn on" certain genes and "turn off" others in a cell. That changed cell gains
the ability to keep dividing without control or order, producing more cells just like it and forming a tumor.

The term "breast cancer" refers to a malignant tumor that has developed from cells in the breast. Usually breast cancer either begins in the cells of the lobules, which are the milk-producing glands, or the ducts, the passages that drain milk from the lobules to the nipple. Less commonly, breast cancer can begin in the stromal tissues, which include the fatty and fibrous connective tissues of
the breast.

Identifying correctly whether a tumor is benign or malignant is vital in deciding what is the best treatment, saving and improving the quality of life. In this project, we used Breast Cancer Wisconsin (Diagnostic) Data Set to create a
model able to predict if a tumor is or not dangerous based in characteristics that were computed from a digitized image of a fine needle aspirate (FNA) of a
breast mass.

Source: BreastCancer.org, Inside Radiology

Problem Statement

A tumor can be benign (not dangerous to health) or malignant (has the potential to be dangerous). Benign tumors are not considered cancerous: their cells are close to normal in appearance, they grow slowly, and they do not invade nearby tissues or spread to other parts of the body. Malignant tumors are cancerous. Left unchecked, malignant cells eventually can spread beyond the original tumor to other parts of the body. As the physical aspects of the malignant tumor differ from the benign tumor cells, we can measure the physical characteristics such as radius (mean of distances from center to points on the perimeter), texture (standard deviation of gray-scale values), perimeter, area, smoothness, compactness, concavity, concave points, symmetry or fractal dimension to understand and create two classes of tumor and identify which class each tumor belongs for new samples.

Main aim is to diagnose patients with breast cancer by analysing the data of patients and classifying them into two categories,having diagnosis results as :

(1) Benign(B) and (2) Malignant(M)

The techniques that I used to solve this problem:

Decision Tree:

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Random Forest or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Extra DecisionTrees This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

Gradient Boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

AdaBoost Classifier is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers.
In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.

XGBoost Classifier initially started as a research project by Tianqi Chen as part of the Distributed (Deep) Machine Learning Community (DMLC) group. Initially, it began as a terminal application which could be configured using a libsvm configuration file. After winning the Higgs Machine Learning Challenge, it became well known in the ML competition circles. Soon after, the Python and R packages were built and now it has packages for many other languages like Julia, Scala, Java, etc. This brought the library to more developers and became popular among the Kaggle community where it has been used for a large number of competitions

Support Vector are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

SGD Classifier this estimator implements regularized linear models with stochastic gradient descent learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning, see the partial_fit method. For best results using the default learning rate schedule, the data should have zero mean and unit variance.
Logistic Regression:

This was developed by statistician David Cox in 1958. The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). It allows one to say that the presence of a risk factor increases the odds of a given outcome by a specific factor. The model itself simply models probability of output in terms of input, and does not perform statistical classification, though it can be used to make a classifier, for instance by choosing a cutoff value and classifying inputs with probability greater than the cutoff as one class, below the cutoff as the other.

Metrics

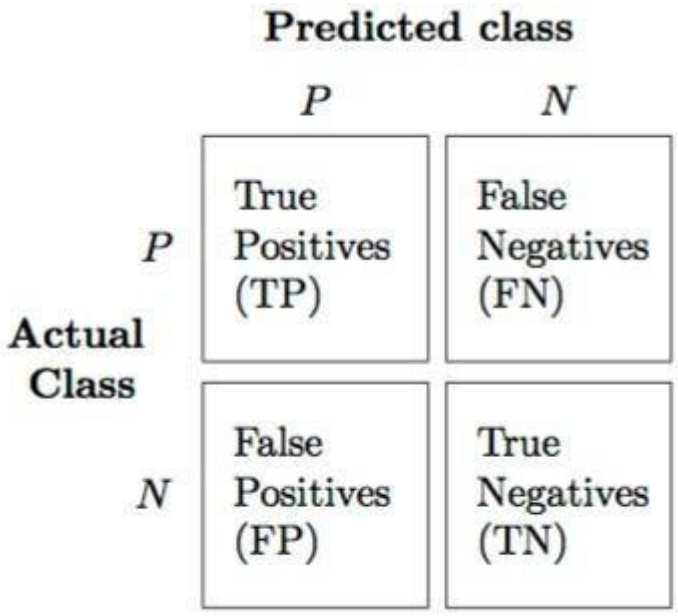
In tumor cells classification is important to avoid false negatives because if a malignant tumor is predict as benign the patient will not receive treatment. That's why F1 is a ideal metric to score our model.

recall = True positive / (True positive + False negative)

precision = True positive / (True positive + False positive)

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

The final model I will analyze the model performance ploting a confusion matrix and score model using F1 Score that is a methot to measure performance of binary classification, the F1 score is a measure of a test's accuracy, is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.



Using F1 Score formula.

In statistical analysis of binary classification, the F1 score is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results returned by the classifier, and r is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at
0.

II. Analysis

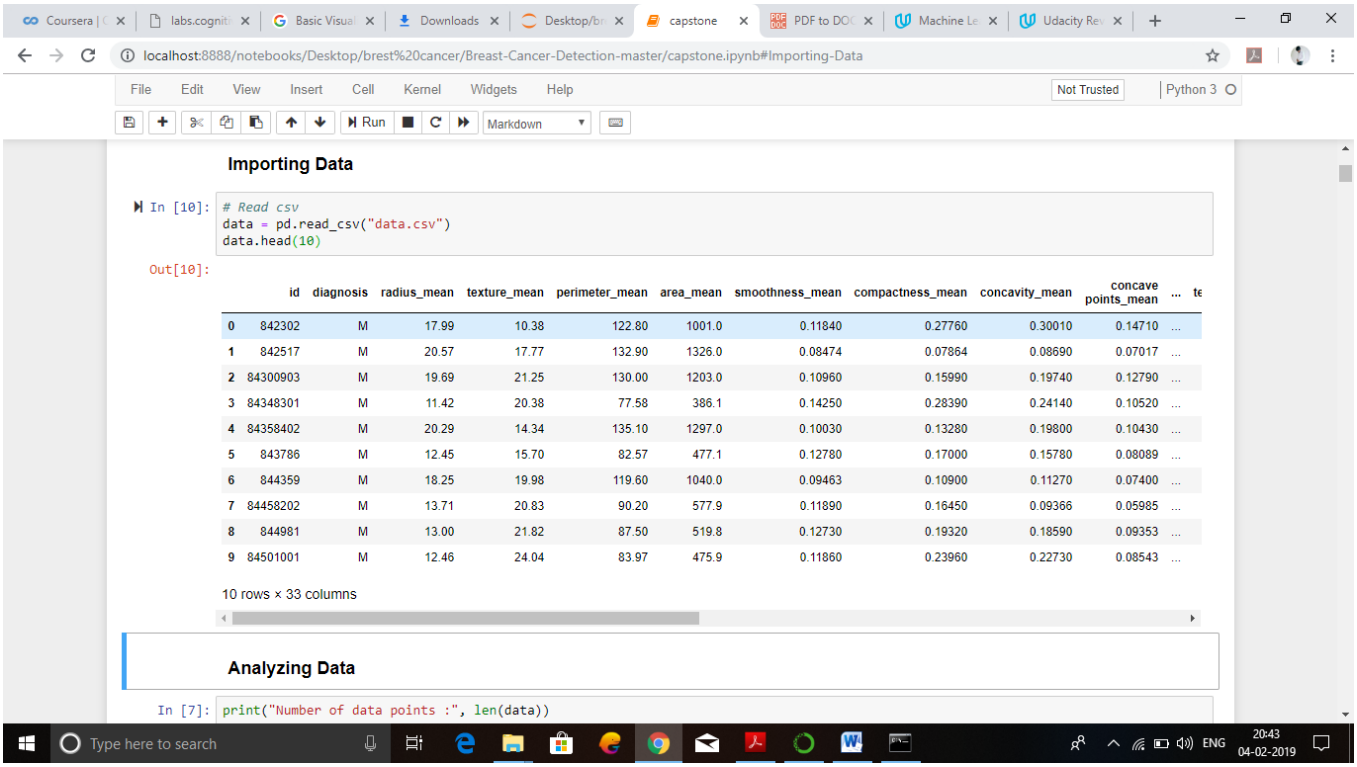
Data Exploration

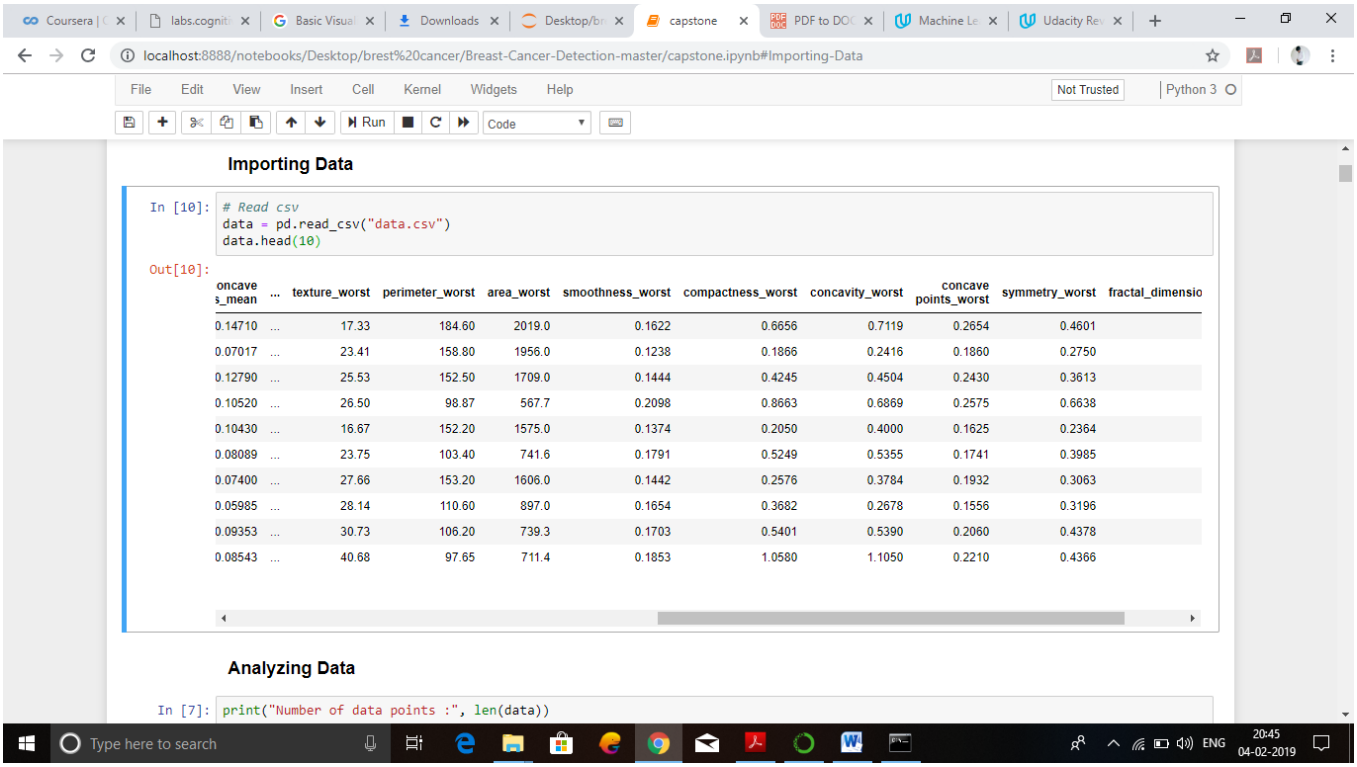
To create a model capable to predict whether a tumor cell is or not malignant I used a labeled dataset:

The Wisconsin Diagnostic Breast Cancer was created in 1995 by:

Dr. William H. Wolberg (General Surgery Dept., University of Wisconsin Clinical Sciences Center),
W. Nick Street (Computer Sciences Dept., University of Wisconsin)
and Olvi L. Mangasarian (Computer Sciences Dept., University of Wisconsin).

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.
Sample data:





The dataset have the following structure:

- Number of instances: 569
- Number of attributes: 32 (ID, diagnosis, 30 real-valued input features)
- Diagnosis(M = malignant, B = benign)
- Missing attribute values: none
- Some Outliers are present. Later they are removed.
- Class distribution: 357 benign, 212 malignant
- All feature values are recoded with four significant digits.
- Ten real-valued features are computed for each cell nucleus:

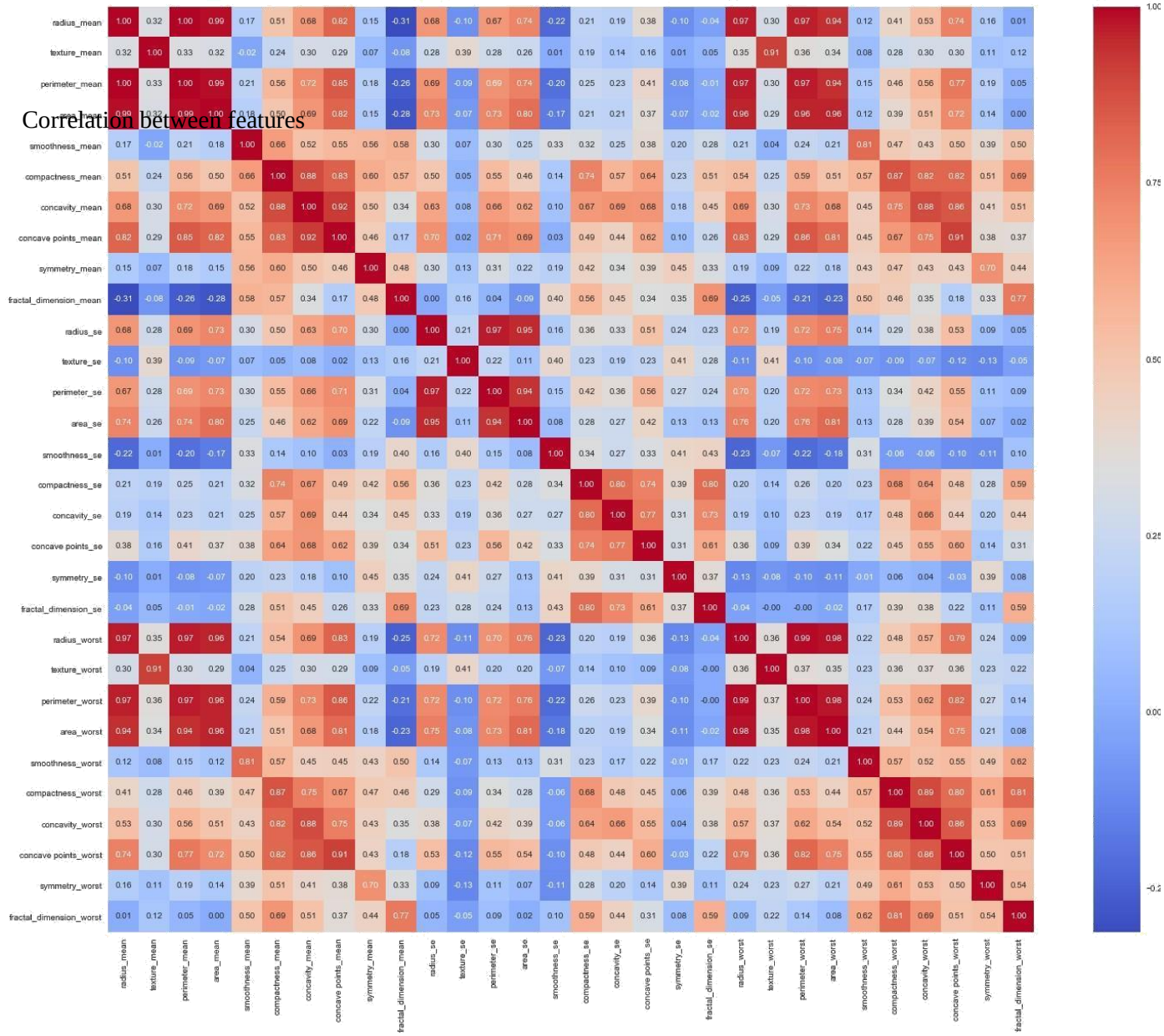
a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2/ area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave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

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

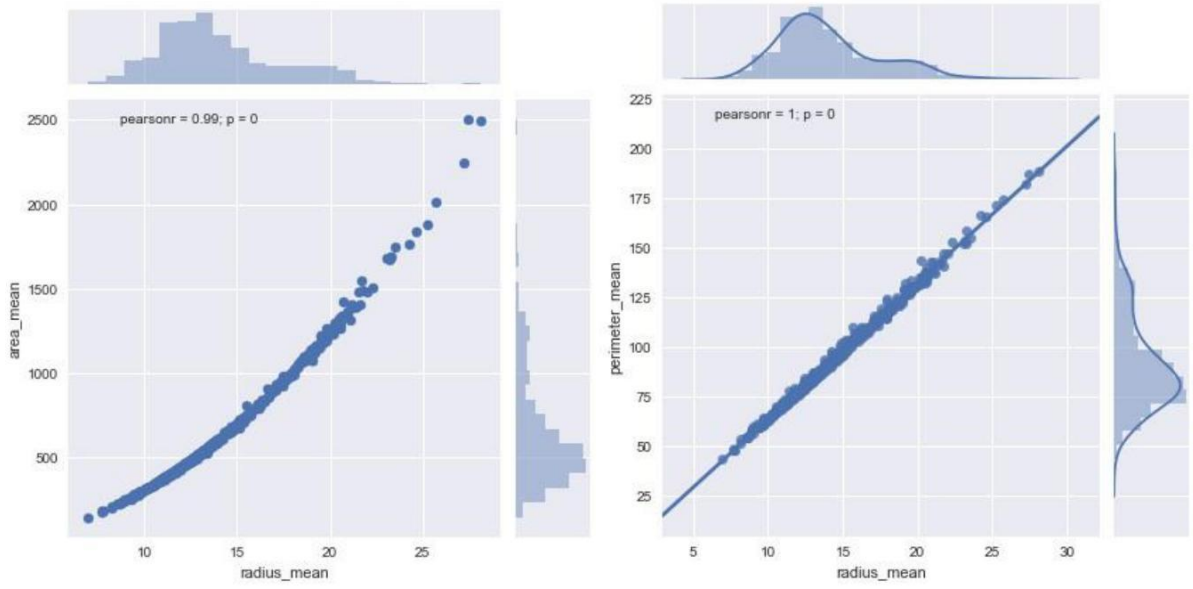
I used this physical characteristics as features to train a machine learning algorithms to create and evaluate models that classifies if a cell is benign or malignant based in this characteristics.

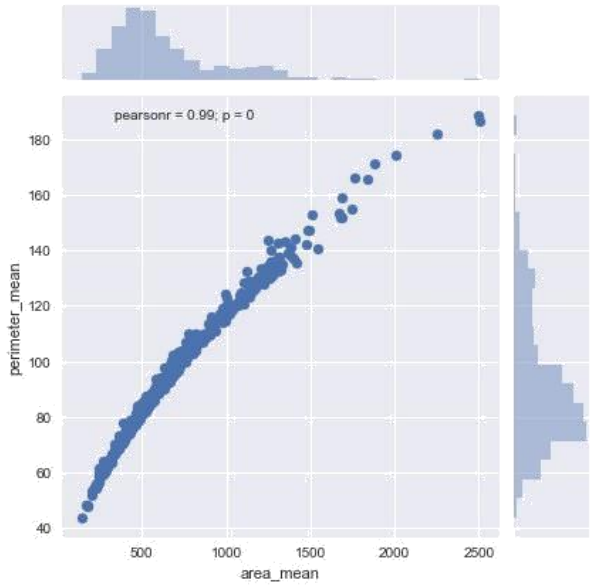
Exploratory Visualization

Correlation between features

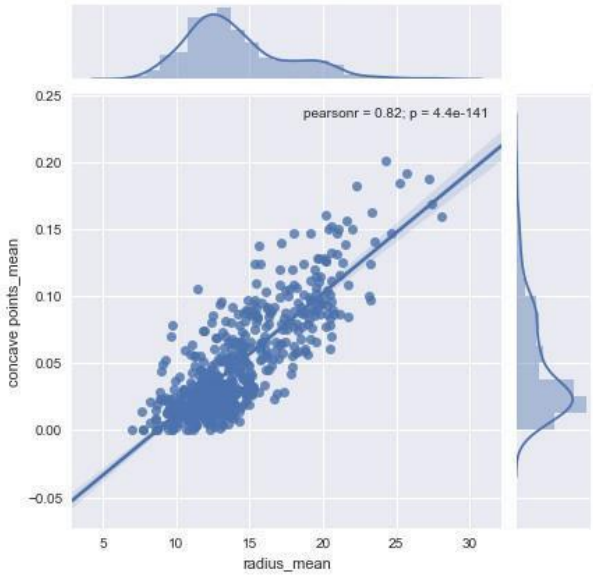


Radius, Perimeter and Area have strong positive correlation

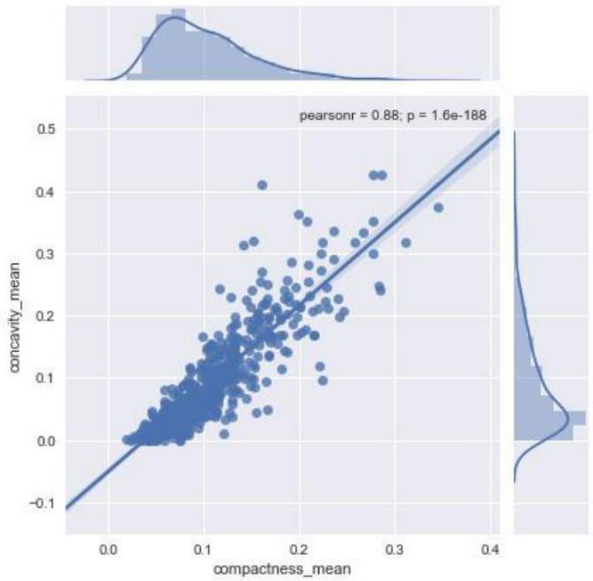


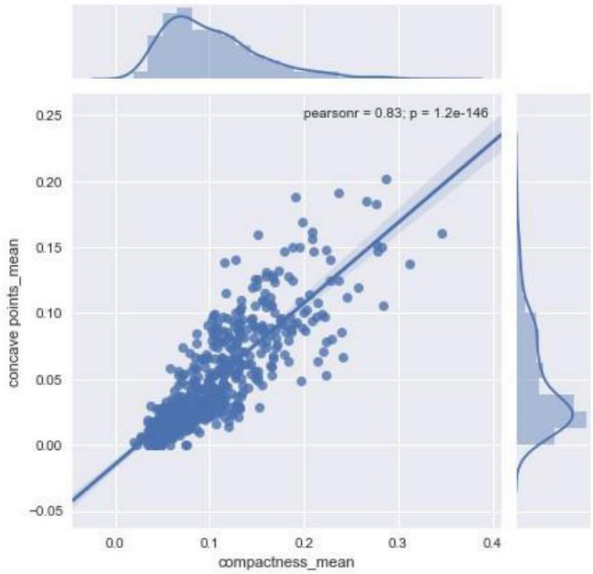


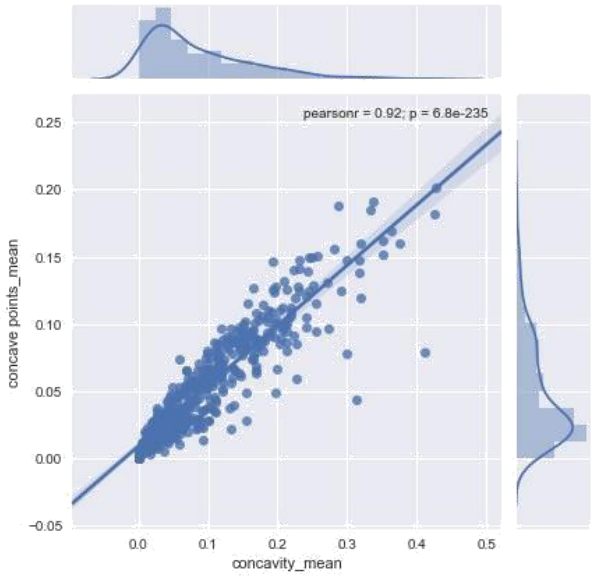
Radius have a positive correlation with Concave Points



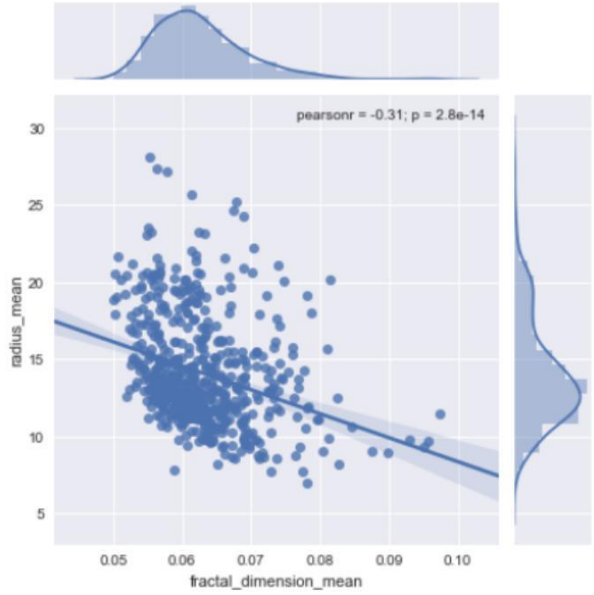
Compactness, Concavity and Concave Points have strong positive correlation

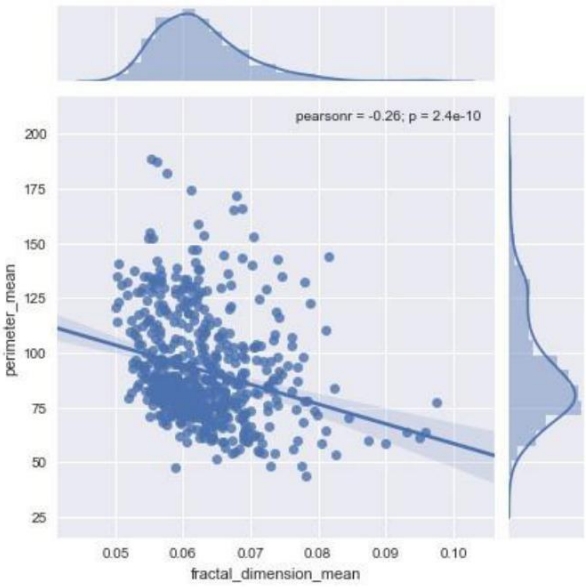


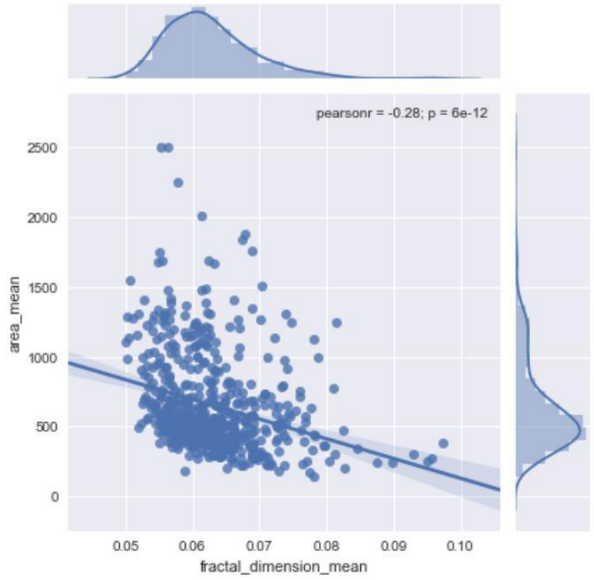




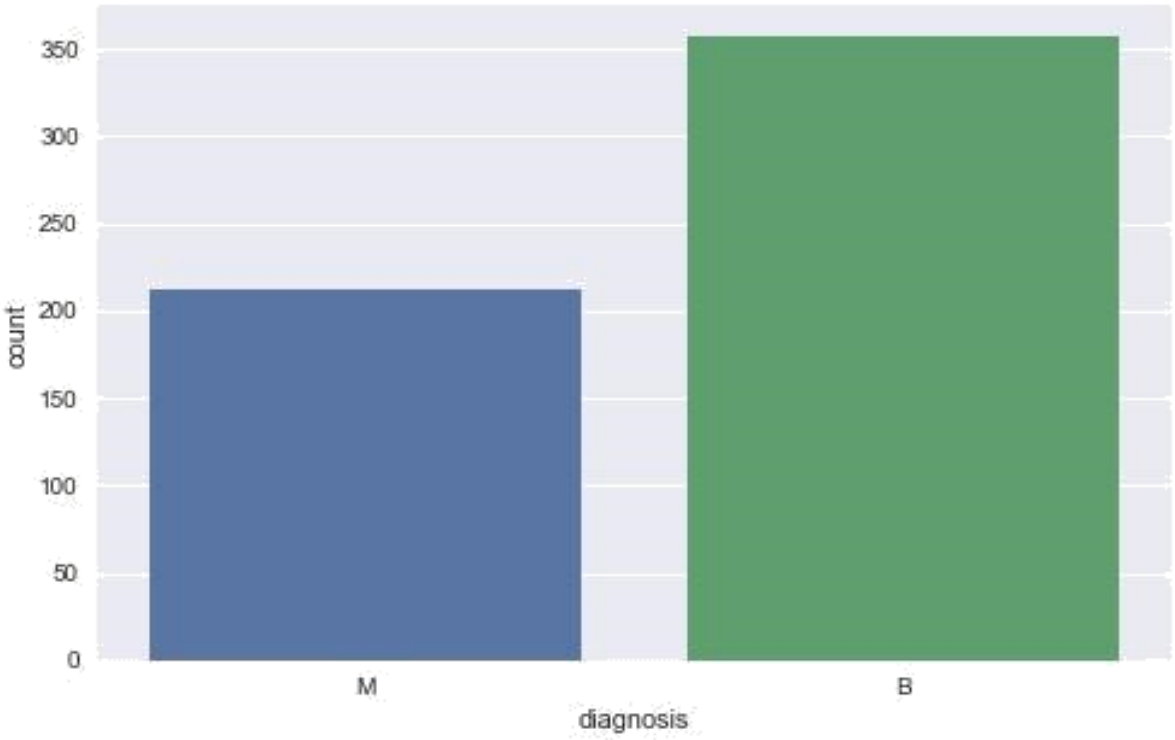
Fractal Dimention have some negative correlation with Radius, Perimeter and Area







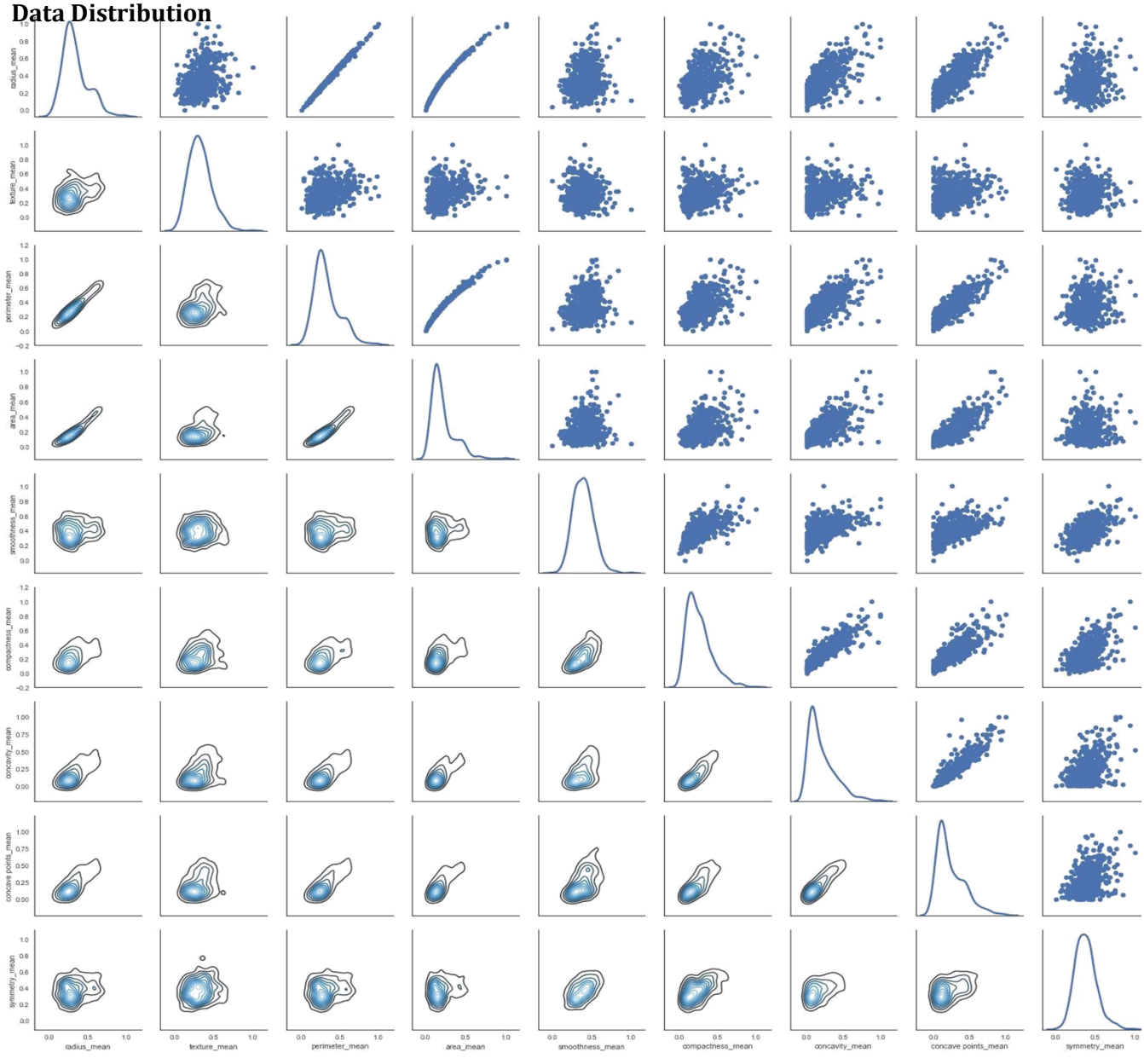
Distribution of Classes



Number of Benign : 357

Number of Malignant : 212

Data Distribution



Algorithms and Techniques

I tested many data manipulation techniques in dataset, I measured result to understand which combination of approaches is more effective

Feature engineering: I decided create two new features, Mean

Volume

```
# Creating a empty list
mean_volume = []
# defining pi
pi = 3.1415

# calculatin mean volume for each mean radius and saving result in mean_volume list for i in range(len(X)):

    #appending result in mean_volume list
    mean_volume.append((math.pow(X["radius_mean"][i], 3)*4*pi)/3)

# Creating a new feature
X["mean_volume"]= mean_volume
```

Measurements Sum

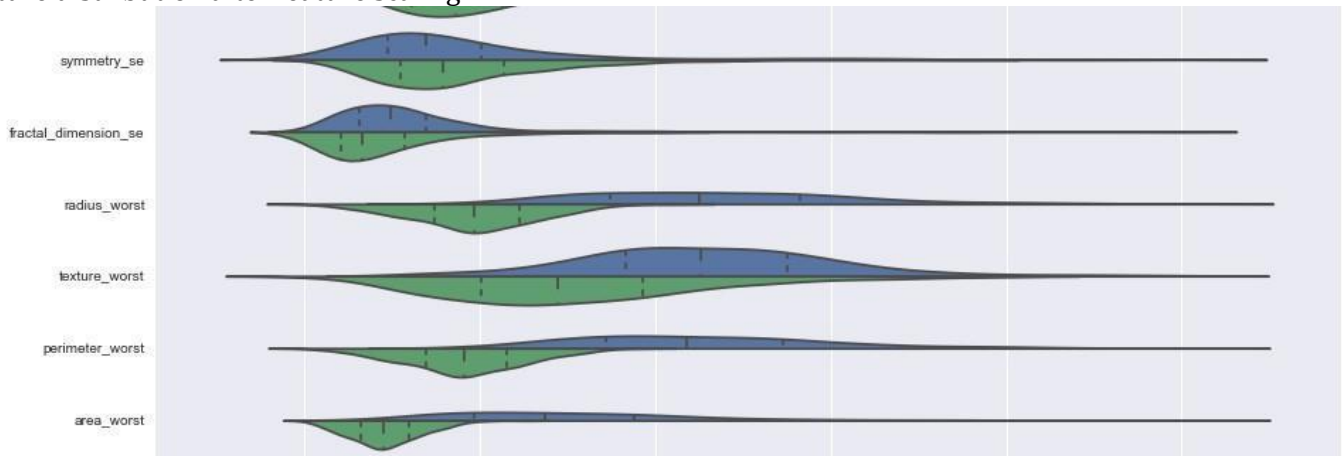
```
# Creating a new feature adding up some phisical measuraments
X["mesuraments_sum_mean"] = X["radius_mean"] + X["perimeter_mean"] + X["area_mean"]
```

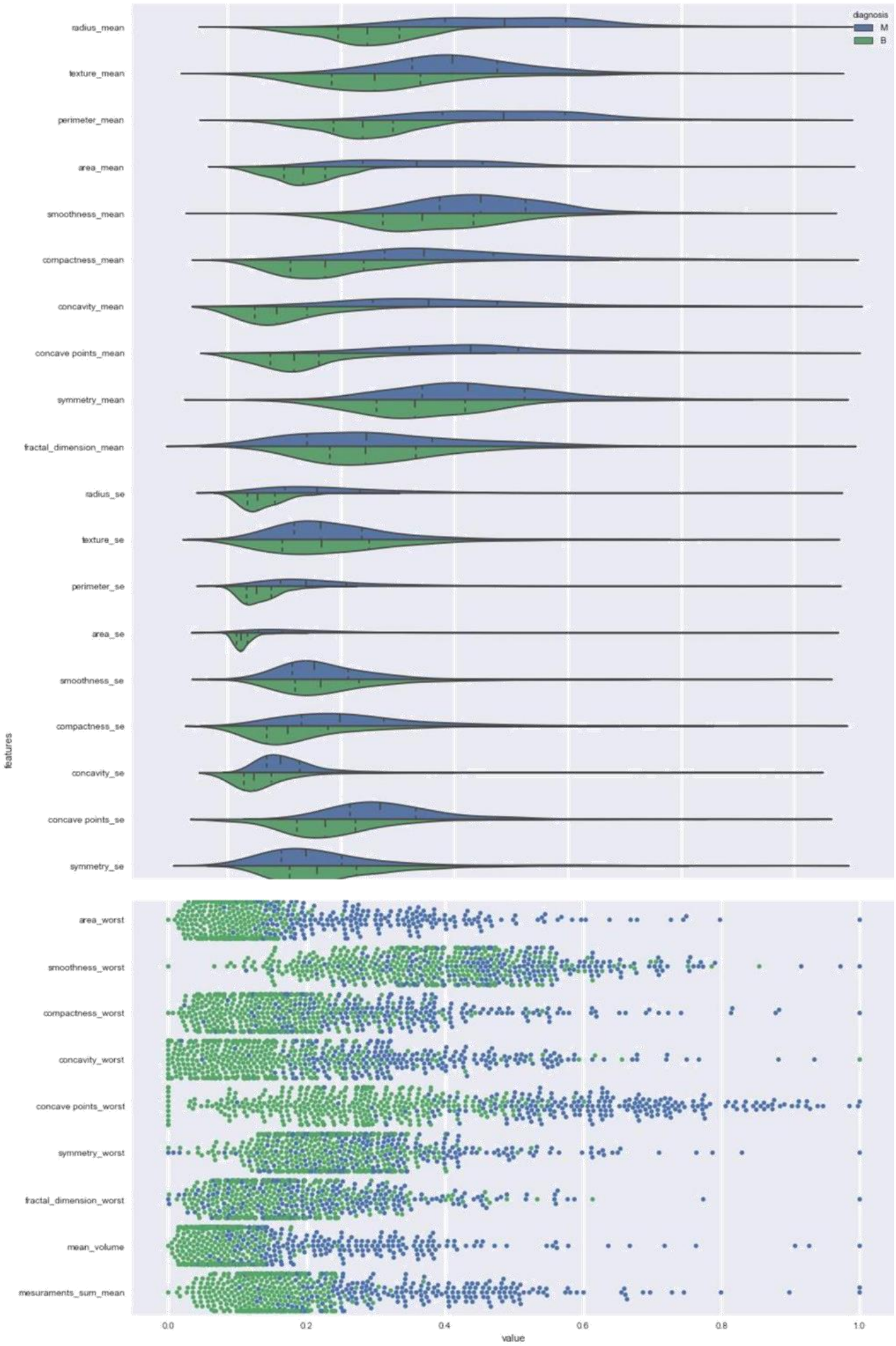
Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, the majority of classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

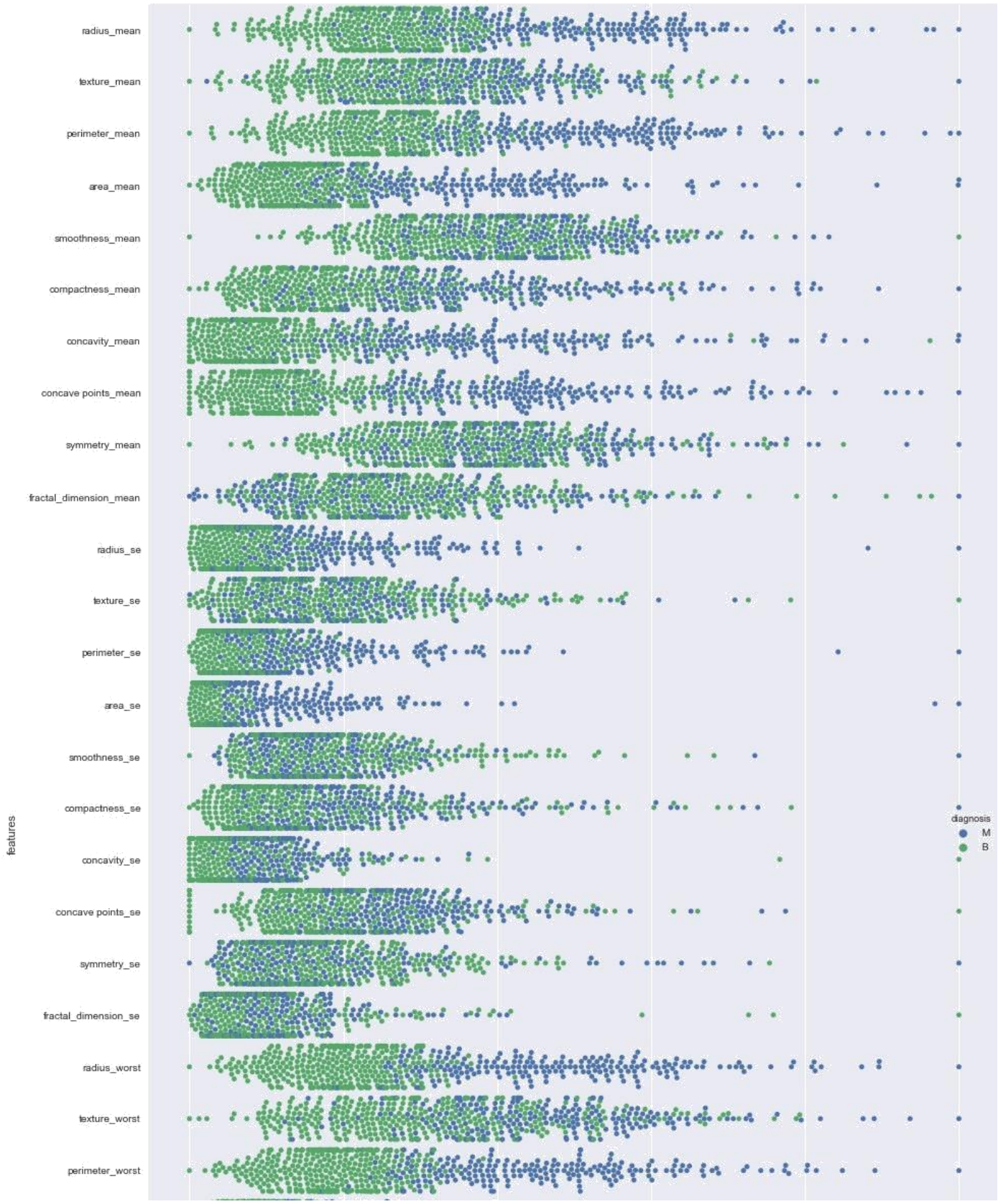
Another reason why feature scaling is applied is that gradient descent converges much faster with feature scaling than without it.[1]

Feature Scaling - Wikipedia

Feature distribution after feature scaling




diagnosis
M
B
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
area_worst
smoothness_worst
compactness_worst
concavity_worst
concave points_worst
symmetry_worst
fractal_dimension_worst
mean_volume
mesuraments_sum_mean
features
0.0
0.2
0.4
0.6
0.8
1.0
value



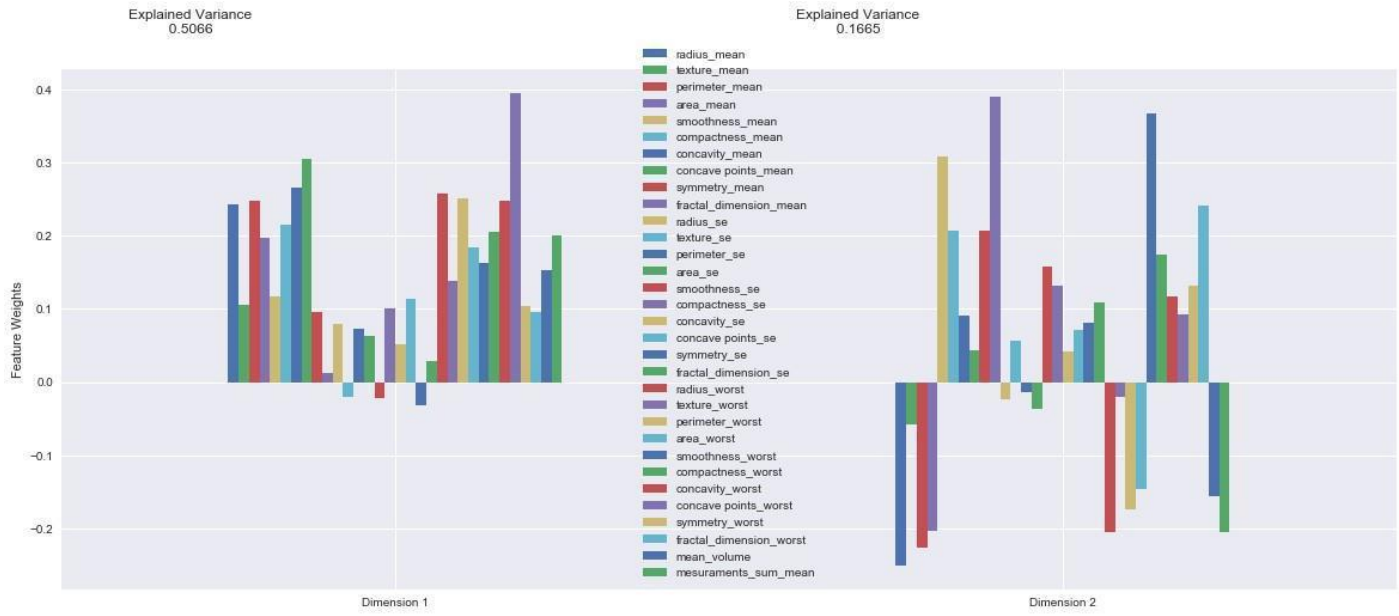
Detect Outliers using [Tukey Method]

```
# For each feature find the data points with extreme high or low values for feature in X.keys():

        # Calculate Q1 (25th percentile of the data) for the given feature Q1 =
        np.percentile(X[feature], 25)

        # Calculate Q3 (75th percentile of the data) for the given feature
```

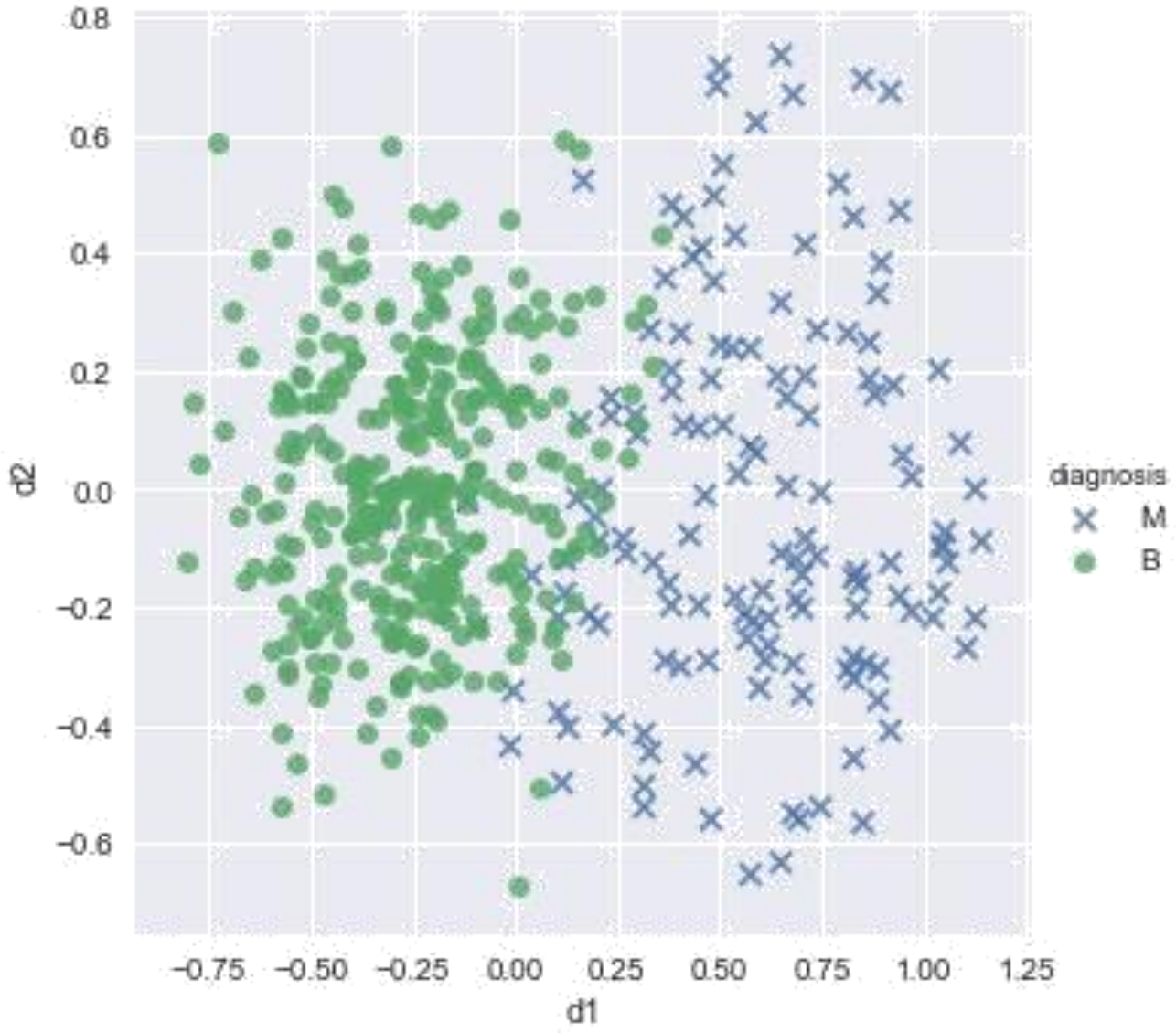
```
Q3 = np.percentile(X[feature], 75)

# Use the interquartile range to calculate an outlier step (1.5 times the int step = (Q3 - Q1) * distance

outliers.append(X[~((X[feature] >= Q1 - step) & (X[feature] <= Q3 + step))].i
```

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.

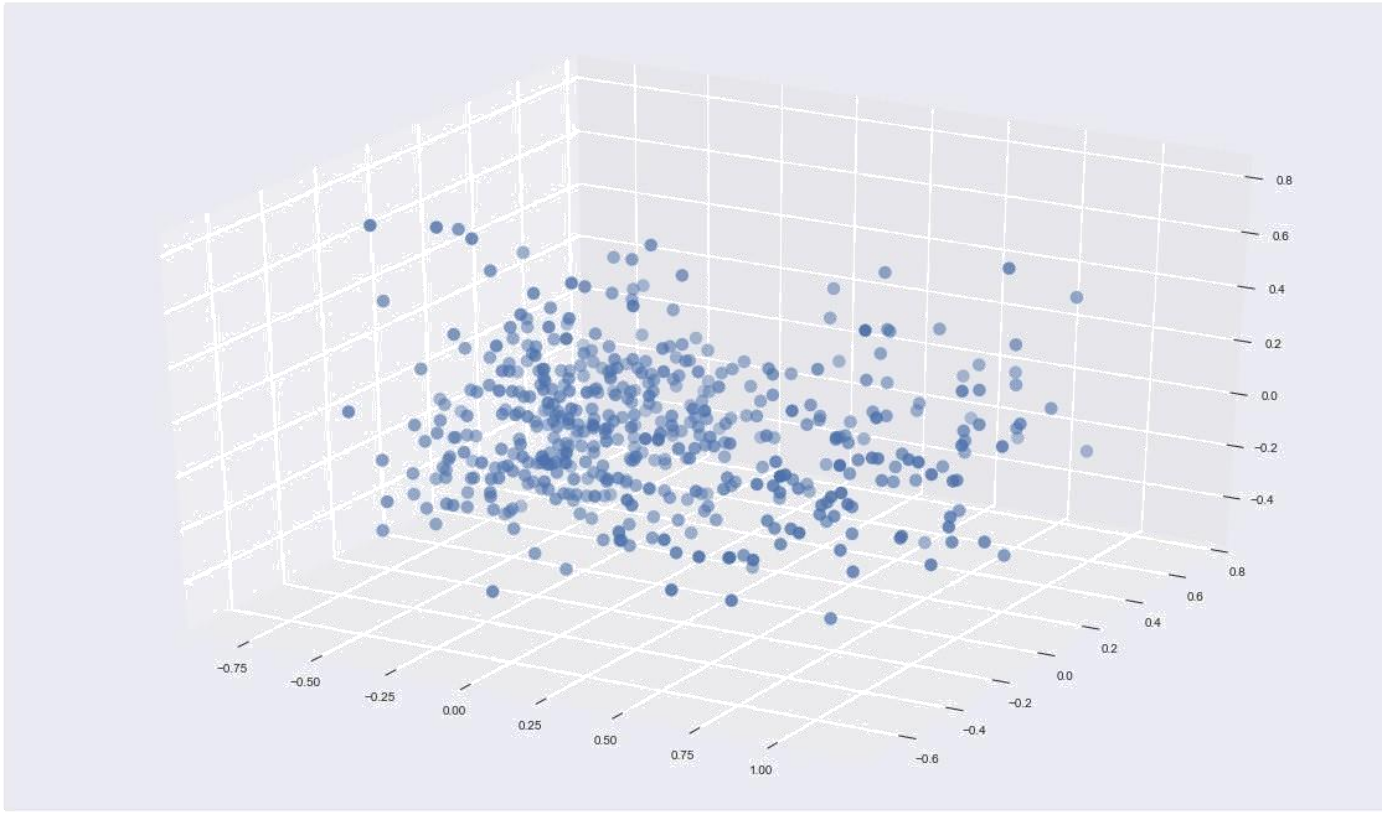
2-dimensional PCA Feature importance



2-dimensional PCA data separation



3-dimensional PCA data distribution

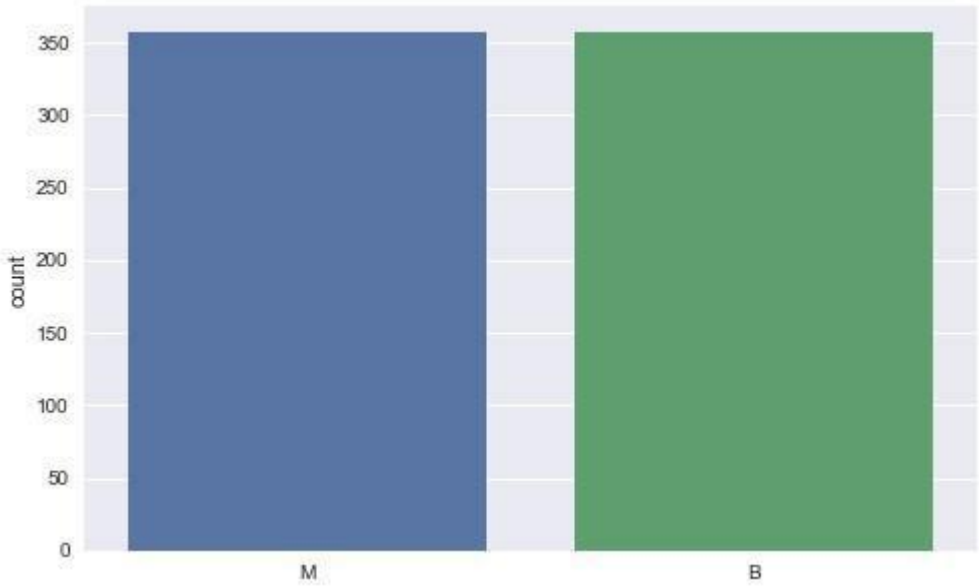


Dataset has more samples of benignant tumors over malignants, it cam make classification algorithm tend to predict more cases to dominant class, I studied impact of generate new samples with three different methods.

Imbalanced Learning

Naive random over-sampling

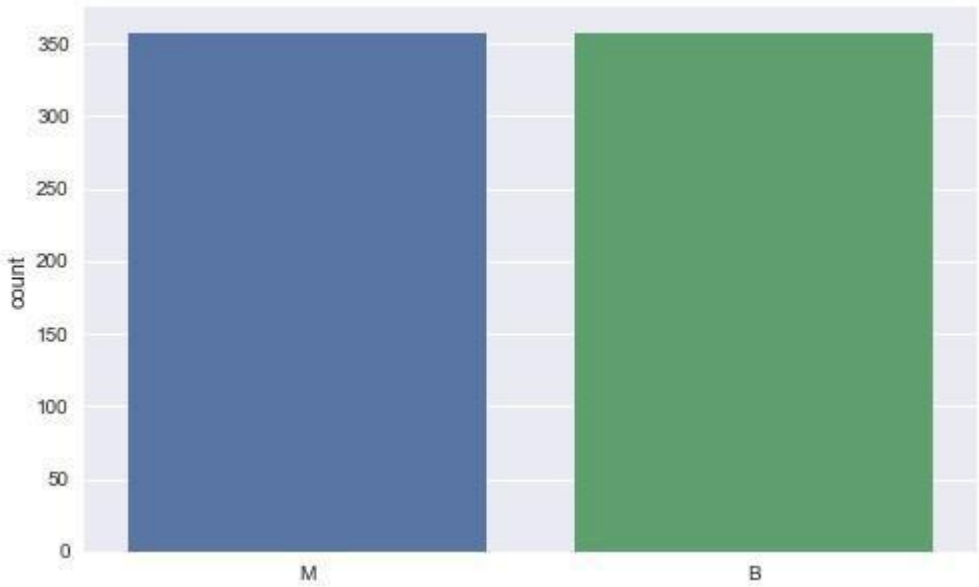
One way to fight this issue is to generate new samples in the classes which are under-represented. The most naive strategy is to generate new samples by randomly sampling with replacement the current available samples. The RandomOverSampler offers such scheme:



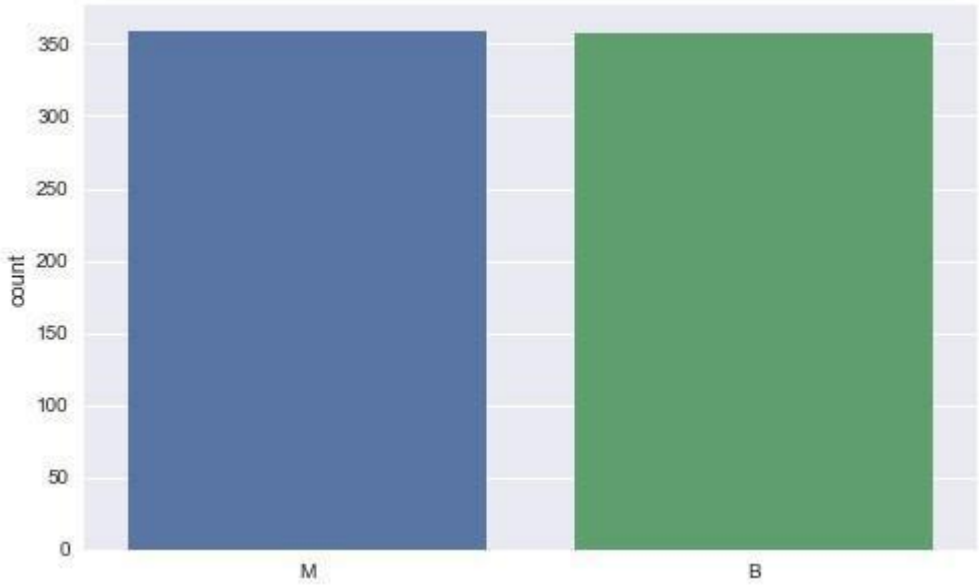
From random over-sampling to SMOTE and ADASYN

Apart from the random sampling with replacement, there is two popular methods to over-sample minority classes: (i) Synthetic Minority Oversampling Technique (SMOTE) and (ii) Adaptive Synthetic (ADASYN) sampling method. These algorithm can be used in the same manner:

SMOTE



ADASYN



Feature Selection using Scikit Learn

Feature selection works by selecting the best features based on univariate statistical tests. It can be seen as a preprocessing step to an estimator. SelectKBest removes all but the k highest scoring features

We going to performe Feature Selection using SelectKBest module in sklearn combination of features.

Best Features using k=5:

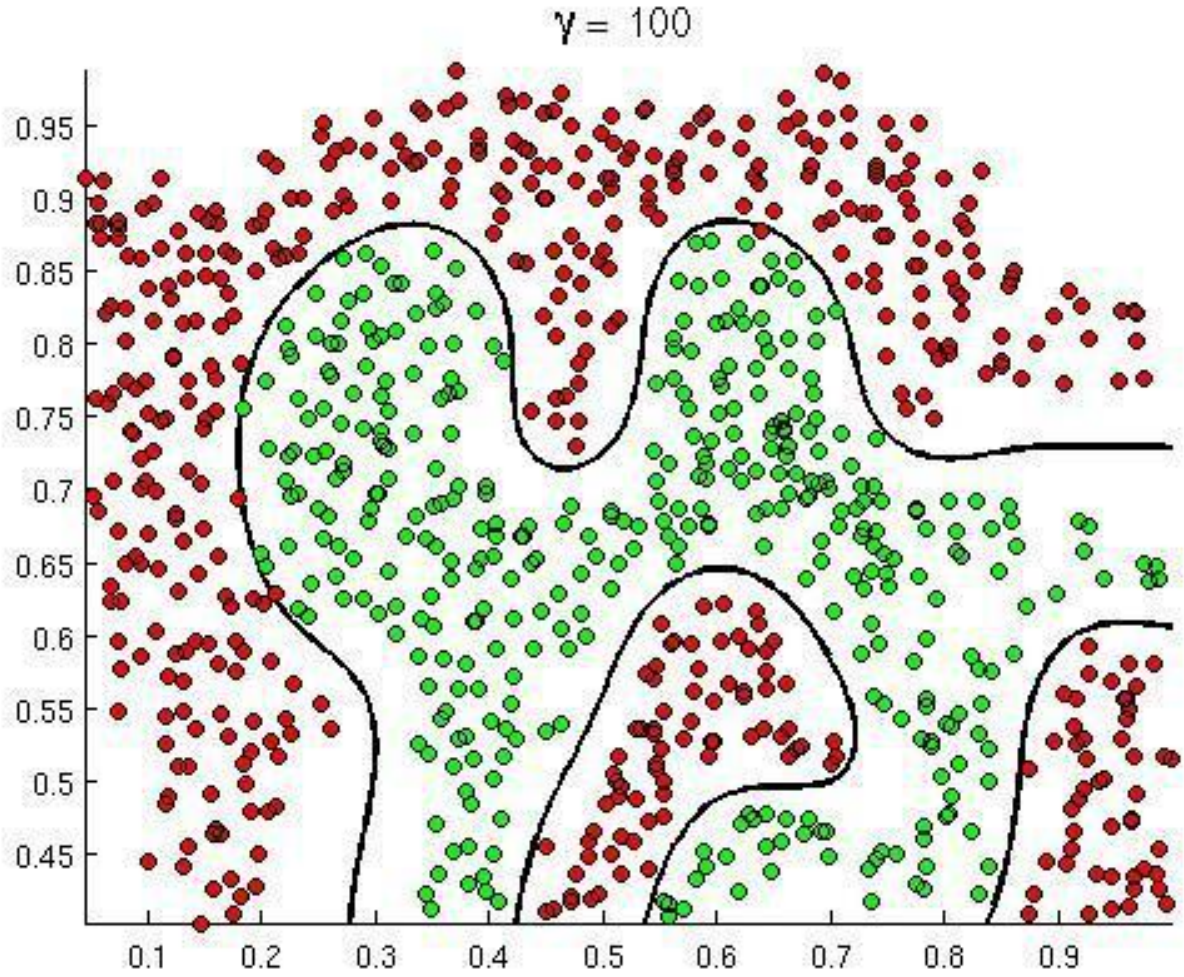
	area_mean	area_se	area_worst	mean_volume	mesuraments_sum_mean
0	1001.0	153.40	2019.0	24387.612775	1141.79
1	1326.0	74.08	1956.0	36456.810913	1479.47
2	1203.0	94.03	1709.0	31975.176401	1352.69
3	386.1	27.23	567.7	6238.412850	475.10
4	1297.0	94.44	1575.0	34988.227503	1452.39

We going to performe Feature Selection using tools like feature_selection, SelectKBest or SelectPercentile, modules in sklearn. And Dimension Reduction to get the optiomal decision bondary/surface mesuring accuracy of different models, combination of features and dimensionality.

Supervised Learning

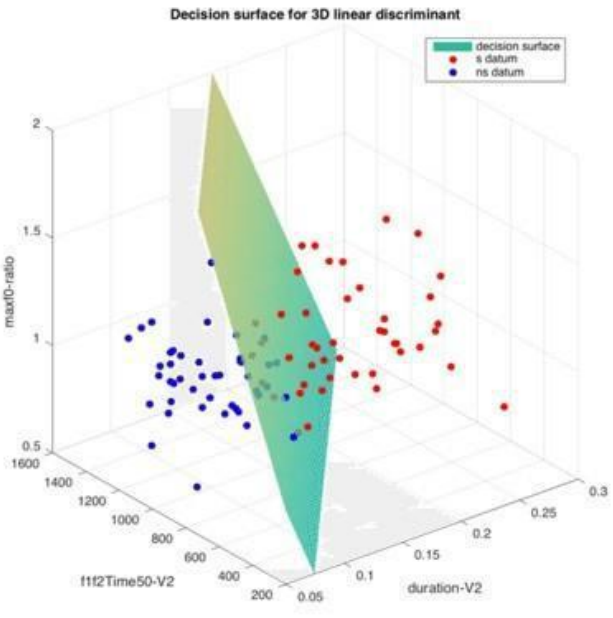
Using a dataset with 32 features going to predict if class a sample belongs, it means that we face a multidimensional classification problem. Our job is understand and prepare data to feed a algorithm wich should find the best decision boundary or in our case, the best decision hypperplane, see:

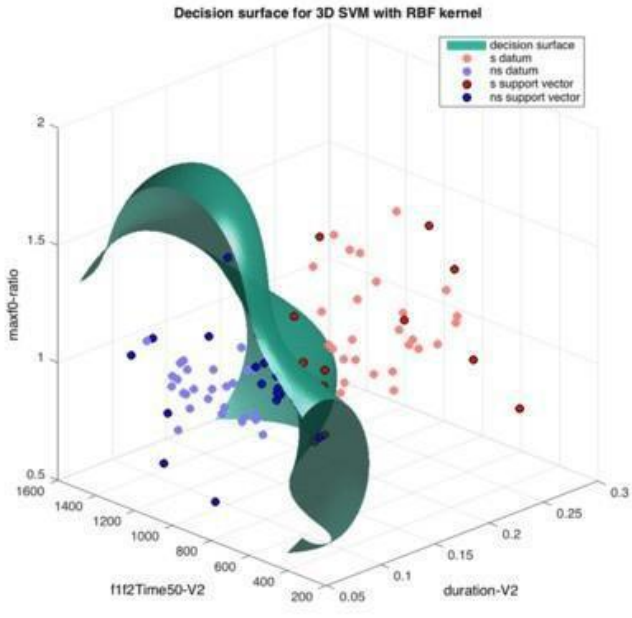
** Decision Boundary 2-dimensional **



When we have 2-dimensional data we can separate data points using a decision boundary.

** Decision Surface - 3 dimensional **





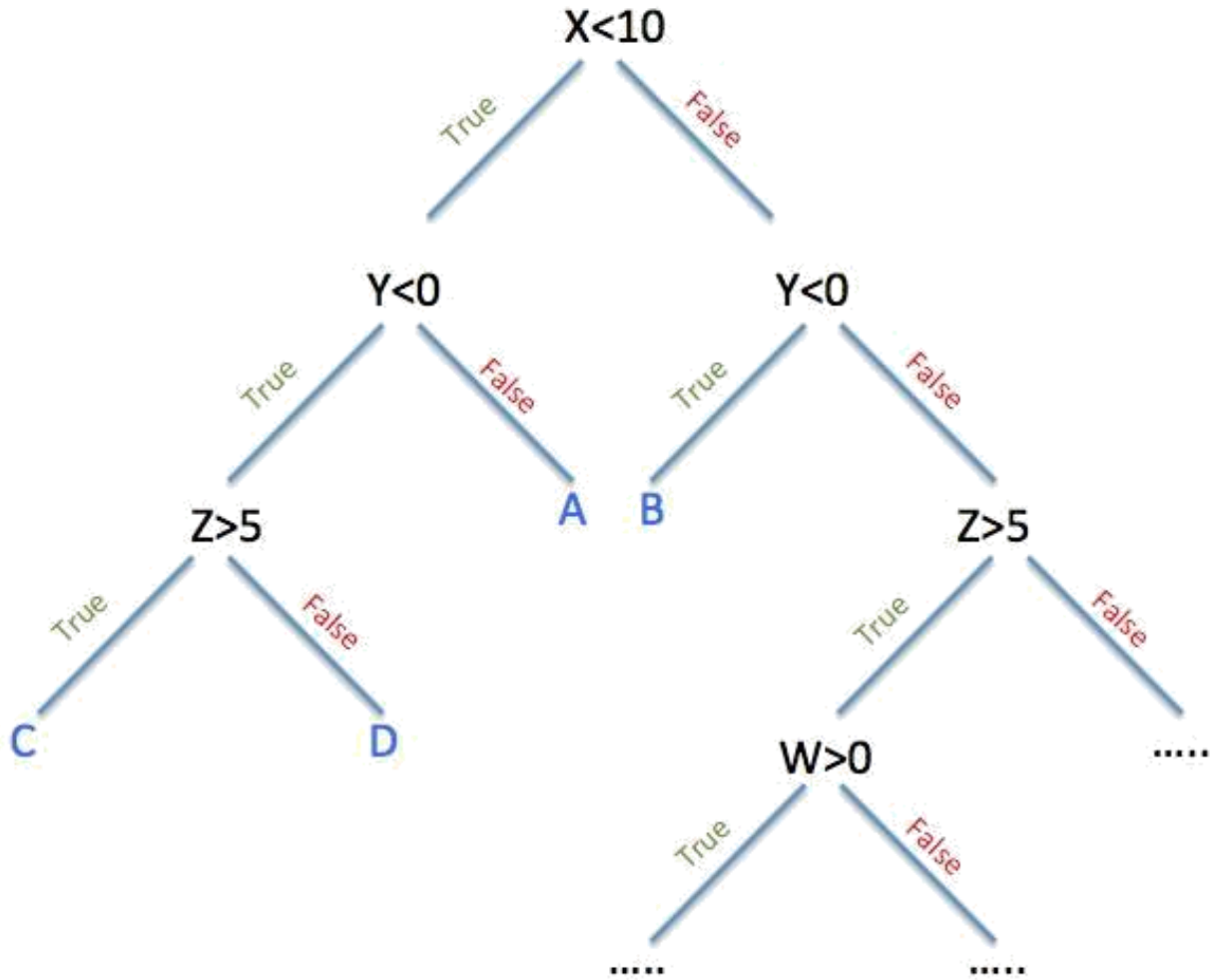
Decision surface, a surface that separates data points in a 3-dimensional space.

The task of selected algorithm is learn from train data to find the best decision surface and correctly predict "M" Malignant or "B" Benignant for unsee test data, that was picked randomly from dataset.

Machine Learning Algorithms

In this project I used 9 different classifiers to analyze they results and get the higher F1 score, I will give a briefly explanation about each one of this classifiers:

Decisions Trees:



A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

Random Forest or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

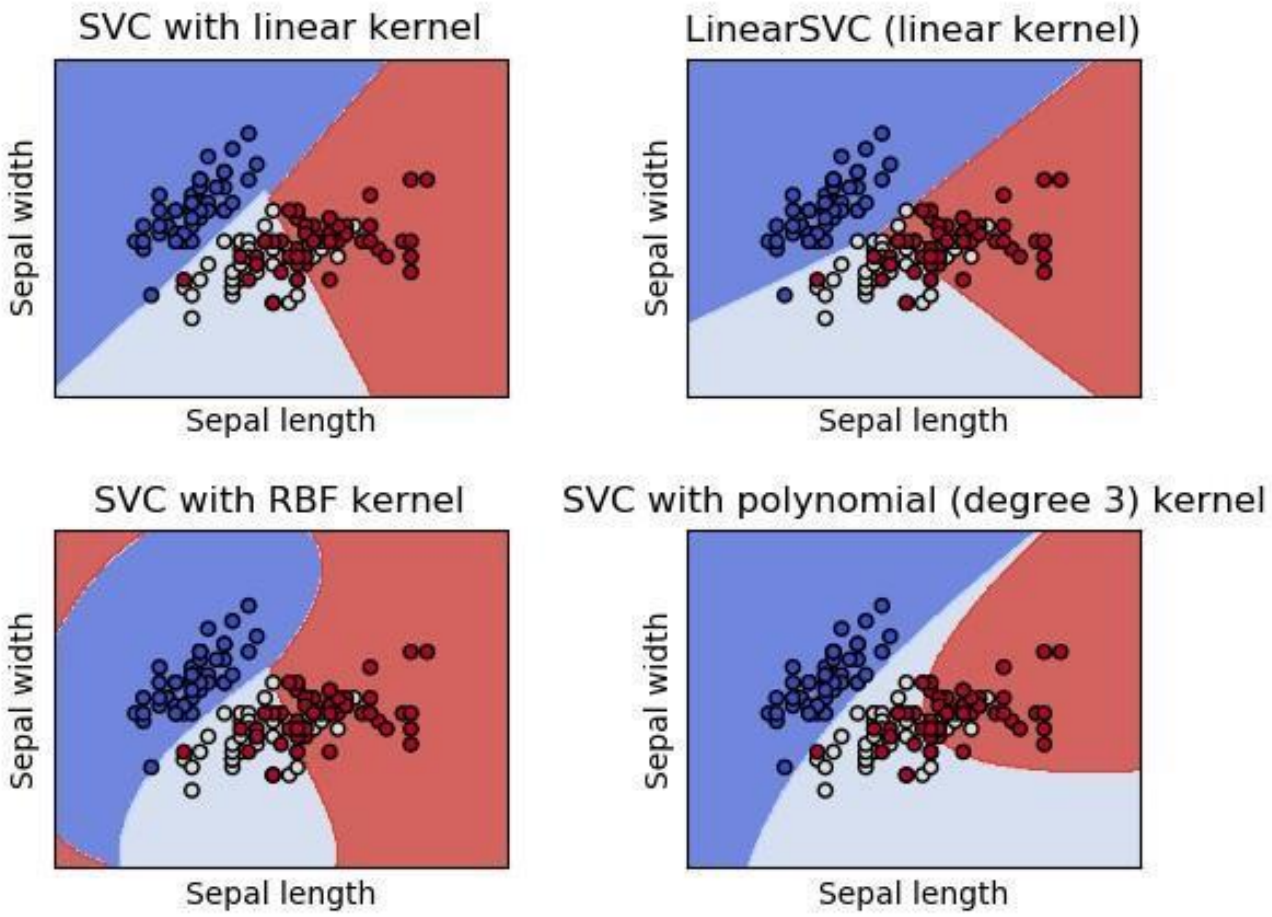
Extra DecisionTrees This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

Gradient Boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

AdaBoost Classifier is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers.
In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.

XGBoost Classifier initially started as a research project by Tianqi Chen as part of the Distributed (Deep) Machine Learning Community (DMLC) group. Initially, it began as a terminal application which could be configured using a libsvm configuration file. After winning the Higgs Machine Learning Challenge, it became well known in the ML competition circles. Soon after, the Python and R packages were built and now it has packages for many other languages like Julia, Scala, Java, etc. This brought the library to more developers and became popular among the Kaggle community where it has been used for a large number of competitions

Support Vector Machines



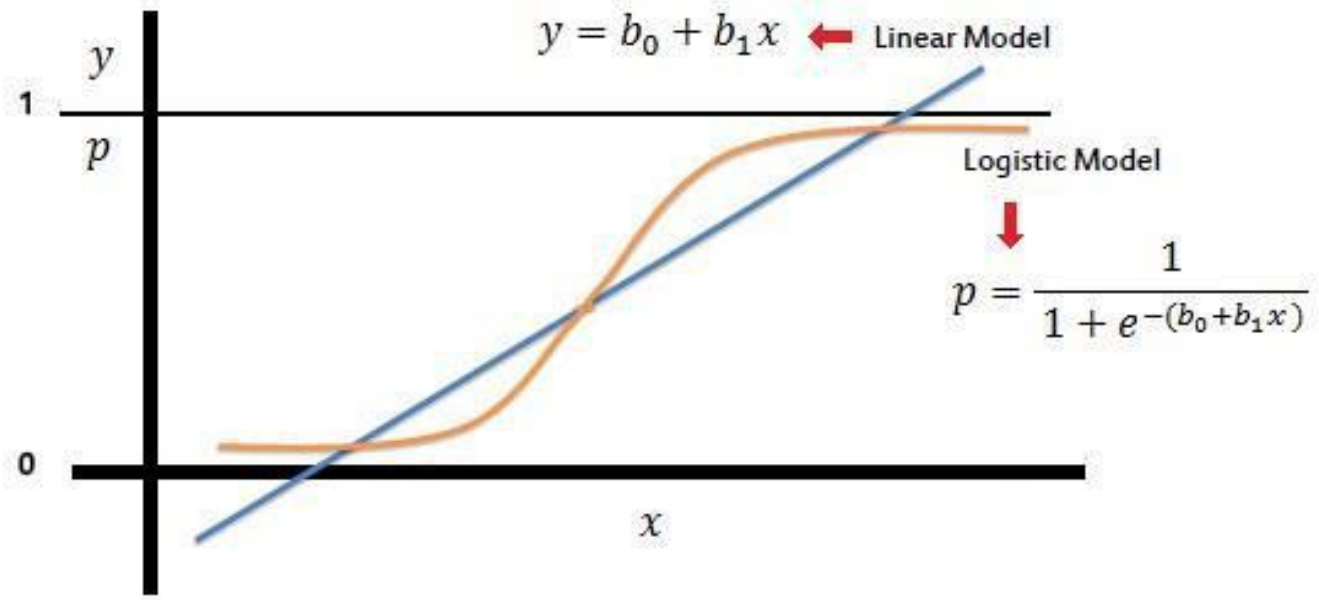
Support Vector are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high- dimensional feature spaces.

SGD Classifier this estimator implements regularized linear models with stochastic gradient descent learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning, see the partial_fit method. For best results using the default learning rate schedule, the data should have zero mean and unit variance.

This implementation works with data represented as dense or sparse arrays of floating point values for the features. The model it fits can be controlled with the loss parameter; by default, it fits a linear support vector machine (SVM).

Logistic Regression



Was developed by statistician David Cox in 1958. The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). It allows one to say that the presence of a risk factor increases the odds of a given outcome by a specific factor. The model itself simply models probability of output in terms of input, and does not perform statistical classification, though it can be used to make a classifier, for instance by choosing a cutoff value and classifying inputs with probability greater than the cutoff as one class, below the cutoff as the other.

Benchmark

To realize how much effective the method is, after optimize our model I'll compare results with results extracted of the following Paper:

Approximate Distance Classification
Adam H. Cannon, Lenore J. Cowen, Carey E. Priebe
Department of Mathematical Sciences
The Johns Hopkins University
Baltimore, MD, 21218

They used k-nearest neighbor to predict cancer in same dataset to diagnosis Breast Cancer and their effectiveness, in this way we can measure and compare results in the same domain.

" The best among these was 5 nearest neighbors, and it achieved only 93.1% classification rates. Our best reported result is 95.6% compared to a reported best known result of 97.5%" - Approximate Distance Classification, pg.4

III. Methodology

Data Preprocessing Data

structure

```
Data columns (total 33 columns):
id                         569 non-null int64
diagnosis                  569 non-null object
radius_mean                569 non-null float64
texture_mean               569 non-null float64
perimeter_mean             569 non-null float64
area_mean                  569 non-null float64
smoothness_mean            569 non-null float64
compactness_mean           569 non-null float64
concavity_mean             569 non-null float64
concave points_mean        569 non-null float64
symmetry_mean              569 non-null float64
fractal_dimension_mean     569 non-null float64
radius_se                  569 non-null float64
texture_se                 569 non-null float64
perimeter_se               569 non-null float64
area_se                    569 non-null float64
smoothness_se              569 non-null float64
compactness_se             569 non-null float64
concavity_se               569 non-null float64
concave points_se          569 non-null float64
symmetry_se                569 non-null float64
fractal_dimension_se       569 non-null float64
radius_worst               569 non-null float64
texture_worst              569 non-null float64
perimeter_worst            569 non-null float64
area_worst                 569 non-null float64
smoothness_worst           569 non-null float64
compactness_worst          569 non-null float64
concavity_worst            569 non-null float64
concave points_worst       569 non-null float64
symmetry_worst             569 non-null float64
fractal_dimension_worst    569 non-null float64
Unnamed: 32                0 non-null float64
```

```
#Save labels in y
y = data["diagnosis"]
```

"diagnosis" is our target, I saved this feature in a 1-dimensional dataset named y

```
# Drop columns
X = data.drop(["id", "diagnosis", "Unnamed: 32"], axis=1)
```

"Unannamed: 32" feature has only NaN (not a number) values
"Id" feature hasn't information to help us classify benignant/malignant tumors. I deleted "Unannamed: 32" and "Id]' features and save in a X dataframe.

Using Train Test Split I divided data in train and test data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42
```

Implementation

Predict if a tumor is malignant or benignant is a classification problem, I choose some of the best classifier algorithms to perform this task. The initial process was very simple, using previously shuffled and divided data:

X_train (train features) and y_train (train labels)

I used Scikit Learn that is nativily instaled from Anaconda.

```
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier from sklearn.ensemble import
GradientBoostingClassifier, AdaBoostClassifier from sklearn.linear_model import LogisticRegression,
SGDClassifier
from sklearn.tree  import  DecisionTreeClassifier from
sklearn.neighbors import  NearestNeighbors from
sklearn.naive_bayes  import  GaussianNB
from sklearn import  svm from
sklearn import  tree
```

And also installed the XGBoost Classifier

```
from xgboost  import  XGBClassifier
```

Classifiers list:

```
#  Random Forest  Classifier
RF_clf =  RandomForestClassifier()

#  Extra Trees Classifier
XT_clf =  ExtraTreesClassifier()

#  Decision Tree  Classifier
DT_clf =DecisionTreeClassifier()

#  Support Vector Machine Classifier
SV_clf = svm.SVC()

#  AdaBoost  Classifier
AD_clf =  AdaBoostClassifier()

#  Gradient  Boosting Classifier
GB_clf =  GradientBoostingClassifier()

#  SGD  Classifier
SG_clf =  SGDClassifier()

#  Logistic  Regression
LR_clf =  LogisticRegression()
```

```
# XGB Classifier
XB_clf = XGBClassifier()

classifiers = [RF_clf, XT_clf, DT_clf, SV_clf, AD_clf, GB_clf, SG_clf, LR_clf, XB_clf classifiers_names = ['Random

Forest ', 'Extra DecisionTrees', 'Decision Tree
                    'Support Vector ', 'AdaBoost Classifier', 'Gradient Boosting
                    'SGD Classifier ', 'Logistic Regression', 'XGB Classifier

parameters = [RF_par, XT_par, DT_par, SV_par, AD_par, GB_par, SG_par, LR_par, XB_par]
```

I feed classifiers with (X, y) using the following function to train, tune using Grid Search, print F1 scores and store results:

```
def tune_compare_clf(X, y, classifiers, parameters, classifiers_names):


    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_stat print "\n" "Train size : ",


    X_train.shape, " and Train labels : ", y_train.shape, "\n

    print "Test size: ", X_test.shape, " and Test labels : ", y_test.shape, "\n", "\n" results = [] print " --

    -- F1 Score ---- ", "\n"

    for clf, par, name in itertools.izip(classifiers, parameters, classifiers_names):
        # Store results in results list
        clf_tuned = GridSearchCV(clf, par).fit(X_train, y_train)
        y_pred = clf_tuned.predict(X_test)
        results.append(y_pred)

        print name, ": %.2f%%" % (f1_score(y_test, y_pred, average='weighted') * 100.0)

    result = pd.DataFrame.from_records(results)

    return result, X_test, y_test
```

Without any manipulation technique as scaling or outliers removing I get the following results: Random

Forest : 95.60%
Extra DecisionTrees : 96.47%
Decision Tree : 94.74%
Support Vector : 47.80%
AdaBoost Classifier : 96.49%
Gradient Boosting : 94.74%
SGD Classifier : 87.42%
Logistic Regression : 96.47%

XGB Classifier : 96.47%

Refinement

Using ** Grid Search** I did a Exhaustive search over specified parameter values for an estimator. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

```
# Random Forest Classifier Parameters to tune
RF_par = {"max_depth": [3, None], "max_features": [1, 3, 10], "min_samples_split": [2
          "min_samples_leaf": [1, 3, 10], "bootstrap": [True, False], "criterion": ["gini

# Extra Trees Classifier Parameters to tune
XT_par = { 'n_estimators': [5, 10, 16], "min_samples_split": [2, 3, 10], "min_samples_lea

# Decision Tree Classifier Parameters to tune
DT_par = { 'splitter': ['best', ], "min_samples_split": [2, 3, 10], "min_samples_leaf"

# Support Vector Machine Classifier Parameters to tune
SV_par = {'kernel': ['rbf'], 'C': [1]}

# AdaBoost Classifier Parameters to tune
AD_par = {'n_estimators':[10, 20, 50, 60], 'learning_rate':[0.1, 0.5, 1.0, 1.5], 'algorit

# Gradient Boosting Classifier Parameters to tune
GB_par = {'loss':['deviance', 'exponential'], 'learning_rate':[0.01, 0.1, 0.5, 1.0],
          "min_samples_split": [2, 3], "min_samples_leaf": [1, 3], 'max_depth':[2, 3,

# SGD Classifier Parameters to tune
SG_par = {'loss':['hinge', 'log', 'squared_hinge', 'perceptron'], 'penalty':['l2', 'l1'
          'alpha':[0.00001, 0.0001, 0.001], 'epsilon':[0.01, 0.1, 0.5]}

# Logistic Regression Parameters to tune
LR_par= {'penalty':['l1','l2'], 'C': [0.5, 1, 5, 10], 'max_iter':[50, 100, 150, 200]}

# XGB Classifier Parameters to tune
XB_par = {'max_depth':[2, 3, 5], 'learning_rate':[0.01, 0.1, 0.5, 1], 'n_estimators':[
```

I defined functions to apply data manipulation techniques cited before and tested different approaches measuring results to find the best classifier configuration.

scaler(X) : The Function receive a Dataframe and return a Scaled Dataframe

selector(X, y, k) : The function receive features and labels (X, y) and a target number to select features (k) and return a new dataset wiht k best features

remove_outliers(X, y, f, distance): The Function receive Features (X) and Label (y) a frequency (f) and Inter-Quartile distance (distance), and return features and labels without outliers (good_X, good_y)

resample(X, y, method): The function receive features and labels (X, y) and a method to balance

data available methods RandomOverSampler, ADASYN, SMOTE. The funcion returns X_resampled, y_resampled

Now we'll test using:

tune_compare_clf(X, y, classifiers, parameters, classifiers_names): a function that tune each algorith to given data and print F1 Scores.
I took this F1 score as my evaluation metric because F1 score works better than the accuracy as my dataset is imbalanced .

IV. Results

Model Evaluation and Validation

To evaluate results I used unsee data (20% of samples randomly chosen from original dataset) to compare results with true result (true label) and calculate precision, recall and F1 score.

Results using each data manipulation technique

Classifier	Original	Scaled	Outiliers Removed	12 Features	10 Features	Resampled Randon	SMOTE	ADASYN
Random Forest	95.60%	96.47%	94.23%	94.68%	95.58%	96.50%	96.50%	96.52%
Extra DecisionTrees	96.47%	95.60%	96.14%	96.47%	**98.24%**	**99.30%**	97.90%	95.83%
Decision Tree	94.74%	94.74%	93.22%	95.58%	94.74%	97.20%	93.70%	94.44%
Support Vector	47.80%	96.45%	50.05%	47.80%	47.80%	37.32%	37.32%	44.34%
AdaBoost Classifier	**96.49%**	96.49%	96.12%	95.60%	94.71%	97.90%	96.50%	95.83%
Gradient Boosting	94.74%	94.74%	94.21%	92.95%	95.60%	96.50%	96.50%	96.52%
SGD Classifier	87.42%	**97.36%**	90.16%	76.39%	24.34%	73.29%	61.44%	74.61%
Logistic Regression	96.47%	**97.36%**	**98.06%**	95.58%	97.35%	98.60%	**98.60%**	**97.92%**
XGB Classifier	96.47%	96.47%	96.16%	**96.49%**	94.74%	95.80%	95.80%	95.83%

We can notice that scale (0 to 1) data give us more consistent results across all classifiers, specially with support vector machine that performs poorly in all scenarios without scaled data.

Remove outliers do not show any improvements as well as feature selection that improved some classifier but worsen other results.

Re-sample techniques to correct the unbalanced dataset shows improvements for most of the classifiers and random re-sample seems to be the better method than Smote and Adasyn.

Results using combination of data manipulation technique

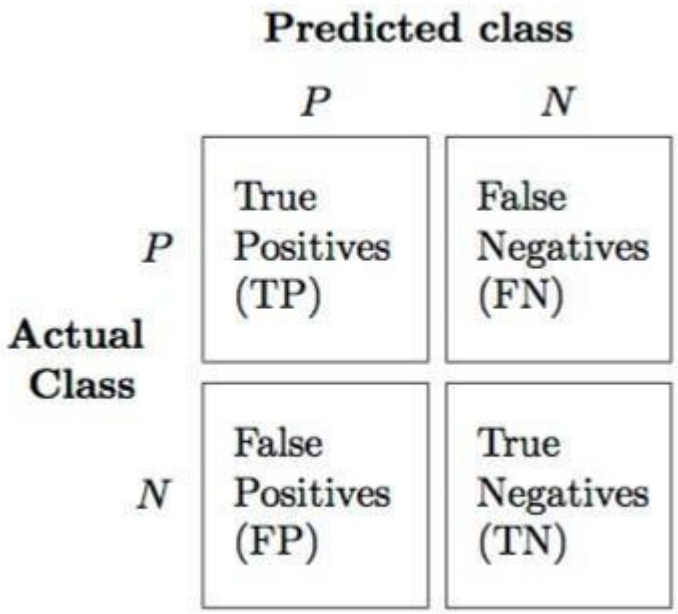
Classifier	Scaled + Outiliers Removed	Scaled + Resampled	Scaled + Outiliers Removed+Resampled	Scaled + Feature + Out. Rem + Resampled
Random Forest	96.14%	**97.90%**	97.83%	96.47%
Extra DecisionTrees	94.21%	96.50%	97.10%	96.47%
Decision Tree	91.24%	97.20%	94.20%	94.74%
Support Vector	**98.05%**	95.81%	97.83%	47.80%
AdaBoost Classifier	96.14%	95.11%	97.10%	**96.49%**
Gradient Boosting	95.20%	96.50%	**99.28%**	95.60%
SGD Classifier	**98.05%**	93.68%	97.83%	85.07%
Logistic Regression	97.08%	95.80%	97.83%	96.47%
XGB Classifier	96.16%	97.20%	98.55%	96.47%

A combination of Scaling, outliers removal and re-sample was the best approach, giving us the highest scores across all classifiers and more reliable results.

The tune_compare() function store prediction of all algorithm in a dataframe, I used this results to create a voting system, using describe() function I get the most common prediction for all classifier creating a high resilient system and avoid fluctuations in performance.

For each sample I save in y_pred_votes the most predict class.

```
y_pred_votes = result.describe().iloc[[2]]
```



71 data points was correctly predicted as Benign tumors

- 65 data points was correctly predicted as Malignant tumors
- 01 data point was wrongly predicted as Malignan tumor
- 01 data point was wrongly predictes as Benign tumor

F1 Score: 0.9855

In statistical analysis of binary classification, the F1 score (also F-score or F-measure) is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results

returned by the classifier, and r is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

Justification

In Approximate Distance Classification, Drs.
Adam H. Cannon, Lenore J. Cowen and Carey E. Priebe reported the following results:

" The best among these was 5 nearest neighbors, and it achieved only 93.1% classification rates.
Our best reported result is 95.6% compared to a reported best known result of 97.5%" -
Approximate Distance Classification, pg.4

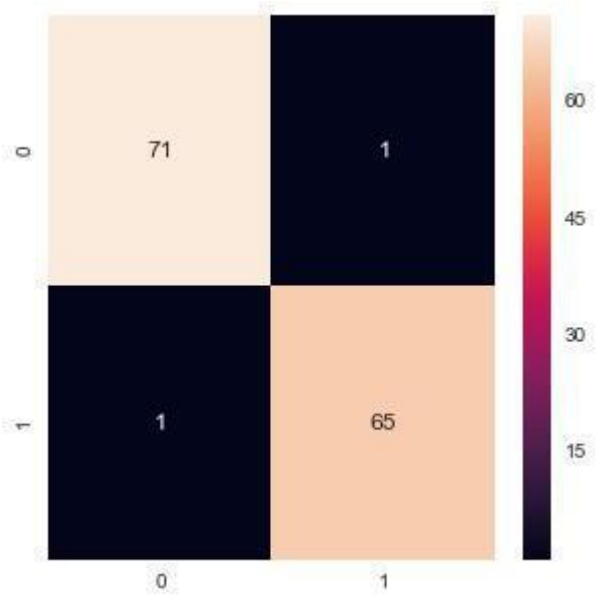
Our best result was 99.30% using F1 scoring method to score a Extra Decision Tree Algorithm and balanced data using Random method. And using "voting system" cited before I got a F1 score of 98.55%.

Our results were significant more accurate that our benchmark model and it prove the power of modern machine learning algorithms against similar methods that were used in Approximate Distance Classification in 1998, october.

V. Conclusion

Free-Form Visualization

In our final result using a "voting system" the model predicted wrongly "only" two cases as we can see in **confusion matrix** bellow:



In disease classification is very important have **low false negative** occurrences because if a patience have a disease and a model predict that they doesn't, the patient will not receive treatment and this may worsen the clinical picture.

- 71 data points was correctly predicted as Benign tumors
- 65 data points was correctly predicted as Malignant tumors
- 01 data point was wrongly predicted as Malignant tumor
- 01 data point was wrongly predicted as Benign tumor

The highest and more consistent results across all classifiers was achieved using Scale, Removing Outliers and Balancing data.

Reflection

In this project I used many data manipulation techniques as feature scaling, remove outliers and balance data creating new samples. Sometimes machine learning is a black box like, is difficult understand the optimal algorithm, parameters, features and process data methods to get the best results. To work around this issue I created a function that test many parameters for algorithm I chose, and using that function I tested different data manipulation techniques, that approach allow me running over several scenarios and found some very efficients configurations.

I expected get scores similar as my benchmark model, but the final model surprisingly surpass the bests results of benchmark model. It show how powerful modern machine learning algorithms are, and how they can be used to turn better our lives improving diagnosis results with reliable predictions.

The Wisconsin Breast Cancer dataset has 30 features and is difficult realize what is and what isn't important to correctly classify a tumor cell. Even more difficult is chosen a algorithm and parameters to get optimal results. I decided use tune methods to support me in hard task of test many parameters combination and create functions to help cover many data manipulation techniques analyzing different scenarios that give us exceptional results.

Improvement

The approach I adopted in this project were try several different algorithm, parameters, methods and combinations to find the best configuration.

I believe that combining new data manipulation techniques, more tuning parameters and algorithms best results can be achieved. But this can increase exponentially the number of train sets which may cause time consumption issues.