

**Homework #2 (Part B)**

\_\_\_\_\_ **Jane Doe** \_\_\_\_\_

(put your name above)

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 be sufficient). If you borrow code and don't provide the source, it is a violation of the Honor Code.

Total grade:  
\_\_70\_\_ out of \_\_70\_\_ points

**ATTENTION:** HW2 has two parts. Please first complete the Quiz “HW2\_Part1” on Canvas. Then, proceed with Part 2 on the following page. You will need to submit (a) a PDF file with your answers and screenshots of Python code snippets as well as Rapidminer repositories and (b) the Python code and Rapidminer repositories.

(70 points) [Mining publicly available data. Please implement the following models with both Rapidminer and 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>

**a** [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> and is also provided as in the appendix of this homework assignment.

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 B or malignant M).

**50 Points (Python):**

- a) (10 points) Load the data. Then, explore the data by reporting summary statistics and a correlation matrix. Show your code.
- b) (12 points) Perform a predictive modeling analysis on this dataset to predict the type (benign B or malignant M) using a k-NN technique (for k=3) and the Logistic Regression technique. Please be specific about what other parameters you specified for your models. Briefly discuss your modeling process (e.g., validation technique, any preprocessing steps, parameters used to build the models, etc.) and show your code. Report the estimated coefficients of the Logistic Regression technique.
- c) (13 points) Compare the generalization performance of the k-NN model with the Logistic Regression model. Make sure you report the confusion matrix, the predictive accuracy, precision, recall, and f-measure. Briefly discuss the results and show your code.
- d) (15 points) What generalization performance metric would you prefer to use in order to choose the best performing model in this context and why? Please be clear about any assumptions you might make when you select the generalization performance metric you would prefer.

### **Exploration and Problem Scoping**

Before beginning any modeling, I needed to ensure I understood the business problem I was trying to address. In this case, the problem I am trying to predict is whether or not a cell mass is cancerous (“Malignant”) or not (“Benign”). I then took a look at the description of the dataset below to see what kind of data I have. Initially, I explored the data, which contained 569 observations in total. The dataset contains 30 numeric independent variables, one ID variable

that won't be used to make predictions, and one dependent variable containing the instance diagnosis.

Corresponding Code:

```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn

import pandas as pd
import numpy as np
import sklearn
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LinearRegression

[ ] import yellowbrick
from yellowbrick.classifier import ClassificationReport
from yellowbrick.classifier import ConfusionMatrix

[ ] url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data'

[ ] df = pd.read_csv(url)

df['malignant'] = df.iloc[:, 1].str.replace('M', '1')
df['malignant'] = df['malignant'].str.replace('B', '0')
df['malignant'] = df['malignant'].astype(int)
```

After making sure I understood the business problem and the data that I have, I began my exploration of the data. After importing my data into Python, I started by exploring the frequency of instances per class. There are 357 “Benign” cases and 212 “Malignant” cases. In order to explore the data further, I use the `.describe()` function to view descriptive statistics, and upon doing this, I found that there were no missing values, so I didn't have to worry about removing these or imputing them with anything. I also used histograms to help understand the distribution of my features as well as to confirm the distribution of my target variable and see if they appear to be skewed. From the histograms, we can see that most of these attributes follow a normal distribution, but some follow a log distribution.

## Corresponding Code:

```
[6] df = df.dropna()  
     df.isna().values.any()
```

```
False
```

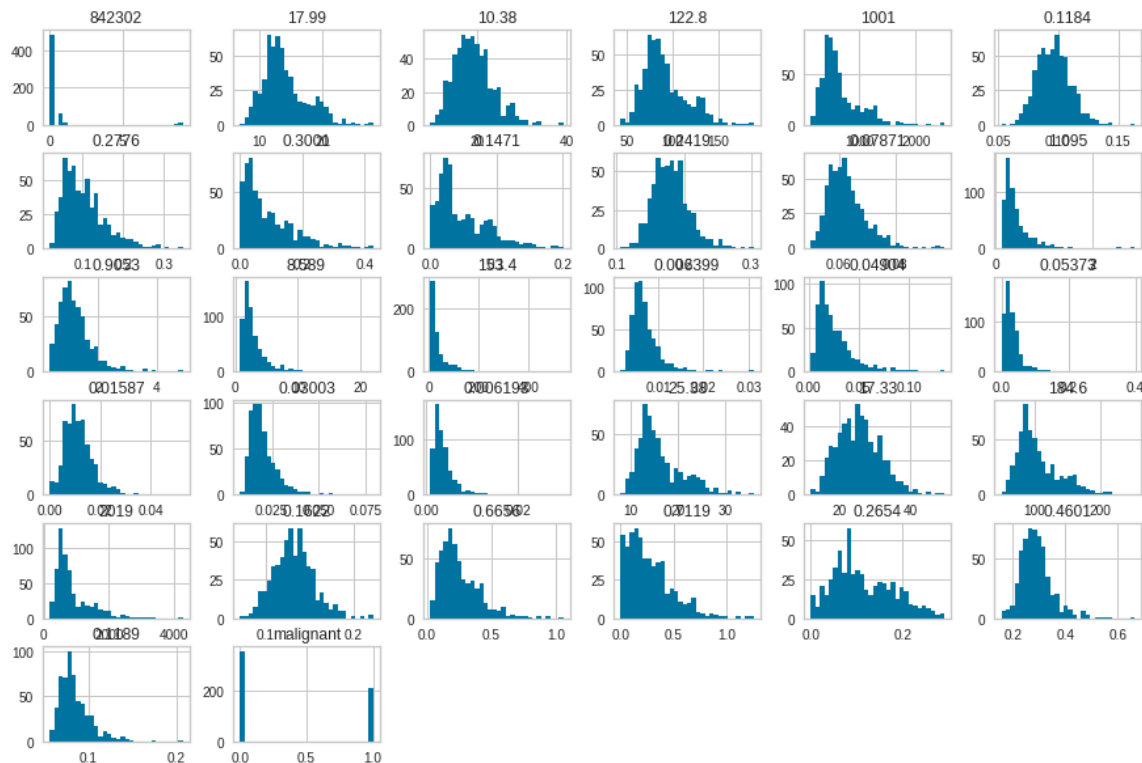
```
[7] df.iloc[:,1].value_counts()
```

```
B    357  
M    211  
Name: M, dtype: int64
```

```
[8] df.describe()
```

```
count      842302      17.99      10.38      122.8      1001      0.1184      0.2776      0.3001  
mean      3.042382e+07  14.120491  19.305335  91.914754  654.279754  0.096321  0.104036  0.088427  
std       1.251246e+08   3.523416   4.288506  24.285848  351.923751  0.014046  0.052355  0.079294  
min       8.670000e+03   6.981000   9.710000  43.790000  143.500000  0.052630  0.019380  0.000000  
25%       8.692225e+05  11.697500  16.177500  75.135000  420.175000  0.086290  0.064815  0.029540  
50%       9.061570e+05  13.355000  18.855000  86.210000  548.750000  0.095865  0.092525  0.061400  
75%       8.825022e+06  15.780000  21.802500 103.875000  782.625000  0.105300  0.130400  0.129650  
max       9.113205e+08  28.110000  39.280000 188.500000 2501.000000  0.163400  0.345400  0.426800
```

```
df.hist(bins=30, figsize=(15, 10))
```

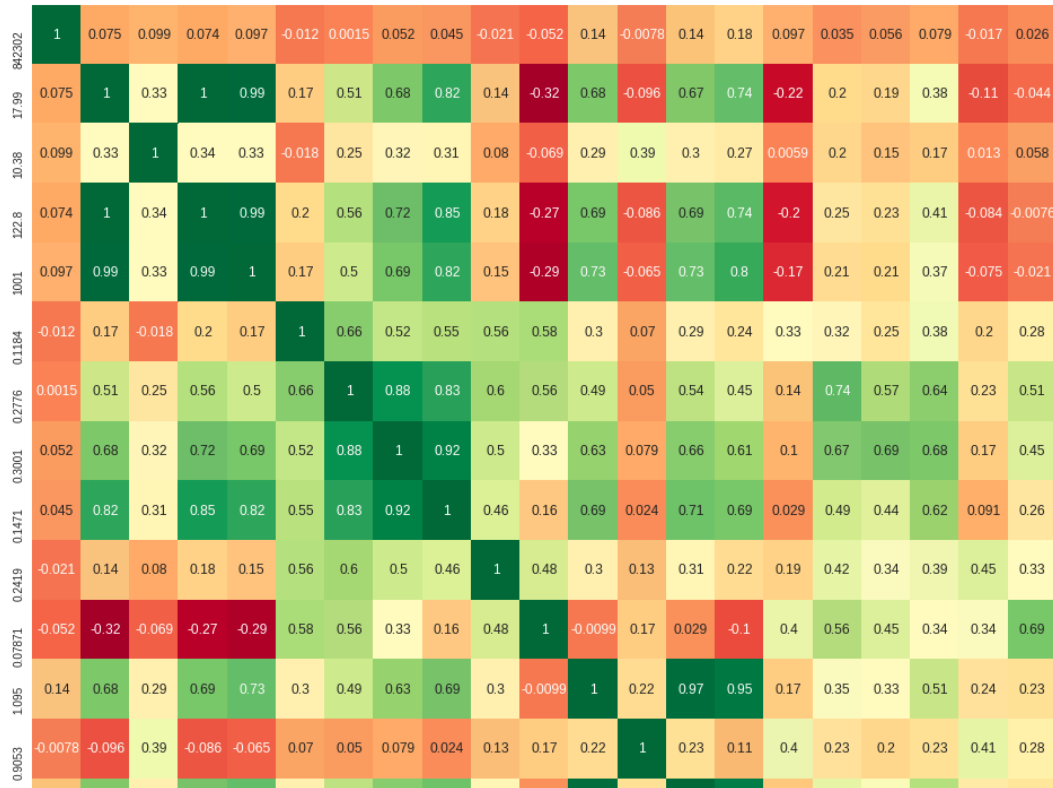


We will now plot a correlation matrix of the features using seaborn. Multicollinearity is a concern in classical statistical analysis and exploratory analytics but it is not a primary concern in predictive analytics. The reason for that is that, in predictive analytics, we mostly care about the predictive power of the model and multicollinearity does not affect the predictive power of the model. Multicollinearity could affect our coefficient estimates or standard error estimates but in machine learning, we are not concerned with individual coefficients or their interpretation. In fact, if you try to remove variables that are highly correlated in this case it would even reduce your models' generalization performance. In any case, the regularized models will naturally take care of any multicollinearity issues.

Corresponding Code:

```
[9] corrmatrix = df.corr()
    top_correlated_features = corrmatrix.index
    plt.figure(figsize=(31,31))
    #plot heat map
    plot=sns.heatmap(df[top_correlated_features].corr(),annot=True,cmap="RdYlGn")
```

✖



## Model Building Process

### Split Validation:

After the data exploration phase, I moved to the model building phase. I started first by splitting the data into training and test sets in order to properly evaluate the predictions with out-of-sample predictions. I chose to use .30 as the splitting criteria for the test set and .70 for the training set. I set 'stratify' equal to y, which ensures that there are enough observations from each class in both the training and the test sets. I didn't edit any other parameters and left the default shuffling parameter set.

Corresponding Code:

```
[ ] #Define Target Variable
y = df.iloc[:,1]
```

```
[ ] #Define Attributes (Note: ID Column is excluded of course)
x = df.iloc[:,2:]
x.columns
```

```
Index(['17.99', '10.38', '122.8', '1001', '0.1184', '0.2776', '0.3001',
       '0.1471', '0.2419', '0.07871', '1.095', '0.9053', '8.589', '153.4',
       '0.006399', '0.04904', '0.05373', '0.01587', '0.03003', '0.006193',
       '25.38', '17.33', '184.6', '2019', '0.1622', '0.6656', '0.7119',
       '0.2654', '0.4601', '0.1189'],
      dtype='object')
```

```
[ ] #Split Validation
X_train, X_test, Y_train, Y_test = train_test_split( x, y, test_size=0.30, random_state=1, stratify=y)
```

### k-NN Model:

Standardization is necessary for the k-NN model to accurately calculate distances without attributes of larger scales influencing the distance calculated disproportionately. In order to account for different possible scaling in my variables, I normalized my features in both the training and test sets using StandardScaler from sklearn by computing the mean and standard deviations to be used for scaling. That is, after retrieving and selecting the data, I then normalized each attribute according to their z-score.

Corresponding Code:

```
#Normalization of Data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

```
[ ] #Stack the Data
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined_std = np.hstack((Y_train, Y_test))
```

The Model Induction and Model Evaluation Processes for 3NN Outlined in Python;  
Corresponding Code:



```

from sklearn import linear_model

#Logistic Regression Model
clf=linear_model.LogisticRegression(C=1e5)
clf=clf.fit(X_train,Y_train)

print('intercept', clf.intercept_) #intercept
print('coefficients', clf.coef_)   #coefficients

#Generate predictions
y_pred = clf.predict(X_test)

# 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): %.2f' % f1_score(Y_train, y_pred_insample, average='macro'))

# Kappa score
print('Kappa score (out-of-sample): ', cohen_kappa_score(Y_test, y_pred))
print('Kappa score (insample): %.2f' % cohen_kappa_score(Y_train, y_pred_insample))

# Build a text report showing the main classification metrics (out-of-sample performance)
print(classification_report(Y_test, y_pred, target_names=['benign', 'malignant']))

Confusion_Matrix=pd.DataFrame(
    confusion_matrix(Y_test, y_pred),
    columns=['Predicted benign', 'Predicted malignant'],
    index=['True bening', 'True malignant']
)
Confusion_Matrix

```

After training and applying my model to the test set, I assessed performance by calculating the accuracy, precision, recall, and f1-score using, among others, the `classification_report` function. I ended up with an out-of-sample of accuracy of 99%, which is really good. We know, however, that accuracy is a simplistic performance metric and can often be misleading. Therefore, I need to take a closer look at the other performance metrics as well. The precision and recall for both “Benign” and “Malignant” are also very high, with the precision for “Malignant” being 100% and Recall being 98%. Having a good Recall in a context like this is incredibly important, as a false negative (classifying a cell as “Benign” when it’s really “Malignant”) in this case would be extremely detrimental to the patient. A false positive, although it would be less costly than a false negative, could also cost a healthy patient financially and emotionally if he or she is misdiagnosed as having cancer. Both our false positives and false negatives are fairly low in this

case (0 false negatives and 1 false positive). I ended up with an f-measure for “Benign” 100% and for “Malignant” of 99%, both of which are very promising for the model.

Accuracy (out-of-sample): 0.99				
Accuracy (in-sample): 1.00				
F1 score (out-of-sample): 0.9937374107306354				
F1 score (in-sample): 1.00				
Kappa score (out-of-sample): 0.987475280158207				
Kappa score (insample): 1.00				
	precision	recall	f1-score	support
benign	0.99	1.00	1.00	107
malignant	1.00	0.98	0.99	64
accuracy			0.99	171
macro avg	1.00	0.99	0.99	171
weighted avg	0.99	0.99	0.99	171
	Predicted benign		Predicted malignant	
True benign	107		0	
True malignant	1		63	

### Logistic Regression Model:

Since I already completed my data exploration phase before the k-NN model, I moved right into the modelling phase. First, I split the data the same way I did for my k-NN model, using a test set size of .30 and a training set size of .70, with shuffled sampling and ‘stratify’ set to ‘y.’ I set up the model and trained it in the same way I had previously, except I used the LogisticRegression classifier. I used a parameter of C equal to 10000, which is the inverse of the regularization factor. Regularization is avoided by setting a high C, but I found that C did not have much of an impact on my model in this case, which indicates that the model is not prone to overfitting for this context and dataset. By default, the model will apply an L2 penalty.

Corresponding Code:

```

from sklearn import linear_model

#Logistic Regression Model
clf=linear_model.LogisticRegression(C=1e5)
clf=clf.fit(X_train,Y_train)

print('intercept', clf.intercept_) #intercept
print('coefficients', clf.coef_)   #coefficients

#Generate predictions
y_pred = clf.predict(X_test)

# 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): %.2f' % f1_score(Y_train, y_pred_insample, average='macro'))

# Kappa score
print('Kappa score (out-of-sample): ', cohen_kappa_score(Y_test, y_pred))
print('Kappa score (insample): %.2f' % cohen_kappa_score(Y_train, y_pred_insample))

# Build a text report showing the main classification metrics (out-of-sample performance)
print(classification_report(Y_test, y_pred, target_names=['benign', 'malignant']))

Confusion_Matrix=pd.DataFrame(
    confusion_matrix(Y_test, y_pred),
    columns=['Predicted benign', 'Predicted malignant'],
    index=['True bening', 'True malignant']
)
Confusion_Matrix

```

After training and applying the model to the test set, I assessed performance by calculating the accuracy, precision, recall, and f1-score using the classification\_report function. I ended up with an out-of-sample accuracy of 96%, which is on-par with what I got using the k-NN model, albeit a little bit lower. All of my performance metrics were pretty similar: precision for both “Benign” has slightly increased to 100% and precision for “Malignant” has slightly decreased to 98%. Recall for “Benign” has slightly decreased to 99% and for “Malignant” has increased to 100%. The f-measure for “Benign” and “Malignant” are exactly the same as before. As a whole, this model was almost the same as my k-NN model, except for the fact that we now have 1 false positive and 0 false negatives.

```

intercept [-2.51567114]
coefficients [[-1.34851142e+01 -4.49103573e+00  2.39143209e+00  3.10561994e-02
 4.46490794e-01  2.38128729e+00  3.42066410e+00  1.41505842e+00
 6.90442918e-01  1.34287777e-01 -7.17208986e-01 -8.05295998e+00
-2.97038454e+00  7.53722183e-01  3.82117664e-02  4.24216838e-01
 6.34940230e-01  1.85261382e-01  1.33728426e-01  4.24217097e-02
-1.55615761e+01  4.08892964e+00  1.04737103e+00  5.07219617e-02
 9.07746875e-01  7.69098739e+00  9.58512651e+00  2.80804259e+00
 2.10710171e+00  7.54726656e-01  3.75149563e+01]]
Index(['17.99', '10.38', '122.8', '1001', '0.1184', '0.2776', '0.3001',
      '0.1471', '0.2419', '0.07871', '1.095', '0.9053', '8.589', '153.4',
      '0.006399', '0.04904', '0.05373', '0.01587', '0.03003', '0.006193',
      '25.38', '17.33', '184.6', '2019', '0.1622', '0.6656', '0.7119',
      '0.2654', '0.4601', '0.1189'],
      dtype='object')
Accuracy (out-of-sample): 0.99
Accuracy (in-sample): 1.00
F1 score (out-of-sample): 0.9937766131673764
F1 score (in-sample): 1.00
Kappa score (out-of-sample): 0.987553679307082
Kappa score (insample): 1.00

```

	precision	recall	f1-score	support
benign	1.00	0.99	1.00	107
malignant	0.98	1.00	0.99	64
accuracy			0.99	171
macro avg	0.99	1.00	0.99	171
weighted avg	0.99	0.99	0.99	171

	Predicted benign	Predicted malignant
True bening	106	1
True malignant	0	64

We can also look at the confusion matrix in the form of a heat map, as shown below.

Corresponding Code:

```

for C in [10000]:
    logregression = LogisticRegression(C=C, random_state=1)

    logregression.fit(X_train, Y_train)

    y_pred_logregression = logregression.predict(X_test)

    print(f"C:{C}")
    print(f"Accuracy: {metrics.accuracy_score(Y_test, y_pred_logregression)}")
    print(f"Precision: {metrics.precision_score(Y_test, y_pred_logregression, pos_label='M')}")
    print(f"Recall: {metrics.recall_score(Y_test, y_pred_logregression, pos_label='M')}")
    print(f"F1 Score: {metrics.f1_score(Y_test, y_pred_logregression, pos_label='M')}")
    print("Confusion Matrix")
    print(metrics.confusion_matrix(Y_test, y_pred_logregression))
    print("\n")

#Visualize performance
visualizer = ClassificationReport(logregression)

visualizer.fit(X_train, Y_train)      # Fit the visualizer and the model
visualizer.score(X_test, Y_test)      # Evaluate the model on the test data
visualizer.poof()

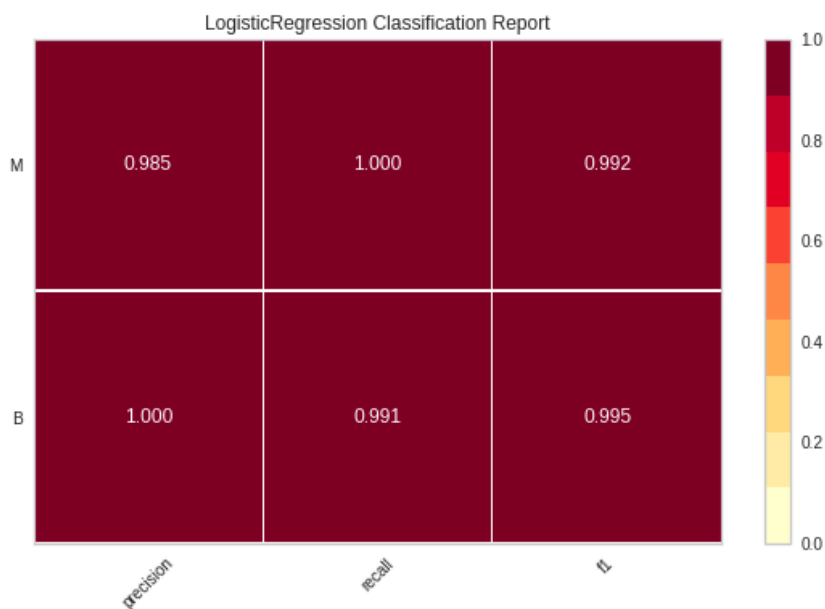
viz = ConfusionMatrix(logregression)
viz.fit(X_train, Y_train)
viz.score(X_test, Y_test)
viz.poof()

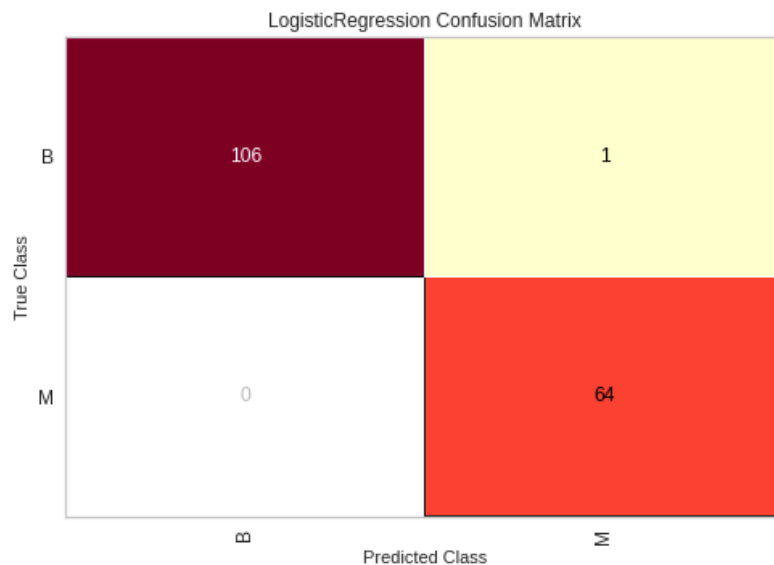
```

```

C:10000
Accuracy:0.9941520467836257
Precision: 0.9846153846153847
Recall: 1.0
F1 Score: 0.9922480620155039
Confusion Matrix
[[106  1]
 [ 0 64]]

```





### Conclusion:

Overall, LR performs extremely similar to our k-NN predictor. Although many of the performance metrics are very similar, I conclude that logistic regression outperforms kNN because it has a better recall. Recall [ True Positive / (True Positive + False Negative) ] is the best metric for evaluating the performance of a classifier on true positive values (aka malignant) as we would like to minimize the False Negatives as much as possible. Specifically, given that we are working with cancer data, our chief objective is to maximize the number of correctly detected True Positive values. These values indicate that a patient has cancer. A False Positive value (aka the patient does not have cancer, but our model indicates that he/she does) is *not as harmful or costly as* not detecting the cancer at all (which would be False Negative). It would be better to recommend a patient that is classified as a False Positive to undergo additional cancer screening tests as opposed to completely missing the fact that a patient has cancer (False Negative).

Another situation where recall or precision would be a better approach as compared to accuracy would be when we have a class imbalance (there is far more data associated with one class than the other).

Further work should be done in order to improve these predictions. One step we can try to improve our predictions in the kNN model is to try other values of K. Changing the k value changes the complexity of the model and perhaps a more complex or less complex model would yield better generalization performance for the underlying phenomenon and dataset. It is important to keep in mind the implications of these predictions. False negatives and false positives should not be viewed as equally costly. Predicting an observation as Malignant when it was in fact Benign implies that a patient has to go in for further testing or a biopsy. However, predicting that an observation is Benign when it is in fact Malignant may be life threatening for a patient. Also, we should note here that although split validation is used, we could also use cross-validation in order to estimate the generalization performance of the models. In fact, given that we have a relatively small dataset, cross-validation would make better use of the limited dataset. I will be using cross validation in the Rapidminer processes.

**20 Points (Rapidminer):**

**Perform a predictive modeling analysis on this dataset to predict the type (benign B or malignant M) using a k-NN technique (for k=3) and the Logistic Regression technique. Compare the generalization performance of the k-NN model with the Logistic Regression model. Make sure you report the confusion matrix, the predictive accuracy, precision, recall, and f-measure.**

We follow the same process on Rapidminer. The corresponding screenshots are shown below.

**a) [20 points] Please show below screenshots of the models you have built using Rapidminer, the results, and the parameters you have specified.**

**a. [8 points] Data Preview**

**i. Screenshot of the summary statistics table in Rapidminer.**

Name	Type	Missing	Statistics			Filter (32 / 32 attributes):	Search for Attribute	
▼ Id att1	Integer	0	Min 8670	Max 911320502	Average 30371831.432			
▼ Label att2	Binominal	0	Least M (212)	Most B (357)	Values B (357), M (212)			
▼ att3	Real	0	Min 6.981	Max 28.110	Average 14.127			
▼ att4	Real	0	Min 9.710	Max 39.280	Average 19.290			
▼ att5	Real	0	Min 43.790	Max 188.500	Average 91.969			
▼ att6	Real	0	Min 143.500	Max 2501	Average 654.889			
▼ att7	Real	0	Min 0.053	Max 0.163	Average 0.096			
▼ att8	Real	0	Min 0.019	Max 0.345	Average 0.104			
▼ att9	Real	0	Min 0	Max 0.427	Average 0.089			
▼ att10	Real	0	Min 0	Max 0.201	Average 0.049			
▼ att11	Real	0	Min 0.106	Max 0.304	Average 0.181			
▼ att12	Real	0	Min 0.050	Max 0.097	Average 0.063			

Showing attributes 1 – 32

Examples: 569 Special Attributes: 2 Regular Attributes: 30

Although the screenshot was not required, a preview of the data is always a good idea.

Open in

Turbo Prep

Auto Model

Filter (569 / 569 examples): 

all

Row No.	att1	att2	att3	att4	att5	att6	att7	att8	att9	att10	att11	att12	att13
1	842302	M	17.990	10.380	122.800	1001	0.118	0.278	0.300	0.147	0.242	0.079	1.09
2	842517	M	20.570	17.770	132.900	1326	0.085	0.079	0.087	0.070	0.181	0.057	0.54
3	84300903	M	19.690	21.250	130	1203	0.110	0.160	0.197	0.128	0.207	0.060	0.74
4	84348301	M	11.420	20.380	77.580	386.100	0.142	0.284	0.241	0.105	0.260	0.097	0.49
5	84358402	M	20.290	14.340	135.100	1297	0.100	0.133	0.198	0.104	0.181	0.059	0.75
6	843786	M	12.450	15.700	82.570	477.100	0.128	0.170	0.158	0.081	0.209	0.076	0.33
7	844359	M	18.250	19.980	119.600	1040	0.095	0.109	0.113	0.074	0.179	0.057	0.44
8	84458202	M	13.710	20.830	90.200	577.900	0.119	0.165	0.094	0.060	0.220	0.075	0.58
9	844981	M	13	21.820	87.500	519.800	0.127	0.193	0.186	0.094	0.235	0.074	0.30
10	84501001	M	12.460	24.040	83.970	475.900	0.119	0.240	0.227	0.085	0.203	0.082	0.29
11	845636	M	16.020	23.240	102.700	797.800	0.082	0.067	0.033	0.033	0.153	0.057	0.38
12	84610002	M	15.780	17.890	103.600	781	0.097	0.129	0.100	0.066	0.184	0.061	0.50
13	846226	M	19.170	24.800	132.400	1123	0.097	0.246	0.206	0.112	0.240	0.078	0.95
14	846381	M	15.850	23.950	103.700	782.700	0.084	0.100	0.099	0.054	0.185	0.053	0.40
15	84667401	M	13.730	22.610	93.600	578.300	0.113	0.229	0.213	0.080	0.207	0.077	0.21
16	84799002	M	14.540	27.540	96.730	658.800	0.114	0.160	0.164	0.074	0.230	0.071	0.37
17	848406	M	14.680	20.130	94.740	684.500	0.099	0.072	0.074	0.053	0.159	0.059	0.47
18	84862001	M	16.130	20.680	108.100	798.800	0.117	0.202	0.172	0.103	0.216	0.074	0.56
19	849014	M	19.810	22.150	130	1260	0.098	0.103	0.148	0.095	0.158	0.054	0.75
20	8510426	B	13.540	14.360	87.460	566.300	0.098	0.081	0.067	0.048	0.189	0.058	0.27
21	8510653	B	13.080	15.710	85.630	520	0.107	0.127	0.046	0.031	0.197	0.068	0.18
22	8510824	B	9.504	12.440	60.340	273.900	0.102	0.065	0.030	0.021	0.181	0.069	0.27
23	8511133	M	15.340	14.260	102.500	704.400	0.107	0.213	0.208	0.098	0.252	0.070	0.43

<

ExampleSet (569 examples, 2 special attributes, 30 regular attributes)

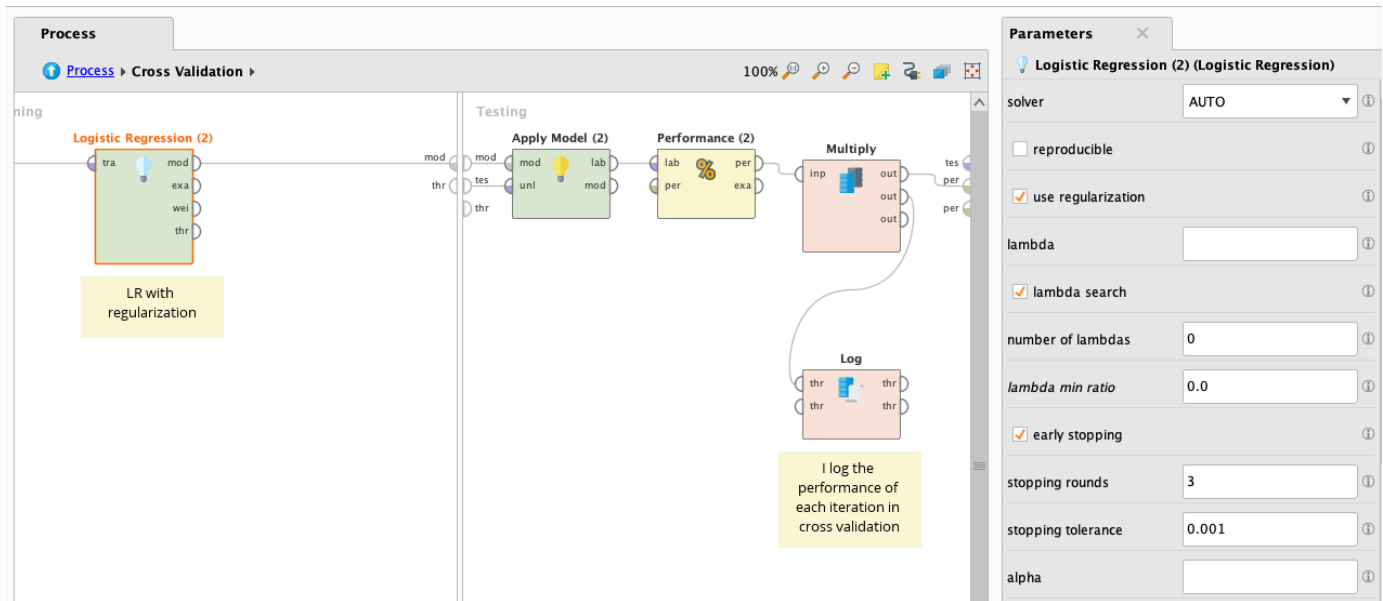
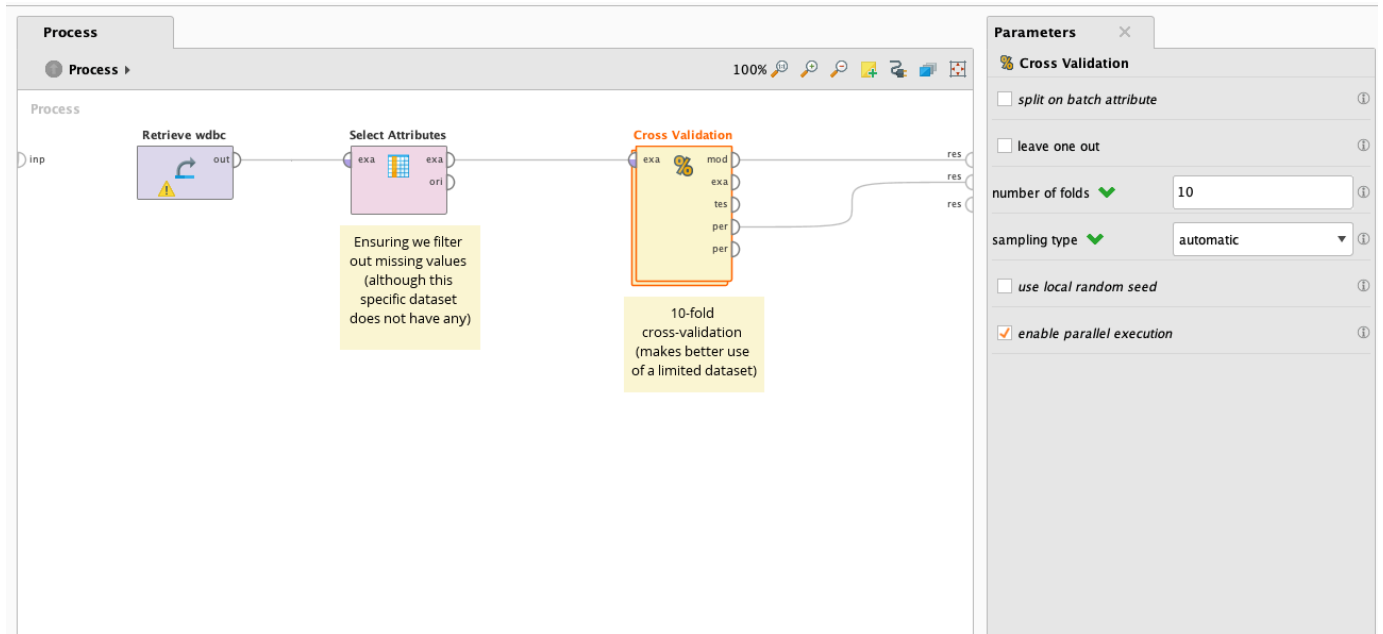
>

## b. [8 points] Logistic Regression

### i. Screenshots for Logistic Regression Model Setup (Rapidminer Processes)



(Insert Screenshots here – 2 screenshots are expected here; one for the upper layer and one inside the validation technique)



ii. Screenshot for Logistic Regression Performance

(Insert Shreenshot here)

## PerformanceVector

```
PerformanceVector:
accuracy: 97.54% +/- 1.69% (micro average: 97.54%)
ConfusionMatrix:
True:   M      B
M:      203    5
B:       9   352
AUC: 0.992 +/- 0.011 (micro average: 0.992) (positive class: B)
precision: 97.65% +/- 3.11% (micro average: 97.51%) (positive class: B)
ConfusionMatrix:
True:   M      B
M:      203    5
B:       9   352
recall: 98.59% +/- 2.01% (micro average: 98.60%) (positive class: B)
ConfusionMatrix:
True:   M      B
M:      203    5
B:       9   352
f_measure: 98.07% +/- 1.30% (micro average: 98.05%) (positive class: B)
ConfusionMatrix:
True:   M      B
M:      203    5
B:       9   352
```

### iii. Screenshot for Logistic Regression Results (Coefficients)

(Insert Shreenshot here)


Attribute	Coefficient	Std. Coefficient
att3	0	0
att4	-0.011	-0.048
att5	0	0
att6	0	0
att7	-0.587	-0.008
att8	20.221	1.068
att9	-13.557	-1.081
att10	-39.906	-1.548
att11	2.469	0.068
att12	11.720	0.083
att13	-7.549	-2.093
att14	0.979	0.540
att15	-0.133	-0.269
att16	-0.038	-1.714
att17	-147.993	-0.444
att18	45.176	0.809
att19	5.502	0.166
att20	-71.456	-0.441
att21	36.996	0.306
att22	376.756	0.997
att23	-0.342	-1.652
att24	-0.329	-2.019

att25	-0.034	-1.149
att26	-0.003	-1.756
att27	-28.991	-0.662
att28	0	0
att29	-4.567	-0.953
att30	-16.139	-1.061
att31	-15.365	-0.951
att32	-34.863	-0.630
Intercept	36.783	-0.180

iv. Screenshot for Logistic Regression Rapidminer Operator Parameters  
(click on Logistic Regression operator and then take a screenshot of the Parameters window on the right)

(Insert Screenshot here)

Parameters ×

 **Logistic Regression (2) (Logistic Regression)**

solver

AUTO ▼ ⓘ

☐ reproducible ⓘ

☒ use regularization ⓘ

lambda

ⓘ

☒ lambda search ⓘ

number of lambdas

0 ⓘ

lambda min ratio

0.0 ⓘ

☒ early stopping ⓘ

stopping rounds

3 ⓘ

stopping tolerance

0.001 ⓘ

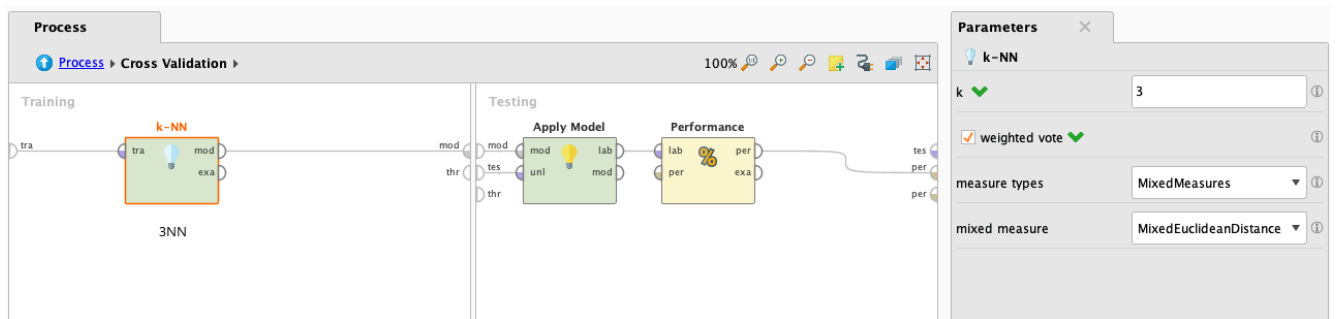
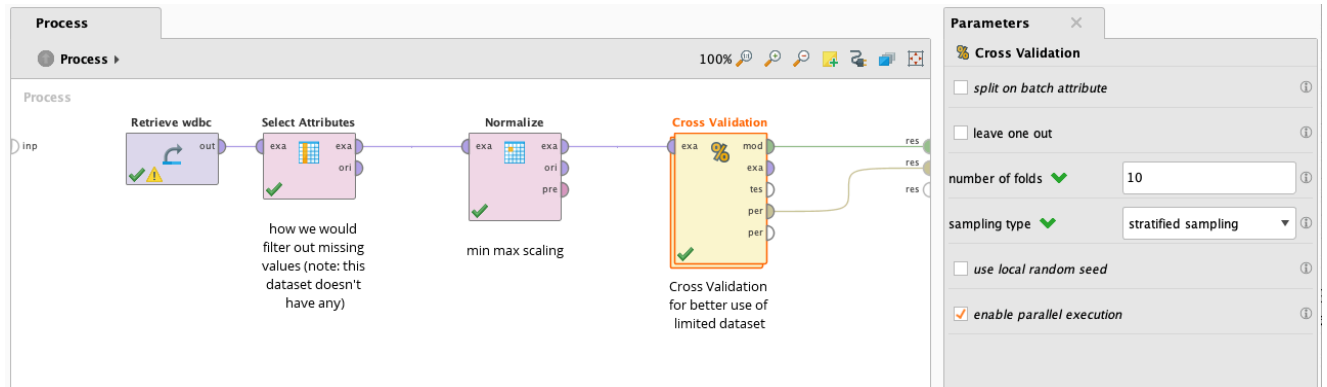
alpha

ⓘ

c. [8 points] kNN

### i. Screenshots for kNN Model Setup (Rapidminer Processes)

(Insert Screenshots here— 2 screenshots are expected here; one for the upper layer and one inside the validation technique)



### ii. Screenshot for kNN Performance

(Insert Screenshot here)

## PerformanceVector

PerformanceVector:

accuracy: 97.02% +/- 3.31% (micro average: 97.01%)

ConfusionMatrix:

True: M B

M: 199 4

B: 13 353

classification\_error: 2.98% +/- 3.31% (micro average: 2.99%)

ConfusionMatrix:

True: M B

M: 199 4

B: 13 353

kappa: 0.934 +/- 0.075 (micro average: 0.936)

ConfusionMatrix:

True: M B

M: 199 4

B: 13 353

precision: 96.73% +/- 4.87% (micro average: 96.45%) (positive class: B)

ConfusionMatrix:

True: M B

M: 199 4

B: 13 353

recall: 98.88% +/- 1.95% (micro average: 98.88%) (positive class: B)

ConfusionMatrix:

True: M B

M: 199 4

B: 13 353

f\_measure: 97.71% +/- 2.45% (micro average: 97.65%) (positive class: B)

ConfusionMatrix:

True: M B

M: 199 4

B: 13 353

iii. Screenshot for kNN Rapidminer Operator Parameters (click on kNN operator and then take a screenshot of the Parameters windows on the right)

(Insert Shreenshot here)

Parameters

k-NN

k ☒ 3

☒ weighted vote ☒

measure types MixedMeasures

mixed measure MixedEuclideanDistance

## **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 ( $\text{perimeter}^2 / \text{area} - 1.0$ )
- 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**