EMORY UNIVERSITY

Introduction to Business Analytics [MSBA]

Homework #2 **(HW2 is Due on Friday, September 21 11.59pm)**

Adil Ashish Kumar

(put your full name above (incl. any nicknames); 5 points)

Note: This is an individual homework. Discussing this homework with your classmates is a violation of the Honor Code. If you borrow code from somewhere else, please add a comment in your code to make it clear what the source of the code is (e.g., a URL would sufficient). If you borrow code and you don't provide the source, it is a violation of the Honor Code.

Total grade: ________ out of ___150___ points

1) (15 points) Would you frame the problem of e-mail spam detection as a supervised learning problem or an unsupervised learning problem? Please justify your answer.

The problem is a supervised learning problem because the goal is to classify an email as spam or not. The target variable in this case would be spam/no spam column having 1 for spam and 0 otherwise. Here, the assumption is that we have a dataset with a column to label an email as spam or not. Unlike in unsupervised learning, there is a specific goal/purpose (spam email prediction) in this problem, which makes it a supervised learning problem. This would be a classification problem since the target variable would be categorical.

2) (15 points) What is a test set and why would you want to use it?

A test set is a part of the data on which we do not train the model. After the model is trained on training data, we evaluate model performance by making predictions on the test dataset and compare the predictions with the target variable in test. Generally, the about 20-30% of the data is set aside for test and about 70-80% of the data would be used to train the model. These numbers may vary based on the size of the dataset. The test dataset should not be used as part of the modeling process, but rather to evaluate the performance of a model.

A test dataset is crucial to generalization performance. Without testing the model on data it has not seen, it is hard to determine the model accuracy. Metrics such as out of sample accuracy, Precision, recall, F measure and error rate give a picture of model performance on test data. With help of test data, we can also create fitting graphs to identify the problem of overfitting by observing error% as model complexity is increased.

3) (20 points) What are the similarities and differences of decision trees and logistic regression? When might you prefer to use one over another?

Differences:

Tree models are more flexible than logistic regression since they can handle nonlinear relationships in the data. In classification trees, decision boundaries are perpendicular to the instance space axes and there can be multiple decision boundaries, but in logistic regression decision boundaries can have any orientation. In Logistic regression, a single decision surface is placed through the entire space, while in tree models, the instance space can be split into small regions. Tree models consider 1 attribute at a time for each split from parent to child node, but logistic regression considers all predictor variables together in model. Model interpretability is better for decision trees since the output of model is a set of rules that is easy to understand, while output of logistic regression is a set of model coefficients which are not easy to understand for non-statistically savvy audience.

Similarities: Both models can be used for classification problems, but logistic regression cannot be used for regression. Both models can overcome overfitting problems – tree models use pre and post pruning techniques while logistic regression uses regularization and feature selection. Probability estimation can be done by both models.

Usage: Logistic regression can be used only for classification problems but trees can be used for both classification and regression problems. Logistic regression is preferred over tree models for smaller datasets. Tree models need more data in general. Tree models can handle datasets with non linear relationships, missing values, better than logistic regression. Tree models have better model comprehensibility than logistic regression. Flexibility of tree models give it an advantage for use on larger datasets.

4) (25 points) You have a fraud detection task (predicting whether a given credit card transaction is "fraud" vs. "non-fraud") and you built a classification model for this purpose. For any credit card transaction, your model estimates the probability that this transaction is "fraud". The following table represents the probabilities that your model estimated for the validation dataset containing 10 records.

Actual Class (from validation data)	Estimated Probability of Record Belonging to Class "fraud"
fraud	0.95
fraud	0.91
fraud	0.75
non-fraud	0.67
fraud	0.61
non-fraud	0.46
fraud	0.42
non-fraud	0.25
non-fraud	0.09
non-fraud	0.04

Based on the above information, answer the following questions:

a) **What is the overall accuracy of your model, if the chosen probability cutoff value is 0.3? What is the overall accuracy of your model, if the chosen probability cutoff value is 0.8?**

Actual Class (from validation data)	Estimated Probability of Record Belonging to Class "fraud"	Prediction (0.3)	Prediction (0.8)	Prediction (0.7)	Prediction (0.4)
fraud	0.95	Fraud	Fraud	Fraud	Fraud
fraud	0.91	Fraud	Fraud	Fraud	Fraud
fraud	0.75	Fraud	non-fraud	Fraud	Fraud
non-fraud	0.67	Fraud	non-fraud	Non	Fraud
fraud	0.61	Fraud	non-fraud	Non	Fraud
non-fraud	0.46	Fraud	non-fraud	Non	Fraud
fraud	0.42	Fraud	non-fraud	Non	Fraud
non-fraud	0.25	non-fraud	non-fraud	Non	Non
non-fraud	0.09	non-fraud	non-fraud	Non	Non
non-fraud	0.04	non-fraud	non-fraud	Non	Non

Cutoff = 0.3, accuracy = 8/10 = 80%

Cutoff =0.8, accuracy = 7/10 =70%

b) **What probability cutoff value should you choose, in order to have Precision fraud = 100% for your model? (Explain.) What is the overall accuracy of your model in this case?**

A cutoff value of 0.7 would result in a 100% precision fraud. The overall accuracy for the model would be 80% Below is the confusion matrix for the same:

	Actual fraud	Actual Non fraud
Predicted fraud	3	0
Predicted Non fraud	2	5

c) **What probability cutoff value should you choose, in order to have Recall fraud = 100% for your model? (Explain.) What is the overall accuracy of your model in this case?**

A cutoff value of 0.4 would result in a 100% recall fraud. The overall accuracy for the model would be 80% Below is the confusion matrix for the same:

	Actual fraud	Actual Non fraud
Predicted fraud	5	2
Predicted Non fraud	0	3

d) **Draw an ROC curve for your model.**

5) (70 points) [Mining publicly available data. Please implement the following models both with Rapidminer and Python but explore the data (e.g., descriptive statistics etc.) just with Python]

Please use the dataset on breast cancer research from this link: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data [Note: Rapidminer can import .data files in the same way it can import .csv files. For Python please read the data directly from the URL without downloading the file on your local disk.**] The description of the data and attributes can be found at this link: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.names . Each record of the data set represents a different case of breast cancer. Each case is described with 30 real-valued attributes: attribute 1 represents case id, attributes 3-32 represent various physiological characteristics, and attribute 2 represents the type (benign or malignant). If the dataset has records with missing values, you can filter out these records using Python and/or Rapidminer before proceeding with the models. Alternatively, if the data set has missing values, you could infer the missing values.**

Perform a predictive modeling analysis on this same dataset using the a) k-NN technique (for k=3) and b) Logistic Regression. Please be specific about what other parameters you specified for your models.

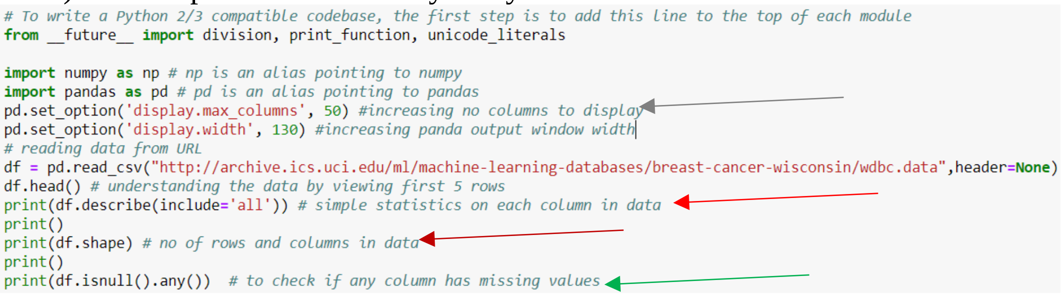
Present a brief overview of your predictive modeling process, explorations, and discuss your results.

Compare the k-NN model with the Logistic Regression model: which performs better? Make sure you present information about the model "goodness" (i.e., confusion matrix, predictive accuracy, precision, recall, f-measure). Please be clear about any assumptions you might make when you choose the best performing model.

Please show screenshots of the models you have built with Rapidminer and Python, show screenshots of the performance results, and the parameters you have specified.

Brief overview of modeling process:

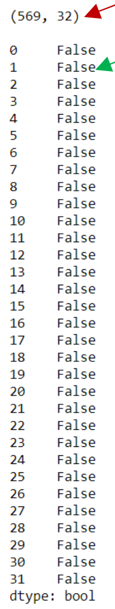
1) Data Input and Preliminary analysis



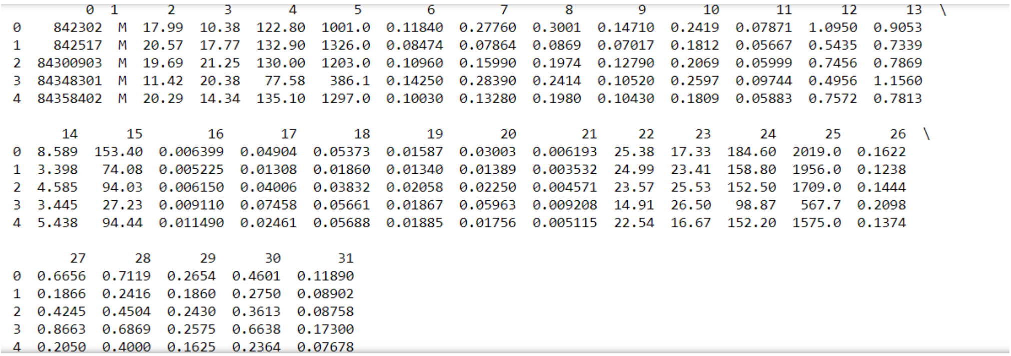
```
# To write a Python 2/3 compatible codebase, the first step is to add this line to the top of each module
from __future__ import division, print_function, unicode_literals

import numpy as np # np is an alias pointing to numpy
import pandas as pd # pd is an alias pointing to pandas
pd.set_option('display.max_columns', 50) #increasing no columns to display
pd.set_option('display.width', 130) #increasing panda output window width
# reading data from URL
df = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data",header=None)
df.head() # understanding the data by viewing first 5 rows
print(df.describe(include='all')) # simple statistics on each column in data
print()
print(df.shape) # no of rows and columns in data
print()
print(df.isnull().any())  # to check if any column has missing values
```

In this step, the data (.data file) is read in python directly from the URL, with "header = None" to indicate the first row is not a header row. The output width and column limit is adjusted to make sure we can explore the data properly since the data has 32 columns. A quick preliminary analysis is done on the data to get an idea about the structure of the data. The df.shape command tells us that the data has 569 rows and 32 columns. The df.isnull().any() command shows there are no missing values in the data, which indicates no missing value treatment needs to be carried out.

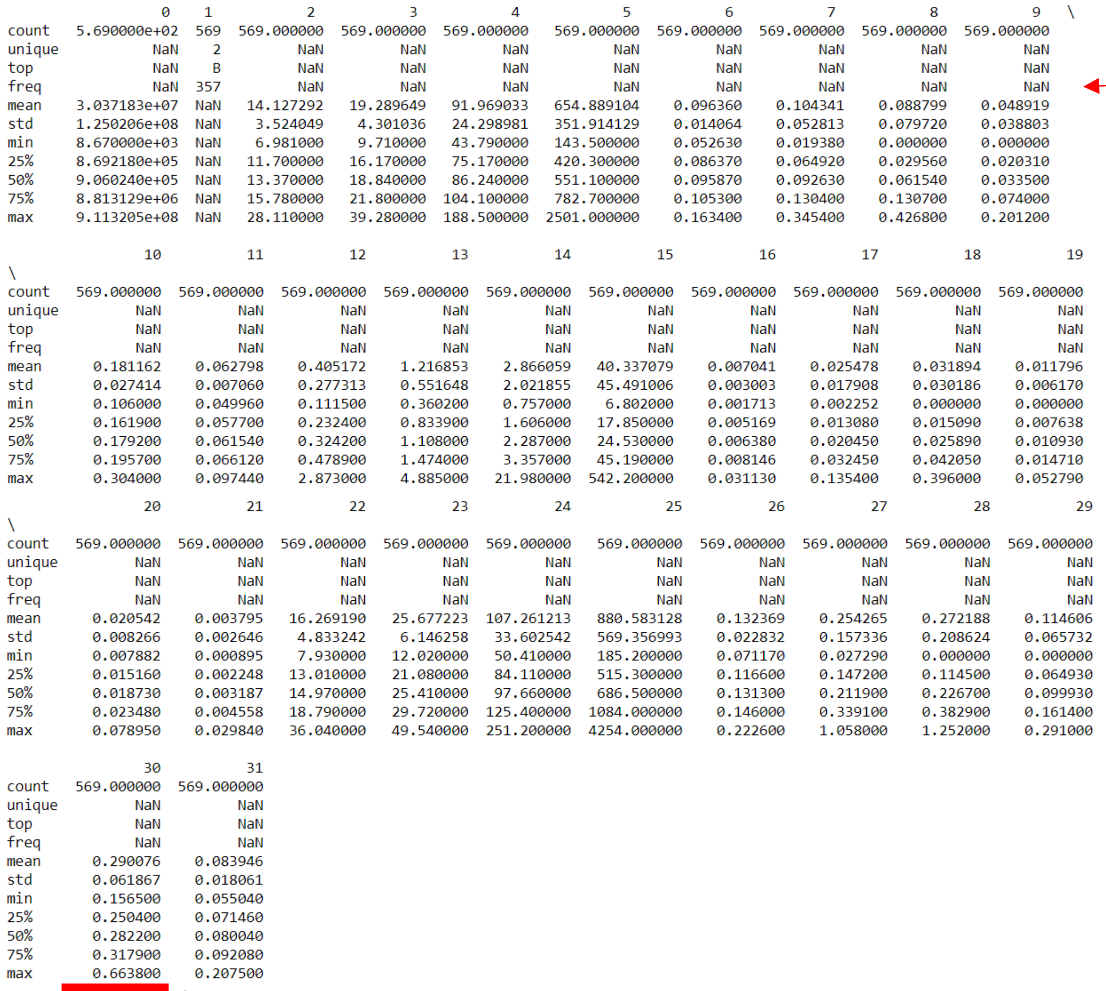
Using df.head(), a sample of the data is printed to get an idea of the structure of the data. As given by the data description, the first column is the case ID, and each row of the data is a unique case. The second column is the target variable, indicating malignancy or benign. All the other columns are numeric attributes.



```
(569, 32)

0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12    False
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
21    False
22    False
23    False
24    False
25    False
26    False
27    False
28    False
29    False
30    False
31    False
dtype: bool
```



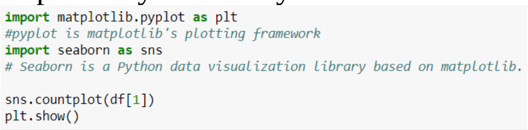
```
         0  1      2      3       4       5        6        7       8        9      10       11      12      13  \
0    842302  M  17.99  10.38  122.80  1001.0  0.11840  0.27760  0.3001  0.14710  0.2419  0.07871  1.0950  0.9053
1    842517  M  20.57  17.77  132.90  1326.0  0.08474  0.07864  0.0869  0.07017  0.1812  0.05667  0.5435  0.7339
2  84300903  M  19.69  21.25  130.00  1203.0  0.10960  0.15990  0.1974  0.12790  0.2069  0.05999  0.7456  0.7869
3  84348301  M  11.42  20.38   77.58   386.1  0.14250  0.28390  0.2414  0.10520  0.2597  0.09744  0.4956  1.1560
4  84358402  M  20.29  14.34  135.10  1297.0  0.10030  0.13280  0.1980  0.10430  0.1809  0.05883  0.7572  0.7813

      14      15        16       17       18       19       20        21     22     23      24      25      26  \
0  8.589  153.40  0.006399  0.04904  0.05373  0.01587  0.03003  0.006193  25.38  17.33  184.60  2019.0  0.1622
1  3.398   74.08  0.005225  0.01308  0.01860  0.01340  0.01389  0.003532  24.99  23.41  158.80  1956.0  0.1238
2  4.585   94.03  0.006150  0.04006  0.03832  0.02058  0.02250  0.004571  23.57  25.53  152.50  1709.0  0.1444
3  3.445   27.23  0.009110  0.07458  0.05661  0.01867  0.05963  0.009208  14.91  26.50   98.87   567.7  0.2098
4  5.438   94.44  0.011490  0.02461  0.05688  0.01885  0.01756  0.005115  22.54  16.67  152.20  1575.0  0.1374

       27      28      29      30       31
0  0.6656  0.7119  0.2654  0.4601  0.11890
1  0.1866  0.2416  0.1860  0.2750  0.08902
2  0.4245  0.4504  0.2430  0.3613  0.08758
3  0.8663  0.6869  0.2575  0.6638  0.17300
4  0.2050  0.4000  0.1625  0.2364  0.07678
```



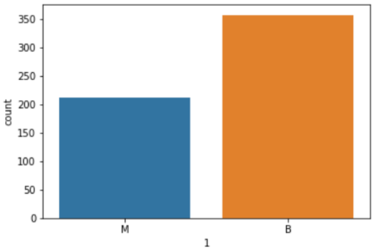
```
                   0    1           2           3           4            5           6           7           8           9  \
count   5.690000e+02  569  569.000000  569.000000  569.000000   569.000000  569.000000  569.000000  569.000000  569.000000
unique           NaN    2         NaN         NaN         NaN          NaN         NaN         NaN         NaN         NaN
top              NaN    B         NaN         NaN         NaN          NaN         NaN         NaN         NaN         NaN
freq             NaN  357         NaN         NaN         NaN          NaN         NaN         NaN         NaN         NaN
mean    3.037183e+07  NaN   14.127292   19.289649   91.969033   654.889104    0.096360    0.104341    0.088799    0.048919
std     1.250206e+08  NaN    3.524049    4.301036   24.298981   351.914129    0.014064    0.052813    0.079720    0.038803
min     8.670000e+03  NaN    6.981000    9.710000   43.790000   143.500000    0.052630    0.019380    0.000000    0.000000
25%     8.692180e+05  NaN   11.700000   16.170000   75.170000   420.300000    0.086370    0.064920    0.029560    0.020310
50%     9.060240e+05  NaN   13.370000   18.840000   86.240000   551.100000    0.095870    0.092630    0.061540    0.033500
75%     8.813129e+06  NaN   15.780000   21.800000  104.100000   782.700000    0.105300    0.130400    0.130700    0.074000
max     9.113205e+08  NaN   28.110000   39.280000  188.500000  2501.000000    0.163400    0.345400    0.426800    0.201200

                10          11          12          13          14          15          16          17          18          19
\
count   569.000000  569.000000  569.000000  569.000000  569.000000  569.000000  569.000000  569.000000  569.000000  569.000000
unique         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
top            NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
freq           NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
mean      0.181162    0.062798    0.405172    1.216853    2.866059   40.337079    0.007041    0.025478    0.031894    0.011796
std       0.027414    0.007060    0.277313    0.551648    2.021855   45.491006    0.003003    0.017908    0.030186    0.006170
min       0.106000    0.049960    0.111500    0.360200    0.757000    6.802000    0.001713    0.002252    0.000000    0.000000
25%       0.161900    0.057700    0.232400    0.833900    1.606000   17.850000    0.005169    0.013080    0.015090    0.007638
50%       0.179200    0.061540    0.324200    1.108000    2.287000   24.530000    0.006380    0.020450    0.025890    0.010930
75%       0.195700    0.066120    0.478900    1.474000    3.357000   45.190000    0.008146    0.032450    0.042050    0.014710
max       0.304000    0.097440    2.873000    4.885000   21.980000  542.200000    0.031130    0.135400    0.396000    0.052790
                20          21          22          23          24           25          26          27          28          29
\
count   569.000000  569.000000  569.000000  569.000000  569.000000   569.000000  569.000000  569.000000  569.000000  569.000000
unique         NaN         NaN         NaN         NaN         NaN          NaN         NaN         NaN         NaN         NaN
top            NaN         NaN         NaN         NaN         NaN          NaN         NaN         NaN         NaN         NaN
freq           NaN         NaN         NaN         NaN         NaN          NaN         NaN         NaN         NaN         NaN
mean      0.020542    0.003795   16.269190   25.677223  107.261213   880.583128    0.132369    0.254265    0.272188    0.114606
std       0.008266    0.002646    4.833242    6.146258   33.602542   569.356993    0.022832    0.157336    0.208624    0.065732
min       0.007882    0.000895    7.930000   12.020000   50.410000   185.200000    0.071170    0.027290    0.000000    0.000000
25%       0.015160    0.002248   13.010000   21.080000   84.110000   515.300000    0.116600    0.147200    0.114500    0.064930
50%       0.018730    0.003187   14.970000   25.410000   97.660000   686.500000    0.131300    0.211900    0.226700    0.099930
75%       0.023480    0.004558   18.790000   29.720000  125.400000  1084.000000    0.146000    0.339100    0.382900    0.161400
max       0.078950    0.029840   36.040000   49.540000  251.200000  4254.000000    0.222600    1.058000    1.252000    0.291000

                30          31
count   569.000000  569.000000
unique         NaN         NaN
top            NaN         NaN
freq           NaN         NaN
mean      0.290076    0.083946
std       0.061867    0.018061
min       0.156500    0.055040
25%       0.250400    0.071460
50%       0.282200    0.080040
75%       0.317900    0.092080
max       0.663800    0.207500
```

The describe function describes each column in the data and gives simple statistics. There are only 2 unique classes in target variable, which is in agreement with the data description. Since we do not have column descriptions, it is difficult to comment on the distribution of the predictor variables.

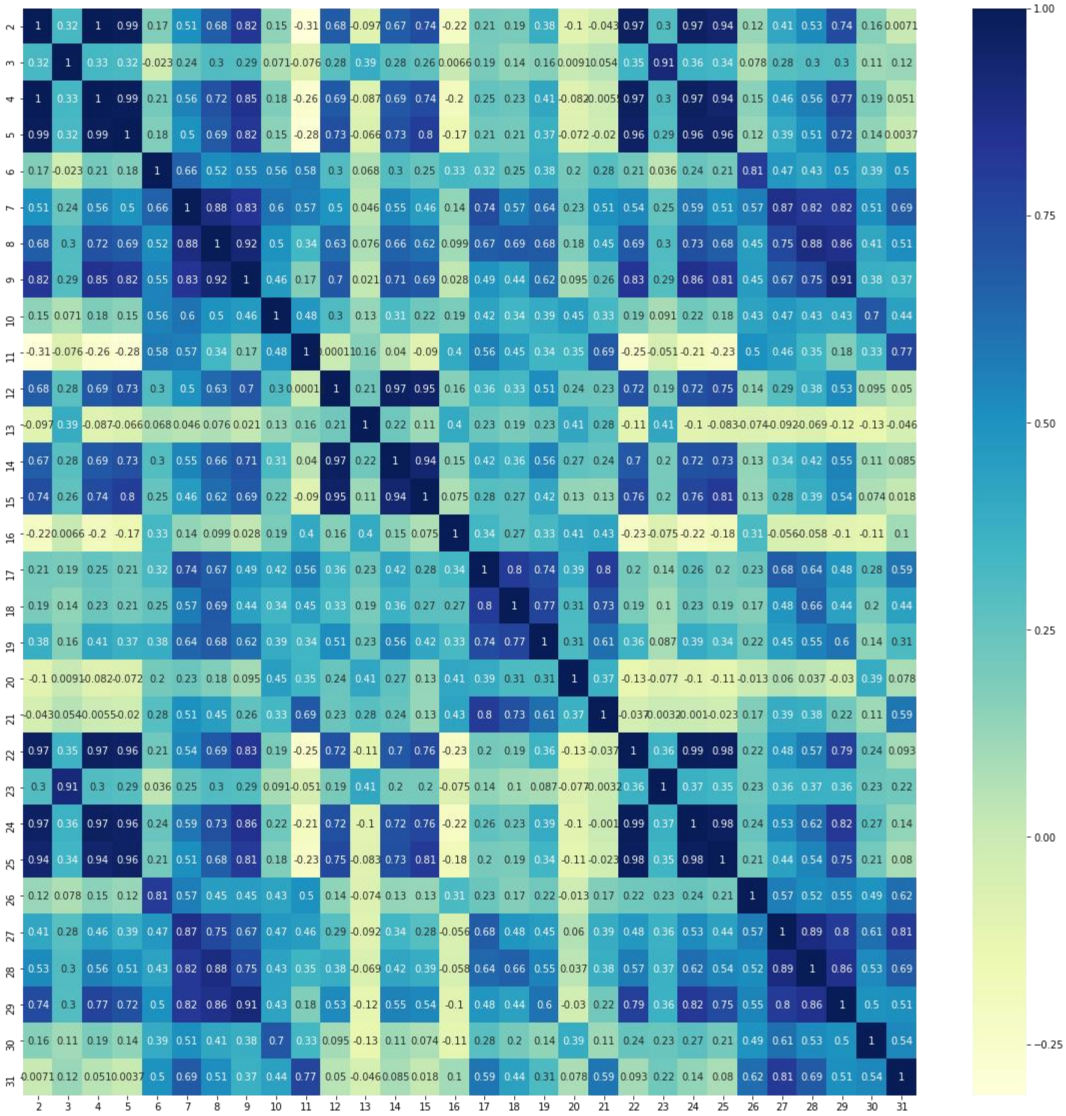
2) Exploratory data analysis & Visualization



```
import matplotlib.pyplot as plt
#pyplot is matplotlib's plotting framework
import seaborn as sns
# Seaborn is a Python data visualization library based on matplotlib.

sns.countplot(df[1])
plt.show()
```

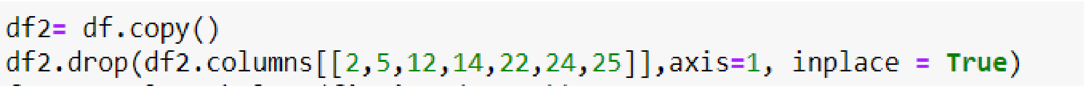


Looking at a histogram of the target variable, we observe that there are 357(63%) instances of benign and 212(37%) instances of malignancy, indicating slight a class imbalance in the data. While this is not an ideal 50:50 distribution, we continue without making any changes to the distribution. There are certain techniques like up-sampling and down-sampling that can be used to alter the class distribution of target variable, but for the sake of simplicity, we leave the target variable as is.



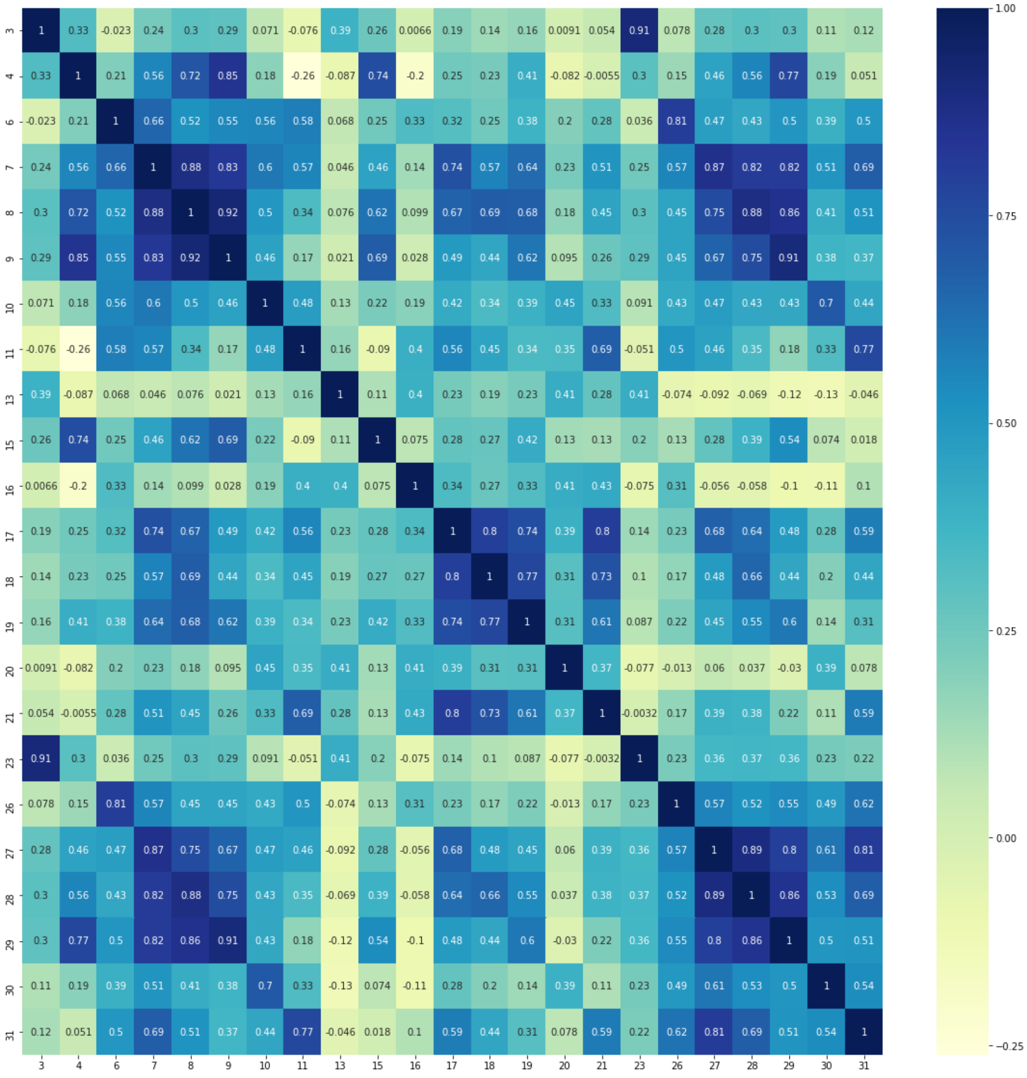
Looking at the correlation heat map, we observe possible cases of multicollinearity. This can be seen in the rather high correlation values(>0.93) between the following pairs of variables – (2,5) (2,22) (2,24) (2,25) (4,5) (4,22) (4,24) (4,25) (5,22) (5,24) (5,25) (12,14) (12,15) (14,15) (22,24) (22,25).

Since multicollinearity may lead to overfitting, we drop the following variables that seem to be causing the multicollinearity – 2,5,12,14,22,24,25



```
df2= df.copy()
df2.drop(df2.columns[[2,5,12,14,22,24,25]],axis=1, inplace = True)
```

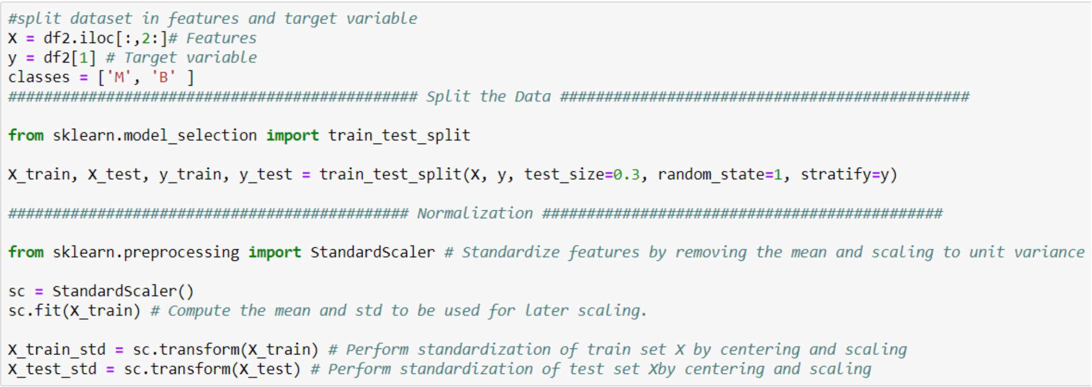
The correlation heatmap after dropping the variables is below, we notice that the extreme correlations of >0.93 are removed, which should have reduced the problem of multicollinearity:



3) Splitting data and standardization

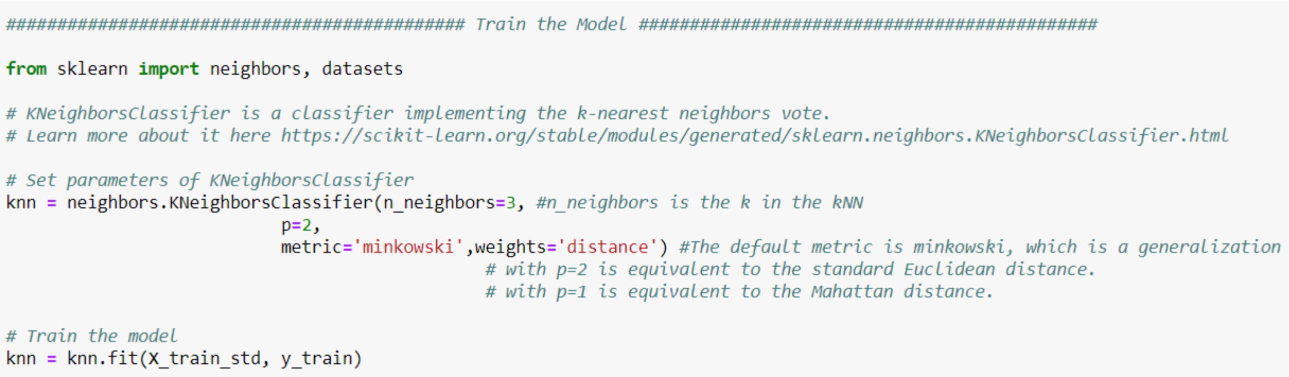
In this step, the data is split into train and test in the ratio 70:30 respectively. The stratified sampling is used to make sure the distribution of both classes is consistent across and test and train data. Since the KNN modeling uses distance as a measure to find nearest neighbors, it is important to standardize the variables so that the distance is unaffected by scaling of the variables. We use the Z score scaling method to standardize the variables.

Splitting data and standardization



```
#split dataset in features and target variable
X = df2.iloc[:,2:]# Features
y = df2[1] # Target variable
classes = ['M', 'B' ]
############################################### Split the Data ###############################################

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)

############################################### Normalization ###############################################

from sklearn.preprocessing import StandardScaler # Standardize features by removing the mean and scaling to unit variance

sc = StandardScaler()
sc.fit(X_train) # Compute the mean and std to be used for later scaling.

X_train_std = sc.transform(X_train) # Perform standardization of train set X by centering and scaling
X_test_std = sc.transform(X_test) # Perform standardization of test set Xby centering and scaling
```

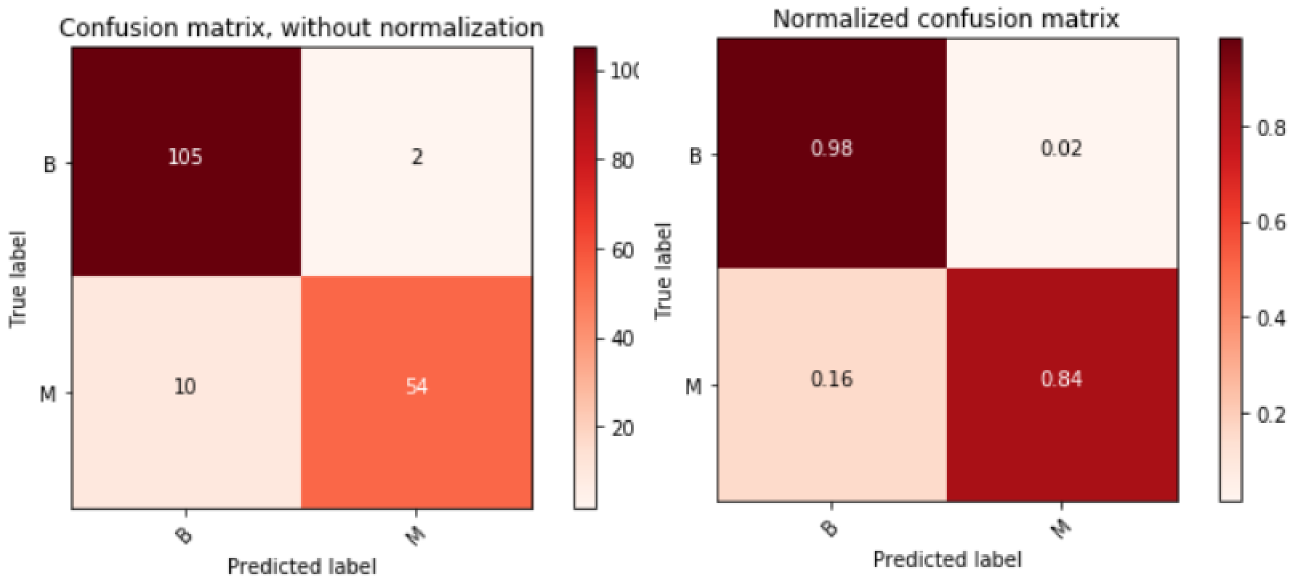
4) KNN model induction

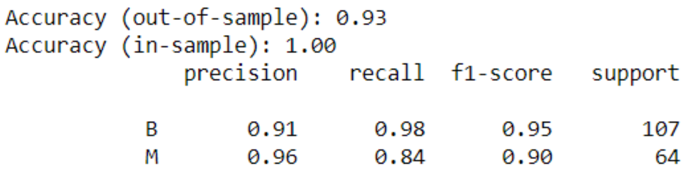


```
############################################## Train the Model ##############################################

from sklearn import neighbors, datasets

# KNeighborsClassifier is a classifier implementing the k-nearest neighbors vote.
# Learn more about it here https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

# Set parameters of KNeighborsClassifier
knn = neighbors.KNeighborsClassifier(n_neighbors=3, #n_neighbors is the k in the kNN
                                     p=2,
                                     metric='minkowski',weights='distance') #The default metric is minkowski, which is a generalization
                                                                # with p=2 is equivalent to the standard Euclidean distance.
                                                                # with p=1 is equivalent to the Mahattan distance.

# Train the model
knn = knn.fit(X_train_std, y_train)
```

In this step, we train a KNN classifier with k=3 on the standardized training and test data. The distance metric used is a generalization of the Euclidean distance, which is specified by p=2. The weights parameter is specified as “distance”, which uses a weighted voting for nearest neighbors, indicating that points nearer will have a greater contribution in class prediction. All other modeling parameters are left as default.

5) KNN model evaluation



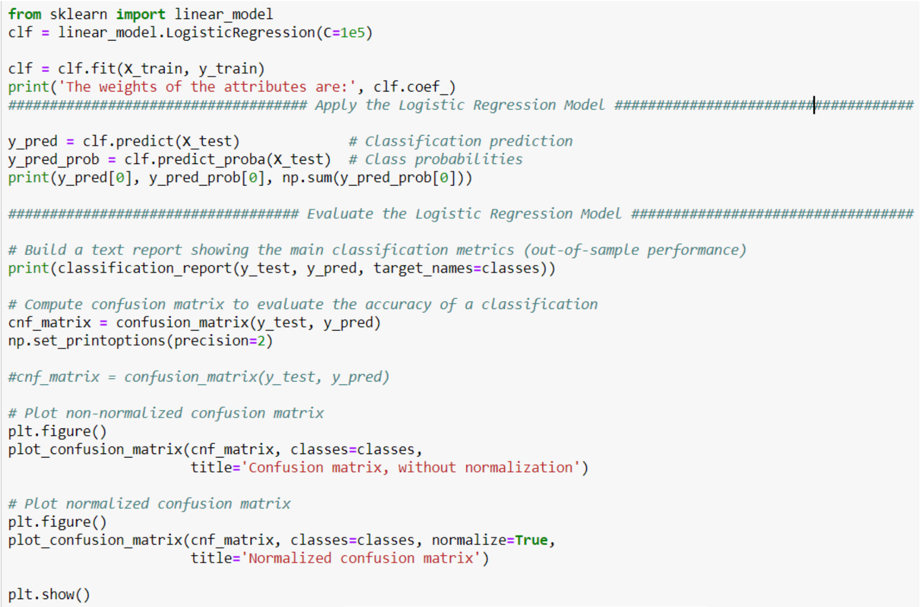


```
Accuracy (out-of-sample): 0.93
Accuracy (in-sample): 1.00
              precision    recall  f1-score   support

           B       0.91      0.98      0.95       107
           M       0.96      0.84      0.90        64
```

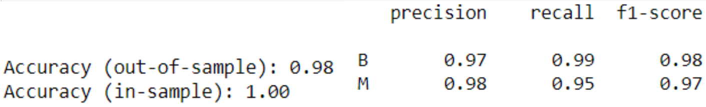
The KNN model has an out of sample accuracy score of 93%. It does a pretty good job with the classification task on both classes, with an F measure of 0.95 for B class and 0.9 for M class, which are very high values, indicating the model is performing well. The precision measure is very high (>0.91) for both classes. The recall metric is nearly =1 for B class (0.98) but not as good for M class (0.84). Looking deeper into this, we see from the confusion matrix that the model wrongly classifies 10 cases of malignancy as Benign. This is an issue since we need to consider the cost of misclassification in both classes. We will look to see if logistic regression can perform better in classifying more malignant classes accurately.

6) Logistic Regression model induction and evaluation

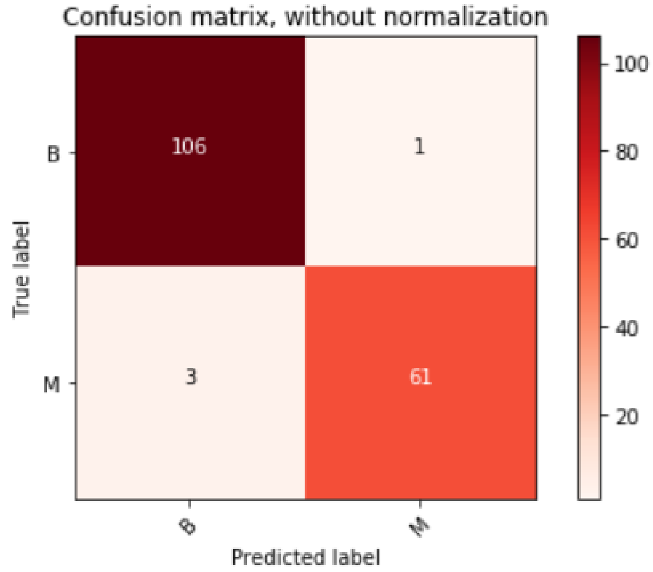
In this step, we train a logistic regression model to carry out the same classification process. It is important to note that we do not use the standardized form of the data here, but rather use the raw unstandardized data. In this case, we use the default penalty option of L2 norm. The C value used is 10000, which is the value for inverse of regularization strength. All other parameters in the modeling are left as default.

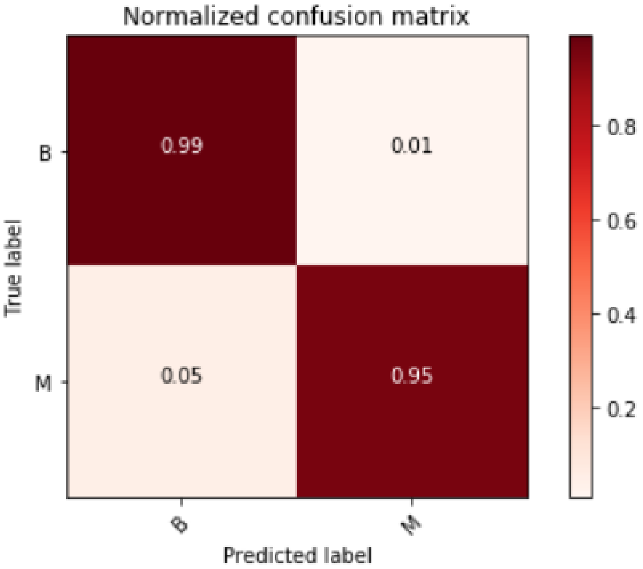


```python
from sklearn import linear_model
clf = linear_model.LogisticRegression(C=1e5)

clf = clf.fit(X_train, y_train)
print('The weights of the attributes are:', clf.coef_)
#################################### Apply the Logistic Regression Model ####################################

y_pred = clf.predict(X_test)             # Classification prediction
y_pred_prob = clf.predict_proba(X_test)  # Class probabilities
print(y_pred[0], y_pred_prob[0], np.sum(y_pred_prob[0]))

################################### Evaluate the Logistic Regression Model ##################################

# Build a text report showing the main classification metrics (out-of-sample performance)
print(classification_report(y_test, y_pred, target_names=classes))

# Compute confusion matrix to evaluate the accuracy of a classification
cnf_matrix = confusion_matrix(y_test, y_pred)
np.set_printoptions(precision=2)

#cnf_matrix = confusion_matrix(y_test, y_pred)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=classes,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=classes, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```



```
                                           precision    recall  f1-score

Accuracy (out-of-sample): 0.98   B       0.97      0.99      0.98
Accuracy (in-sample): 1.00       M       0.98      0.95      0.97
```





The logistic regression model has an out of sample accuracy of 98%. The F measure for B class is 0.98, and for M class is 0.97. These measures are almost =1, indicating the model is performing very well in classifying both classes. Unlike the KNN model, where recall for class M was lower than that of B, here recall scores for both classes are very high (0.95 for M and 0.99 for B). The precision scores for both classes are near perfect as well (0.97 for B and 0.98 for M). On all metrics, logistic regression model has a superior performance in comparison to KNN model.

7) KNN vs Logistic regression

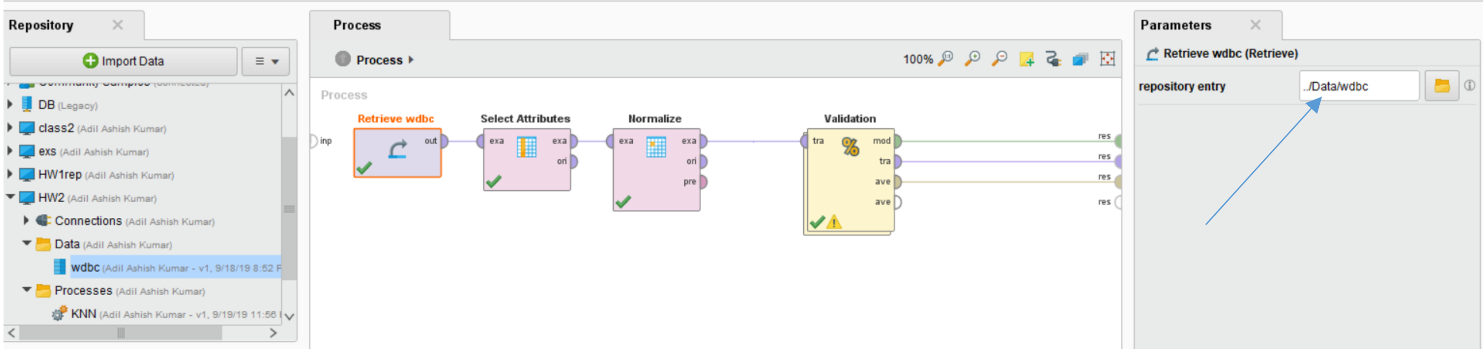
In terms of out of sample accuracy, Logistic regression (98%) performs better than KNN (93%). The F measures for Logistic regression for both classes "B", "M" are better than KNN. Just on performance, it seems like logistic regression performs better than KNN.

An important differentiating factor is the F measure for both models. In KNN, the model predicts about 16% of the malignant cases as benign. This is a major issue, as there are serious costs and consequences involved in wrongly predicting occurrence of a disease like cancer. Thus, I would prefer to use Logistic regression model, as it has much better accuracy in predicting both benign and malignant classes, which are evident from the F measures for B and M classes.

An important consideration is that logistic regression may not be very suitable for very large datasets, so the size of data should be considered when choosing the model for deployment. Another important point is that logistic regression model may not be able to handle complexities in data like missing values, non linearities etc, so additional data preparation maybe needed if the data has such issues. Model comprehensibility is also an important factor and logistic regression is not an easy model to understand. Here I assume that the dataset size is small enough to be suitable for logistic regression. I also assume that model generalization performance is more important than model interpretability. Thus, I would suggest using the logistic regression model.

Rapid miner implementation

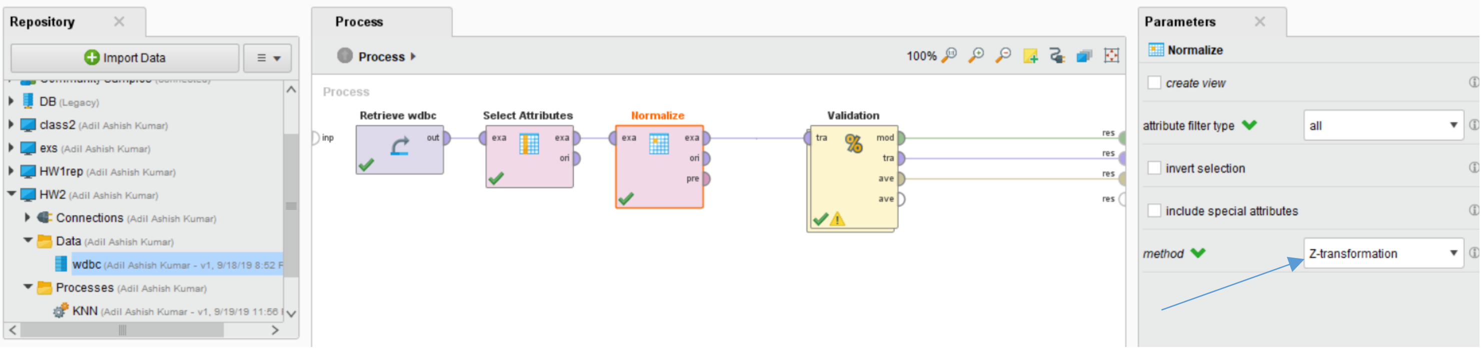
1) KNN



Retrieve wdbc (Retrieve)

repository entry ../Data/wdbc

The modeling process in rapidminer is similar to that of python. First the data is read in using the retrieve module. The select attribute module is used to drop the case ID and other columns that were causing multicollinearity with help of subset option. Then the data is normalized using the Z-score method, where mean of data is subtracted from all values and divided by the std deviation. We use a 70:30 split for train and test data respectively in the validation module. The stratified sampling method is used so that class distribution is identical for target variable in both train and test data


Repository
Import Data
DB (Legacy)
class2 (Adil Ashish Kumar)
exs (Adil Ashish Kumar)
HW1rep (Adil Ashish Kumar)
HW2 (Adil Ashish Kumar)
Connections (Adil Ashish Kumar)
Data (Adil Ashish Kumar)
wdbc (Adil Ashish Kumar - v1, 9/18/19 8:52
Processes (Adil Ashish Kumar)
KNN (Adil Ashish Kumar - v1, 9/19/19 11:56
Process
100%
Retrieve wdbc
Select Attributes
Normalize
Validation
Parameters

Normalize

☐ create view

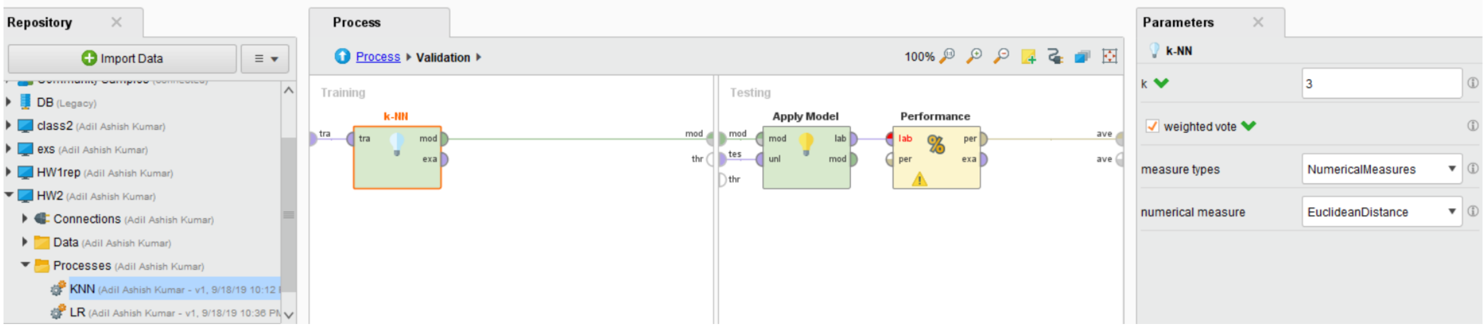
attribute filter type all

☐ invert selection

☐ include special attributes

method Z-transformation

Under the validation module, we use the knn module, the apply module and the performance module. In KNN module, we use K=3, choose the distance measure as Euclidean distance. Since all predictor attributes in our model are numerical, we choose the measure type as “numerical”. In the performance module, I have chosen to see metrics like accuracy, precision, recall and f measure.


Repository
Import Data
DB (Legacy)
class2 (Adil Ashish Kumar)
exs (Adil Ashish Kumar)
HW1rep (Adil Ashish Kumar)
HW2 (Adil Ashish Kumar)
Connections (Adil Ashish Kumar)
Data (Adil Ashish Kumar)
Processes (Adil Ashish Kumar)
KNN (Adil Ashish Kumar - v1, 9/18/19 10:12
LR (Adil Ashish Kumar - v1, 9/18/19 10:36 PM
Process
Validation
100%
Training
Testing
k-NN
Apply Model
Performance
Parameters

k-NN

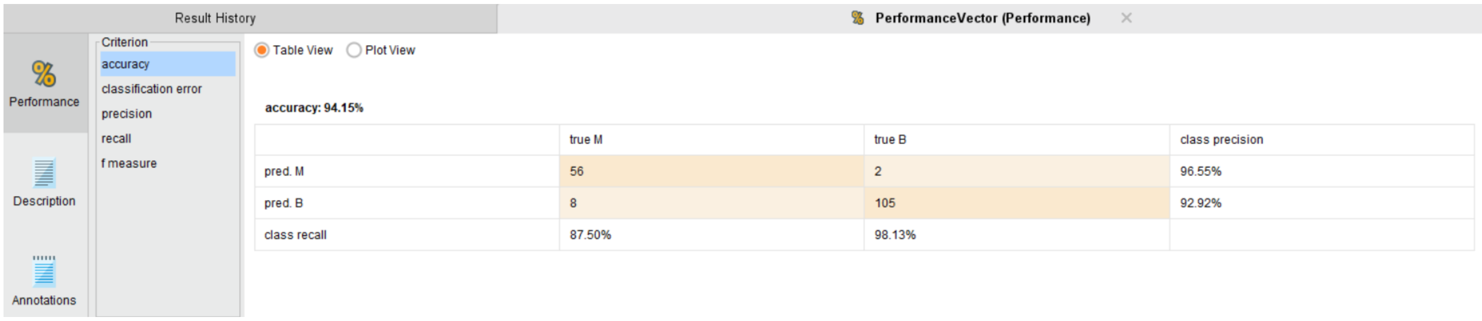
k 3

☑ weighted vote

measure types NumericalMeasures

numerical measure EuclideanDistance

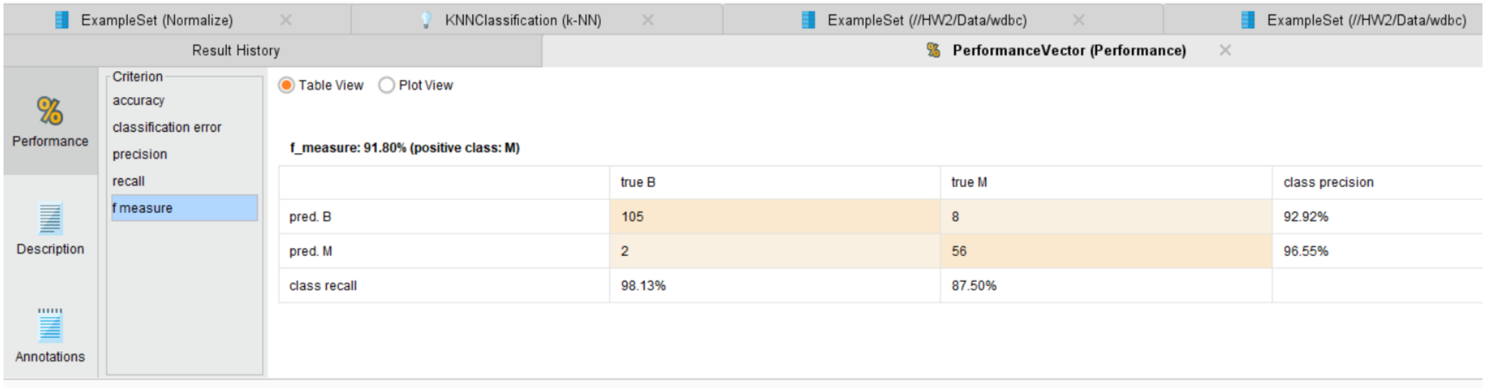
On running the model, we get an out of sample accuracy of 94.15%. The F measure for class “M” is 91.8%. The F measure and accuracy are high (>0.9), indicating the model is doing a good job with the classification task. The confusion matrix shows that the model is doing a good job with predicting both classes.



● Table View ○ Plot View

accuracy: 94.15%

	true M	true B	class precision
pred. M	56	2	96.55%
pred. B	8	105	92.92%
class recall	87.50%	98.13%	

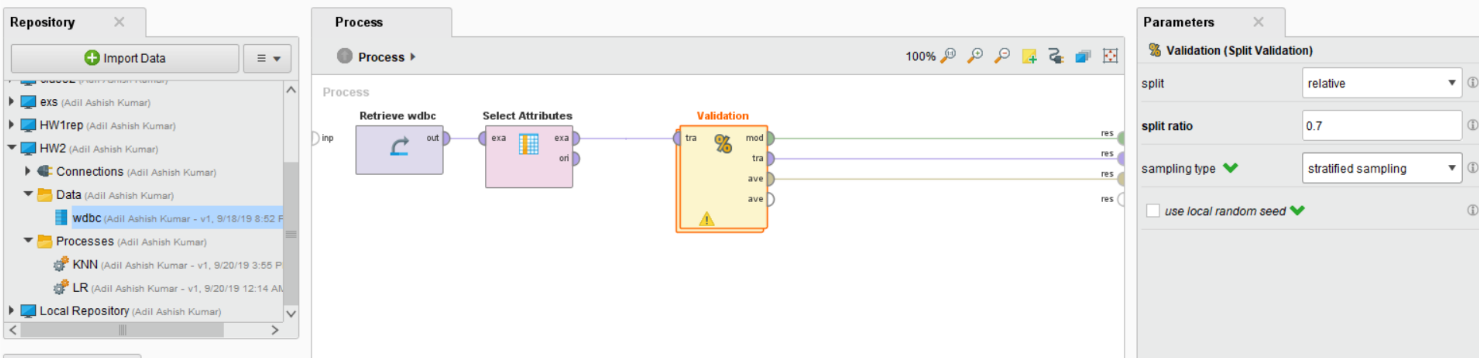


● Table View ○ Plot View

f_measure: 91.80% (positive class: M)

	true B	true M	class precision
pred. B	105	8	92.92%
pred. M	2	56	96.55%
class recall	98.13%	87.50%	

2) Logistic regression


Repository
Import Data
exs (Adil Ashish Kumar)
HW1rep (Adil Ashish Kumar)
HW2 (Adil Ashish Kumar)
Connections (Adil Ashish Kumar)
Data (Adil Ashish Kumar)
wdbc (Adil Ashish Kumar - v1, 9/18/19 8:52 P
Processes (Adil Ashish Kumar)
KNN (Adil Ashish Kumar - v1, 9/20/19 3:55 P
LR (Adil Ashish Kumar - v1, 9/20/19 12:14 AM
Local Repository (Adil Ashish Kumar)
Process
Process
100%
Retrieve wdbc
Select Attributes
Validation
inp
out
exa
exa
ori
tra
mod
tra
ave
ave
res
res
res
res
Parameters
Validation (Split Validation)

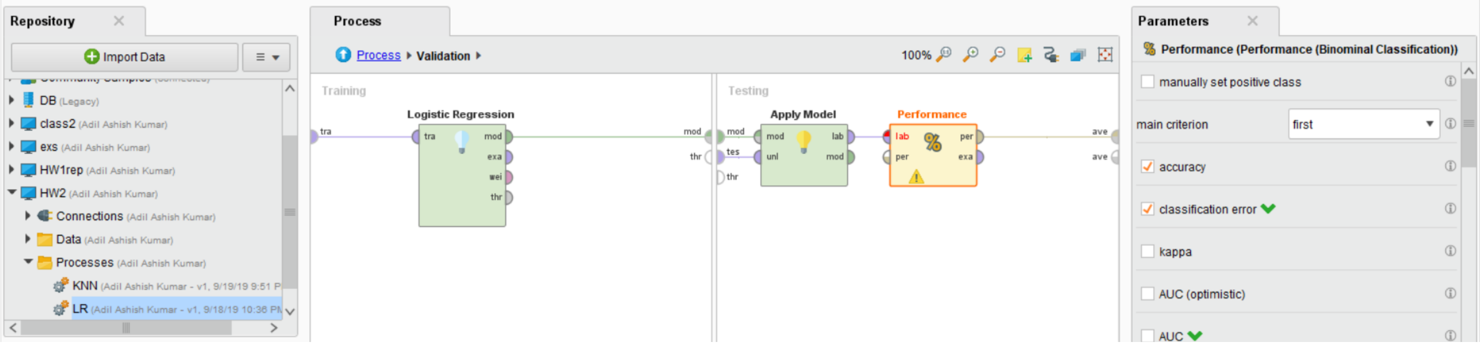
split: relative

split ratio: 0.7

sampling type: stratified sampling

□ use local random seed

In the logistic regression modeling process, we do not use the standardization process on the data, but the raw data itself. The select attribute module is used to filter out columns that were seen to cause multicollinearity during data exploration process. In the validation module, we set a 70:30 split for train and test data respectively with stratified sampling. Under the validation module, we use the logistic regression module along with apply model and performance modules. We leave the modeling settings as default for logistic regression.



□ manually set positive class

main criterion: first

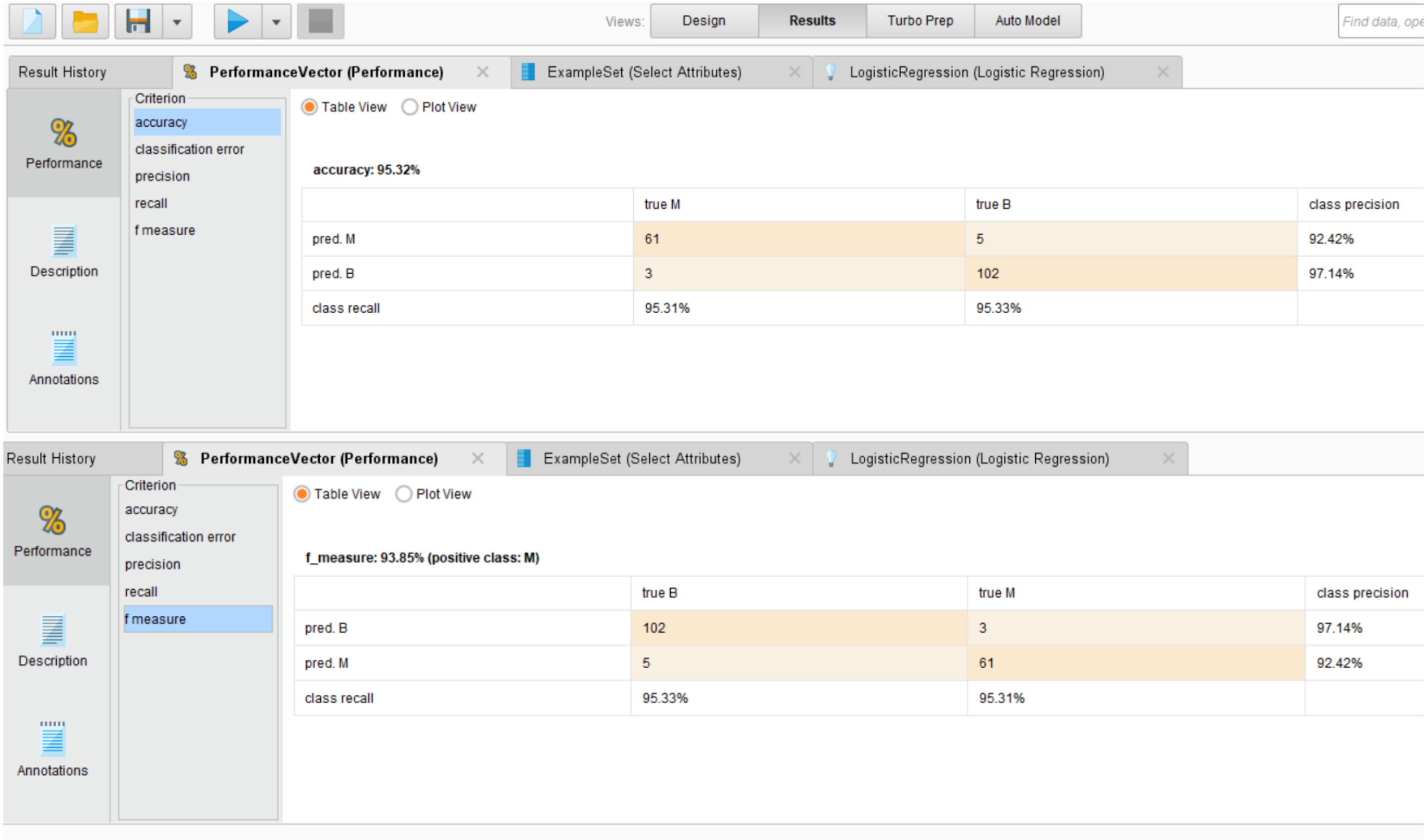
☑ accuracy

☑ classification error

□ kappa

□ AUC (optimistic)

□ AUC



accuracy: 95.32%

	true M	true B	class precision
pred. M	61	5	92.42%
pred. B	3	102	97.14%
class recall	95.31%	95.33%	

f_measure: 93.85% (positive class: M)

	true B	true M	class precision
pred. B	102	3	97.14%
pred. M	5	61	92.42%
class recall	95.33%	95.31%	

The accuracy out of sample is 95.32%. The F measure for class M is 93.85%. In this case as well, the performance of the logistic regression model is better than the KNN model on both out of sample accuracy and F measures for both classes.