Data Analysis of Wisconsin Breast Cancer Data

Remya Kannan
College of Computing and Digital Media
Depaul University
Chicago, Illinois

Introduction:

Breast cancer stands to be the second leading cause of death among women. Statistics claim that one in every four are affected by breast cancer. The major challenge with the health care institutions is providing quality services, and this is especially important in underprivileged countries. Faulty clinical decisions could result in severe consequences. With medical datasets at our disposal there is a need to combine tools and the existing machine learning algorithms to intelligentially extract knowledge to predict the occurrence through accurate diagnosis. While many studies have been done to build models to accurately predict the diagnosis, there is a continual need for the search of an effective and efficient model to do the same.

Various literature has introduced new concepts in the model building process of breast cancer diagnosis. Machine learning concepts such as classification, clustering, etc. have been incorporated and models developed with good results. Further, new concepts developed involving hybrid models that combine techniques to improve the model. The objective of this study is to find a model that helps predict the occurrence of breast cancer based on the diagnostic results with the highest possible accuracy. The UCI breast cancer data set is used for this purpose.

The key to understanding the diagnostics for breast cancer, is learning the cause. With it being the leading cause of death among women globally, it is vital to learn the occurrence and the symptoms to effectively detect and prevent it. With development in technology, there are several ways to check the diagnostics to prevent the occurrence of the breast cancer. The tests conducted by mammograms, X-Rays, etc. provide significant amount of data that can be studied using the machine learning techniques such as SVM, Fuzzy techniques, ANN, etc. Hybrid methods combining various classifiers and clustering algorithms have been incorporated to develop a model with increased accuracy.

This study focuses on the use of the above-mentioned data mining techniques, to develop a model that can predict the occurrence of breast cancer given the diagnostics data. Firstly, the dataset is explored and cleaned for use in the data mining process. Then, classification techniques such as Random Forest, decision tree, SVM, LDA, PCA, KNN and logistic regression are explored and compared for evaluation. Finally, conclusion is drawn to what model best helps predict the occurrence of breast cancer.

This report is divided as follows: section 1 briefly describes the data highlighting the relevant features of the dataset used in this study, section 2 explains how the data was cleaned in preparation for model building, data analysis using the various techniques mentioned above is explored in section 3, section 4 highlights the experimental results that discusses the performance of the model, section 5 interprets and analyses the results followed by conclusion in section 6.

1. Data description:

The Wisconsin Breast Cancer dataset from the UCI Machine Learning Repository is used, to distinguish malignant (cancerous) from benign (non-cancerous) samples. A brief description of these datasets is shown in Table 1.

Dataset	No. of attributes	No. of instances	No. of classes
Wisconsin Breast Cancer (Original)	11	699	2
Wisconsin Diagnosis Breast Cancer (WDBC)	32	569	2
Wisconsin Prognosis Breast Cancer (WPBC)	34	198	2

Table 1. Data Description of Wisconsin Breast Cancer Dataset

2. Data preprocessing:

The data was acquired from the UCI repository and it consists of attributes that characterize data into people with or without heart disease. Data partition aims to split the data into subsets of training and testing data. It is a process by which mutually exclusive data is partitioned using the 10-fold cross validation. To avoid bias, the records are randomly selected for partition. Partitioning helps reduce computation time during model implementation.

Feature extraction is another important component of data preprocessing and involves reduction of the attributes that don't contribute to the model. Fig.1. shows the heat map which indicates that there is high correlation within some dependent variables. PCA can thus be performed to reduce the dimensionality and multicollinearity which in turn avoids redundancy.

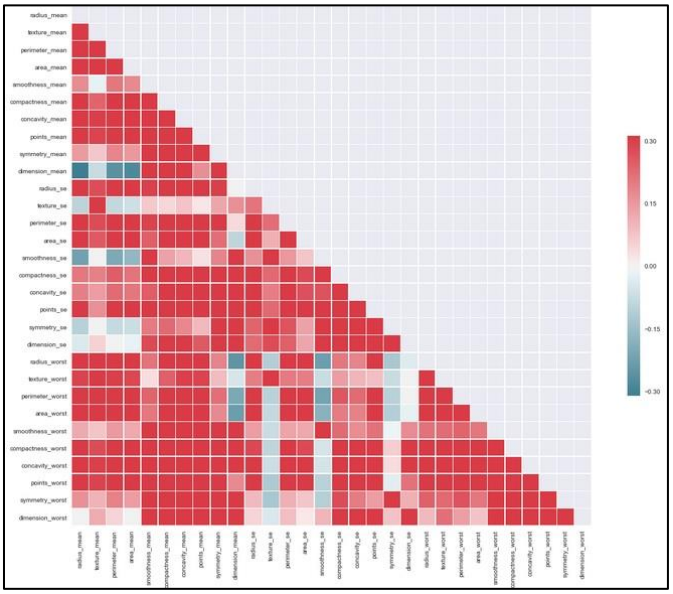


Fig.1. Heatmap indicating correlation

Data cleaning takes the raw data with the selected features to clean them of the missing values and transforms it to be used by the model. It takes places over individual attributes before they are fed to the classifiers. Missing values in medical dataset can be serious as this could mean loss of

information that is detrimental to diagnosis. There is a need for missing value imputation, that are replaced by the mean or 0. For this dataset, there were no missing values. The data was not noisy and there were very few outliers that affected the model.

3. Data Analysis:

Following the data cleaning process, the next stage of the study moves on to the exploratory analysis of the data. There is a need to find out if the variables are correlated or if outliers are present before we proceed to the model building process to avoid error in prediction. Thus, the exploratory analysis can be divided into bivariate analysis and multivariate analysis for this study.

3.1.Bivariate Analysis:

The correlation graph is plotted to remove multicollinearity as seen in Fig.2. We observe that the radius, perimeter, and area attributes are highly correlated as expected from their relation. Additionally, compactness_mean, concavity_mean and concavepoint_mean is highly correlated. Thus, the parameters selected are: perimeter_mean, texture_mean, compactness_mean, symmetry_mean.

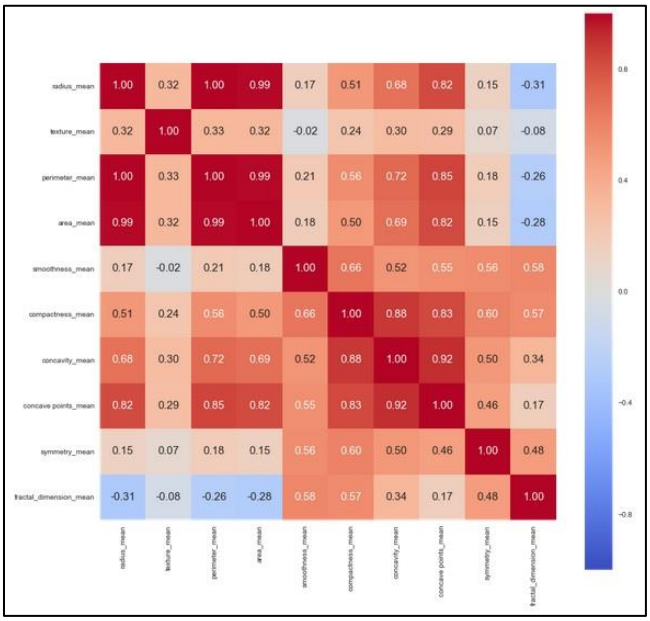


Fig.2. Correlation plot to detect multicollinearity

3.2.Multivariate Analysis:

PCA can be used as a technique for dimensionality reduction. It helps in finding those attributes that contribute towards the model building process. Thus, the results from PCA helped determine that not all the 30 attributes are useful for model building and only 10 attributes contribute to any variance in the data. As seen from the summary of the results below, we observe that it takes at least 10 attributes to explain 95% of the variance in the data and hence proves the importance of the selected attributes for model building.

```
0.44272026  0.63243208  0.72636371  0.79238506  0.84734274  0.88758796
0.9100953   0.92598254  0.93987903  0.95156881  0.961366    0.97007138
0.97811663  0.98335029  0.98648812  0.98915022  0.99113018  0.99288414
0.9945334   0.99557204  0.99657114  0.99748579  0.99829715  0.99889898
0.99941502  0.99968761  0.99991763  0.99997061  0.99999557  1.
```

4. Experimental Results:

This section highlights the results obtained from the classification techniques used in the study. The various classification techniques used in the study are: Random Forest, SVM, Decision Tree, KNN, Logistic Regression, Naïve Bayes, and LDA.

4.1.Random Forest:

The accuracy of the Random Forest model in the initial run is 91%. On including all the features, the accuracy of the Random Forest model has increased. This means that there were important features in the feature list. The accuracy is thus increased to 97%.

4.2.SVM:

The accuracy of the SVM model in the initial run is 85%. This means there is room for improvement. When all the features are included the accuracy of the SVM model significantly decreased to 71%. We thus observe that multicollinearity affects the SVM model but the Random Forest model is more immune. There is thus a need to tune the parameters. On including the important features as shown below, we observe that the accuracy of the SVM model increased to 96%.

```
concave points_mean      0.213517
concavity_mean           0.193938
area_mean                0.147238
radius_mean              0.141010
perimeter_mean           0.129064
texture_mean             0.054795
compactness_mean         0.044956
smoothness_mean          0.030024
symmetry_mean            0.024209
fractal_dimension_mean   0.021249
dtype: float64
```

4.3.Decision Tree:

As shown in the results below, on running the decision tree with 10-fold cross validation the overall accuracy of the decision tree is 93%. On using grid search tuning technique, we observe that the accuracy is increased to 95%.

decision tree (using "entropy" as selection criteria)

In [71]:
```
from sklearn import tree
treeclf = tree.DecisionTreeClassifier(criterion='entropy', min_samples_split=20, max_depth= 5)
treeclf = treeclf.fit(pcs, y)
print 'pridicted: ', treeclf.predict(pcs)
print 'Score: ', treeclf.score(pcs, y)
```

```
pridicted:  [0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1
 0 1 1 1 1 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 1 0 0 0
 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0
 0 1 1 0 1 1 1 1 0 0 1 0 1 0 0 1 1 1 1 1 1 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1 1 0 1 0 1 0 1 0 0 1 0 1 0
 0 1 1 1 0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 1 1 0 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0
 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1
 1 1 0 1 1 0 0 1 1 1 1 0 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 1 0 1 0 1 0 1 0 0 0
 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 1 1 0 1 0
 0 1 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0
 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 1 0
 1 1 0 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 1 1 0 1 1 0 1 1 1 1
 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0
 1 0 1 0 0 1 0 0 0 0 0 1 0 1]
Score:  0.980667838313
```

In [72]:
```
#10-fold cross-validation
cv_scores_DT = cross_validation.cross_val_score(treeclf, x, y, cv=10)
print '10-fold cross-validation scores: ', cv_scores_DT
```

```
10-fold cross-validation scores:  [ 0.91  0.91  0.93  0.95  0.98  0.91  0.93  0.95  0.96  0.93]
```

In [74]:
```
#overall average accuracy
print("Overall Accuracy of Decision Tree: %0.2f (+/- %0.2f)" % (cv_scores_DT .mean(), cv_scores_DT .std() * 2))
```

```
Overall Accuracy of Decision Tree: 0.94 (+/- 0.04)
```

Tuning parameters using Grid Search

In [135]:
```
# Decision tree classifier tuning
data_X= data[prediction_var]
data_y= data["diagnosis"]
```

In [136]:
```
# Function for grid search
def Classification_model_gridsearchCV(model,param_grid,data_X,data_y):
    clf = GridSearchCV(model,param_grid,cv=10,scoring="accuracy")
    clf.fit(train_X,train_y)
    print("The best parameter found on development set is :")
    print(clf.best_params_)
    print("the bset estimator is ")
    print(clf.best_estimator_)
    print("The best score is ")
    print(clf.best_score_)
```

In [137]:
```
param_grid = {'max_features': ['auto', 'sqrt', 'log2'],
              'min_samples_split': [2,3,4,5,6,7,8,9,10],
              'min_samples_leaf':[2,3,4,5,6,7,8,9,10] }

# Grid search considers all combinations of parameters and applies to the model to find the best parameter
model= DecisionTreeClassifier()
Classification_model_gridsearchCV(model,param_grid,data_X,data_y)
```

```
The best parameter found on development set is :
{'max_features': 'auto', 'min_samples_leaf': 8, 'min_samples_split': 7}
the bset estimator is
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=8,
            min_samples_split=7, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
The best score is
0.947236180905
```

We observe the score increased to 95%.

4.4.KNN:

From Fig.3 we observe that k=6 returns good accuracy. The overall accuracy of the model is 97%.

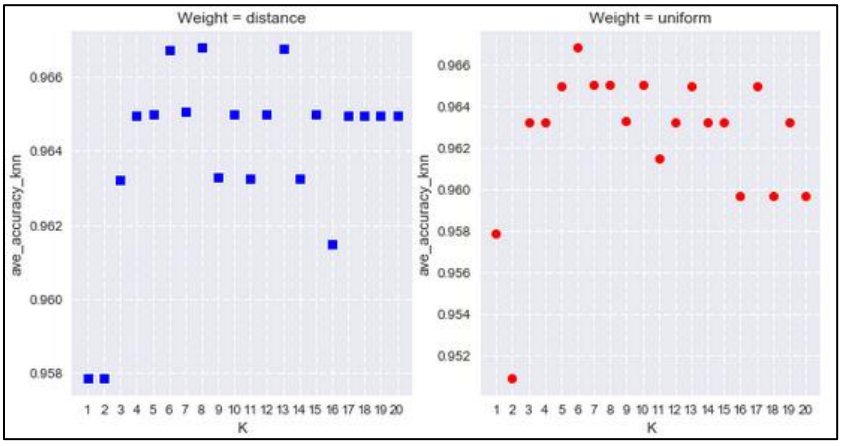
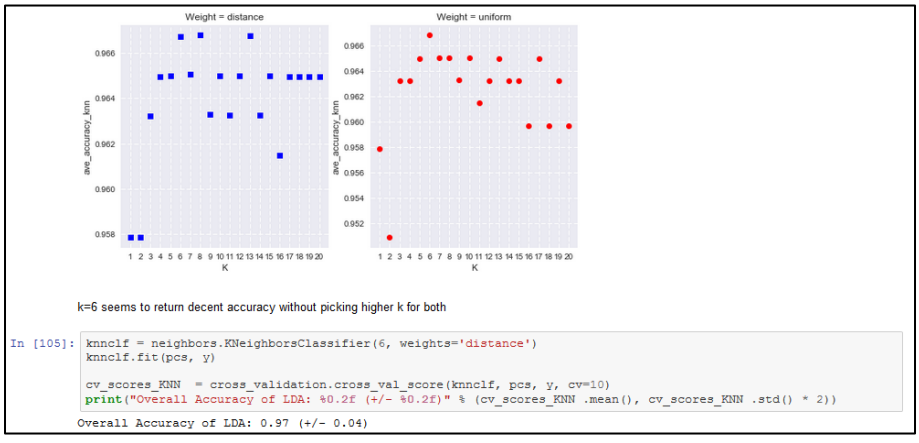


Fig.3. knn plot indicating k=6 is ideal



4.5. Naïve Bayes:

From the results below, we observe that the model reaches an accuracy of 92% on using 10-fold cross validation.

Naive Bayes (Gaussian)

```
In [66]: from sklearn import naive_bayes
         nbclf = naive_bayes.GaussianNB()
         nbclf = nbclf.fit(pcs, y)
         print 'pridicted: ', nbclf.predict(pcs)
         print 'Score: ', nbclf.score(pcs, y)
```

```
pridicted:  [0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1
 0 0 1 1 1 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 1 0 1 0
 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0
 1 1 1 0 1 1 1 1 1 0 0 1 0 1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0
 0 1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 0 0
 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0
 0 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 1 0 1 1
 1 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 0 1 0 1 1 1 0 0 1 1 1 1 0 0 1 0 1 0 1 0 0 0
 0 1 0 0 1 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 1 1 0 1 0
 0 1 0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0
 1 0 0 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0
 0 1 1 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 1 1 0 0 1 0 1 1 1 1
 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0
 1 0 1 0 0 1 0 0 0 0 0 0 0 1]
Score:  0.919156414763
```

```
In [73]: #10-fold cross-validation
         cv_scores_NB = cross_validation.cross_val_score(nbclf, pcs, y, cv=10)
         print '10-fold cross-validation scores: ', cv_scores_NB
```

```
10-fold cross-validation scores:  [ 0.93  0.9   0.89  0.91  0.91  0.96  0.95  0.93  0.88  0.91]
```

```
In [70]: #overall average accuracy
         print("Overall Accuracy of Naive Bayes (Gaussian): %0.2f (+/- %0.2f)" % (cv_scores_NB.mean(), cv_scores_NB.std()
         * 2))
```

```
Overall Accuracy of Naive Bayes (Gaussian): 0.92 (+/- 0.05)
```

4.6.Linear Discriminant Analysis:

From the results below, we observe that the model implementing LDA has an accuracy of 96% on using 10-fold cross validation.

Linear discriminant analysis (LDA)

```
In [78]: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
In [79]: ldclf = LinearDiscriminantAnalysis()
         ldclf = ldclf.fit(pcs, y)
         print 'pridicted: ', ldclf.predict(pcs)
         print 'Score: ', ldclf.score(pcs, y)
```

```
pridicted:  [0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1
 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0
 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0
 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 1 1 0 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0
 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1
 1 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 0 1 0 1 1 1 0 0 1 1 1 1 0 0 1 0 1 0 1 0 0 0
 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 1 1 0 1 0
 0 1 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 1 0
 1 0 0 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 1 0
 0 1 0 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 1 1 0 0 1 0 1 1 1 1
 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0
 1 0 0 0 0 1 0 0 0 0 0 0 0 1]
Score:  0.957820738137
```

```
In [80]: #10-fold cross-validation
         cv_scores_LDA = cross_validation.cross_val_score(ldclf, x, y, cv=10)
         print '10-fold cross-validation scores: ', cv_scores_LDA
```

```
10-fold cross-validation scores:  [ 0.93  0.91  0.96  0.95  0.93  1.    0.95  1.    0.96  0.98]
```

```
In [81]: #overall average accuracy
         print("Overall Accuracy of LDA: %0.2f (+/- %0.2f)" % (cv_scores_LDA.mean(), cv_scores_LDA.std() * 2))
```

```
Overall Accuracy of LDA: 0.96 (+/- 0.06)
```

4.7.Logistic Regression:

The model using logistic regression returned the best results. It has an accuracy of 98%! The results are shown below.

Logistic Regression

```
In [90]: from sklearn.linear_model import LogisticRegression
```

```
In [112]: lgr = LogisticRegression()
          lgr.fit(pcs, y)

          cv_scores_lgr = cross_validation.cross_val_score(lgr, pcs, y, cv=10)
          print("Overall Accuracy of LDA: %0.2f (+/- %0.2f)" % (cv_scores_lgr.mean(), cv_scores_lgr.std() * 2))
```

```
Overall Accuracy of LDA: 0.98 (+/- 0.05)
```

Comparison of accuracy over the models:

A comparison of all the models with the various classification techniques above indicate that Logistic regression is the best model with an accuracy of 98%.

	Decision Tree	KNN	LDA	Logistic	Naive Bayes
Accuracy	0.936863	0.966723	0.959705	0.975619	0.917345

Now that the model is built on the training dataset, it is tested on the test dataset and evaluated using performance metrics such as ROC.

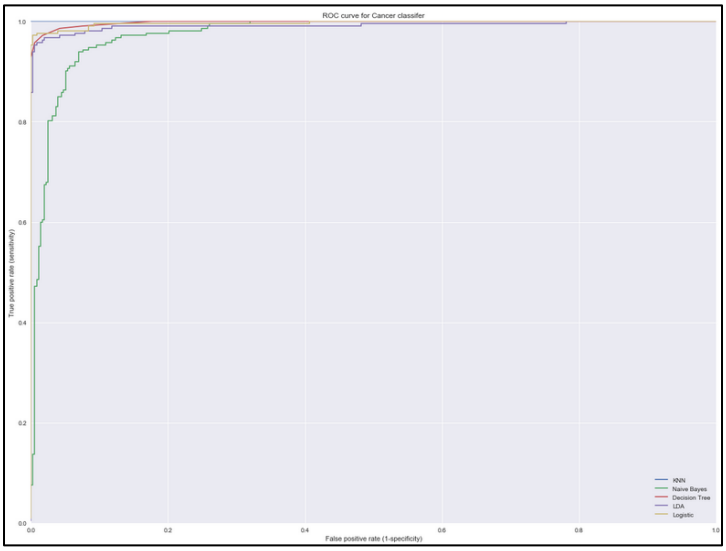


Fig.4. ROC curve for model comparison

From Fig.4., we observe that the model using Naïve Bayes classification technique is the worst model and the models with KNN and logistic regression techniques are the better models.

5. Experimental Analysis & Conclusion:

The objective of this study is to find a model using classification techniques that can predict the diagnosis of the presence or absence of breast cancer in a patient based on the data from the various tests such as mammogram, XRays, etc. The Wisconsin Breast Cancer dataset is used for this purpose. This could aid many medical institutions to predict the occurrence of breast cancer given certain parameters and in turn help prevent it.

This study started with the exploration of the dataset. After the initial data analysis of missing values imputation, feature selection and check for correlation, the data was deemed good for model building.

In this study, various classification and prediction models were designed and developed. The model was used to diagnose and classify the presence or absence of breast cancer. The model with Random Forest including all the important features gave an accuracy of 97%. The model using SVM initially had an accuracy of 85% and when all the features were included it brought down the accuracy significantly to 71%. This means important features need to be selected and using the

grid search tuning techniques, the important features are selected and the final model has an accuracy of 96%. LDA model has an accuracy of 96%.

Yet another classification model uses the decision tree classifier with 93%. On using the grid search tuning techniques, the accuracy increased to 95%. KNN classification model with k set to 6 has an accuracy of 97% while the Naïve Bayes model has an accuracy of 92%. The best model is the logistic regression model with an accuracy of 98%.

We now calculate the ROC to rate the model's performance to support the 10-fold cross validation and accuracy. Looking at the ROC curve in Fig.4., we observe that the plot is towards the true positive side indicating that very few/none of the classes were misclassified. We also observe that the Naïve Bayes model has the worst performance confirming with the lowest accuracy rate and the Logistic regression and KNN model is the best model. As the ROC gets closer to the optimal point of perfection, AUC gets closer to 1. The AUC value for the model with Logistic regression model is 0.89 indicating that the model is appropriate for prediction. This is very important especially in the dataset under consideration as any misclassification error can prove to be fatal. **Thus, we may conclude that we are 95% confident that the logistic regression model can predict the diagnosis of a patient with breast with an accuracy of 98% as the p-value is <0.05**.

For future work, a variety of ensemble learning techniques may be applied such as Adaboost and stacking algorithms can be used in datasets to develop a better model. This study can also extend over other datasets to see the impact of various learning techniques in different fields of analysis. There is a never-ending search for a model that can provide results with high accuracy with minimum computational complexity that can be used in a variety of fields as sensitive as the medical data that could potentially help the society and the less privileged.

Appendix:

Python code for Data Analysis:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.cross_validation import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn import metrics
```

Import Data

```
data =
pd.read_csv("C:/Users/remya/Desktop/Depaul_5th_quarter/CSC_478/project/data.c
sv",header=0)
# Lets take a look at the data
```

```
print(data.head(2))
       id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0  842302         M        17.99         10.38           122.8     1001.0
1  842517         M        20.57         17.77           132.9     1326.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017

      ...       texture_worst  perimeter_worst  area_worst  smoothness_worst
\
0     ...               17.33            184.6      2019.0            0.1622
1     ...               23.41            158.8      1956.0            0.1238

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst
\
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN

[2 rows x 33 columns]
# Lets take a look at the type of data
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
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
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
# We drop the column Unnamed: 32
data.drop("Unnamed: 32",axis=1,inplace=True)
#  Lets take a look at the columns now
data.columns
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
# We dont need the id attribute either
data.drop("id",axis=1,inplace=True)
# Dividing the data according to the features by categories
features_mean= list(data.columns[1:11])
features_se= list(data.columns[11:20])
features_worst=list(data.columns[21:31])
print(features_mean)
print("-----------------------------------")
print(features_se)
print("-----------------------------------")
print(features_worst)
['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']
-----------------------------------
['radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst',
'smoothness_worst', 'compactness_worst', 'concavity_worst', 'concave
points_worst', 'symmetry_worst', 'fractal_dimension_worst']
# Starting with features_mean
data['diagnosis']=data['diagnosis'].map({'M':1,'B':0})
```

Data Exploration

```
data.describe()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	0.372583	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	...	16.269190	25.677223	107.261213	880.583128	0.132369	0.254265	0.272188	0.114606	0.290076	0.083946
std	0.483918	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	...	4.833242	6.146258	33.602542	569.356993	0.022832	0.157336	0.208624	0.065732	0.061867	0.018061
min	0.000000	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	...	7.930000	12.020000	50.410000	185.200000	0.071170	0.027290	0.000000	0.000000	0.156500	0.055040
25%	0.000000	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	...	13.010000	21.080000	84.110000	515.300000	0.116600	0.147200	0.114500	0.064930	0.250400	0.071460
50%	0.000000	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	...	14.970000	25.410000	97.660000	686.500000	0.131300	0.211900	0.226700	0.099930	0.282200	0.080040
75%	1.000000	15.780	21.800000	104.100000	782.70	0.105300	0.130400	0.130700	0.074000	0.195700	...	18.790000	29.720000	125.400000	1084.000	0.146000	0.339100	0.382900	0.161400	0.317900	0.092080

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
		000			0000										000						
max	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	...	36.040000	49.540000	251.200000	4254.000000	0.222600	1.058000	1.252000	0.291000	0.663800	0.207500

8 rows × 31 columns

```
# Determine the frequency of cancer stages
sns.countplot(data['diagnosis'],label="Count")
<matplotlib.axes._subplots.AxesSubplot at 0x221e66e4eb8>
```

From the graph observe that there is more number of bengin stage cancer which can be cured!

Feature Selection

```
# We plot the correlation graph to remove multicollinearity
corr = data[features_mean].corr()
plt.figure(figsize=(14,14))
sns.heatmap(corr, cbar = True,  square = True, annot=True, fmt=
'.2f',annot_kws={'size': 15},
           xticklabels= features_mean, yticklabels= features_mean,
           cmap= 'coolwarm')
<matplotlib.axes._subplots.AxesSubplot at 0x221e615aeb8>
```

Analysis:

1. The radius, parameter and area are highly correlated as expected from their relation.
2. compactness_mean, concavity_mean and concavepoint_mean are highly correlated.
3. Thus, the parameters selected are perimeter_mean, texture_mean, compactness_mean, symmetry_mean.

```
prediction_var =
['texture_mean','perimeter_mean','smoothness_mean','compactness_mean','symmet
ry_mean']
# Split the data into training and testing sets
train, test = train_test_split(data, test_size = 0.3)
# Dimensions
print(train.shape)
print(test.shape)
(398, 31)
(171, 31)
train_X = train[prediction_var]
train_y=train.diagnosis
test_X= test[prediction_var]
test_y =test.diagnosis
# Simple random forest model
model=RandomForestClassifier(n_estimators=100)
# Model fitting
model.fit(train_X,train_y)
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=100, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
# Prediction
prediction=model.predict(test_X)
# Accuracy
metrics.accuracy_score(prediction,test_y)
0.94152046783625731
```

The accuracy of the Random Forest model is 91%. We need to explore other techniques to improve the accuracy.

```
# SVM
model = svm.SVC()
model.fit(train_X,train_y)
prediction=model.predict(test_X)
metrics.accuracy_score(prediction,test_y)
0.87134502923976609
```

SVM model has an accuracy of 85%. This means there is room for improvement. The next step is to consider all features.

```
prediction_var = features_mean
train_X= train[prediction_var]
train_y= train.diagnosis
test_X = test[prediction_var]
test_y = test.diagnosis
model=RandomForestClassifier(n_estimators=100)
model.fit(train_X,train_y)
prediction = model.predict(test_X)
metrics.accuracy_score(prediction,test_y)
0.95906432748538006
```

On including all the featares the accuracy of the Random Forest model has increased. This means that there were important features in the feature list. Extracting the useful features for model building and prediction is the next step.

```
# looking at the importance of the features
featimp = pd.Series(model.feature_importances_,
index=prediction_var).sort_values(ascending=False)
print(featimp)
# Considering all features for the SVM model
model = svm.SVC()
model.fit(train_X,train_y)
prediction=model.predict(test_X)
metrics.accuracy_score(prediction,test_y)
concave points_mean       0.213517
concavity_mean            0.193938
area_mean                 0.147238
radius_mean               0.141010
perimeter_mean            0.129064
texture_mean              0.054795
compactness_mean          0.044956
smoothness_mean           0.030024
symmetry_mean             0.024209
fractal_dimension_mean    0.021249
dtype: float64
0.7192982456140351
```

The accuracy of the SVM model decreased significantly!

```
# Considering the 5 most important features
prediction_var=['concave points_mean','perimeter_mean' , 'concavity_mean' ,
'radius_mean','area_mean']
train_X= train[prediction_var]
train_y= train.diagnosis
test_X = test[prediction_var]
test_y = test.diagnosis
model=RandomForestClassifier(n_estimators=100)
model.fit(train_X,train_y)
prediction = model.predict(test_X)
metrics.accuracy_score(prediction,test_y)
0.91228070175438591
model = svm.SVC()
model.fit(train_X,train_y)
prediction=model.predict(test_X)
metrics.accuracy_score(prediction,test_y)
0.77192982456140347
```

From the above analysis, we observe that multicollinearity affects the SVM model but the random forest classifier is more immune.

```
prediction_var = features_worst
train_X= train[prediction_var]
train_y= train.diagnosis
test_X = test[prediction_var]
test_y = test.diagnosis
```

```
model = svm.SVC()
model.fit(train_X,train_y)
prediction=model.predict(test_X)
metrics.accuracy_score(prediction,test_y)
0.64327485380116955
```

The low accuracy confirms the need to tune the parameters!

```
model=RandomForestClassifier(n_estimators=100)
model.fit(train_X,train_y)
prediction = model.predict(test_X)
metrics.accuracy_score(prediction,test_y)
0.98245614035087714
featimp = pd.Series(model.feature_importances_,
index=prediction_var).sort_values(ascending=False)
print(featimp)
perimeter_worst            0.287027
concave points_worst       0.218910
area_worst                 0.176132
radius_worst               0.129253
concavity_worst            0.057210
texture_worst              0.033661
smoothness_worst           0.027317
compactness_worst          0.026631
symmetry_worst             0.022085
fractal_dimension_worst    0.021775
dtype: float64
prediction_var = ['concave
points_worst','radius_worst','area_worst','perimeter_worst','concavity_worst'
]
train_X= train[prediction_var]
train_y= train.diagnosis
test_X = test[prediction_var]
test_y = test.diagnosis
model=RandomForestClassifier(n_estimators=100)
model.fit(train_X,train_y)
prediction = model.predict(test_X)
metrics.accuracy_score(prediction,test_y)
0.9707602339181286
# SVM
model = svm.SVC()
model.fit(train_X,train_y)
prediction=model.predict(test_X)
metrics.accuracy_score(prediction,test_y)
0.64912280701754388
```

Scatterplot to identify the variables between two classes

```
color_function = {0: "blue", 1: "red"} # Red is for malignant and blue is for
benign
colors = data["diagnosis"].map(lambda x: color_function.get(x))
pd.scatter_matrix(data[features_mean], c=colors, alpha = 0.5, figsize = (15,
15));
```

Analysis:

1. Radius, area and perimeter have a strong relationship.

```
# Features for prediction
features_mean
['radius_mean',
 'texture_mean',
 'perimeter_mean',
 'area_mean',
 'smoothness_mean',
 'compactness_mean',
 'concavity_mean',
 'concave points_mean',
 'symmetry_mean',
 'fractal_dimension_mean']
predictor_var =
['radius_mean','perimeter_mean','area_mean','compactness_mean','concave
points_mean']
def model(model,data,prediction,outcome):
    kf = KFold(data.shape[0], n_folds=10)
prediction_var =
['radius_mean','perimeter_mean','area_mean','compactness_mean','concave
points_mean']
#  Cross-validation
def classification_model(model,data,prediction_input,output):
    model.fit(data[prediction_input],data[output])
    predictions = model.predict(data[prediction_input])
    accuracy = metrics.accuracy_score(predictions,data[output])
    print("Accuracy : %s" % "{0:.3%}".format(accuracy))
    kf = KFold(data.shape[0], n_folds=5)
    error = []
    for train, test in kf:
        train_X = (data[prediction_input].iloc[train,:])
        train_y = data[output].iloc[train]
        model.fit(train_X, train_y)
        test_X=data[prediction_input].iloc[test,:]
        test_y=data[output].iloc[test]
        error.append(model.score(test_X,test_y))
        print("Cross-Validation Score : %s" %
"{0:.3%}".format(np.mean(error)))
# Decision Tree  classifier
model = DecisionTreeClassifier()
prediction_var =
['radius_mean','perimeter_mean','area_mean','compactness_mean','concave
points_mean']
outcome_var= "diagnosis"
classification_model(model,data,prediction_var,outcome_var)
Accuracy : 100.000%
Cross-Validation Score : 86.842%
Cross-Validation Score : 87.281%
Cross-Validation Score : 89.181%
Cross-Validation Score : 90.132%
Cross-Validation Score : 90.512%
```

Analysis:

If the accuracy is 100% there is signs of overfitting. However, the cross-validation scores are not great and hence we dont consider accuracy alone.

```
# KNN
model = KNeighborsClassifier()
classification_model(model,data,prediction_var,outcome_var)
Accuracy : 90.510%
Cross-Validation Score : 76.316%
Cross-Validation Score : 80.263%
Cross-Validation Score : 85.965%
Cross-Validation Score : 86.623%
Cross-Validation Score : 86.820%
```

The cross-validation scores are not good.

```
# Random Forest
model = RandomForestClassifier(n_estimators=100)
classification_model(model,data,prediction_var,outcome_var)
Accuracy : 100.000%
Cross-Validation Score : 85.088%
Cross-Validation Score : 88.596%
Cross-Validation Score : 90.643%
Cross-Validation Score : 91.667%
Cross-Validation Score : 91.563%
model = svm.SVC()

classification_model(model,data,prediction_var,outcome_var)
Accuracy : 96.661%
Cross-Validation Score : 56.140%
Cross-Validation Score : 65.789%
Cross-Validation Score : 69.883%
Cross-Validation Score : 72.807%
Cross-Validation Score : 74.706%
# Logistic regression
model=LogisticRegression()
classification_model(model,data,prediction_var,outcome_var)
Accuracy : 89.279%
Cross-Validation Score : 78.070%
Cross-Validation Score : 82.018%
Cross-Validation Score : 86.550%
Cross-Validation Score : 87.939%
Cross-Validation Score : 89.112%
```

Tuning parameters using Grid Search

```
# Decision tree classifier tuning
data_X= data[prediction_var]
data_y= data["diagnosis"]
# Function for grid search
def Classification_model_gridsearchCV(model,param_grid,data_X,data_y):
    clf = GridSearchCV(model,param_grid,cv=10,scoring="accuracy")
    clf.fit(train_X,train_y)
```

```
    print("The best parameter found on development set is :")
    print(clf.best_params_)
    print("the bset estimator is ")
    print(clf.best_estimator_)
    print("The best score is ")
    print(clf.best_score_)
param_grid = {'max_features': ['auto', 'sqrt', 'log2'],
              'min_samples_split': [2,3,4,5,6,7,8,9,10],
              'min_samples_leaf':[2,3,4,5,6,7,8,9,10] }

# Grid search considers all combinations of parameters and applies to the
model to find the best parameter
model= DecisionTreeClassifier()
Classification_model_gridsearchCV(model,param_grid,data_X,data_y)
The best parameter found on development set is :
{'max_features': 'auto', 'min_samples_leaf': 8, 'min_samples_split': 7}
the bset estimator is
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=8,
            min_samples_split=7, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
The best score is
0.947236180905
```

We observe the score increased to 95%.

```
# KNN
model = KNeighborsClassifier()

k_range = list(range(1, 30))
leaf_size = list(range(1,30))
weight_options = ['uniform', 'distance']
param_grid = {'n_neighbors': k_range, 'leaf_size': leaf_size, 'weights':
weight_options}
Classification_model_gridsearchCV(model,param_grid,data_X,data_y)
The best parameter found on development set is :
{'leaf_size': 1, 'n_neighbors': 9, 'weights': 'uniform'}
the bset estimator is
KNeighborsClassifier(algorithm='auto', leaf_size=1, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=9, p=2,
           weights='uniform')
The best score is
0.924623115578
# SVM
model=svm.SVC()
param_grid = [
              {'C': [1, 10, 100, 1000],
               'kernel': ['linear']
              },
              {'C': [1, 10, 100, 1000],
               'gamma': [0.001, 0.0001],
               'kernel': ['rbf']
              },
 ]
Classification_model_gridsearchCV(model,param_grid,data_X,data_y)
```

```
The best parameter found on development set is :
{'C': 1000, 'kernel': 'linear'}
the bset estimator is
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
The best score is
0.934673366834
```

Analysis:

SVM works the best when optimal parameters are chosen, with increase in accuracy from 70% to 93%

yuehchao wu_Final Project

```
cd /users/jasonwu/desktop/Final_prj_478
/Users/jasonwu/Desktop/Final_prj_478
#libarary
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.style.use('ggplot')
%matplotlib inline
#load the dataset
data = pd.read_csv("wisc_bc_data.csv",na_values='?')
data.head(5)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	points_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	points_worst	symmetry_worst	dimension_worst
0	8710402	B	12.32	12.39	78.85	464.1	0.10280	0.06981	0.03987	0.037000	...	13.50	15.64	86.97	549.1	0.13185	0.12666	0.12420	0.09391	0.28	0.06771
1	8910251	B	10.60	18.95	69.28	346.4	0.09688	0.11470	0.06387	0.026042	...	11.88	22.94	78.28	424.8	0.12	0.25	0.19160	0.07926	0.29	0.07587
2	905520	B	11.04	16.83	70.92	373.2	0.10770	0.07804	0.03046	0.024800	...	12.41	26.44	79.93	471.4	0.13	0.14	0.10670	0.07431	0.29	0.07881

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	points_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	points_worst	symmetry_worst	dimension_worst
3	868871	B	11.28	13.39	73.00	384.8	0.11640	0.11360	0.04635	0.04796	...	11.92	15.77	76.53	434.0	0.13667	0.18222	0.08669	0.08611	0.21002	0.06784
4	9012568	B	15.19	13.21	97.65	711.8	0.07963	0.06934	0.03393	0.02657	...	16.20	15.73	104.50	819.1	0.11426	0.17437	0.13620	0.08178	0.2487	0.06766

5 rows × 32 columns

```
data.shape
(569, 32)
```

preprocessing

```
#check for missing values
print "attribute with missing values: \n"
print [col for col in data.columns if data[col].isnull().any()]
attribute with missing values:

[]
```

- no missing values

```
#drop unused columed

data.drop('id',axis=1,inplace=True)
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
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
points_mean          569 non-null float64
symmetry_mean        569 non-null float64
```

```
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
points_se            569 non-null float64
symmetry_se          569 non-null float64
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
points_worst         569 non-null float64
symmetry_worst       569 non-null float64
dimension_worst      569 non-null float64
dtypes: float64(30), object(1)
memory usage: 137.9+ KB
```

- Target: diagnosis
- Training : the rests of varaibles
- Goal : build the model(Classifier) that acuurately predict (classify) the data

Exploration

```
# bar plot for target attribute :diagnosis
fig = plt.figure(figsize=(5, 5))
ax=fig.add_subplot(1,1,1)
ax.set_title('diagnosis')
data['diagnosis'].value_counts().plot(kind='bar',color='blue')
fig.tight_layout()
plt.show()
```

- ~350 obeservation are diagnosed as B (malignant)
- ~200 obeservation are diagnosed as M (benign)

```
# correlation between training attributes

#visulize the correlation
import seaborn as sns
# Generate a mask for the upper triangle
mask = np.zeros_like(data.iloc[:,1:].corr(), dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(20, 15))

# Generate a custom diverging colormap
```

```
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(data.iloc[:,1:].corr(), mask=mask, cmap=cmap, vmax=.3,
            square=True,linewidths=.5, cbar_kws={"shrink": .5}, ax=ax)
plt.yticks(rotation=0)
plt.xticks(rotation=90)
plt.show()
```

- heat map show high correlation with in some dependent varaibles, PCA will be performed to reduce the dimensionality andmuticolinearity since high correlation within depedent varaibles may indicate redundency

```
#seperate targe and training attributs

x= data.iloc[:,1:]
y= pd.get_dummies(data.diagnosis)
y=y.M
0    0
1    0
2    0
3    0
4    0
Name: M, dtype: uint8
x.head()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	points_mean	symmetry_mean	dimension_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	points_worst	symmetry_worst	dimension_worst
0	12.32	12.39	78.85	464.1	0.10280	0.06981	0.03987	0.03700	0.1959	0.05955	...	13.50	15.64	86.97	549.1	0.1385	0.1266	0.12420	0.09391	0.2827	0.06771
1	10.60	18.95	69.28	346.4	0.09688	0.11470	0.06387	0.02642	0.1922	0.06491	...	11.88	22.94	78.28	424.8	0.1213	0.2515	0.19160	0.07926	0.2940	0.07587
2	11.04	16.83	70.92	373.2	0.10770	0.07804	0.03046	0.02480	0.1714	0.06340	...	12.41	26.44	79.93	471.4	0.1369	0.1482	0.10670	0.07431	0.2998	0.07881
3	11.28	13.39	73.00	384.8	0.11640	0.11360	0.04635	0.04796	0.1771	0.06072	...	11.92	15.77	76.53	434.0	0.1367	0.1822	0.08669	0.08611	0.2102	0.06784

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	points_mean	symmetry_mean	dimension_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	points_worst	symmetry_worst	dimension_worst
4	15.19	13.21	97.65	711.8	0.07963	0.06934	0.03393	0.02657	0.1721	0.05544	...	16.20	15.73	104.50	819.1	0.11426	0.17637	0.13620	0.08178	0.2487	0.06766

5 rows × 30 columns

```
y.head()
0    0
1    0
2    0
3    0
4    0
Name: M, dtype: uint8
```

- Target : y (M=0,B=1)

PCA

normalized x

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
x_std = scaler.fit(x).transform(x)
x_std.shape
(569, 30)
#pca on standardized Xs
from sklearn import decomposition
pca = decomposition.PCA(n_components=30)
pca.fit(x_std)
print pca.explained_variance_ratio_.cumsum()
[ 0.44272026  0.63243208  0.72636371  0.79238506  0.84734274  0.88758796
  0.9100953   0.92598254  0.93987903  0.95156881  0.961366    0.97007138
  0.97811663  0.98335029  0.98648812  0.98915022  0.99113018  0.99288414
  0.9945334   0.99557204  0.99657114  0.99748579  0.99829715  0.99889898
  0.99941502  0.99968761  0.99991763  0.99997061  0.99999557  1.        ]
```

- 10 pcs can explained ~95% variation.
- we can reduce the size from 30 dim to 10 dim and only lose 5% of variation so why not

Transform the data into a reduced dimension space(10 components)

```
pca = decomposition.PCA(n_components=10)
```

```
np.set_printoptions(precision=2,suppress=True)
pcs=pca.fit(x_std).transform(x_std)
print pcs
[[-2.51  0.11 -0.5  ..., -0.03  0.51  0.34]
 [-1.46  1.69  1.17 ...,  0.14  0.36 -0.49]
 [-2.93  0.38 -0.89 ..., -0.9   0.19 -0.4 ]
 ...,
 [-0.39 -0.99 -2.59 ...,  0.21 -0.1   0.46]
 [-1.17 -0.47 -0.48 ..., -0.41 -0.55 -0.17]
 [ 3.64 -1.96 -0.83 ..., -1.52 -1.83  0.53]]
pcs.shape
(569, 10)
```

Modeling

```
from sklearn import cross_validation
```

Naive Bayes (Gaussian)

```
from sklearn import naive_bayes
nbclf = naive_bayes.GaussianNB()
nbclf = nbclf.fit(pcs, y)
print 'pridicted: ', nbclf.predict(pcs)
print 'Score: ', nbclf.score(pcs, y)
pridicted:  [0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0
1 0 0 0 1
 0 0 1 1 1 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 0 1 0
 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0
 1 1 1 0 1 1 1 1 0 0 1 0 1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 1 1 1 1 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0
 0 1 1 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 1 1 0 0
 1 0 0 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0
 0 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1
 1 1 0 1 1 0 0 1 1 1 1 0 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 1 0 1 0 0 0
 0 1 0 0 1 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 1 1 0 1 0
 0 1 0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 1 0
 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0
 0 1 1 0 1 0 1 0 1 1 1 1 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 1 1 0 0 1 0 1 1 1 1
 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0
 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1]
Score:  0.919156414763
#10-fold cross-validation
cv_scores_NB = cross_validation.cross_val_score(nbclf, pcs, y, cv=10)
print '10-fold cross-validation scores: ', cv_scores_NB
10-fold cross-validation scores:  [ 0.93  0.9   0.89  0.91  0.91  0.96  0.95
0.93  0.88  0.91]
#overall average accuracy
print("Overall Accuracy of Naive Bayes (Gaussian): %0.2f (+/- %0.2f)" %
(cv_scores_NB.mean(), cv_scores_NB.std() * 2))
Overall Accuracy of Naive Bayes (Gaussian): 0.92 (+/- 0.05)
```

decision tree (using "entropy" as selection criteria)

```
from sklearn import tree
```

```
treeclf = tree.DecisionTreeClassifier(criterion='entropy',
min_samples_split=20, max_depth= 5)
treeclf = treeclf.fit(pcs, y)
print 'pridicted: ', treeclf.predict(pcs)
print 'Score: ', treeclf.score(pcs, y)
pridicted:  [0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0
1 0 0 0 1
 0 1 1 1 1 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 1 0 0 0
 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0
 0 1 1 0 1 1 1 1 0 0 1 0 1 0 0 1 1 1 1 1 1 0 0 1 0 1 0 0 1 0 1 1 1 1 1 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1 1 0 1 0 1 0 1 0 0 1 0 1 0
 0 1 1 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 1 1 0 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0
 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1
 1 1 0 1 1 0 0 1 1 1 1 0 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 1 0 1 0 0 0
 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 1 0 1 0
 0 1 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0
 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0 1 1 0 1 1 1 1
 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0
 1 0 1 0 0 1 0 0 0 0 0 1 0 1]
Score:  0.980667838313
#10-fold cross-validation
cv_scores_DT = cross_validation.cross_val_score(treeclf, x, y, cv=10)
print '10-fold cross-validation scores: ', cv_scores_DT
10-fold cross-validation scores:  [ 0.91  0.91  0.93  0.95  0.98  0.91  0.93
0.95  0.96  0.93]
#overall average accuracy
print("Overall Accuracy of Decision Tree: %0.2f (+/- %0.2f)" % (cv_scores_DT
.mean(), cv_scores_DT .std() * 2))
Overall Accuracy of Decision Tree: 0.94 (+/- 0.04)
```

Linear discriminant analysis (LDA)

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
ldclf = LinearDiscriminantAnalysis()
ldclf = ldclf.fit(pcs, y)
print 'pridicted: ', ldclf.predict(pcs)
print 'Score: ', ldclf.score(pcs, y)
pridicted:  [0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0
1 0 0 0 1
 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0
 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0
 0 1 1 0 1 1 1 1 0 0 1 0 1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 1 1 1 1 1 0 1 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 0 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0
 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 1 1 0 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0
 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1
 1 1 0 1 1 0 0 1 1 1 1 0 0 0 1 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 1 0 1 0 0 0
 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 1 0
 0 1 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0
 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0
 0 1 0 0 1 0 1 0 1 1 1 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 1 1 1 1
 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0
```

```
 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1]
Score:  0.957820738137
#10-fold cross-validation
cv_scores_LDA = cross_validation.cross_val_score(ldclf, x, y, cv=10)
print '10-fold cross-validation scores: ', cv_scores_LDA
10-fold cross-validation scores:  [ 0.93  0.91  0.96  0.95  0.93  1.    0.95
1.    0.96  0.98]
#overall average accuracy
print("Overall Accuracy of LDA: %0.2f (+/- %0.2f)" % (cv_scores_LDA.mean(),
cv_scores_LDA.std() * 2))
Overall Accuracy of LDA: 0.96 (+/- 0.06)
```

KNN

```
from sklearn import neighbors
# experiment K = 0~20 and weights = 'distance' then visulize it

fig= plt.figure(figsize=(10,5))
K= list(range(1,21))

for i in range(1,21):

        #Knn wiht weights='distance'
        knnclf = neighbors.KNeighborsClassifier(i, weights='distance')
        knnclf.fit(pcs, y)

        cv_scores_KNN = cross_validation.cross_val_score(knnclf, pcs, y,
cv=10)
        ave_accuracy_knn=cv_scores_KNN .mean()

        ax1=fig.add_subplot(121)
        ax1.set_title('Weight = distance')
        ax1.set_xlabel('K')
        ax1.set_ylabel('ave_accuracy_knn')
        plt.plot(i,ave_accuracy_knn,'bs')
        plt.grid(linestyle='--')
        plt.xticks(K)

        #Knn wiht weights='uniform'
        knnclf = neighbors.KNeighborsClassifier(i, weights='uniform')
        knnclf.fit(pcs, y)

        cv_scores_KNN  = cross_validation.cross_val_score(knnclf, pcs, y,
cv=10)
        ave_accuracy_knn=cv_scores_KNN.mean()

        ax2=fig.add_subplot(122)
        ax2.set_title('Weight = uniform')
        ax2.set_xlabel('K')
        ax2.set_ylabel('ave_accuracy_knn')
        plt.plot(i,ave_accuracy_knn,'ro')
        plt.grid(linestyle='--')
        plt.xticks(K)


plt.show()
```

k=6 seems to return decent accuracy without picking higher k for both

```
knnclf = neighbors.KNeighborsClassifier(6, weights='distance')
knnclf.fit(pcs, y)

cv_scores_KNN  = cross_validation.cross_val_score(knnclf, pcs, y, cv=10)
print("Overall Accuracy of LDA: %0.2f (+/- %0.2f)" % (cv_scores_KNN .mean(),
cv_scores_KNN .std() * 2))
Overall Accuracy of LDA: 0.97 (+/- 0.04)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression()
lgr.fit(pcs, y)

cv_scores_lgr = cross_validation.cross_val_score(lgr, pcs, y, cv=10)
print("Overall Accuracy of LDA: %0.2f (+/- %0.2f)" % (cv_scores_lgr.mean(),
cv_scores_lgr.std() * 2))
Overall Accuracy of LDA: 0.98 (+/- 0.05)
```

comparing accuracy across models

```
d = {'KNN': cv_scores_KNN .mean(), 'Logistic': cv_scores_lgr.mean(),'LDA':
cv_scores_LDA.mean(),'Decision Tree': cv_scores_DT .mean(),'Naive Bayes':
cv_scores_NB.mean()}
df = pd.DataFrame(d,index=["Accuracy"])
df
```

	Decision Tree	KNN	LDA	Logistic	Naive Bayes
Accuracy	0.936863	0.966723	0.959705	0.975619	0.917345

- Logistic Regression have slightly better accuracy than others

Model Evaluation

ROC Curve

```
knnP=knnclf.predict_proba(pcs)[:,1]
nbP=nbclf.predict_proba(pcs)[:,1]
treeP=treeclf.predict_proba(pcs)[:,1]
ldaP=ldclf.predict_proba(pcs)[:,1]
lgrP=lgr.predict_proba(pcs)[:,1]
models=[knnP,nbP,treeP,ldaP,lgrP]
label=['KNN','Naive Bayes','Decision Tree','LDA','Logistic']
from sklearn import metrics
# plotting ROC curves
plt.figure(figsize=(20, 15))
```

```
for i in range(len(models)):
    fpr, tpr,thresholds= metrics.roc_curve(y,models[i])
    plt.plot(fpr,tpr,label=label[i])
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.title('ROC curve for Cancer classifer')
plt.xlabel('False positive rate (1-specificity)')
plt.ylabel('True positive rate (sensitivity)')
plt.legend(loc=4,)
<matplotlib.legend.Legend at 0x11b723490>
```

From the ROC curve:

- Naive Bayee has the worst performace than others. KNN and logistic stand out to be a better models in our cases.

Conclusion

After examine all the classifcation methods with the reduced features(30 attributes to 10 pcs which cover ~95% of varaition), We think the best classifaciton technique for breast cancer is logistic regression which give us ~98% average accuracy with 10 fold cross validation. Roc curves also comfrim that logistic regression have good performace.