Homework #2

MAYUR KUMAR__

(put your full name above (incl. any nicknames))

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.

I would frame the problem of e-mail spam detection as a supervised learning problem , because

- Like in supervised learning Problem, The goal is very clear here ,which is to classify an email whether it is a spam or not. A standard binary classification task
- The Response variable in this case would be spam/ no spam which can be encoded as 1 for spam or 0 for no spam
- Assumption we are making here is that our dataset would have a labelled column to label each mail as spam or not

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

A test dataset is a part of the whole dataset and is independent of the training dataset, but follows the same probability distribution as the training dataset.

Once we train the model using train data set, we want to know the performance of that model. In order to evaluate model, we use test data. Without test data, it's hard to determine the model's performance.

It helps us determine whether our model is overfitting, underfitting or should we increase complexity or decrease complexity of the model.

Test data set can be derived from the whole dataset in the following manners depending upon the goal we want to achieve.

1)Split validation:

We split the whole dataset into train and test usually in the ratio of 80-20 or 70-30

2)Nested holdout testing:

We split the whole dataset into train, validation and test usually in the ratio of 50-25-25 or 60-20-20.

3) K fold Cross validation :

We divide the whole dataset into K parts, and use each part as a test dataset in each iteration.

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

1)DATAMINING TASKS:

- DT can be used for classification and Regressions tasks, while Logistic Regression is used exclusively for Regression

2)PROBABILITIES:

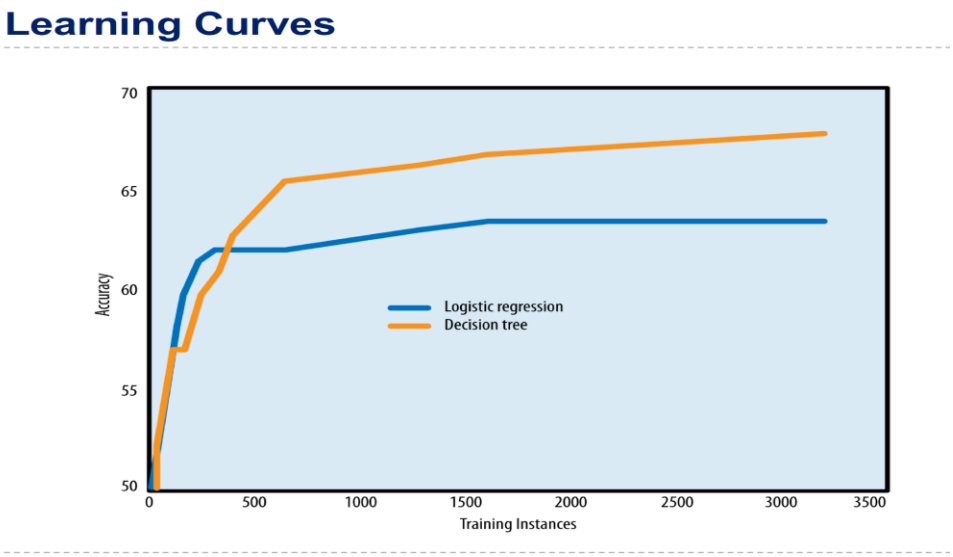
- DT classifies first and then can give probabilities for each prediction based on frequency based approach coupled with laplace correction, While Logistic Regression predicts the probabilities for each prediction and then classifies based on cutoff value we give.

3)DECISION BOUNDARIES:

- DT has multiple decision boundaries which could divide the entire space into very small regions, while Logistic Regression can only have one decision boundary which means it can only bisect the space.
- DT can only have decision boundaries that are perpendicular to axis, but Logistic Regression has a decision boundary which can orient in any direction.
- DT doesn't have curved decision boundaries, but Logistic Regression can have a curved shape decision boundary when we add polynomial terms in the equation.

4)DATA SIZE:

- Logistic Regression performs better on small datasets, while DT might not as DT will overfit for small datasets. DT needs more data in general. How each of them performs depending upon the size of the data can be seen as below.



5)MULTICOLLINEARITY:

- Both of the models behave unstable if there is multi collinearity ,so We choose to drop some variables in both of the model

6)USAGE OF ATTRIBUTES:

- DT uses one attribute at a time to build the entire tree, while Logistic Regression uses all variables together in model

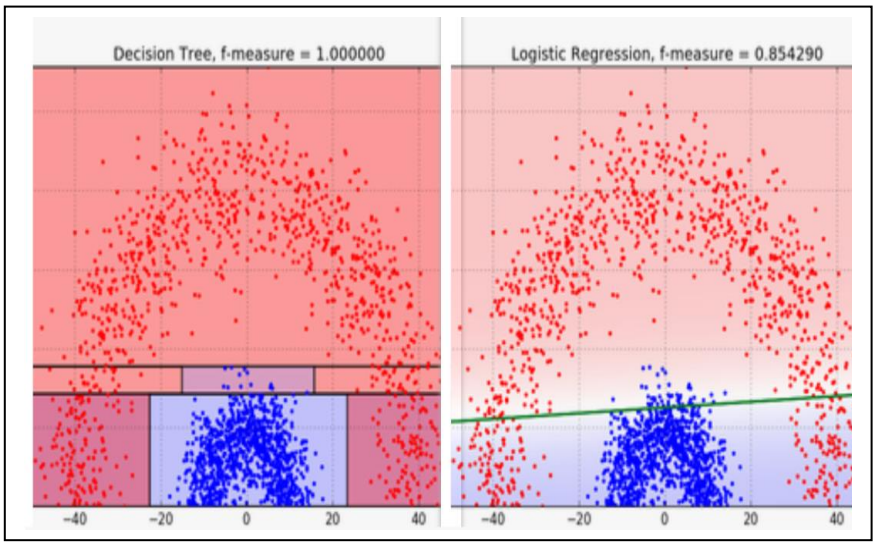
7)MODEL COMPREHENSIBILITY:

- DT has much better comprehensibility than Logistic Regression. Looking at the path from leaf node to parents node, we can easily interpret why we got a certain prediction.

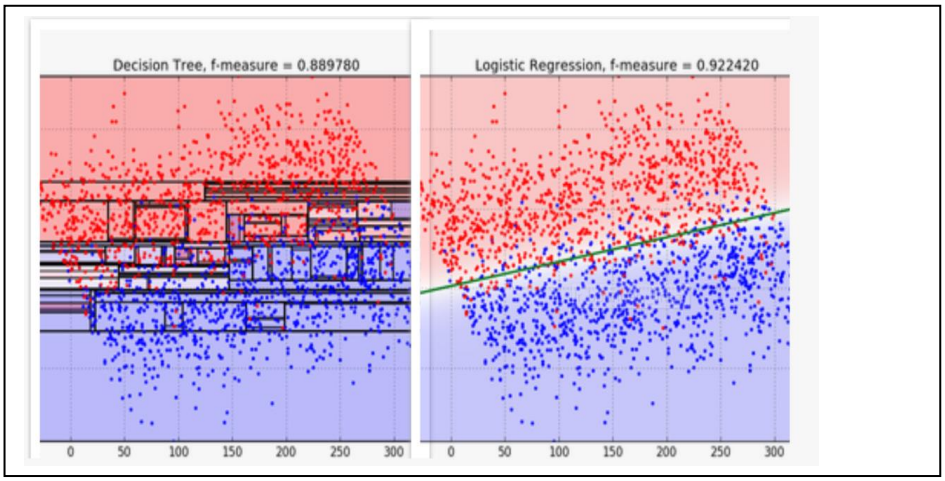
8)NONLINEAR RELATIONSHIPS:

- In the case of non linear relationships, DT performs better. This can be visually explained as follows.

  CASE 1: In this example where the two classes are separated by a decidedly non-linear boundary, we see that trees can better capture the division, leading to superior classification performance.



CASE 2: when classes are not well-separated, trees are susceptible to overfitting the training data, so that Logistic Regression's simple linear boundary generalizes better.



9)MISSING VALUES:

- DT is Robust to Missing values than Logistic Regression

10)COMPLEXITY:

- DT uses max_depth to increase complexity, while Logistic Regression uses more number of attributes, transformed variables, and polynomial terms of variables to increase complexity in the model

11)OVERFITTING:

Differences:

- DT uses pre pruning and post pruning techniques to overcome Overfitting, while Logistic regression uses Regularization and feature selection

Similarities:

- If we use large number of variables, both of them tend to overfit because of curse of dimensionality, so we need to optimize variable selection to avoid overfitting in both of them

SUMMARY OF WHEN TO CHOOSE ONE OVER OTHER:

Choose Logistic over DT:

- If we have small dataset with not much of missing values, and comprehensibility is not our key issue, then prefer Logistic Regression
- When classes are not well-separated as I explained in point 8 ,then prefer Logistic Regression

Choose DT over Logistic

- If your data has non linear relationships
- If you want to go for flexibility
- If you want interpretability/ comprehensibility

4) (30 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"	Cut off =0.3	Cut off = 0.8	PRECISION= 100%	RECALL= 100%
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
fraud	0.61	fraud	non-fraud	non-fraud	fraud
non-fraud	0.46	fraud	non-fraud	non-fraud	fraud
fraud	0.42	fraud	non-fraud	non-fraud	fraud
non-fraud	0.25	non-fraud	non-fraud	non-fraud	non-fraud
non-fraud	0.09	non-fraud	non-fraud	non-fraud	non-fraud
non-fraud	0.04	non-fraud	non-fraud	non-fraud	non-fraud

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?

Cutoff =0.8

		ACTUAL VALUES	
		FRAUD	NON-FRAUD
PREDICTED VALUES	FRAUD	TP = 5	FP = 2
	NON-FRAUD	FN = 0	TN = 3

ACCURACY = 8/10=80%

Cutoff =0.3

		ACTUAL VALUES	
		FRAUD	NON-FRAUD
PREDICTED VALUES	FRAUD	TP = 2	FP = 0
	NON-FRAUD	FN = 3	TN = 5

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?

Precision= TP / TP + FP

		ACTUAL VALUES	
		FRAUD	NON-FRAUD
PREDICTED VALUES	FRAUD	TP = 3	FP = 0
	NON-FRAUD	FN = 2	TN = 5

In order for precision to be 100%, FP should be 0. This can happen if we choose cutoff >0.67 and <0.75. To be more specific, I would choose 0.7

Accuracy in this case would be 80%

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?

Recall = TP / TP+ FN

		ACTUAL VALUES	
		FRAUD	**NON-FRAUD**
PREDICTED VALUES	**FRAUD**	**TP = 3**	**FP = 2**
	NON-FRAUD	**FN = 0**	**TN = 5**

In order for our Recall to be 100%, FN should be 0. This can happen if we choose cutoff <0.42. To be more specific, I would choose cutoff=0.4

Accuracy in this case would be 80%

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.

Appendix (Data Description)

1.Title: Wisconsin Diagnostic Breast Cancer (WDBC)

Results:

- predicting field 2, diagnosis: B = benign, M = malignant

2. Number of instances: 569

3. Number of attributes: 32 (ID, diagnosis, 30 real-valued input features)

4. Attribute information

1) ID number
2) Diagnosis (M = malignant, B = benign)
3-32)

Ten real-valued features are computed for each cell nucleus:

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

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

All feature values are recoded with four significant digits.

5. Missing attribute values: none

6. Class distribution: 357 benign, 212 malignant

1)DATA UNDERSTANDING:

- **Import the data:**

Data can be directly imported from the link as follows. Since none of the columns have been labelled, we specify header =None . Now the first row will be considered as one of the instances , and not the header.

```
#Importing data directly from the website
data = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data" ,header=None)
```

- **Structure of the dataset:**

```
print(data.shape)
print()
print(data.info())
print()
print(data.head())
print()
type(data)
```

With the help of above commands, we could get a good idea of how our dataset looks like.

Number of Observations	Explanatory variables	Response Variable	Index column
569	30 Columns from 2 to 32	Column2	Column 1

Our Dataset has a fair number of observations, which help us to model better as it seems like a quality dataset.

- **Response Variable:**

Our Response Variable has two classes.

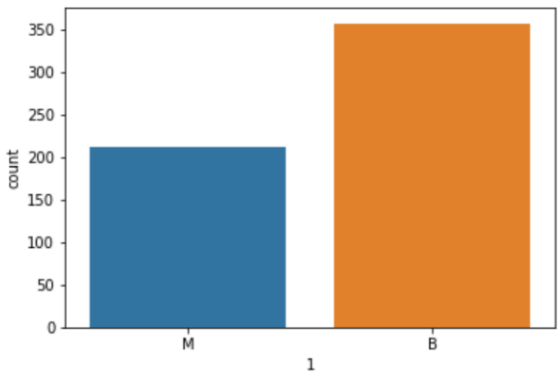
1)Malignant = Harmful Tumor

2)Benign = Not harmful Tumor

  - ***Distribution of our response variable:***

```
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(data[1])
plt.show()
```



It shows that instances of benign(67%) are quite more than Malignant(33%). This is an unbalanced dataset. Hence **stratifying** while splitting the data into train and test would be a good idea.

2)DATA EXPLORATION:

In order to build a model to predict Malgnant or Benign from the provided dataset, It's good to examine the statistics of each attribute. This can be done by describe function in python. It gives statistics for each variable. Statistics for each variable seems fine. Nothing aberrant as yet.

```
#To see statistics of each variable
print(data.describe(include='all'))
```

```
                 0    1           2           3           4           5   \
count   5.690000e+02  569  569.000000  569.000000  569.000000  569.000000
unique           NaN    2         NaN         NaN         NaN         NaN
top              NaN    B         NaN         NaN         NaN         NaN
freq             NaN  357         NaN         NaN         NaN         NaN
mean    3.037183e+07  NaN   14.127292   19.289649   91.969033   654.889104
std     1.250206e+08  NaN    3.524049    4.301036   24.298981   351.914129
min     8.670000e+03  NaN    6.981000    9.710000   43.790000   143.500000
25%     8.692180e+05  NaN   11.700000   16.170000   75.170000   420.300000
50%     9.060240e+05  NaN   13.370000   18.840000   86.240000   551.100000
75%     8.813129e+06  NaN   15.780000   21.800000  104.100000   782.700000
max     9.113205e+08  NaN   28.110000   39.280000  188.500000  2501.000000

                6           7           8           9   ...          22  \
count   569.000000  569.000000  569.000000  569.000000  ...  569.000000
unique         NaN         NaN         NaN         NaN  ...         NaN
top            NaN         NaN         NaN         NaN  ...         NaN
freq           NaN         NaN         NaN         NaN  ...         NaN
mean      0.096360    0.104341    0.088799    0.048919  ...   16.269190
std       0.014064    0.052813    0.079720    0.038803  ...    4.833242
min       0.052630    0.019380    0.000000    0.000000  ...    7.930000
25%       0.086370    0.064920    0.029560    0.020310  ...   13.010000
50%       0.095870    0.092630    0.061540    0.033500  ...   14.970000
75%       0.105300    0.130400    0.130700    0.074000  ...   18.790000
max       0.163400    0.345400    0.426800    0.201200  ...   36.040000

               23          24          25          26          27  \
count   569.000000  569.000000  569.000000  569.000000  569.000000
unique         NaN         NaN         NaN         NaN         NaN
top            NaN         NaN         NaN         NaN         NaN
freq           NaN         NaN         NaN         NaN         NaN
mean     25.677223  107.261213  880.583128    0.132369    0.254265
std       6.146258   33.602542  569.356993    0.022832    0.157336
min      12.020000   50.410000  185.200000    0.071170    0.027290
25%      21.080000   84.110000  515.300000    0.116600    0.147200
50%      25.410000   97.660000  686.500000    0.131300    0.211900
75%      29.720000  125.400000 1084.000000    0.146000    0.339100
max      49.540000  251.200000 4254.000000    0.222600    1.058000

               28          29          30          31
count   569.000000  569.000000  569.000000  569.000000
unique         NaN         NaN         NaN         NaN
top            NaN         NaN         NaN         NaN
freq           NaN         NaN         NaN         NaN
mean      0.272188    0.114606    0.290076    0.083946
std       0.208624    0.065732    0.061867    0.018061
min       0.000000    0.000000    0.156500    0.055040
25%       0.114500    0.064930    0.250400    0.071460
50%       0.226700    0.099930    0.282200    0.080040
75%       0.382900    0.161400    0.317900    0.092080
max       1.252000    0.291000    0.663800    0.207500
```

- **DATA CHECK:**

```
#To see if there are any missing values
print(data.isnull().any().sum())
```

We see that there are no missing values. No need of Data cleaning. We are good to go ahead

3)DATA VISUALIZATION:

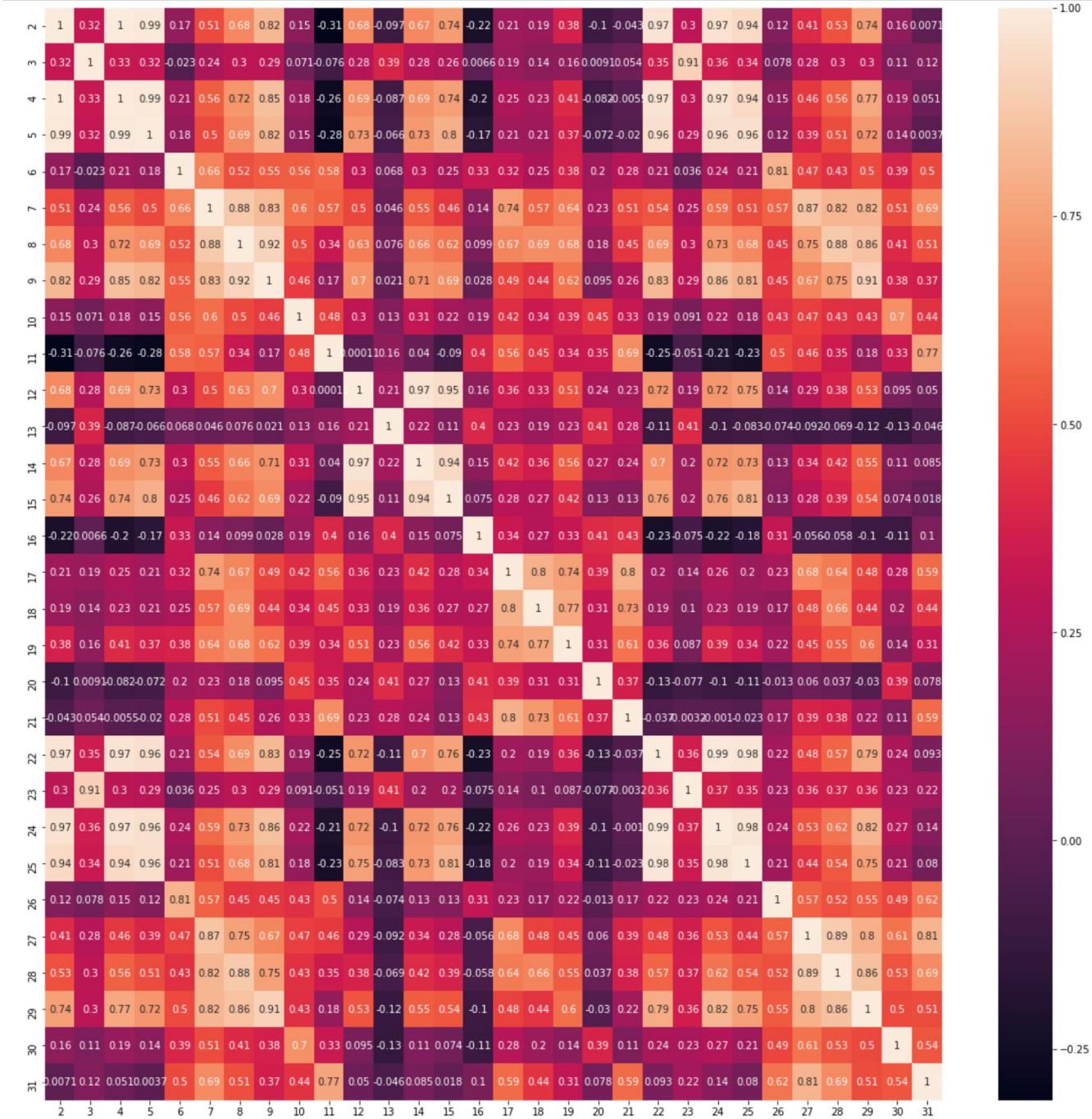
It is helpful to have a look at a histogram of each individual attribute/feature. This allows us to see the distribution of classes within each feature. This can help us determine if the individual feature itself would be enough in predicting malignant/benign. Also, plots between each independent variable vs Response variable could give some insights. All of these graphs can be visualized using ***sns.pairplot(data)*.** Looking at the graphs, we see that there aren't much of insights.

Correlation:

For a deeper look at our data, we can also look at the correlation between features. This can help us identify which features are most related to our target variable. Especially, It is imperative to see correlation amongst Independent variables to avoid multi collinearity. With the following code we could get a correlation matrix

```
f, ax = plt.subplots(figsize=(20,20))
# creating correlation matrix
corr = data.iloc[:,2:].corr()

#using a heatmap to visualize the correlation matrix
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values, annot=True)
```



Looking at the correlation heat map, we see that the following pairs are highly correlated with a correlation value >=0.94 in each of the following pairs. We could select only one of them from each pair to avoid multicollinearity and make the model more stable.

Highly correlation feature pairs	Correlation value
2,5	0.99
2,22	0.97
2,24	0.97
2,25	0.94
4,5	0.99
4,22	0.96
4,24	0.97
4,25	0.94
5,22	0.96
5,24	0.96
5,25	0.96
12,14	0.97
12,15	0.95
14,15	0.94
22,24	0.99
22,25	0.98

4)SPLIT THE DATA:

Before modelling, It is important to split the data into train and test. Doing so, we can train the model with the train data set and evaluate its performance with test(out of sample) We split the data into train and test in the ratio of 70/30 here.

Since, it is an unbalanced dataset, it is good to do stratifying, because we could get the same ratio of malignant and benign cases in both train and test. We just want our test data to have substantial number of malignant cases in test as well.

```
X= data.iloc[: , 2: ]
Y= data.iloc[: , 1]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=1,stratify=Y)

print(X.head())
print(Y.head())
```

Once we split the data, we want to see how the target variable has been split across train, and test

```
######################################### Distribution Target Variable ########################################

print('Labels counts in y:', Y.value_counts())
print('Labels counts in y_train:', Y.value_counts())
print('Labels counts in y_test:', Y.value_counts())
```

```
Labels counts in y: B    357
M    212
Name: 1, dtype: int64
Labels counts in y_train: B    357
M    212
Name: 1, dtype: int64
Labels counts in y_test: B    357
M    212
Name: 1, dtype: int64
```

It seems like a fair split.

5)NORMALIZATION:

Since in the case of Knn, we use distance as a measure to find the nearest neighbours, it is important to standardize the variables so that the distance is unaffected by scaling of the variables We are using Z-score scaling method to standardize the variables.

```
sc= StandardScaler()
sc.fit(X_train)

X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

6)BUILD THE MODEL WITH KNN:

Let's build the model with following parameters initially

- k=3.
- P =2, indicating Eucledian distance
- Weights = 'uniform'

Also do the prediction on out of sample , name it as Y_pred
Insample, name it as Y_pred_insample

```
clf_knn = KNeighborsClassifier(n_neighbors=3 , p=2 , metric ='minkowski' )
clf_knn.fit(X_train_std, Y_train)

#Prediction for test data set and train dataset
Y_pred= clf_knn.predict(X_test_std)
Y_pred_insample = clf_knn.predict(X_train_std)
```

EXPERIMENTING MODEL WITH DIFFERENT PARAMETERS:

We could try different combinations of parameters such as

1)Eucledian vs Manhattan distance

2) Weights = uniform (All points in each neighborhood are weighted equally)

vs distance**(**weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away)

7)EVALUATION:

We evaluate the model by computing the confusion matrix using following code for 4models we tried.

```
# Compute confusion matrix
cnf_matrix = confusion_matrix(Y_test, Y_pred)
np.set_printoptions(precision=2)

class_outcomes = ["M" ,"B"]
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes= class_outcomes,
                      title='Confusion Matrix')
```

We get performance generalization through the following code

```
print("Model Metrics:")
print()
print("Accuracy: " + str(accuracy_score(Y_test,Y_pred)*100))
print("Positive Precision: " + str(precision_score(Y_test, Y_pred, pos_label = 'M')))
print("Negative Precision: " + str(precision_score(Y_test, Y_pred, pos_label = 'B')))
print("Positive Recall: " + str(recall_score(Y_test, Y_pred, pos_label = 'M')))
print("Negative Recall: " + str(recall_score(Y_test, Y_pred, pos_label = 'B')))
print("Positive F-Measure: " + str(f1_score(Y_test, Y_pred, pos_label = 'M')))
print("Negative F-Measure: " + str(f1_score(Y_test, Y_pred, pos_label = 'B' )))
```

Upon experimenting, we found out the following performances by each model we tried.

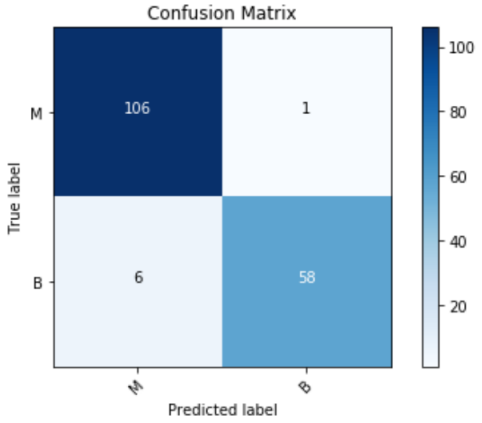
Parameter Combinations	ACCURACY	PRECISION+	PRECISION-VE	RECALL+	RECALL-	fmeasure+	fmeasure-
P=2, weights=uniform	95.9%	98.3%	94.6%	90.6%	99%	94.3%	96.8%
P=2, weight=distance	95.9%	98.3%	94.6%	90.6%	99%	94.3%	96.8%
P=1, weight=uniform	94%	96%	93.75%	89%	98.1%	92%	95%
P=1 ,weight=distance	94.73%	96.6%	93.75%	89%	98%	92.6%	95.8%

We see that there is some change in performance when we change eucledian to manhattan , but insignificant change when we change weights metric

I also tried removing various attributes which are highly correlated with eachother , but the model's overall performance didn't change much

FINAL MODEL:

Our final Model would be the one with k=3, weight =distance , p=2(Eucledian). It has the following confusion matrix



ACCURACY= 95.9%
PRECISION +VE= 98.3%
PRECISION -VE=94.6%
RECALL +VE = 90.6%
REALL -VE =99%
FMEASURE +VE= 94.3%
FMEASURE -VE =96.8%

NOTE:

Our final model has Recall +ve 90.6% which could be an issue, because we are misclassifying 9.4% of the M cases as B. We would discuss this at the end when we are comparing models

ARE WE OVERFITTING?

Looking at the performance on test data, It doesn't seem like overfitting,
Since K=3 is very small. Smaller the K, higher the chances for overfitting. Let's just compare insample and out sample test performance.

```
#Comparision for insampleand outsample
# Accuracy
print('Accuracy (out-of-sample): %.2f' % accuracy_score(Y_test, Y_pred))
print('Accuracy (in-sample): %.2f' % accuracy_score(Y_train, Y_pred_insample))

# F1 score
print('F1 score (out-of-sample): ', f1_score(Y_test, Y_pred, average='macro'))
print('F1 score (in-sample)    : ', f1_score(Y_train, Y_pred_insample, average='macro'))


# Build a text report showing the main classification metrics (out-of-sample performance)
print(classification_report(Y_test, Y_pred, target_names=class_outcomes))
```

```
Accuracy (out-of-sample): 0.96
Accuracy (in-sample): 0.99
F1 score (out-of-sample):  0.9555629802873371
F1 score (in-sample)    :  0.9864973978653675
              precision    recall  f1-score   support

           M       0.95      0.99      0.97       107
           B       0.98      0.91      0.94        64

   micro avg       0.96      0.96      0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171
```

Looking at Accuracy, both of them are close enough, which says the model is good and not really overfitting.

LOGISTIC REGRESSION:

TRAIN THE MODEL:

Let's train the model and do the prediction on test data. We do two types of prediction here.
1)Classification
2)Probabilities

```
clf_logistic = linear_model.LogisticRegression(C=1e5)

clf_logistic.fit(X_train, Y_train)
print('The weights of the attributes are:', clf_logistic.coef_)

#classification predictions
Y_pred_logistic= clf_logistic.predict(X_test)

#class probabilities
Y_pred_logistic_prob = clf_logistic.predict_proba(X_test)

print()
print(Y_pred_logistic[0] ,Y_pred_logistic_prob[0])
print(np.sum(Y_pred_logistic_prob[0]))
```

We can also check the weights of the attributes by ***clf_logsitic.coef*** command

Fllowing are the weights of the attributes:

```
[-6.16e+00 -1.78e-01 7.86e-01 8.27e-03 3.22e+00 -3.87e-01 3.38e+00 5.84e+00
5.72e+00 -7.30e-01 -4.23e+00 -4.42e+00 9.59e-01 1.59e-01 8.78e-01 -5.17e+00 -
6.67e+00 4.90e-01 -1.95e-01 -9.90e-01 -1.53e+00 6.82e-01 -8.01e-02 3.20e-02
8.84e+00 -2.52e+00 3.66e+00 1.21e+01 1.29e+01 -9.21e-01]]
```

- We could try experimenting different parameters such as changing the C value. C value helps us customize the regularization.

- Higher C value means lower regularization. Lower C value means Higher regularization.

- We do regularization because we want to add a little bias to the regression line in a hope that it would fit well for the test data. We are increasing it's performance on test data but decreasing performance on train data. regularization helps add penalty to the attributes , and tries to lower the multi collinearity effect.

EVALUATION:

We could try 5 orders of C value. 10,100,1000,10000,100000

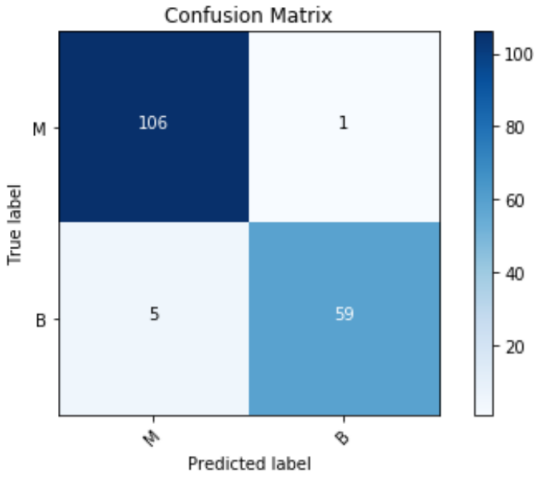
C value	ACCURACY	PRECISION+	PRECISION-VE	RECALL+	RECALL -	fmeasure+	fmeasure -
1	95.3%	98.3%	94.6%	90.6%	99%	94.3%	96.8%
10	95.9%	98.3%	94.6%	90.6%	99%	94.3%	96.8%
100	95.9%	98.3%	94.6%	90.6%	99.1%	94%	96%
1000	94.73%	98.3%	94.6%	90.6%	99%	94.6%	96.8%
10000	94.73%	98.3%	94.6%	90.6%	99%	94.6%	96.8%
100000	96.49%	98.3%	94.6%	92%	99%	94.3%	96.8%

Upon Experimenting different C values, we see that Model's performance doesn't vary much. But when C=1e5 , we are getting a very slightly better Accuracy and Recall+ values. So I would go with the model with C=100000, which essentially says that lamda is infitesimally small, which implies no penalty, implying no regularization is needed for this model.

FINAL MODEL:

```
#Compute_confusion_matrix
cnf_matrix = confusion_matrix(Y_test, Y_pred_logistic)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes= class_outcomes,
                      title='Confusion Matrix')
```




```
print("Logistic Regression Model Metrics :")
print()
print("Accuracy: " + str(accuracy_score(Y_test,Y_pred_logistic)*100))
print("Positive Precision: " + str(precision_score(Y_test, Y_pred, pos_label = 'M')))
print("Negative Precision: " + str(precision_score(Y_test, Y_pred, pos_label = 'B')))
print("Positive Recall: " + str(recall_score(Y_test, Y_pred_logistic, pos_label = 'M')))
print("Negative Recall: " + str(recall_score(Y_test, Y_pred, pos_label = 'B')))
print("Positive F-Measure: " + str(f1_score(Y_test, Y_pred, pos_label = 'M')))
print("Negative F-Measure: " + str(f1_score(Y_test, Y_pred, pos_label = 'B' )))
```

```
Logistic Regression Model Metrics :

Accuracy: 96.49122807017544
Positive Precision: 0.9830508474576272
Negative Precision: 0.9464285714285714
Positive Recall: 0.921875
Negative Recall: 0.9906542056074766
Positive F-Measure: 0.943089430894309
Negative F-Measure: 0.9680365296803651
```

COMPARING KNN AND LOGSITIC MODELS:

	KNN	LOGISTIC
ACCURACY	95.9%	96.49%
PRECISION +VE	98.3%	98.3%
PRECISION -VE	94.6%	94.6%
RECALL +VE	90.6%	92%
RECALL -VE	99%	99%
FMEASURE +VE	94.3%	94.3%
FMEASURE -VE	96.8%	96.8%

In all metrics, Logistic performs better than or same as Knn.

IMPORTANT NOTE:

Our False positives and False Negatives have different costs in our context.

Predicting Benign as Malignant is not that of an issue, but predicting malignant as Benign is a big issue, because it's a big risk. Patient's life would be at stake if we misclassify patient with malignant tumor with benign tumor. Thus we have different costs.

That means our Recall+ve should be as high as possible. So more than just Accuracy, Recall+ve is an important metric to compare between models.

Recall +ve = TP / (TP + FN) . FN should be as low as possible. FN basically means, you predicted some one as benign but in reality the patient is malignant. We want to avoid this case as much as possible.

Since Logsitic has higher Recall+ve than Knn. I will choose **Logistic Regression Model** that I built.

RAPID MINER:

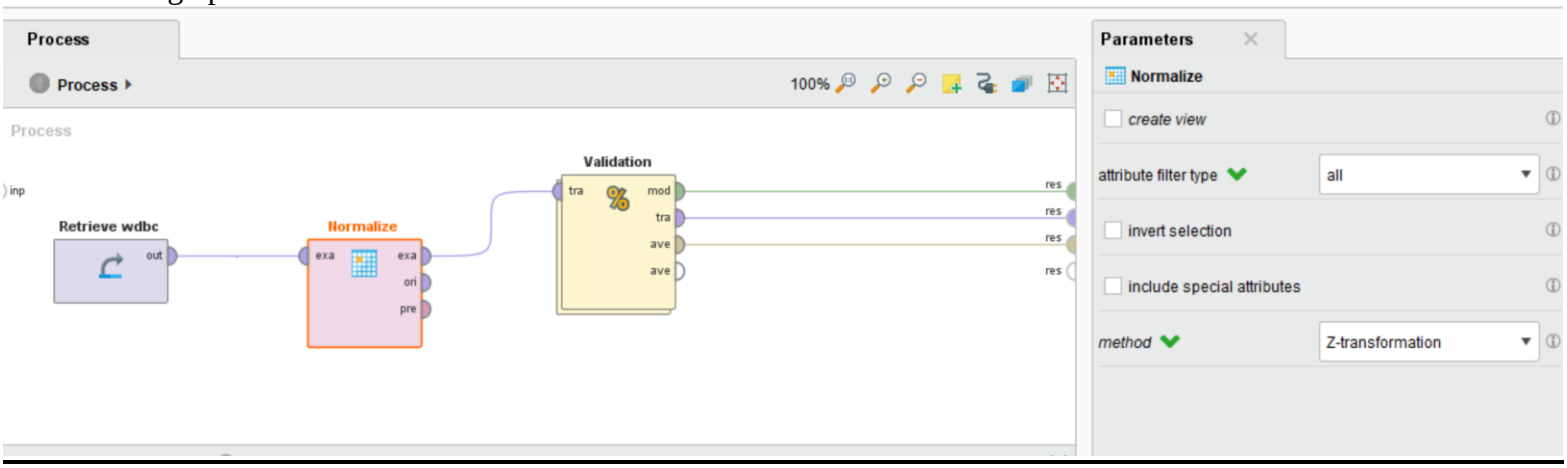
KNN:

The process taken in RapidMiner mirrors that of Python.

DESIGN:

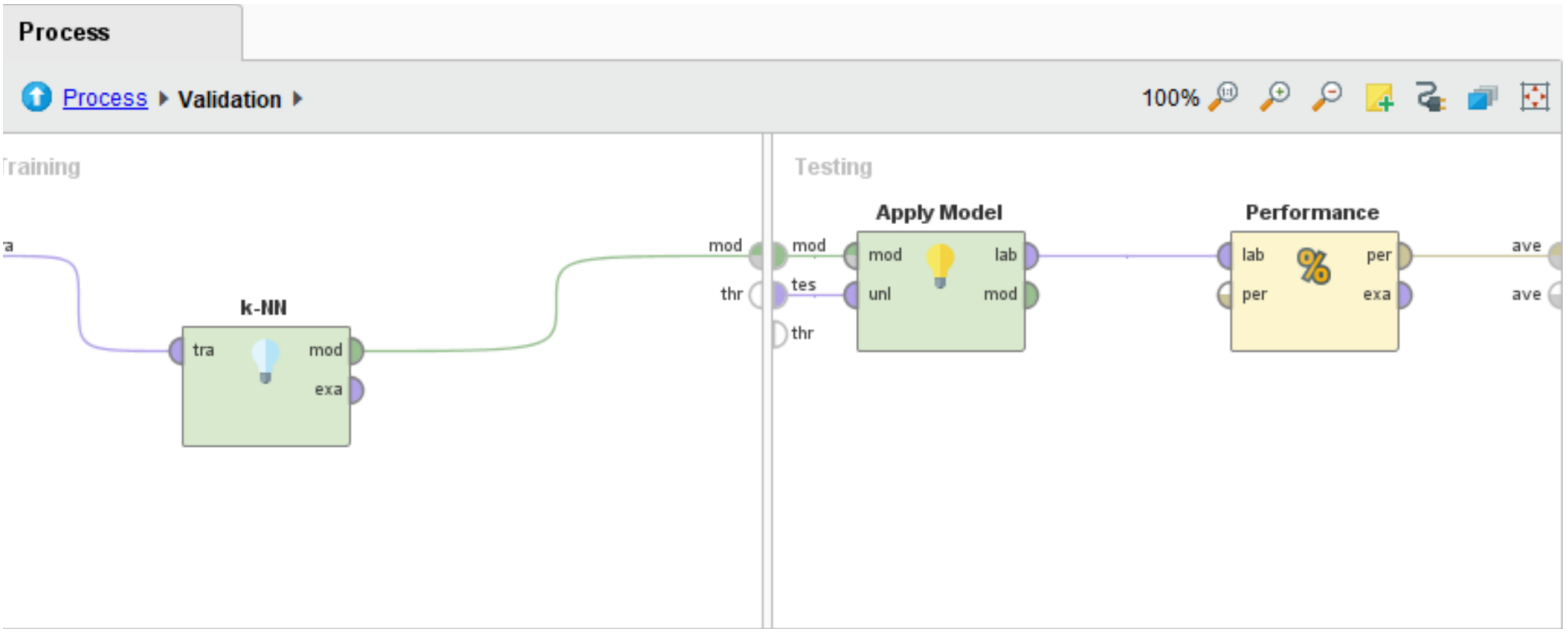
- Once we input the data, label the target variable as label , and index column as id.
- We use retrieve operator to read the data, then normalize operator to normalize the variables. I use z-transformation method
- We use Split validation operator to split the data into train and test. The ratio I chose is 70-30 , and I chose stratifying strategy here.

Our Design process so far looks like this:

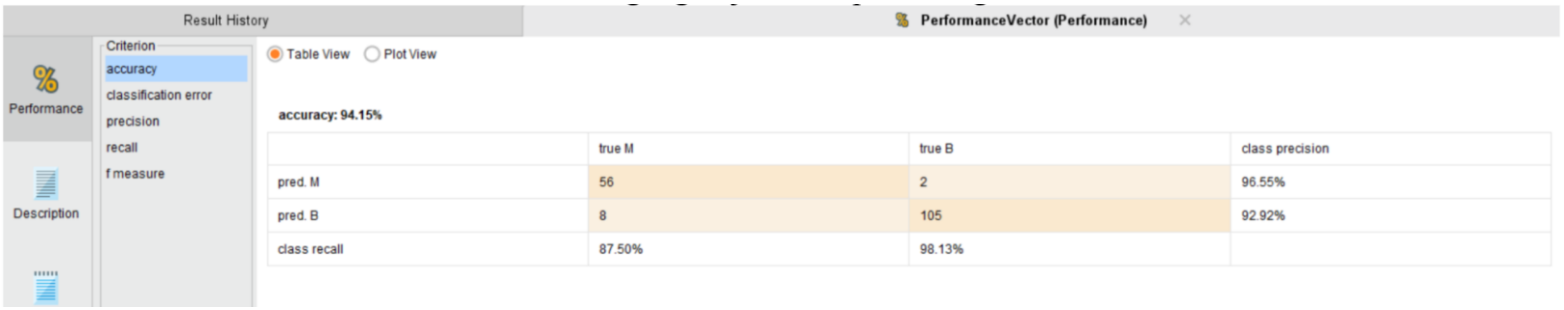


- Split validation has double layer, which has training on left , and testing on right.
- Add Knn operator on the left, and choose k=3,and Eucledian distance
- Add Apply Model, and Generalization performance on the right

Final Design inside split validation



After running, we see results as follows,



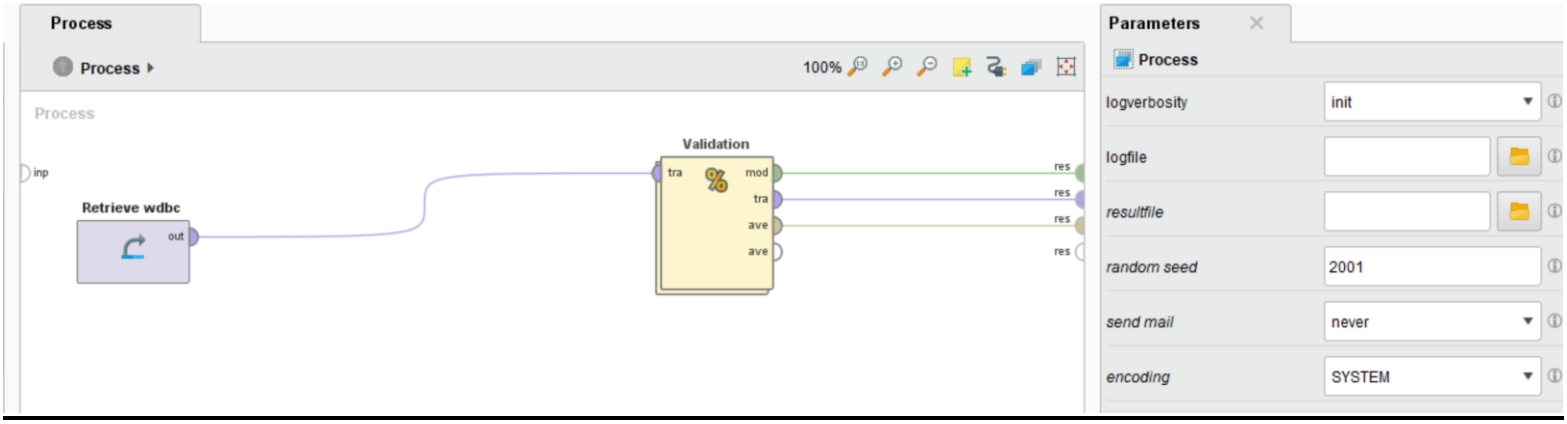
	true M	true B	class precision
pred. M	56	2	96.55%
pred. B	8	105	92.92%
class recall	87.50%	98.13%	

ACCURACY = 94.15%

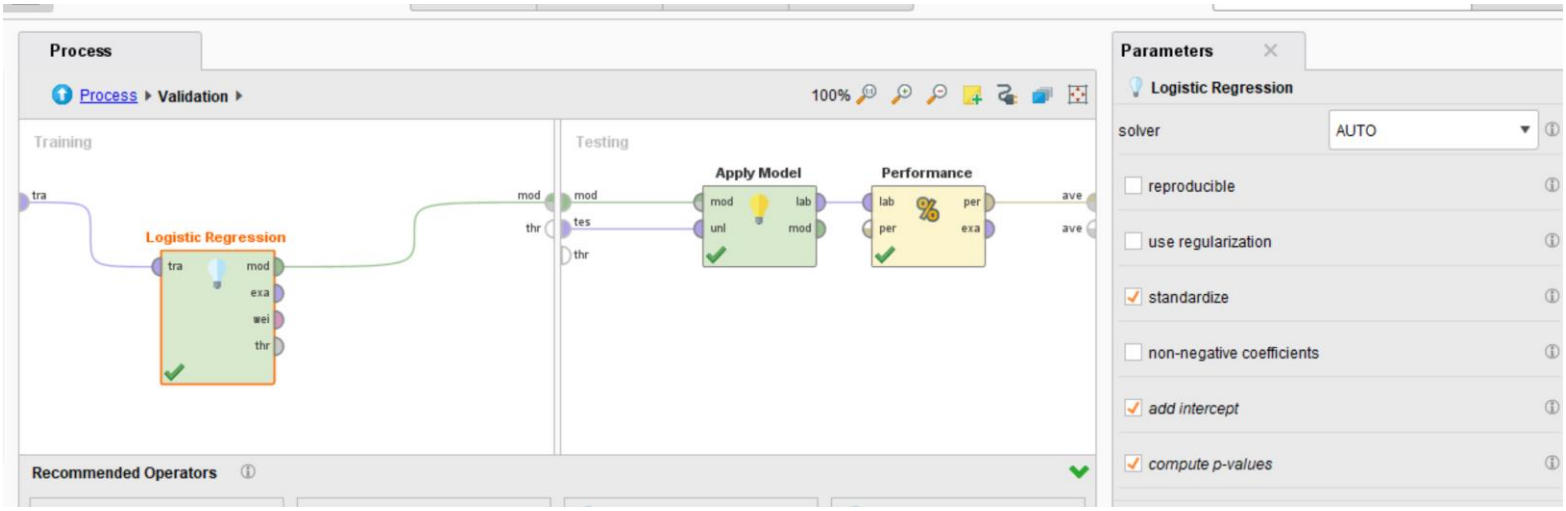
RECALL +VE = 87.5%

LOGISTIC REGRESSION:

DESIGN:

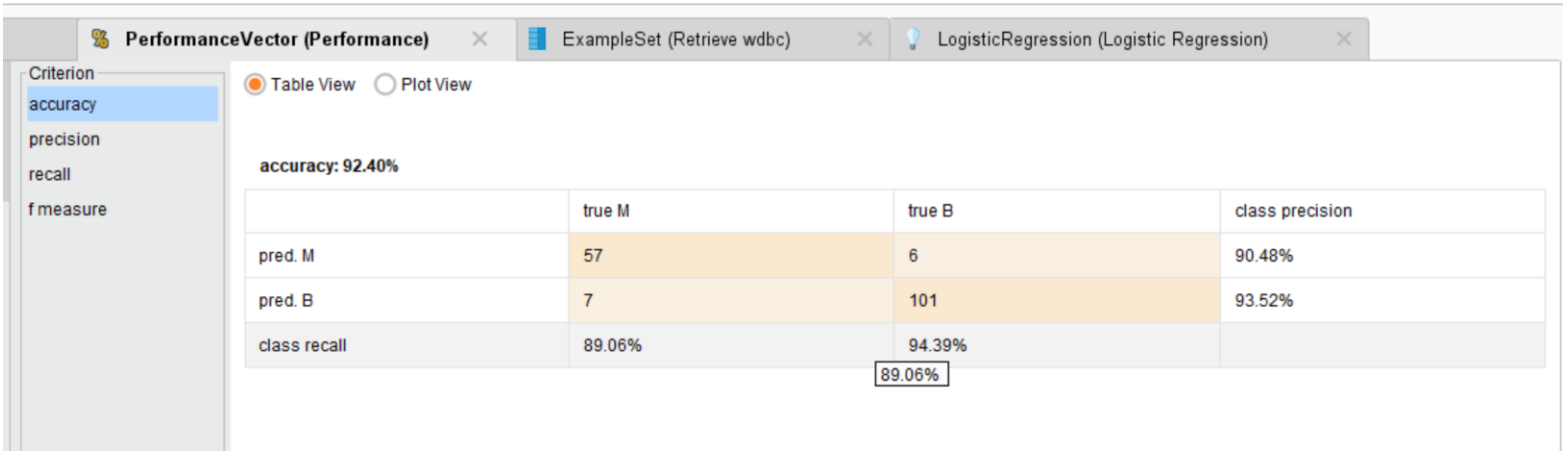


DESIGN INSIDE SPLIT VALIDATION:



I experimented with regularization and tried multiple Lamda values. Without regularization, I got the best model.

RESULTS OF MY FINAL BEST MODEL:



COMPARING BOTH MODELS ON RAPIDMINER:

	Knn	Logistic
ACCURACY	94.15%	92.4%
PRECISION+	91%	90.48%
PRECISION-	92%	93.52%
RECALL +	87.5%	94.39%
RECAL -	98.2%	89.06%
F MEASURE+	90%	93.95%

Knn has better accuracy , but Recall+ is the most important metric , and Logistic has better Recall+ , so I would go with Logsitic Regression over Knn.