# Evaluation of SVM, K-NN and Decision Tree with Random Forest in Breast Cancer Diagnosis

Jack Crum  Liuqing Yang  Sixian Zhu  Biying Zhu

## Introduction

Cancer has become one of the leading causes of mortality around the world and its diagnosis is an important research topic in medical domain. According to The World Health Organization, there were around 14 million new cases and 8.2 million death related to cancer worldwide in 2012. Breast cancer is found to be the fifth major cause of cancer death and second leading cause of death among women. National Breast Cancer Foundation estimates that over 252,710 women in the United States will be diagnosed with breast cancer each year, and more than 40,500 will die.

Early detection and diagnosis are the most effective ways to prevent death from breast cancer. However, one of the major challenges in medical community is to extract meaningful and effective information from diagnosis data. Machine Learning is an adaptive process that obtains meaningful patterns from data and makes predictions based on available information, therefore it can be an effective classification and recognition tool in breast cancer diagnosis. This project evaluates three machine learning classification models for their accuracy in medical diagnosis, the goal is to find the best model that suits clinical breast cancer classification and reduce the possibility of human error in diagnosis.

## Data

Wisconsin Breast Cancer Database created by Dr. William H. Wolberg, University of Wisconsin, is used to study the performance of classification algorithm. The dataset is extracted from UCI online machine-learning repository. The Wisconsin Breast Cancer Database (WBCD) contains 699 instances in total, 30 of them holds missing values and the rest of them are taken for use in this study.  Each sample has a binary classification outcome, which is either malignant or benign. Except for ID number and diagnosis, each instance also consists of ten real-valued features generated from image analysis of fine needle aspirates (FNA) of breast masses to describe the cell nucleus. The ten features are:

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)

For each feature, the mean, standard error and mean of the three largest values (“worst”) are also calculated, so there are 30 features in the dataset. Figure_1 summarizes the diagnosis results by plotting the count of binary outcomes. Overall the class is distributed with 357 (62.74%) benign and 212 (37.6%) malignant. When mapping the outcome, malignant is represented by 1, and 0 is assigned to benign.

**FIgure_1 Distribution of Malignant and Benign Diagnosis**

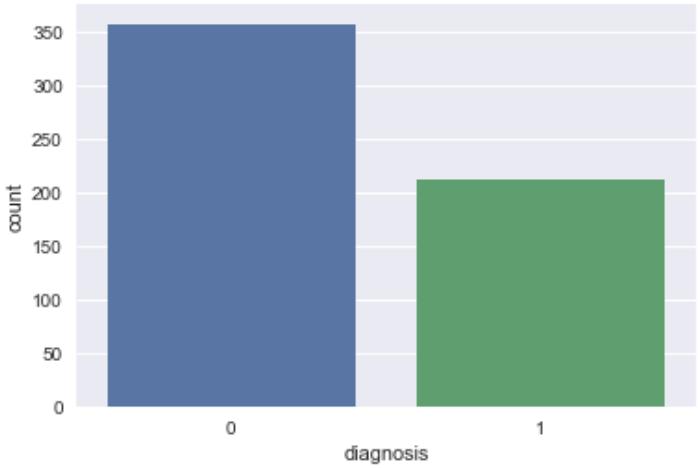
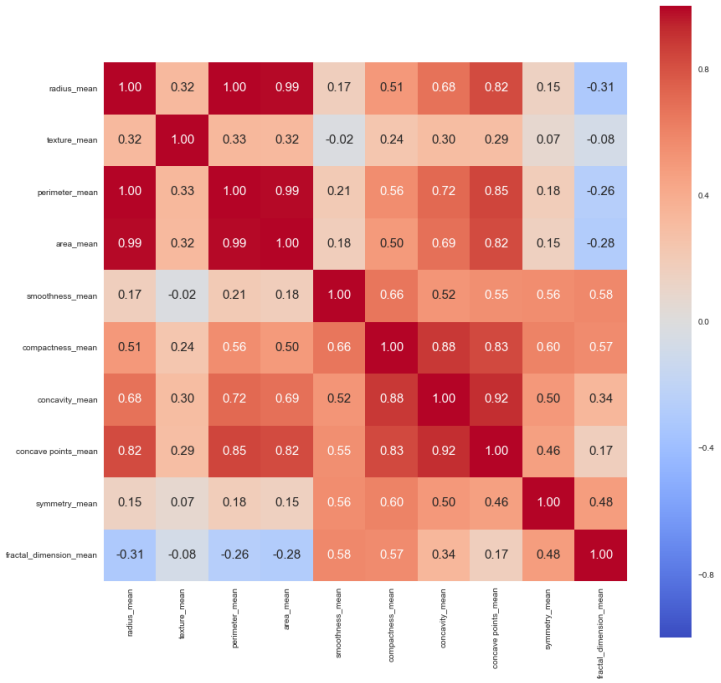



Figure_2 shows a heatmap of correlations between feature means. It is highly possible that noises are introduced due to the existence of high correlations, resulting in incorrect predictions. Samples that constantly receive wrong predictions will be discussed further in this paper. Another limitation is that database was established in 1992, therefore to improve the model in this paper more updated data may be added into the database.

**Figure_2 Correlation between Means of Features**



## Methodology

The data was split into training and testing data so that all models could be evaluated on the same testing data and therefore be comparable to each other. Both the training and testing datasets were normalized (scaling all values to between 0 and 1) to account for features on different scales and prevent them from having greater influence on the model.

The K-Nearest Neighbors algorithm is one of the most commonly used algorithm in machine learning classification. It is a non-parametric learning method based on instances. All data are assumed be be points in the N-dimensional space and K-NN allows the classification of a new point by calculating distances from all other points[1]. The proper performance of the method is influenced by the choice of k, which is the number of neighbors, and the choice of the distance function. By applying different classification rules of K-NN method we are able to analyze results from different distances and values of k, and the performance will be evaluated by accuracy rate.

Another model adopted in this study is Support Vector Machine (SVM). SVM is widely used for data classification and regression which allows the model to reduce generalization error. By applying SVM to binary classification, input vectors will be mapped onto a new higher dimensional feature space and an optimal separating hyperplane will be constructed[2].

Decision tree is a powerful learning method for breast cancer prediction and classification. It creates a classification model that map observations of nodes to conclusions about target values[3]. Leaves represent the class labels, and branches are conjunctions of features. In this study, decision tree is employed to construct a binary predictive model to diagnose whether the tumor is malignant or benign based upon feature values associated with each sample. Random Forest Classifier is used remove the inherent flaws of Decision Tree of overfitting and poor generalizing by randomly assigning features to the nodes, creating many decision trees, and taking the average results to determine which features are the most important to produce optimal breast cancer diagnosis results.

Logistic Regression examines the relationship between a binary dependent variable and explanatory variables. In this case, we adopt logistic regression only for samples that cannot be classified correctly by the three models above.

---

[1] Sarkar M, Leong TY. Application of K-nearest neighbors algorithm on breast cancer diagnosis problem. *Proceedings of the AMIA Symposium*. 2000:759-763.

[2] Huang M-W, Chen C-W, Lin W-C, Ke S-W, Tsai C-F. SVM and SVM Ensembles in Breast Cancer Prediction. Hernandez-Lemus E, ed. *PLoS ONE*. 2017;12(1):e0161501. Doi:10.1371

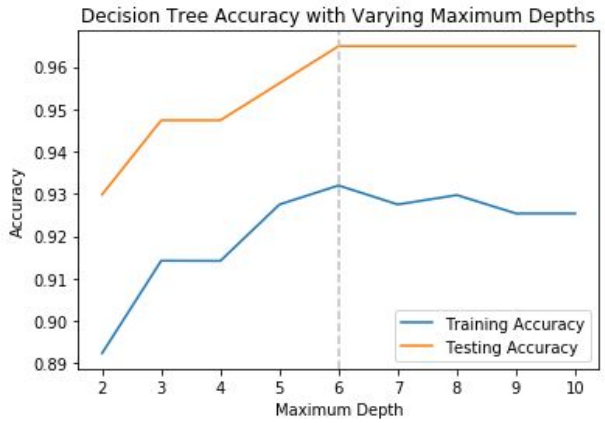
[3] Ronak Sumbaly, N Vishnusri and S Jeyalatha. Article: Diagnosis of Breast Cancer using Decision Tree Data Mining Technique. *International Journal of Computer Applications* 98(10):16-24, July 2014.

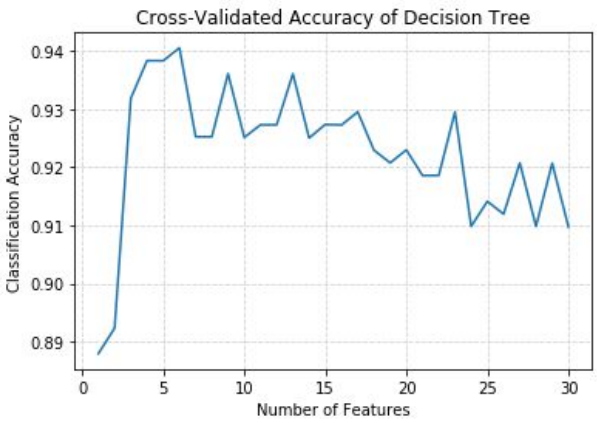
**Experimentation and Results**

Decision Tree:

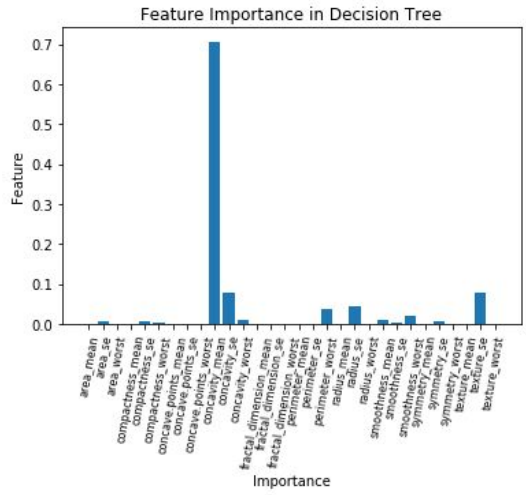
A binary Decision Tree Classifier takes data with one target value containing two classes and splits the data at each node using a series of Boolean (True or False) tests. All values that pass the test are separated from all the values that fail the test. This model was chosen as decision trees provide a comprehensive and simple means of understand how the data is analyzed and optimally split to develop good classification results.

A decision tree splits nodes by the feature that produces the maximum information gain from one level of the tree to the next. The tree starts the split on the feature that provides the greatest information gain and stops splitting when all end nodes are pure (of one class) or no more information can be gained from further division. In this tree, gini impurity is the metric used to determine information gain. Gini impurity is the measure of how often a randomly chosen observation from the dataset would be incorrectly classified if it was randomly labeled according to the class distribution of the subset. A gini impurity of 0 at each end node is optimal.




A decision tree has multiple parameters that can be tuned for optimal results. The first parameter tuned was maximum depth. As a decision tree tends to overfit on the training data and poorly generalize the testing data, it is vital to prune the tree for maximum depth and number of features. The training and testing accuracies were calculated and plotted at each maximum depth from 2 to 10. The graph shows that 6 is the optimal maximum depth for accuracy. Second, a 5-fold cross validation of classification accuracy was plotted for each possible number of features, 1 to 30. The graph shows optimal testing accuracy is found with a decision tree containing 6 features. The decision tree is pruned to contain 6 features and have a maximum depth of 6.



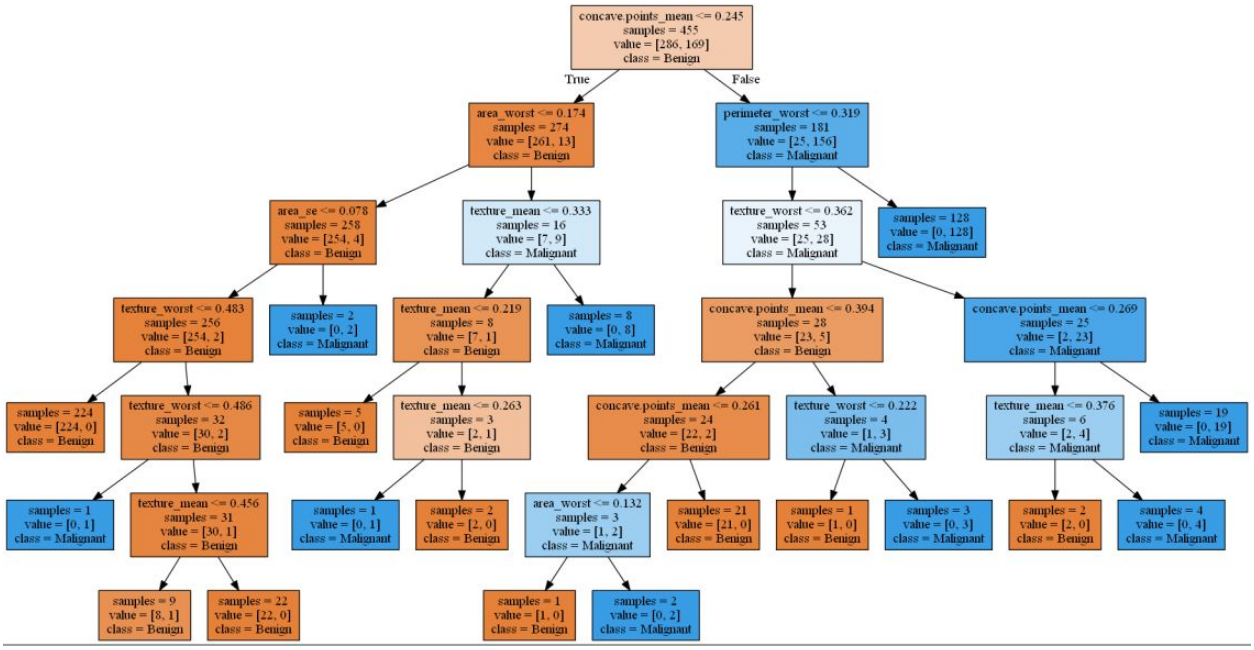

| | Predicted Benign | Predicted Malignant |
|---|---|---|
| True Benign | 69 | 2 |
| True Malignant | 4 | 39 |

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Benign | 0.95 | 0.97 | 0.96 | 71 |
| Malignant | 0.95 | 0.91 | 0.93 | 43 |
| avg / total | 0.95 | 0.95 | 0.95 | 114 |

Accuracy: 0.940583426236

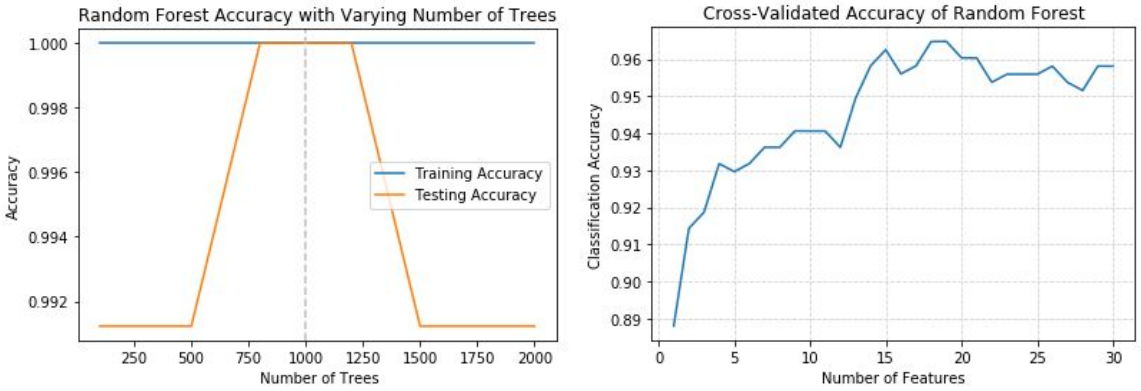
To determine which features to use and reduce dimensionality, feature importance is calculated, sorted, and the top 6 are selected. In this case, the 6 features chosen are concave.points_mean, perimeter_worst, texture_worst, area_worst, texture_mean, and area_se, with concave.points_mean having significantly more importance than any other feature. The training set and testing sets are subsetted to only include these features and the model is fit again. A confusion matrix and other metrics are calculated to show the predictive capabilities of the decision tree model. Precision measures the ratio of correct positive predictions to all positive predictions and is calculated by  True Positive / (True Positive + False Positive) as shown in the confusion matrix. Recall/Sensitivity measures how often positive values are predicted correctly and is calculated by True Positive / (True Positive + False Negative). F1 score is the harmonic mean of recall and precision. The optimized decision tree scores decently in both measurements as well as classification accuracy, where it scores 0.94058. However, the decision tree scores a 0.91 for malignant sensitivity, which means it only correctly classified 39/43 true malignant cases as malignant. While in many industries a 0.91 true positive rate would be adequate, in cancer detection, a high sensitivity is necessary for catching as many cases as possible.

The optimized decision tree is visualized with the split, total values, and class ratios in each node. As can be seen, all end nodes are pure except for one which only contains one incorrect value and is color coded by class (orange = “Benign”, blue = “Malignant”). Although there is one strong benign end node with 224 benign values and 0 malignant values, one strong malignant end node with 128 malignant values and 0 benign values, all other end nodes consist of very few total values, implying the model may overfit the data.



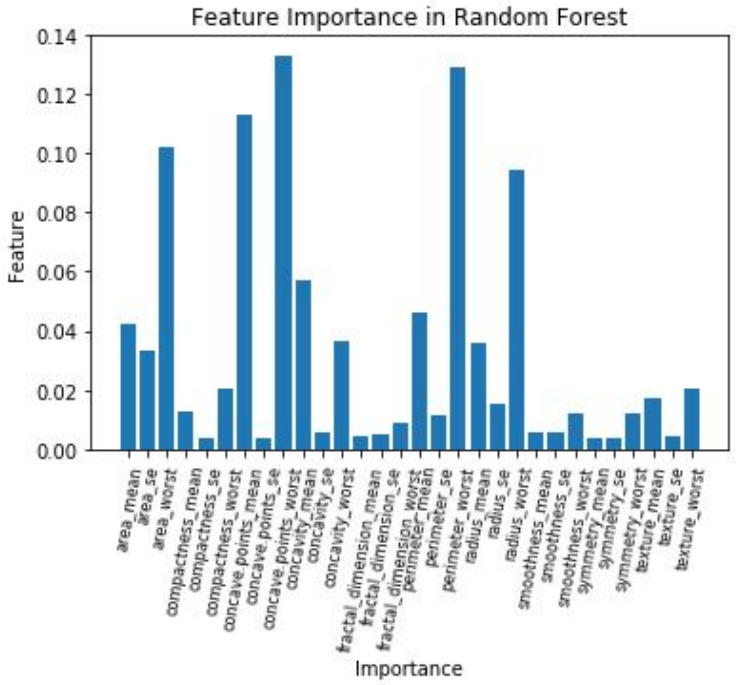

Random Forest:

Random forests are aggregations of several decision tree models. A single decision tree is very fast but not too accurate. The earlier comparisons between the decision tree and a simple linear model illustrated this. While random forests take longer to run, they are very accurate and robust to overfitting. Random forests improve accuracy by fitting many trees to a bootstrap sample of the original dataset. This is a technique called “bootstrap aggregation” or “bagging”. In addition to making trees by bagging, random forests change how the classification or regression trees are constructed. In standard trees, each node is split using the split with the most information gain among all the features. In a random forest, each node is split using the best among a subset of features randomly chosen at that node.




First, we determine the number of trees in the forest and the depth of each tree that optimizes accuracy in the testing data. It is necessary to check training and testing accuracy to prevent

overfitting on the training set and poor performance on the testing set. As shown by the graphs above, the optimal number of trees in the forest is 1000 and the optimal maximum depth is 15. Though 18 features provides a slight gain in accuracy, it is not a large enough gain to overcomplicate the model. The parameters of the optimal random forest are set.



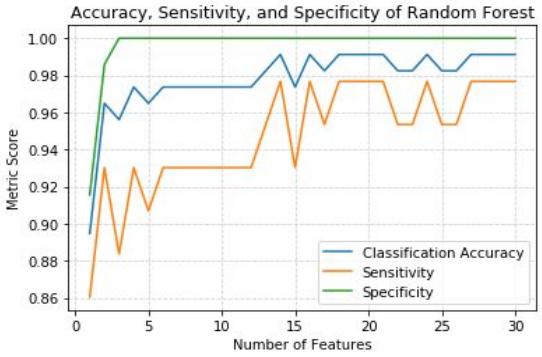

| feature | importance |
|---|---|
| concave.points_worst | 0.133067 |
| perimeter_worst | 0.129365 |
| concave.points_mean | 0.112895 |
| area_worst | 0.101934 |
| radius_worst | 0.094653 |
| concavity_mean | 0.056994 |
| perimeter_mean | 0.046408 |
| area_mean | 0.042356 |
| concavity_worst | 0.036295 |
| radius_mean | 0.035819 |
| area_se | 0.033368 |
| compactness_worst | 0.020715 |
| texture_worst | 0.020187 |
| texture_mean | 0.017271 |
| radius_se | 0.015584 |
| compactness_mean | 0.012886 |
| symmetry_worst | 0.012159 |
| smoothness_worst | 0.012044 |
| perimeter_se | 0.011193 |
| fractal_dimension_worst | 0.008931 |
| smoothness_se | 0.005797 |
| smoothness_mean | 0.005719 |
| concavity_se | 0.005467 |
| fractal_dimension_se | 0.005031 |
| texture_se | 0.004387 |
| fractal_dimension_mean | 0.004205 |
| symmetry_se | 0.003925 |
| compactness_se | 0.003812 |
| symmetry_mean | 0.003771 |
| concave.points_se | 0.003763 |

With the Random Forest classifier from sklearn, we can measure feature importance as the averaged gini impurity decrease computed from all 1000 decision trees in the forest. As seen on the graph of feature importance in the random forest, many more features have an importance in the model than in the decision tree. The top features include concave.points_worst, perimeter_worst, concave.points_mean, area_worst, and radius_worst, most of which had no importance on the decision tree. As the optimal maximum depth was determined to be 15, the top 15 important features were selected for the model.

| | Predicted Benign | Predicted Malignant |
|---|---|---|
| True Benign | 71 | 0 |
| True Malignant | 3 | 40 |

Accuracy: 0.973684210526

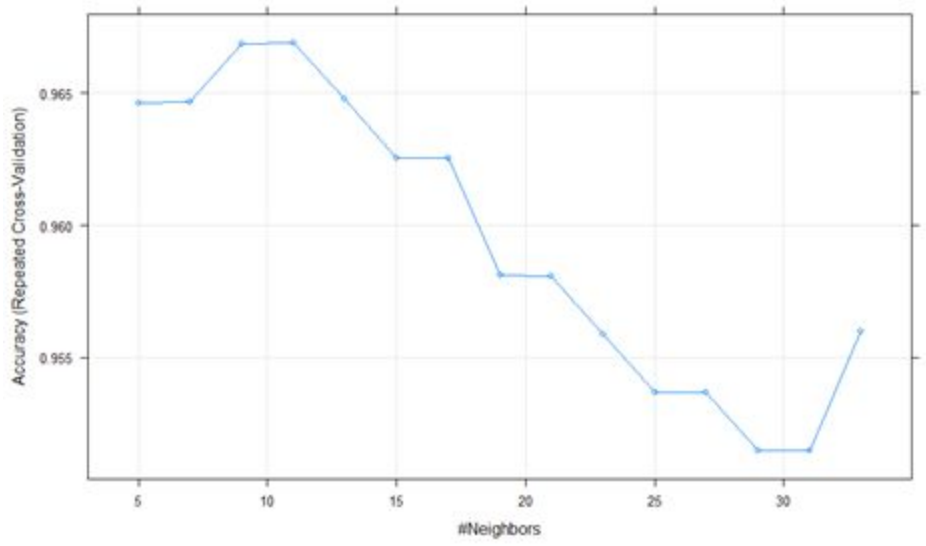
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Benign | 0.96 | 1.00 | 0.98 | 71 |
| Malignant | 1.00 | 0.93 | 0.96 | 43 |
| avg / total | 0.97 | 0.97 | 0.97 | 114 |




The metrics of the Random Forest model are computed with a confusion matrix. The model misclassified 3 false negative results (true malignant, predicted benign). While an accuracy rate of 0.9737 is very high, the recall/sensitivity on malignancy is 0.93, slightly higher than the decision tree. This means that the model predicted 40/43 cases of true malignancy correctly. As stated above, while in many industries a 0.93 true positive rate would be adequate, in our specific problem of cancer detection, a high sensitivity is necessary for catching as many cases as possible. As the graph above shows, sensitivity is an issue with the random forest model even when more features are included. Overall, the random forest model provides a very robust model with high accuracy and limited effects from overfitting though some issues remain.

KNN:

Considering the many potential structures of our data, we also used the lazy-learning K-nearest-neighbor classification. Since the scale of each feature varies much, we first normalized the data and then rely on 10-fold cross validation to tune the parameter k, number of nearest observations that will be considered. After conduct 10-fold cross validation on the train data (all features are included), we get the following plot:



Clearly when k=11 get the highest accuracy, 0.967 and when k becomes larger the accuracy becomes worse. Then we applied the trained model to the test data, the prediction accuracy is 0.9825, even higher than the averaged accuracy in the training and the corresponding confusion matrix:

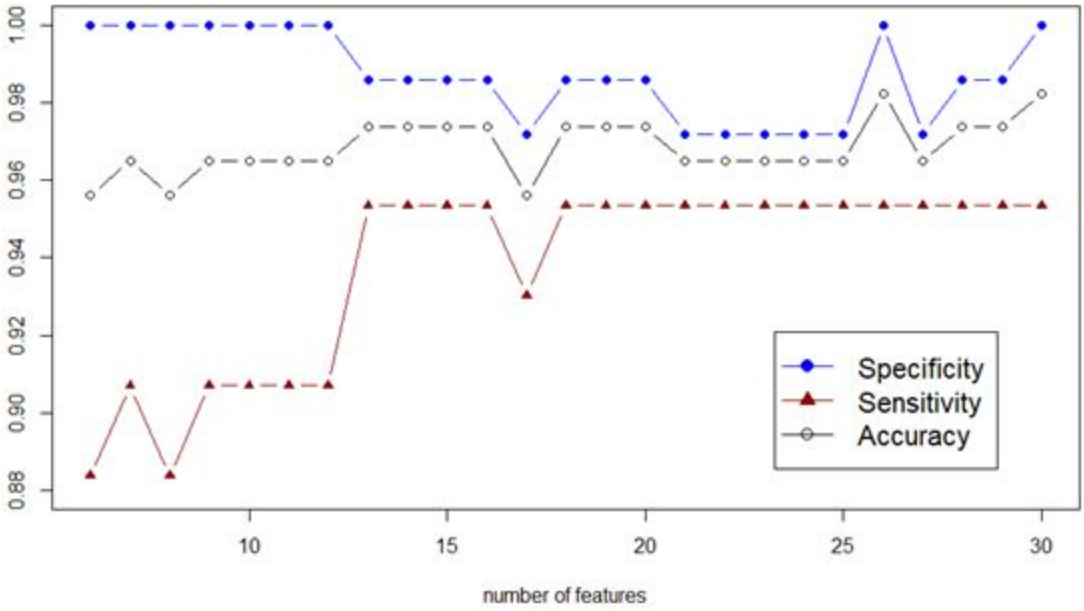
| | **Predicted Malignant** | **Predicted Benign** |
|---|---|---|
| **True Malignant** | **41** | **2** |
| **True Benign** | **0** | **71** |

Luckily, all Benign patients are correctly predicted but with 2 Malignant patients wrongly predicted as Benign, they are ID: 889403 and 91594602. Since we used all the feature in the previous model, we start thinking about if we leave some features out, will the prediction become even better?

To do this, we first weight the importance of each feature by using Random Forest, then conduct the following algorithm:

1. Sort the features by their importance as a vector F.
2. Leave the first feature (most unimportant one) out for both training and test data.
3. Conduct 10-fold cross validation to tune the k and rely on the resulting value to do the prediction on testing data.
4. Calculate and record the Accuracy, sensitivity (proportion of correctly predicted as Malignant), specificity (proportion of correctly predicted as Benign).
5. Let the leave one out F to become the new F.
6. Check if the length of F >1 then back to step1, otherwise stop.

Through above algorithm we get the plot:




Sensitivity, the capability of our knn model to correctly capture breast cancer, does not decrease with the number of feature until we leave 13 features out. Sensitivity, in this case, is most importance because if a malignant patient is predicted to be Benign, he or she may not going to do biopsy and thus miss the best time window for cancer treatment. The plot tells us, no matter what, there always two malignant patients cannot be found out, patients 889403 and patient 91594602.If talking about the overall performance, when leaving out four least important features which are “compactness_se” “fractal_dimension_mean” “symmetry_mean” and “symmetry_se”, all the three evaluations are the same with the full model. Thus those four features together may not provide useful information for our prediction under KNN.

SVM:

In this data set, the response feature, diagnosis has two values of B and M. The prediction of the response can be considered as the classification of a new data point.

Support vector machine (SVM) can be considered as an extension of the perceptron. For perceptron, all the independent variables are the input **P** of the perceptron, computed by specific weight **W** and bias **b**, we get the net-output (**WP + b**), and afterwards, the hard-limit function f get f (**WP + b**) as the final prediction. The perceptron algorithm minimizes the misclassification error. However, in the SVM, our optimization objective is to maximize the margin, the distance between the separating hyperplane (decision boundary) and the training samples that are closest to this hyperplane. By maximizing the margin in SVM, we fit a model generalizes well according to the training data set.

To do this, we:

1. Standardize the data using StandardScaler from sklearn.processing;
2. Split the data into train test and validation sets;
3. Do model selection for different C (penalty parameter) values using train for fitting and validation for evaluating;
4. Fit the combined data set of train and validation with the selected model, evaluate it with test set;

The accuracy of the model with specific C value in validation set:

| **C** | **0.1** | **0.2** | **0.3** | **0.4** | **0.5** |
|---|---|---|---|---|---|
| **Accuracy** | 0.9708029 | 0.9635036 | 0.9708029 | 0.9708029 | 0.9708029 |
| **C** | **0.6** | **0.7** | **0.8** | **0.9** | **1.0** |
| **Accuracy** | 0.9781021 | 0.9635036 | 0.9635036 | 0.9635036 | 0.9635036 |

As a result, we select the C value of 0.6.

The accuracy (right frequency/all frequency) of the selected model in the test set is 0.973684210526.

The confusion matrix is

| | **Predicted Malignant** | **Predicted Benign** |
|---|---|---|
| **True Malignant** | **41** | **2** |
| **True Benign** | **1** | **70** |

The observations in test set predicted wrong are shown below:

| **id** | **diagnosis** | **prediction** |
|---|---|---|
| 889403 | M | B |
| 91594602 | M | B |
| 921386 | B | M |

Logistic Regression

As this paper mentioned previously, there are several samples that cannot be diagnosed correctly all the time due to noises. ID numbers of these samples are 889403, 91594602, 86603, 921386. Individual predictions are made on these samples by logistic regression. Please find the attached code for more details. Results of predictions for these specific samples are shown in the table below.

**Table_1 Individual Logistic Regression Prediction for Specific Samples**

| ID | Prediction Result |
|---|---|
| 889403 | 0.1239494 |
| 91594602 | 0.3499236 |
| 866083 | 0.3816406 |
| 921386 | 0.8701244 |

## Conclusion

In this paper, performances of K-NN, SVM and Decision Tree with Random Forest were evaluated for breast cancer diagnosis using Wisconsin Breast Cancer Database. Overall significant prediction results were achieved, K-NN outperforms Decision Tree, Random Forest, and SVM in terms of accuracy rate and had the highest sensitivity along with SVM. However, malignant sensitivity remains an issue in all models and would require further tuning and testing for optimal results.

**Table_2 Performance In terms of Classification Accuracy Rate**

| | Decision Tree | Random Forest | K-NN | SVM |
|---|---|---|---|---|
| Accuracy Rate | 0.940583426236 | 0.973684210526 | 0.9825 | 0.973684210526 |
| Sensitivity/Recall | 0.91 | 0.93 | 0.9534883721 | 0.9534883721 |

Since Wisconsin Breast Cancer Dataset was created in 1992, adding more updated samples into the dataset for conducting further study with K-NN and Decision Tree may produce better accuracy in modern breast cancer diagnosis.

## References

**Sarkar M, Leong TY**. Application of K-nearest neighbors algorithm on breast cancer diagnosis problem. *Proceedings of the AMIA Symposium*. 2000:759-763.
**Huang M-W, Chen C-W, Lin W-C, Ke S-W, Tsai C-F**. SVM and SVM Ensembles in Breast Cancer Prediction. Hernandez-Lemus E, ed. *PLoS ONE*. 2017;12(1):e0161501. Doi:10.1371
**Ronak Sumbaly, N Vishnusri and S Jeyalatha.** Article: Diagnosis of Breast Cancer using Decision Tree Data Mining Technique. *International Journal of Computer Applications* 98(10):16-24, July 2014.

## Attachment

Code for K-NN: :

```
#data preprocessing
###author: Liuqing Yang
tt=read.csv('data.csv',header=T)
tt=tt[,-33]
length(which(is.na(tt)=='T'))#check if there is missing value
#> length(which(is.na(tt)=='T'))
#[1] 0
#there is no missing value

r1=length(which(tt$diagnosis=='M'))/length(tt$diagnosis)#ratio of malignant
r2=1-r1 #ratio of benign

#normalize data
nmlz=function(x){
  return((x-min(x))/(max(x)-min(x)))
}

tt_nmlz=apply(tt[,-c(1,2)],2,nmlz)
tt_nmlz=cbind.data.frame(tt[,c(1,2)],tt_nmlz)

write.csv(tt_nmlz,file = "data_nmlz.csv",row.names = F)#extract normalized data out.

#standardlize data
stdz=function(x){
  return((x-mean(x))/sqrt(var(x)))
}

tt_stdz=apply(tt[,-c(1,2)],2,stdz)
tt_stdz=cbind.data.frame(tt[,c(1,2)],tt_stdz)
write.csv(tt_stdz,file = "data_stdz.csv",row.names = F)#extract standardlized data out.

#sepereate data for training and testing
n=length(tt[,1])
n_train=floor(n*0.8) #let 80% in the original data for training.
n_test=n-n_train
set.seed(1234)
indx=sample(seq(1,n),n_train)
train=tt[indx,]
test=tt[-indx,]
train_nmlz=tt_nmlz[indx,]
test_nmlz=tt_nmlz[-indx,]
train_stdz=tt_stdz[indx,]
test_stdz=tt_stdz[-indx,]
write.csv(train,file="train.csv",row.names=F)
write.csv(test,file="test.csv",row.names = F)
```

```
write.csv(train_nmlz,file="train_nmlz.csv",row.names=F)
write.csv(test_nmlz,file="test_nmlz.csv",row.names = F)
KNN:
library(randomForest)
train=read.csv("train_nmlz.csv")
#importance for features
fit=randomForest(factor(diagnosis)~., data=train[,-1])
varimpt=as.data.frame(importance(fit))
varimpt=cbind(names = rownames(varimpt), varimpt)
x=varimpt[order(varimpt$MeanDecreaseGini,decreasing = T),]
barplot(as.vector(x[1:30,2]),names.arg=1:30,main="Importances for each feature (after sorting)"
       ,ylab="MeanDecreaseGini")
x[,1]# the features sorted by importance.
y=as.character(x[,1])
range=match(y,colnames(train[,-c(1,2)]))
train_s=train[,c(2,range+2)]

library(caret)
library(e1071)
test=read.csv("test_nmlz.csv")
test_s=test[,c(2,range+2)]
#setting 10-fold cross validation
ctrl <- trainControl(method="repeatedcv", number=10, repeats=1)
#leave features out algorithm
accuracy=rep(NA,25)
sensitivity=rep(NA,25)
specificity=rep(NA,25)
for(i in 30:6){
set.seed(12345)
knnFit1 <- train(diagnosis ~ ., data=train_s[,1:i], method="knn",
              trControl=ctrl, metric="Accuracy", tuneLength=15,
              preProc=c("range"))
knnFit1
plot(knnFit1)

knnPredict1 <- predict(knnFit1, newdata=test_s[,1:i])
cmat1= confusionMatrix(knnPredict1, factor(test_s$diagnosis), positive="M")
cmat1

accuracy[31-i]=cmat1$overall[1]
sensitivity[31-i]=cmat1$byClass[1]
specificity[31-i]=cmat1$byClass[2]
}

#plot accuracy, sensitivity, specifity
plot(30:6,sensitivity,ylim=c(0.88,1),xlab="number of features",type="b",
    col="dark red",pch=17,ylab=" ")
lines(30:6,specificity,xlab="number of features",type="b",pch=16,col="blue")
```

```
lines(30:6,accuracy,xlab="number of features",type="b",pch=1,col="black")
legend(cex=1.4,"bottomright",inset=0.08,c("Specificity","Sensitivity","Accuracy"),pch=c(16,17,1),
       lty=c(1,1,1),col=c("blue","dark red","black"))

#print out IDs that failed to correctly predicted as Malignant
failM=which(knnPredict1!=as.factor(test$diagnosis))
test$id[failM]
```

Code for SVM:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

###################################
######   data processing    ######
###################################
#import data
cancer_train = pd.read_csv('C:/Users/Admin/Desktop/Fall 2017 GWU Courses/Machine Learning/Final
Project/train.csv')
cancer_test = pd.read_csv('C:/Users/Admin/Desktop/Fall 2017 GWU Courses/Machine Learning/Final
Project/test.csv')
diag_mapping = {'B': 0,'M': 1}
cancer_train['diagnosis'] = cancer_train['diagnosis'].map(diag_mapping)
cancer_test['diagnosis'] = cancer_test['diagnosis'].map(diag_mapping)
#setting train set and test set
cancer_train_data= cancer_train.drop('diagnosis', 1)
cancer_train_data= cancer_train_data.drop('id', 1)
cancer_test_data= cancer_test.drop('diagnosis', 1)
cancer_test_data= cancer_test_data.drop('id', 1)
y_train = cancer_train['diagnosis']
y_test  = cancer_test['diagnosis']
X_train = cancer_train_data
X_test  = cancer_test_data


#############################
#####       SVM     ########
#############################
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
```

```
svm = SVC(kernel='linear', C=1.0, random_state=0)
svm.fit(X_train_std, y_train)


#Model Evaluation
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
y_predict = svm.predict(X_test_std)
acc = accuracy_score(y_test, y_predict)
conf_forest = pd.DataFrame(confusion_matrix(y_test, y_predict),
            columns=['Predicted Benign', 'Predicted Malignant'],
            index=['True Benign', 'True Malignant'])
print(conf_forest, "\n")
print(acc)


#Detecting Wrong ID
id = pd.DataFrame(cancer_test['id'])
y_test = pd.DataFrame(y_test)
y_predict = pd.DataFrame(y_predict)
c_table = pd.concat([id, y_test, y_predict], axis=1)
print(c_table[c_table.iloc[:,1] != c_table.iloc[:,2]])

########################################
######   meaningful features    ######
########################################

#meaningful featrues for original
from sklearn.ensemble import RandomForestClassifier
feat_labels = cancer.columns[2:]
forest = RandomForestClassifier(random_state=1, n_jobs=-1)
forest.fit(X_train, y_train)
importances = forest.feature_importances_
indices = np.argsort(importances)[::-1]
for f in range(X_train.shape[1]):
      print("%2d) %-*s %f" % (f + 1, 30, feat_labels[f], importances[indices[f]]))


Code for Logistic Regression:
#load orginal datasets
setwd("C:/Users/Admin/Desktop/Fall 2017 GWU Courses/Machine Learning/Final Project")
getwd()
test=read.csv("test.csv", header=TRUE,sep=",")
train=read.csv("train.csv", header=TRUE,sep=",")
train$diagnosis1[train$diagnosis =='B'] <-0
train$diagnosis1[train$diagnosis =='M'] <-1
test$diagnosis1[test$diagnosis =='B'] <-0
test$diagnosis1[test$diagnosis =='M'] <-1
```

```
logitModel <- glm(diagnosis1 ~ radius_mean+texture_mean + perimeter_mean + area_mean +
                    smoothness_mean + compactness_mean + concavity_mean,
                  data=train, family=binomial(link="logit"))
new1=test[test$id==889403,]
new2=test[test$id==91594602,]
new3=test[test$id==866083,]
new4=test[test$id==921386 ,]
predict(logitModel, new1, type="response")
predict(logitModel, new2, type="response")
predict(logitModel, new3, type="response")
predict(logitModel, new4, type="response")


> predict(logitModel, new1, type="response")
       47
0.1239494
> predict(logitModel, new2, type="response")
       97
0.3499236
> predict(logitModel, new3, type="response")
       19
0.3816406
> predict(logitModel, new4, type="response")
      107
0.8701244
```

Decision Tree and Random Forest:

```
##### 1) Import Packages #####

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report as cr

##### 2) Importing Training and Testing Data #####
```

```
import os
os.chdir('C:/Users/sjcrum/Documents/GitHub/Machine-Learning-Final')

#Original training and testing data
train_original = pd.read_csv("train.csv")
test_original = pd.read_csv("test.csv")
#Normalized training and testing data
train = pd.read_csv("train_nmlz.csv")
test = pd.read_csv("test_nmlz.csv")
#selecting feature columns for training and testing and assigning target column
train_feature_cols = train.iloc[:, 2:]
X_train = train_feature_cols
y_train = train.diagnosis
test_feature_cols = test.iloc[:, 2:]
X_test = test_feature_cols
y_test = test.diagnosis
#Mapping B to 0 and M to 1 in in diagnosis (target) column
y_train = y_train.map({'B':0, 'M':1})
y_test = y_test.map({'B':0, 'M':1})
y_train = pd.to_numeric(y_train)
y_test = pd.to_numeric(y_test)


##### 3)Create Decision Tree ######

# Assign variables as empty lists
train_accuracy = []
test_accuracy = []
maxdepth = []
#Test Decision Tree model with different maximum depths
for n in range(2,11):
        model = DecisionTreeClassifier(max_depth = n, random_state = 0)
        model.fit(X_train, y_train)
        #10-fold cross validation for the Decision Tree accuracy on the training data
        train_acc = np.mean(cross_val_score(model, X_train, y_train, cv=10))
        test_acc = model.score(X_test, y_test)

        #append data to above lists
        train_accuracy.append(train_acc)
        test_accuracy.append(test_acc)
        maxdepth.append(n)

#Plot training and testing accuracy vs accuracy
plt.plot(maxdepth,train_accuracy, label = "Training Accuracy")
plt.plot(maxdepth,test_accuracy, label = "Testing Accuracy")
plt.title("Decision Tree Accuracy with Varying Maximum Depths")
plt.xlabel("Maximum Depth")
plt.ylabel("Accuracy")
```

```
#Vertical line to show maximum accuracy
plt.axvline(x=6, color = "silver", linestyle='dashed')
plt.legend()
plt.show()
dtree = DecisionTreeClassifier(criterion='gini', max_depth=6, random_state = 0)

#Function that plots feature importance
def plot_features(model, model_name):

        model.fit(X_train, y_train)

        #Feature Importance
        features = pd.DataFrame({'feature':X_train.columns.values,
'importance':model.feature_importances_})
        features_sorted = features.sort_values(by = ['importance'], ascending = False)
        print(features_sorted)

        #Plotting feature importance
        plt.bar(features_sorted.feature, features.importance)
        plt.title("Feature Importance in {}".format(model_name))
        plt.xticks(fontsize =  8, rotation = 80)
        plt.xlabel("Importance")
        plt.ylabel("Feature")
        plt.show()

plot_features(dtree, "Decision Tree")
#set max depth to optimal depth
dtree = DecisionTreeClassifier(criterion='gini', max_depth=6, random_state = 0)
#Function that plots classication accuracy, sensitivity, and specificity
# and plots 5-fold cross-validated accuracy
def plot_feature_length(model, model_name):

        total_features = []
        accuracy = []
        sense = []
        spec = []
        cv = []

        for n in range(1, 31):

        model.fit(X_train, y_train)

        features = pd.DataFrame({'feature':X_train.columns.values,
'importance':model.feature_importances_})
        features_sorted = features.sort_values(by = ['importance'], ascending = False)

        important_features = features_sorted['feature'].head(n)
        X_train_feat = X_train.loc[:, important_features]
```

```
X_test_feat = X_test.loc[:, important_features]

#Fitting the smaller model
model.fit(X_train_feat, y_train)

#Predict target values
y_predict = model.predict(X_test_feat)

#Accuracy
acc_small = accuracy_score(y_test, y_predict)
accuracy.append(acc_small)
total_features.append(n)

#Confusion Table
conf = pd.DataFrame(confusion_matrix(y_test, y_predict),
columns=['Predicted Benign', 'Predicted Malignant'],
index=['True Benign', 'True Malignant'])

TP = conf.iloc[1,1]
TN = conf.iloc[0,0]
FP = conf.iloc[0,1]
FN = conf.iloc[1,0]

#Sensitivity calculation
sensitivity = recall_score(y_test, y_predict)
#Specificity calculation
specificity = TN / (TN + FP)

sense.append(sensitivity)
spec.append(specificity)
#Cross Validation Score
cv_acc = np.mean(cross_val_score(model, X_train_feat, y_train, cv = 5))
cv.append(cv_acc)

plt.plot(total_features, accuracy, label = "Classification Accuracy")
plt.plot(total_features, sense, label = "Sensitivity")
plt.plot(total_features, spec, label = "Specificity")
plt.grid(True, linestyle='dashed', c = "lightgrey")
plt.title("Accuracy, Sensitivity, and Specificity of {}".format(model_name))
plt.xlabel("Number of Features")
plt.ylabel("Metric Score")
plt.legend()
plt.show()

plt.plot(total_features, cv)
plt.grid(True, linestyle='dashed', c = "lightgrey")
plt.title("Cross-Validated Accuracy of {}".format(model_name))
plt.xlabel("Number of Features")
```

```
        plt.ylabel("Classification Accuracy")
        plt.show()
plot_feature_length(dtree, "Decision Tree")

#Function to get optimal number of features for the decision tree model
def optimal_features_scores(model, n):

        model.fit(X_train, y_train)

        #Get top n features
        features = pd.DataFrame({'feature':X_train.columns.values,
'importance':model.feature_importances_})
        features_sorted = features.sort_values(by = ['importance'], ascending = False)

        #Dataset with only top n features
        important_features = features_sorted['feature'].head(n)
        X_train_feat = X_train.loc[:, important_features]
        X_test_feat = X_test.loc[:, important_features]

        model.fit(X_train_feat, y_train)

        y_predict = model.predict(X_test_feat)

        acc = accuracy_score(y_test, y_predict)
        conf_tree = pd.DataFrame(confusion_matrix(y_test, y_predict),
        columns=['Predicted Benign', 'Predicted Malignant'],
        index=['True Benign', 'True Malignant'])
        print(conf_tree, "\n")
        print("Accuracy: ", acc)
        print("\n")
        #Other metrics to show model quality
        target_names = ['Benign', 'Malignant']
        print(cr(y_test, y_predict, target_names=target_names))

optimal_features_scores(dtree, 6)
#Making the optimal Decision Tree
dtree = DecisionTreeClassifier(criterion='gini', max_depth=6, random_state = 0)
dtree.fit(X_train, y_train)
features_dec = pd.DataFrame({'feature':X_train.columns.values,
'importance':dtree.feature_importances_})
features_sorted_dec = features_dec.sort_values(by = ['importance'], ascending = False)
important_features_dec = features_sorted_dec['feature'].head(6)
X_train_dec = X_train.loc[:, important_features_dec]
X_test_dec = X_test.loc[:, important_features_dec]
dtree_opt = DecisionTreeClassifier(criterion='gini', max_depth=6, random_state = 0)
dtree_opt.fit(X_train_dec, y_train)
print("Accuracy: ", np.mean(cross_val_score(dtree_opt, X_train_dec, y_train, cv = 5)))
```

```
#Export visualization of decision tree
from sklearn.tree import export_graphviz
export_graphviz(dtree_opt,out_file="opttree.dot",class_names=['Benign',"Malignant"],
        feature_names=X_train_dec.columns.values,impurity=False,filled=True)


##### Random Forest Model ######

#Assign feature labels
feat_labels = X_train.columns
#Assign variables to empty lists
train_forest_accuracy = []
test_forest_accuracy = []
maxforestdepth = []
#Testing Random Forests with varying max_depths
for n in range(1,31):
        forest = RandomForestClassifier(criterion='gini', max_depth=n, n_estimators = 1000,
random_state = 0)
        #fit the random forest to the training data
        forest.fit(X_train, y_train)

        #Accuracy scores of Random Forest for training and testing
        #Traded computing speed for lack of cross validation
        train_forest_acc = forest.score(X_train, y_train)
        test_forest_acc = forest.score(X_test, y_test)
        #Append to above lists
        train_forest_accuracy.append(train_forest_acc)
        test_forest_accuracy.append(test_forest_acc)
        maxforestdepth.append(n)

#Plot testing and training accuracy vs Maximum Depth
plt.plot(maxforestdepth,train_forest_accuracy, label = "Training Accuracy")
plt.plot(maxforestdepth,test_forest_accuracy, label = "Testing Accuracy")
plt.title("Random Forest Accuracy with Varying Maximum Depths")
plt.xlabel("Maximum Depth")
plt.ylabel("Accuracy")
plt.axvline(x=8, color = "silver", linestyle='dashed')
plt.legend()
plt.show()
#Assign variables to empty lists
train_forest_accuracy_est = []
test_forest_accuracy_est = []
forest_est = []
#Forest size estimates
est = [100,500,800, 900, 1000, 1100, 1200, 1500, 2000]
for n in est:
        forest = RandomForestClassifier(criterion='gini', max_depth=8, n_estimators = n, random_state =
0)
```

```
        #fit the random forest to the training data
        forest.fit(X_train, y_train)

        #Accuracy scores of Random Forest for training and testing
        #Traded computing speed for lack of cross validation
        train_forest_acc = forest.score(X_train, y_train)
        test_forest_acc = forest.score(X_test, y_test)
        #Append to above lists
        train_forest_accuracy_est.append(train_forest_acc)
        test_forest_accuracy_est.append(test_forest_acc)
        forest_est.append(n)

#Plot testing and training accuracy vs Number of Trees
plt.plot(forest_est,train_forest_accuracy_est, label = "Training Accuracy")
plt.plot(forest_est,test_forest_accuracy_est, label = "Testing Accuracy")
plt.title("Random Forest Accuracy with Varying Number of Trees")
plt.xlabel(" Number of Trees")
plt.ylabel("Accuracy")
plt.axvline(x=1000, color = "silver", linestyle='dashed')
plt.legend()
plt.show()

#Fit the Random Forest model with all features with best max depth and number of trees
rf =  RandomForestClassifier(criterion='gini', max_depth=8, n_estimators = 1000, random_state = 0)
rf.fit(X_train, y_train)
#Plot best features
plot_features(rf, "Random Forest")

#Predict target values
y_predict_rf = rf.predict(X_test)
#User defined function above
plot_feature_length(rf, "Random Forest")
#User defined function above
optimal_features_scores(rf, 15)

#Making the optimal Random Forest
rf =  RandomForestClassifier(criterion='gini', max_depth=8, n_estimators = 1000, random_state = 0)
rf.fit(X_train, y_train)
features_rf = pd.DataFrame({'feature':X_train.columns.values, 'importance':rf.feature_importances_})
features_sorted_rf = features_rf.sort_values(by = ['importance'], ascending = False)
important_features_rf = features_sorted_rf['feature'].head(15)
X_train_small = X_train.loc[:, important_features_rf]
X_test_small = X_test.loc[:, important_features_rf]
rf_small =  RandomForestClassifier(criterion='gini', max_depth=15, n_estimators = 1000, random_state =
0)
rf_small.fit(X_train_small, y_train)
```

```
#Calculated and plot ROC Curve and AUC
def plot_roc(model, feature_X, feature_y, model_name):
        model.fit(feature_X, feature_y)
        y_prob = model.predict_proba(feature_X)[:, 1]
        fpr, tpr, thresholds = roc_curve(y_test, y_prob)
        plt.plot(fpr, tpr, linewidth = 5)
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.0])
        plt.title("ROC Curve for Cancer Prediction by {}".format(model_name))
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.grid(True)
        plt.show()

        auc_score = roc_auc_score(y_test, y_prob)
        print(auc_score)
        return(auc_score)
#Calculating the AUC (Area Under the Curve) Score
plot_roc(dtree_opt, X_test, y_test, "Decision Tree")
plot_roc(rf_small, X_test_small, y_test, "Random Forest Small")




###### Plotly Graph #######
## Making a plotly graph to show category separation from only top three features
#Top three features as shown by random forest
a = train_original["compactness_mean"]
b = train_original["concavity_se"]
c = train_original["area_mean"]

#Importing packages and setting credentials
import plotly
import plotly.plotly as py
import plotly.graph_objs as go
plotly.tools.set_credentials_file(username='jackcrum', api_key='CueWpz8W4OROnmTR9mko')
#Mapping colors onto original training data
size_mapping = {'B': "blue",'M': "red"}
train_original['color'] = train_original['diagnosis'].map(size_mapping)
#Create the 3D scatter plot
trace1 = go.Scatter3d(x=a, y=b, z=c, mode='markers', marker=dict(size=4, color=train_original["color"],
opacity=0.8))
#Set the layout
data = [trace1]
layout = go.Layout(title='Plot Title', xaxis=dict(title='Growth Mean Radius'), yaxis=dict(title='Growth Mean
Texture'))
#Plot the figure to my plotly account
fig = go.Figure(data=data, layout=layout)
```

```
py.iplot(fig, filename='3d-scatter-tumor')
```