Post Block Assignment 1

Applied machine learning

Jeandre de Bruyn 19768206

Theoretical Questions

1. The process A1 which assigns a numerical value to each unique value of the categorical feature is a worse approach than A2 in which the categorical feature is binary encoded because the numerical encoding assumes ordinality in the categorical feature. If there is no ordinality in the categorical features then assigning numerical values creates a new relationship between the original categorical values that never existed.
   Example: If the categorical feature "colour" had the values (red, green, blue, yellow) and these values were assigned the numerical values (1,2,3,4) respectively then it assumes that blue is more than red and will lead to poor performance in a model that uses the numerically encoded colour feature.
2.

Machine Learning algorithm	Data Quality Aspect	Is it robust?	Motivation
Classification tree	Missing Values	yes	The process through which the classification tree is induced is robust to missing values.
	Noise	yes	Pruning process favours generalisation performance and as such the pruning process will remove noise
	Outliers	yes	Pruning process favours generalisation performance and as such the pruning process will remove outliers
	Skew class distribution	no	Skew class distributions make minority classes likely to be pruned off when they should not be. Oversampling a minority class can reduce skewness.
Regression tree	Missing Values	yes	should there be missing values then the algorithm uses surrogate splits to assign an instance a class even though it is computationally expensive
	Noise	yes	Regression tree is robust to noise granted that it is zero mean noise or if not zero mean that the deviation of the noise is small.
	Outliers	no	Post pruning will remove outliers in the depth of the tree but will not remove the outliers higher up. Can be avoided by clamping values within a certain range
Model Tree	Missing Values	yes	should there be missing values then the algorithm uses surrogate splits to assign an instance a class even though it is computationally expensive
	Noise	yes	Model tree is robust to noise granted that it is zero mean noise or if not zero mean that the deviation of the noise is small.
	Outliers	no	Since the model tree also uses MSE like regression trees, the post pruning process removes the deep outliers but not the ones higher up. Clamping the feature values to a certain range can reduce outliers
KNN for classification	Missing Values	no	Missing values can change the class that the entity gets assigned to as the missing value could have changed the result of the majority voting. Imputation should be used for missing values or ignoring the feature if the feature is small and has little effect.

	Noise	no	The smaller the value of k the more sensitive KNN is to noise, but the larger K is made to ignore noise effects the less precise the algorithm becomes. Trying to find the right balance of K to fight the noise yet retain accuracy will lead to better results.
	Outliers	yes	Due to the nature of the algorithm the outliers will not be included in the group of nearest neighbours and won't have an effect on the result. Will only have an effect at very large k and even then majority voting should fight effect of outlier.
	Skew class distribution	no	The majority class will have a significant effect on the algorithm outcome due to the system using majority voting, especially as k increases. The way to mitigate this is to oversample the minority class
KNN for regression	Missing Values	yes	Due to the nature of the regression using average values for prediction missing values won't have a large effect on the outcome.
	Noise	no	With low values of K the regression model is still sensitive to noise. Trying to find the right balance of K to fight the noise yet retain accuracy will lead to better results.
	Outliers	yes	Robust against feature based outliers (input vector), the same as classification where the outlier will not be part of the nearest neighbour group.

3. Choosing features with strong predictive power is important for machine learning algorithms because feeding an algorithm unrelated information is going to produce poor results. Not only this but making sure that the data that is used as the input for the algorithm needs to be in the correct data type and format or there will be no predictions at all.
4. (A)

P(ym) for each class:

3	2	4	6	5	8	7
0.125	0.125	0.125	0.25	0.125	0.125	0.125

Log2 for each P(ym):

3	2	4	6	5	8	7
-3	-3	-3	-2	-3	-3	-3

-SUM of P(ym)*log$_2$P(ym):

3	2	4	6	5	8	7	SUM
0.125	0.125	0.125	0.25	0.125	0.125	0.125	
-3	-3	-3	-2	-3	-3	-3	
-0.375	-0.375	-0.375	-0.5	-0.375	-0.375	-0.375	2.75

(B) Even numbers

	2	4	6	8	
freq	1	1	2	1	
P	0.2	0.2	0.4	0.2	
log2	-2.3219281	-2.3219281	-1.3219281	-2.3219281	
times	-0.4643856	-0.4643856	-0.5287712	-0.4643856	1.92192809
					SUM

Odd Numbers:

	3	5	7	
freq	1	1	1	
P	0.33333333	0.33333333	0.33333333	
log2	-1.5849625	-1.5849625	-1.5849625	
times	-0.5283208	-0.5283208	-0.5283208	1.5849625
				SUM

Finding Hx(D):

	Fraction in decimal	Odd and even H(D)	Po*H(Do)
5/8	0.63	1.92192809	1.20
3/8	0.38	1.5849625	0.59
SUM			1.80

Gain = 2.75-1.80 = 0.95

PBA1
Question 2

May 22, 2021

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.neighbors import KNeighborsClassifier
import sklearn.metrics
from matplotlib import pyplot
import pandas_profiling
from imblearn.over_sampling import SMOTE
```

[2]:
```
def without_hue(plot, feature):
    total = len(feature)
    for p in plot.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height()/total)
        x = p.get_x() + p.get_width() / 2 - 0.05
        y = p.get_y() + p.get_height()
        ax.annotate(percentage, (x, y), size = 12)
    plt.show()
```

0.1 1

[3]:
```
breast_train = pd.read_csv('/Users/jeandre/Desktop/Applied Machine Learning/
↪Post Block Assignemnt 1/breastCancerTrain.csv', sep='\t')
```

[4]:
```
breast_test = pd.read_csv('/Users/jeandre/Desktop/Applied Machine Learning/Post␣
↪Block Assignemnt 1/breastCancerTest.csv', sep='\t')
```

[5]:
```
breast = pd.concat([breast_test,breast_train])
breast.replace('?',value = np.nan,inplace=True)
```

[6]:
```
print(type(breast))
```

```
<class 'pandas.core.frame.DataFrame'>
```

[7]:
```
breast.head()
```

[7]:
```
         id diagnosis radius_mean texture_mean perimeter_mean area_mean  \
0    842517         M       20.57        17.77          132.9      1326
1    843786         M       12.45         15.7          82.57     477.1
2  84610002         M       15.78        17.89          103.6       781
3    846381         M       15.85        23.95          103.7     782.7
4   8510824         B       9.504        12.44          60.34     273.9

  smoothness_mean compactness_mean concavity_mean concave points_mean  …  \
0         0.08474          0.07864         0.0869             0.07017  …
1          0.1278             0.17         0.1578             0.08089  …
2          0.0971           0.1292        0.09954             0.06606  …
3         0.08401           0.1002        0.09938             0.05364  …
4          0.1024          0.06492        0.02956             0.02076  …

  perimeter_worst area_worst smoothness_worst compactness_worst  \
0           158.8       1956           0.1238            0.1866
1           103.4      741.6           0.1791            0.5249
2           136.5       1299           0.1396            0.5609
3             112      876.5           0.1131            0.1924
4           65.13      314.9           0.1324            0.1148

  concavity_worst concave points_worst symmetry_worst fractal_dimension_worst  \
0          0.2416                0.186          0.275                 0.08902
1          0.5355               0.1741         0.3985                 0.12440
2          0.3965                0.181         0.3792                 0.10480
3          0.2322               0.1119         0.2809                 0.06287
4         0.08867              0.06227          0.245                 0.07773

  gender    Bratio
0      F  0.065443
1      F  0.481372
2      F  0.789345
3      F  0.532400
4      F  0.475519

[5 rows x 34 columns]
```

[8]:
```
breast.dropna(inplace=True)
breast.drop('gender',axis=1,inplace =True)
column_list = list(breast.columns.values)
column_list.remove('id')
column_list.remove('diagnosis')
print(column_list)
for col in column_list:
    breast[col] = pd.to_numeric(breast[col], downcast="float")
```

```
['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean',
'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se',
'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se', 'concavity_se',
'concave points_se', 'symmetry_se', 'fractal_dimension_se', 'radius_worst',
'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst',
'compactness_worst', 'concavity_worst', 'concave points_worst',
'symmetry_worst', 'fractal_dimension_worst', 'Bratio']
```

1 General data quality issues:

- Double Tabbed heading in a tab delimited file.
- Gender column contains only female entries.
- All columns contain mixed data types; strings and numbers.
- Some values in columns are represented as a question mark.
- Large outlier in smoothness_mean
- Eleven entries contain only zeros for six features.
- Skew class distribution, 50% more negative cases than positive.

1.1 2

a)

1.2 Issues addressed to enable analysis:

- Double tabbed heading fixed and csv succesfully read into pandas.without doing this the data was impossible to manipulate.
- cast mixed data types to float except for id and diagnosis. Mixed data types are difficult to work with in terms of visulizations etc.
- drop gender feature since all entries are female. Does not provide any information.
- replaced all question marks with NaN and then dropped any rows containing NaN. This takes the datset from 569 entries to 457 entries. Question marks are not valid inputs for our modelling process.

1.3 Issues found in analysis

- Extreme large outlier in smoothness_mean, only one such outlier. Remove since it only affects a single row
- 6 Features containing 11 zeros: "[concavity_mean , concave points_mean , concabity_se , concave_points_se , concavity_worst , concave points_worst]" Wont be changing these entries since the features are all very low valued.

1.4 b)

1.4.1 any other transform:

Remove ID since it is unique for every entry

Carry out the fixes onto the individual train and test sets. Previously only implemented on the two sets combined.

[9]:
```
for df in [breast_test,breast_train]:
    df.replace('?',value = np.nan,inplace=True)
    df.dropna(inplace=True)
    df.drop('gender',axis=1,inplace =True)
    column_list = list(df.columns.values)
    column_list.remove('id')
    column_list.remove('diagnosis')
    for col in column_list:
        df[col] = pd.to_numeric(df[col], downcast="float")
```

c)

[10]:
```
X_test = breast_test[list(breast_test.columns[2:33])]
X_train = breast_train[list(breast_train.columns[2:33])]
X_train # check that we have the correct columns
```

[10]:
```
     radius_mean  texture_mean  perimeter_mean    area_mean  smoothness_mean  \
0      17.990000     10.380000      122.800003  1001.000000          0.11840
1      19.690001     21.250000      130.000000  1203.000000          0.10960
2      11.420000     20.379999       77.580002   386.100006          0.14250
3      20.290001     14.340000      135.100006  1297.000000          0.10030
5      13.710000     20.830000       90.199997   577.900024          0.11890
..           …             …              …           …              …
379     9.333000     21.940001       59.009998   264.000000          0.09240
385    11.200000     29.370001       70.669998   386.000000          0.07449
386    15.220000     30.620001      103.400002   716.900024          0.10480
387    20.129999     28.250000      131.199997  1261.000000          0.09780
388    16.600000     28.080000      108.300003   858.099976          0.08455

     compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
0             0.27760         0.30010              0.14710         0.2419
1             0.15990         0.19740              0.12790         0.2069
2             0.28390         0.24140              0.10520         0.2597
3             0.13280         0.19800              0.10430         0.1809
5             0.16450         0.09366              0.05985         0.2196
..                …             …                    …            …
379           0.05605         0.03996              0.01282         0.1692
385           0.03558         0.00000              0.00000         0.1060
386           0.20870         0.25500              0.09429         0.2128
```

```
387           0.10340        0.14400              0.09791         0.1752
388           0.10230        0.09251              0.05302         0.1590

     fractal_dimension_mean  …  texture_worst  perimeter_worst   area_worst  \
0                   0.07871  …      17.330000       184.600006  2019.000000
1                   0.05999  …      25.530001       152.500000  1709.000000
2                   0.09744  …      26.500000        98.870003   567.700012
3                   0.05883  …      16.670000       152.199997  1575.000000
5                   0.07451  …      28.139999       110.599998   897.000000
..                      …  …            …              …            …
379                 0.06576  …      25.049999        62.860001   295.799988
385                 0.05502  …      38.299999        75.190002   439.600006
386                 0.07152  …      42.790001       128.699997   915.000000
387                 0.05533  …      38.250000       155.000000  1731.000000
388                 0.05648  …      34.119999       126.699997  1124.000000

     smoothness_worst  compactness_worst  concavity_worst  \
0             0.16220            0.66560          0.71190
1             0.14440            0.42450          0.45040
2             0.20980            0.86630          0.68690
3             0.13740            0.20500          0.40000
5             0.16540            0.36820          0.26780
..                …                …              …
379           0.11030            0.08298          0.07993
385           0.09267            0.05494          0.00000
386           0.14170            0.79170          1.17000
387           0.11660            0.19220          0.32150
388           0.11390            0.30940          0.34030

     concave points_worst  symmetry_worst  fractal_dimension_worst    Bratio
0                 0.26540          0.4601                  0.11890  0.162878
1                 0.24300          0.3613                  0.08758  0.751106
2                 0.25750          0.6638                  0.17300  0.465537
3                 0.16250          0.2364                  0.07678  0.969993
5                 0.15560          0.3196                  0.11510  0.440695
..                    …             …                      …        …
379               0.02564          0.2435                  0.07393  0.217251
385               0.00000          0.1566                  0.05905  0.228154
386               0.23560          0.4089                  0.14090  0.435295
387               0.16280          0.2572                  0.06637  0.164469
388               0.14180          0.2218                  0.07820  0.097137

[294 rows x 31 columns]
```

[11]:
```
Y_test = breast_test["diagnosis"]
Y_train = breast_train["diagnosis"]
```

```
values = {'B':1, 'M':0}

Y_test = Y_test.map(values)
Y_train = Y_train.map(values)
Y_train
```

[11]:
```
0      0
1      0
2      0
3      0
5      0
      ..
379    1
385    1
386    0
387    0
388    0
Name: diagnosis, Length: 294, dtype: int64
```

1.4.2 Classification tree approach:

A classification tree is trained to overfit on the training set. This is done by splitting the dataset further and further untill each subset is as homogeneous as possible i.e. contains only one class with each split attempting to maximize the homogeneity in the following subsets.

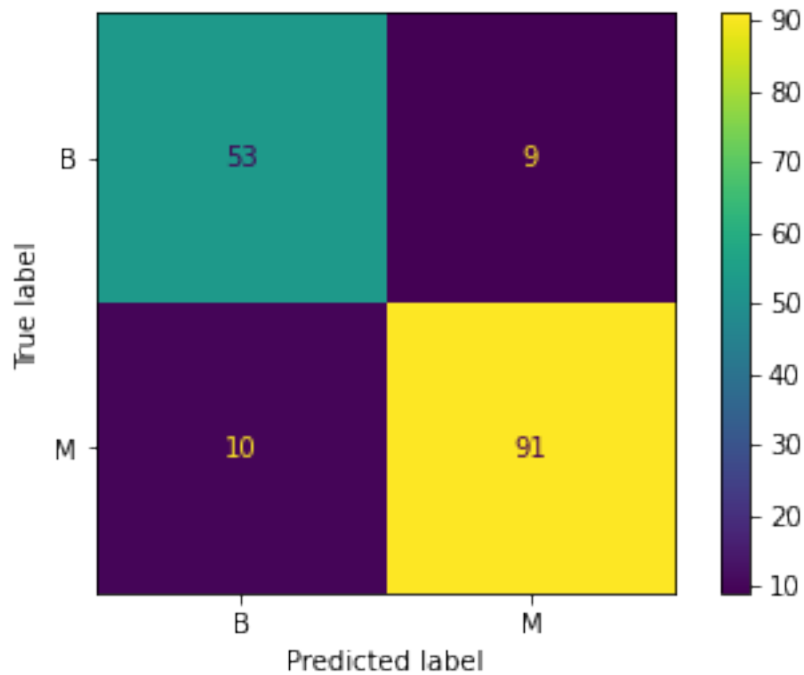
The approach to finding optimal splits is a recursive one by which the function splits the set into subsets recursively untill they are homogeneous at which point the recursion ends.

[12]:
```
treeinst = tree.DecisionTreeClassifier(random_state=0)
treeclf = treeinst.fit(X_train, Y_train)
D_pred = treeclf.predict(X_test)
print("Accuracy score =",sklearn.metrics.accuracy_score(Y_test,D_pred))
print("Precision score =",sklearn.metrics.
 ↪precision_score(Y_test,D_pred,average='weighted'))
print("Recall = ",sklearn.metrics.
 ↪recall_score(Y_test,D_pred,average='weighted'))
```

```
Accuracy score = 0.8834355828220859
Precision score = 0.8838572402376083
Recall =  0.8834355828220859
```

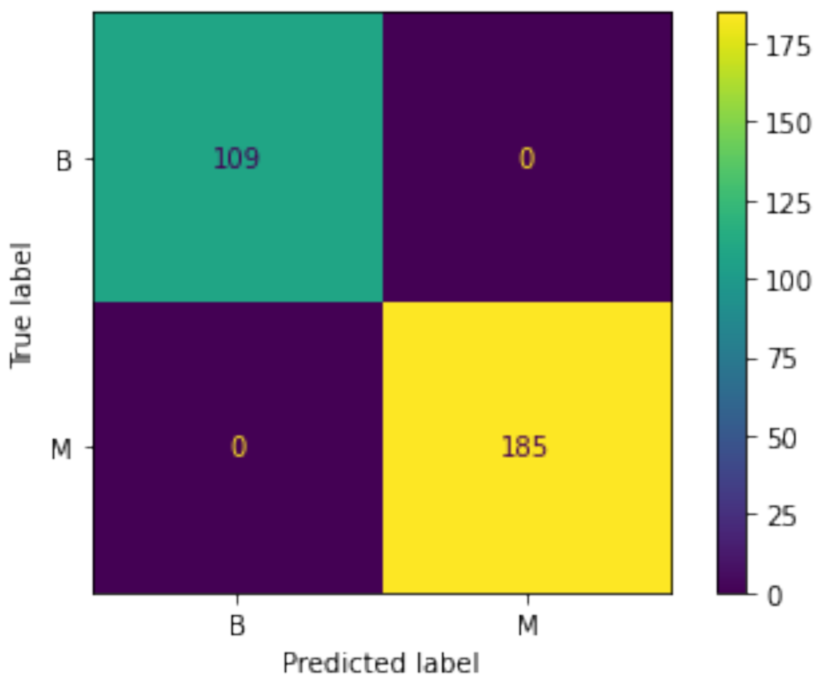
[13]:
```
sklearn.metrics.
 ↪plot_confusion_matrix(treeclf,X_test,Y_test,display_labels=values) # TEST SET
```

[13]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fe143640fa0>
```




```
[14]: sklearn.metrics.
      ↪plot_confusion_matrix(treeclf,X_train,Y_train,display_labels=values) # TRAIN␣
      ↪SET
```

```
[14]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
      0x7fe13efd1340>
```



Complete overfit on the training set

d)

1.4.3 Post Pruning process used:

Minimal cost-complexity pruning.

Minimal cost complexity uses cost complexity defined as: $R_\alpha(T) = R(T) + \alpha|\widetilde{T}|$ where T is a given tree and $\tilde{T}$ being the number of terminal nodes.

As the name implies this method aims to minimize the $R_\alpha(T)$ of a tree. This is done by evaluating the cost complexity of a node and using the equation $\alpha_{eff}(t) = \frac{R(t)-R(T_t)}{|T|-1}$ to calculate $\alpha_{eff}(t)$, a non terminal node with the lowest value for $\alpha_{eff}(t)$ will be the weakest link and pruned. The pruning process continues untill the value for $\alpha_{eff}(t)$ is greater than ccp_alpha

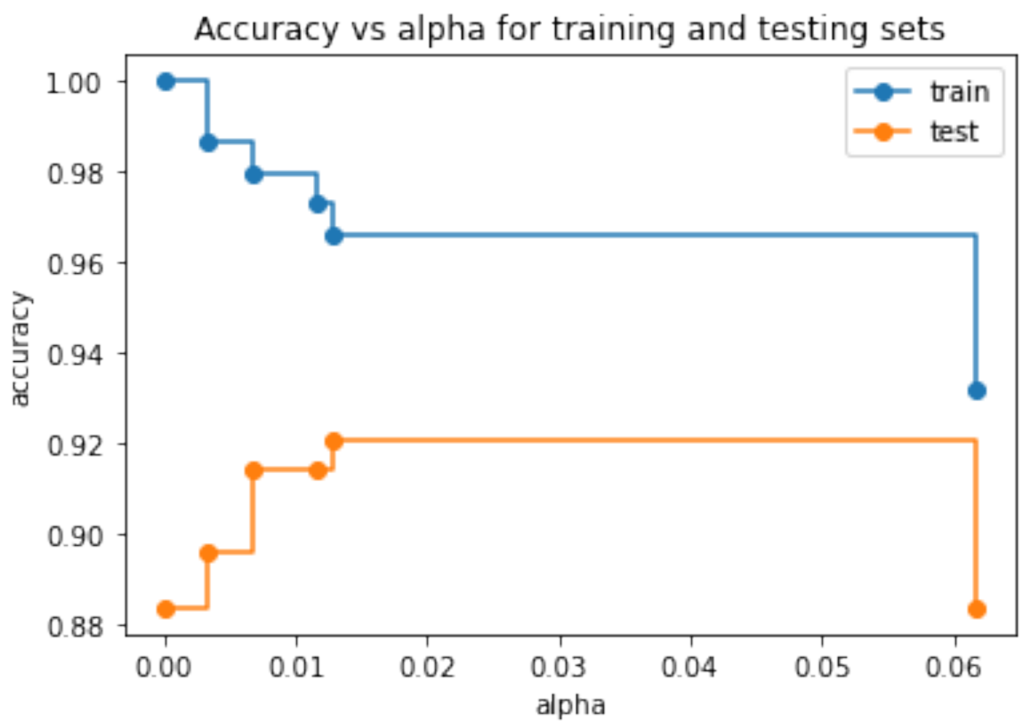
[15]:
```
path = treeclf.cost_complexity_pruning_path(X_train, Y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

[16]:
```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train, Y_train)
    clfs.append(clf)
```

[17]:
```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]
```

[18]:
```
train_scores = [clf.score(X_train, Y_train) for clf in clfs]
test_scores = [clf.score(X_test, Y_test) for clf in clfs]

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.show()
```



[19]:
```
print(ccp_alphas)
```

```
[0.         0.00332701 0.00665951 0.01166181 0.01276977 0.06172553]
```

alpha = 0.0128 provides the best test accuracy

[20]:
```
ccp_alpha = ccp_alphas[4]
print(ccp_alpha)
```

```
0.012769774214623466
```

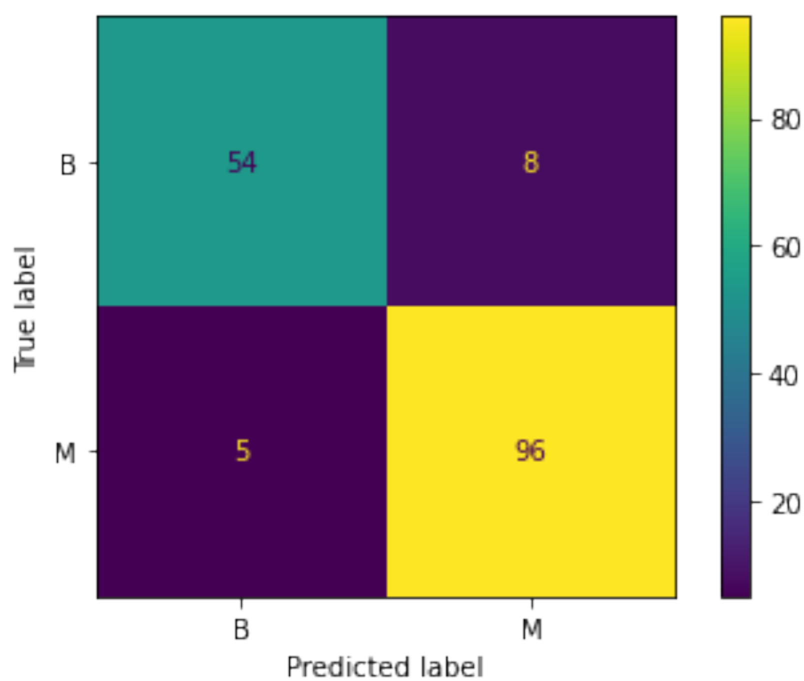
[21]:
```
treeinst = tree.DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
treeclf = treeinst.fit(X_train, Y_train)
D_pred = treeclf.predict(X_test)
print("Accuracy score =",sklearn.metrics.accuracy_score(Y_test,D_pred))
print("Precision score =",sklearn.metrics.
 ↪precision_score(Y_test,D_pred,average='weighted'))
print("Recall = ",sklearn.metrics.
 ↪recall_score(Y_test,D_pred,average='weighted'))
sklearn.metrics.confusion_matrix(Y_test,D_pred)
```

```
Accuracy score = 0.9202453987730062
Precision score = 0.9201014229609425
Recall =  0.9202453987730062
```

[21]:
```
array([[54,  8],
       [ 5, 96]])
```

[22]:
```
sklearn.metrics.
 ↪plot_confusion_matrix(treeclf,X_test,Y_test,display_labels=values) # TEST SET
```

[22]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fe143d3b280>
```

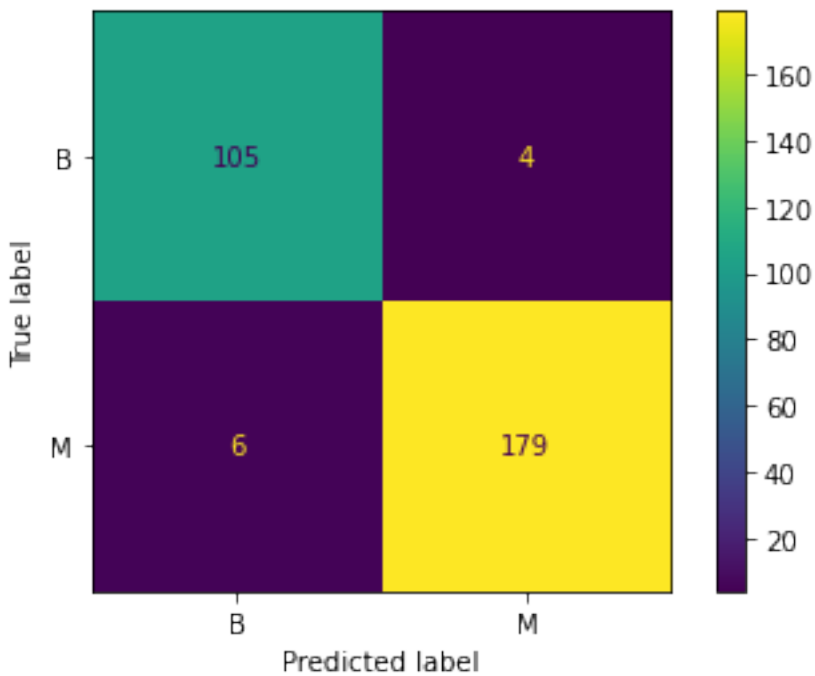


54/62= 87.1% accuracy for B

96/101 = 95% accuracy for M

[23]:
```
sklearn.metrics.
 ↪plot_confusion_matrix(treeclf,X_train,Y_train,display_labels=values) # TRAIN␣
 ↪SET
```

[23]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fe143d41dc0>



e) Pruning helps provide a more accurate model by eliminating nodes that contribute the most to overfitting the training set, i.e. pruning eliminates the nodes that contain outliers. As a result of the pruning the tree becomes less overfitted and therefore a more generalized model which improves accuracy overall and on the testing set.

1.5 3) KNN

1.5.1 Data Quality issues:

Many of the same as the Decision tree including:

- Double tabbed heading fixed and csv succesfully read into pandas.without doing this the data was impossible to manipulate.
- cast mixed data types to float except for id and diagnosis. Mixed data types are difficult to work with in terms of visulizations etc.
- drop gender feature since all entries are female. Does not provide any information.
- replaced all question marks with NaN and then dropped any rows containing NaN. This takes the datset from 569 entries to 457 entries. Question marks are not valid inputs for our modelling process.

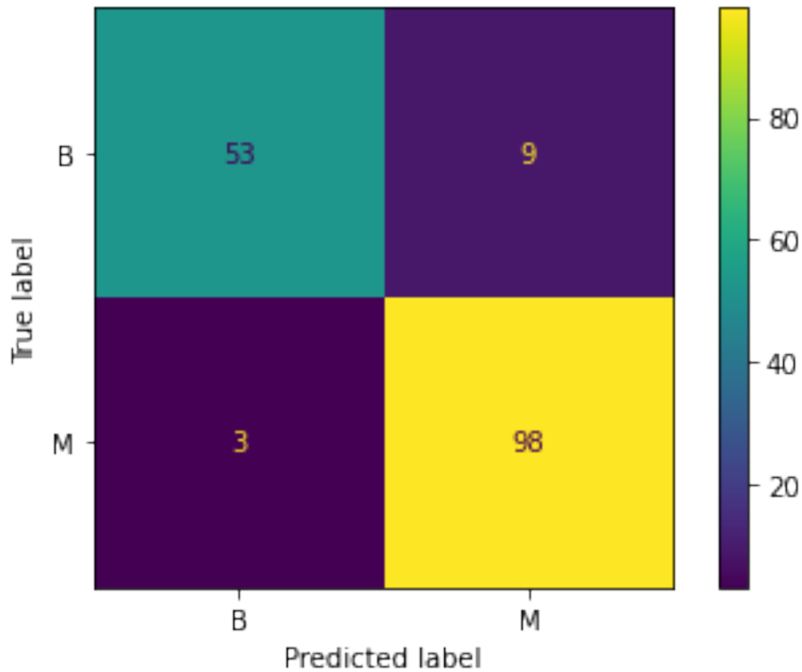
1.6 Issues found in analysis

- Extreme large outlier in smoothness_mean, only one such outlier. Remove since it only affects a single row
- 6 Features containing 11 zeros: "[concavity_mean , concave points_mean , concabity_se , concave_points_se , concavity_worst , concave points_worst]" Wont be changing these entries since the features are all very low valued and could be valid entries.
- High noise in the data, will have to pick the right value for K to find a balance against robustness for noise and precision.

[24]:
```
k=5
neigh = KNeighborsClassifier(n_neighbors=k)
neigh.fit(X_train,Y_train)
K_pred = neigh.predict(X_test)
print("K is =",k)
print("Accuracy score =",sklearn.metrics.accuracy_score(Y_test,K_pred))
print("Precision score =",sklearn.metrics.
 ↪precision_score(Y_test,K_pred,average='weighted'))
print("Recall = ",sklearn.metrics.
 ↪recall_score(Y_test,K_pred,average='weighted'))
sklearn.metrics.plot_confusion_matrix(neigh,X_test,Y_test,display_labels=values)
```

```
K is = 5
Accuracy score = 0.9263803680981595
Precision score = 0.9275045664157526
Recall =  0.9263803680981595
```

[24]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fe147551850>
```



53/62= 85.4% accuracy for B

98/101 = 97% accuracy for M

Given the setting with regards to breast cancer it is much more important to catch as many of the cases as possible of malignant tumors, having a few false alarms when misidentifying a benign tumor as malignant is worth it to have as few as possible malignant tumors slip through without identification. For this reason the KNN is the better solution as it has the better sensitivity to identifying malignant tumors while having around the same base accuracy score as the decision tree. Only 3% of the malignant cases were missed in the KNN method versus 5% for the decision tree.

```
[ ]:
```

Question 3

May 22, 2021

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.tree import DecisionTreeClassifier, plot_tree,␣
 ↪DecisionTreeRegressor
import sklearn.metrics
from matplotlib import pyplot
import pandas_profiling
```

[2]:
```
def without_hue(plot, feature):
    total = len(feature)
    for p in plot.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height()/total)
        x = p.get_x() + p.get_width() / 2 - 0.05
        y = p.get_y() + p.get_height()
        ax.annotate(percentage, (x, y), size = 12)
    plt.show()
```

[3]:
```
bike = pd.read_csv('/Users/jeandre/Desktop/Applied Machine Learning/Post Block␣
 ↪Assignemnt 1/SeoulBikeData.csv',encoding='latin1')
```

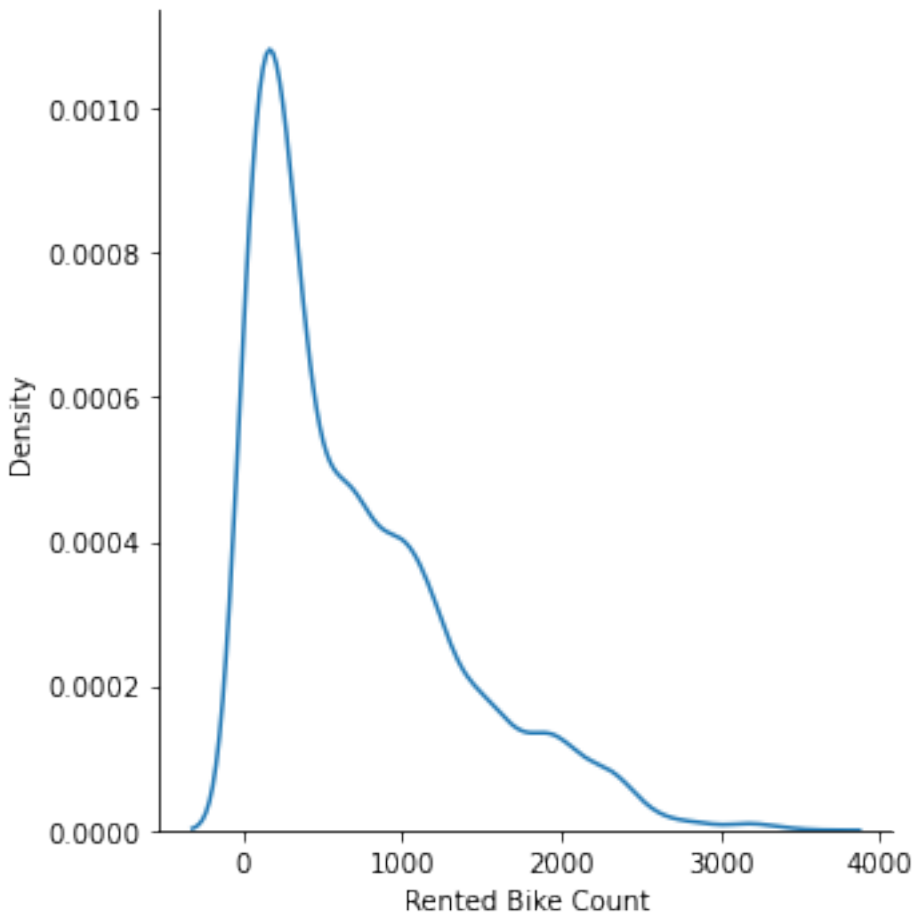
0.0.1 Data quality findings and actions:

- Date will be removed as it has extreme high cardinality and time of year is represented by season.
- Need to encode seasons values since strings are not compatible with the modeling process used. Will use dummy encoding and dropping first column.
- The Amount of rented bikes will need to be binned. Having the model predict a distinct value of 2166 possible combinations is going to lead to poor results. With a maximum of 3556 and minimum of 0 it will be binned into quartiles. This provides a good balance between accuracy and business value.
- Removal of Dew point temperature due to its very high correlation between itself and temperature.

- Removal of “Functioning days” since there is no reason to predict the amount of bikes that will be rented when the rental business is closed.
- Dummy encoding of the holiday variable to make it compatible with the modeling processes used.
- Creation of a new vector based hour system due to the linear time system not representing the closeness of 23 and 1.
- Removal of snowfall and rainfall since 95%+ are zeros.

[4]:
```
sns.displot(data = bike, x = "Rented Bike Count", kind = "kde" ) # distribution␣
↪of "average feature"
```

[4]:
```
<seaborn.axisgrid.FacetGrid at 0x7fbe6e5e4f40>
```



[5]:
```
bike.drop('Dew point temperature(°C)',axis=1,inplace =True)
bike.drop('Functioning Day',axis=1,inplace =True)
bike.drop('Date',axis=1,inplace =True)
bike.drop('Rainfall(mm)',axis=1,inplace =True)
bike.drop('Snowfall (cm)',axis=1,inplace =True)
```

[6]:

```
dummy_season = pd.get_dummies(bike['Seasons'],drop_first=True)
bike.drop('Seasons',axis=1,inplace =True) # no use for this after we have the␣
↪dummy encoding

dummy_holiday = pd.get_dummies(bike['Holiday'],drop_first=True)
bike.drop('Holiday',axis=1,inplace =True) # no use for this after we have the␣
↪dummy encoding
```

[7]:

```
#creating new feature based on hours that better represents the ralationhip by␣
↪converting it from an ordinal linear
# relationship to a circular one. better shows the delta between 23 and 1 for␣
↪eg.
bike["x"] = np.sin(np.radians(bike["Hour"]*15)) # sin and cos switched so that␣
↪0 and 24 are situated at the top.
bike["y"] = np.cos(np.radians(bike["Hour"]*15)) # Thus we have a vector that␣
↪represents the time in a circular domain.
bike.head(2)
```

[7]:

```
   Rented Bike Count  Hour  Temperature(°C)  Humidity(%)  Wind speed (m/s)  \
0                254     0             -5.2           37               2.2
1                204     1             -5.5           38               0.8

   Visibility (10m)  Solar Radiation (MJ/m2)         x         y
0              2000                      0.0  0.000000  1.000000
1              2000                      0.0  0.258819  0.965926
```

[8]:

```
X = bike[list(bike.columns[2:])]
X = X.merge(dummy_season,left_index=True, right_index=True) # adding in the␣
↪dummy encodings of seasons
X = X.merge(dummy_holiday,left_index=True, right_index=True) # adding in the␣
↪dummy encodings of holidays
X.head(2)
```

[8]:

```
   Temperature(°C)  Humidity(%)  Wind speed (m/s)  Visibility (10m)  \
0             -5.2           37               2.2              2000
1             -5.5           38               0.8              2000

   Solar Radiation (MJ/m2)         x         y  Spring  Summer  Winter  \
0                      0.0  0.000000  1.000000       0       0       1
1                      0.0  0.258819  0.965926       0       0       1

   No Holiday
0           1
1           1
```

```
[9]: bike["Rented Bike Count"] = pd.to_numeric(bike["Rented Bike Count"],␣
     ↪downcast="float")
     Y = bike["Rented Bike Count"]
     Y.head(2)
```

```
[9]: 0    254.0
     1    204.0
     Name: Rented Bike Count, dtype: float32
```

```
[10]: X_train, X_test, Y_train, Y_test = sklearn.model_selection.
      ↪train_test_split(X,Y, test_size = 0.4, random_state = 0)
```

1 KNN

1.0.1 Description:

KNN is an uncomplicated classification and regression algorithm that assigns the class of an unknown entity based on the class of its k nearest neighbors (k being decided by the implementer of the algorithm). The algorithm takes the unknown entities from the test set and the known entities that it bases its decisions on from the training set.

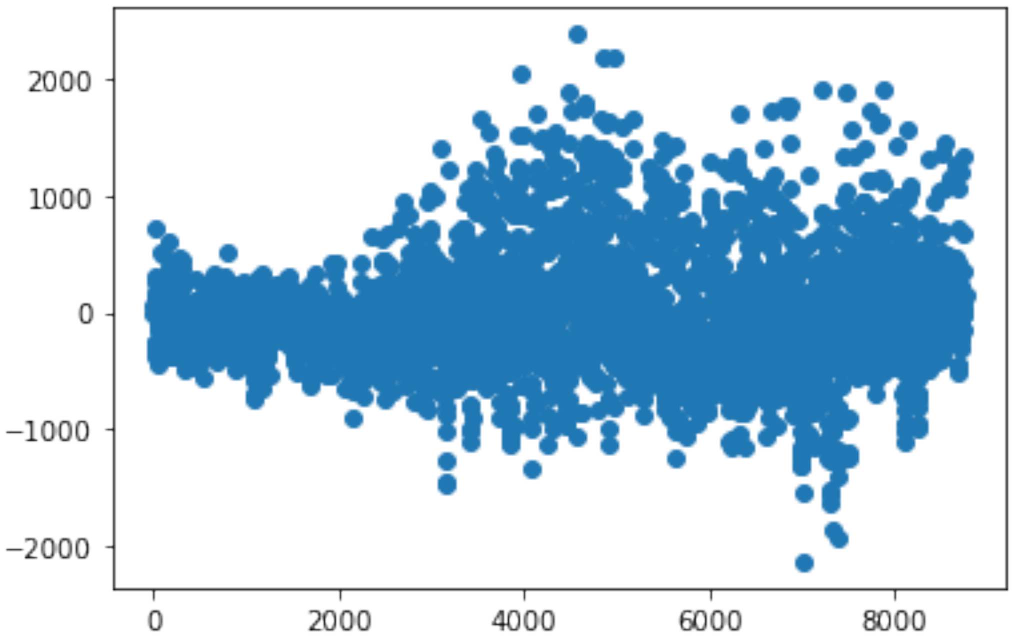
Below, after several K's tested it was found that k=12 was the best performing value for k.

```
[11]: k=12
      neigh = KNeighborsRegressor(n_neighbors=k)
      neigh.fit(X_train,Y_train)
      K_pred = neigh.predict(X_test)
      print("K is =",k)
      print("Score = ",neigh.score(X_test,Y_test))
```

```
K is = 12
Score =  0.40867017742648526
```

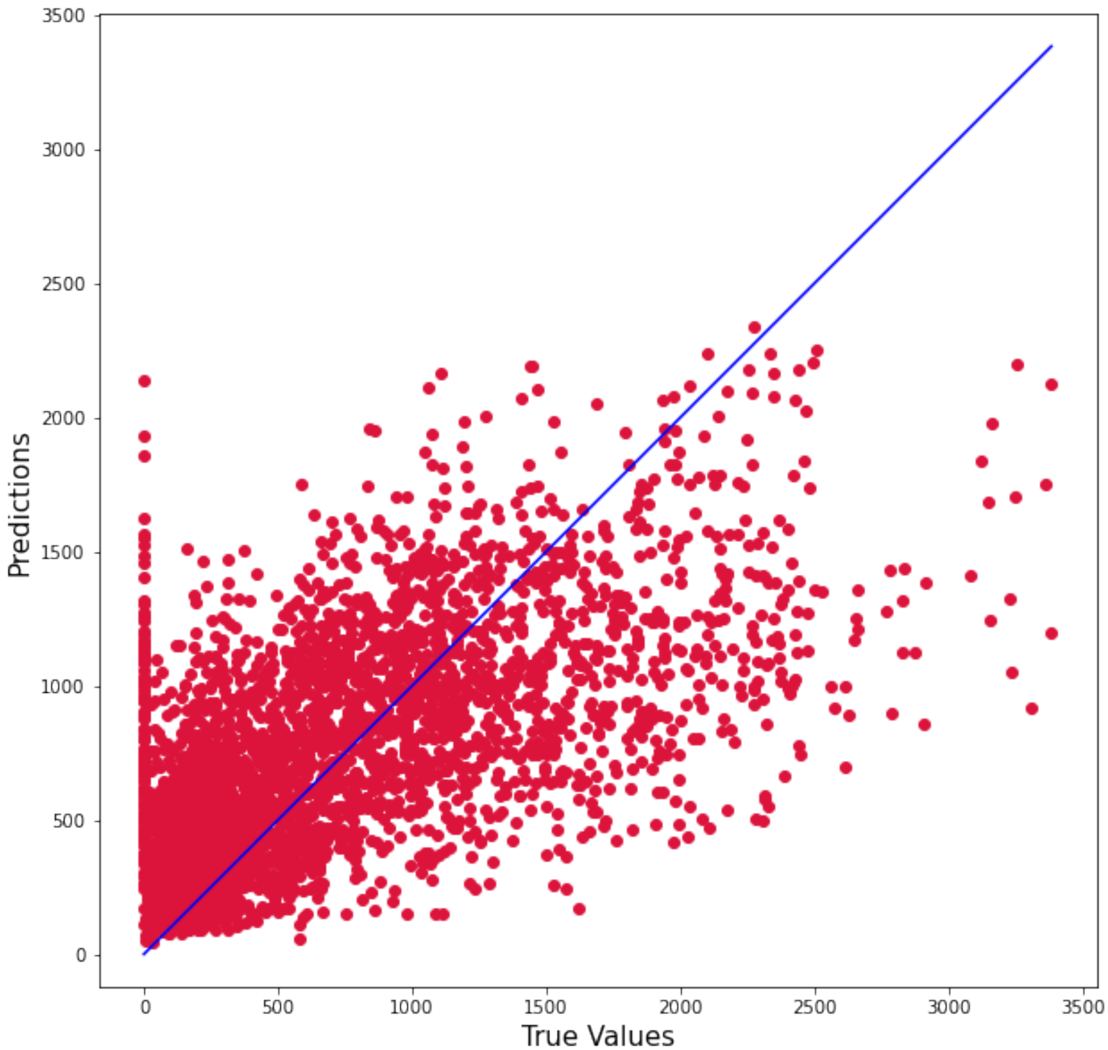
```
[12]: plt.plot(Y_test - K_pred,marker='o',linestyle='')
```

```
[12]: [<matplotlib.lines.Line2D at 0x7fbe6e71a4f0>]
```



[13]:
```
plt.figure(figsize=(10,10))
plt.scatter(Y_test, K_pred, c='crimson')

p1 = max(max(K_pred), max(Y_test))
p2 = min(min(K_pred), min(Y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.show()
```



2 Decision Tree

2.0.1 Description:

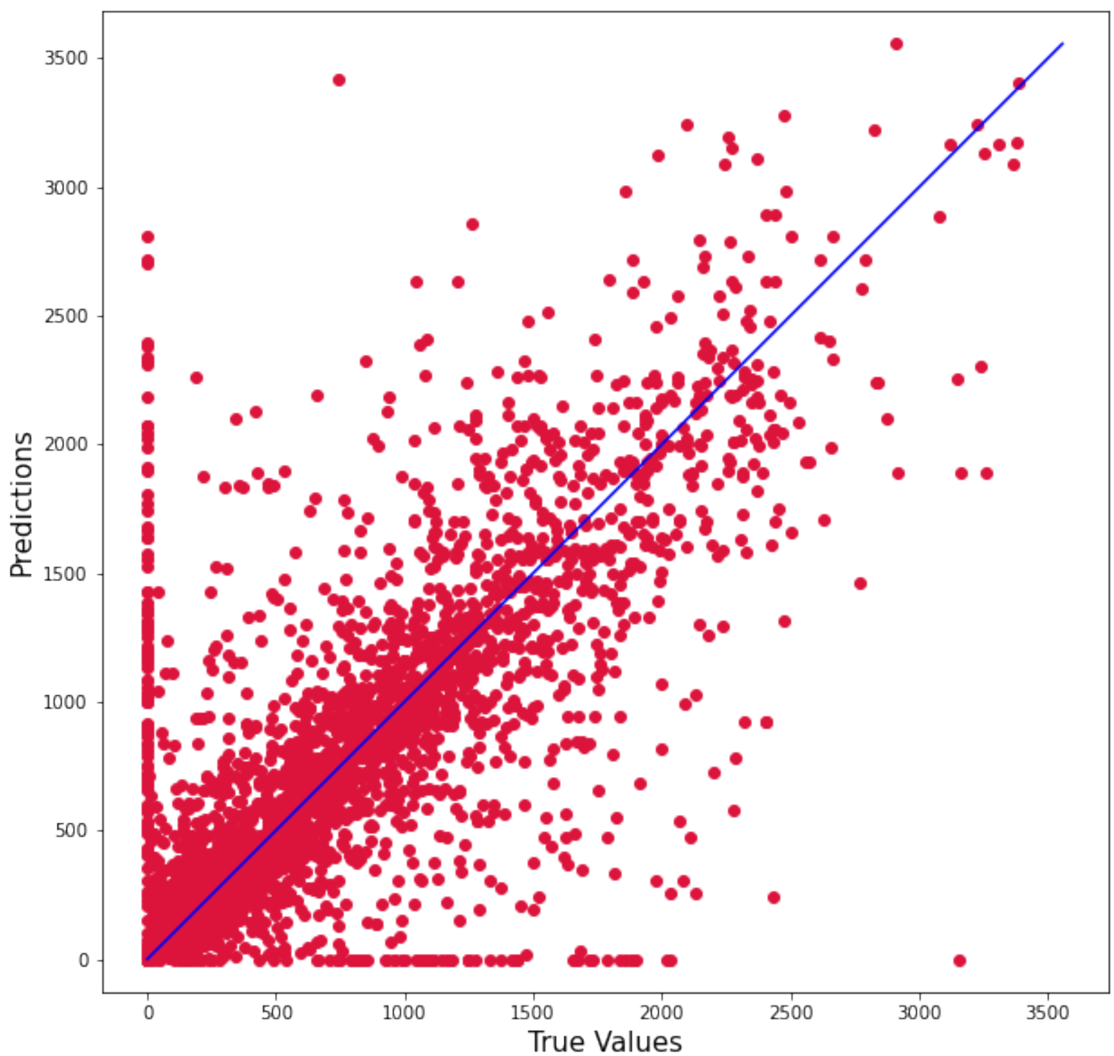
Decision trees classify unknown entities from the testing set by following a tree where each node describes a decision boundary in the state space. The algorithm follows this tree until it terminates at a node at which point it knows what to classify the unknown entity as. The decision boundaries are learnt from the training set by subdividing each state space recursively untill it reaches maximum homogeneity, thus the tree is overfitted to the training set. In order to further generalise the model the tree is pruned to remove the most overfitting branches and make the model more accurate in a general usecase. In this instance minimal cost complexity is used to prune the tree.

```
[14]: treeinst = tree.DecisionTreeRegressor(random_state=0)
      treeclf = treeinst.fit(X_train, Y_train)
      D_pred = treeclf.predict(X_test)
      print("Score = ",treeclf.score(X_test,Y_test))
```

```
Score =  0.5297513506229128
```

```
[15]: plt.figure(figsize=(10,10))
      plt.scatter(Y_test, D_pred, c='crimson')

      p1 = max(max(D_pred), max(Y_test))
      p2 = min(min(D_pred), min(Y_test))
      plt.plot([p1, p2], [p1, p2], 'b-')
      plt.xlabel('True Values', fontsize=15)
      plt.ylabel('Predictions', fontsize=15)
      plt.axis('equal')
      plt.show()
```



from the two plots of KNN and Decision trees comparing predicted values versus true values it is clear to see that the Decision tree has its values closer to the blue line (representing 1:1 match between predicted and true values). The decision tree also has a much higher R^2 score, 0.53 vs 0.41. KNN has much fewer occurrences of predicting a zero when the true value is some other value than the decision tree (seen by the line of zero values at the bottom of the graph) yet both mispredict higher values when the true value is zero. KNN has a trend of underpredicitng the true value.

The decision tree is the better pick for this regression task of predicting bike rentals. There was no pruning of the decision tree in this implementation.