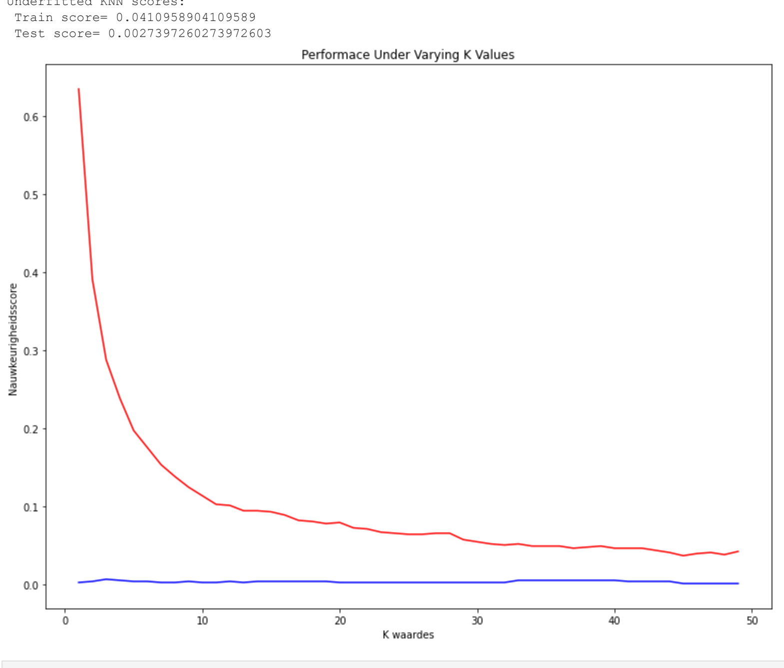
```python
#Initiatie en fitting van hiervoor genoemde modellen
overfitted_knn = KNeighborsClassifier(n_neighbors=1).fit(X_train,y_train)
underfitted_knn = KNeighborsClassifier(n_neighbors=50).fit(X_train,y_train)

#Meting en presentatie van hiervoor genoemde modellen
overfitted_train_score=overfitted_knn.score(X_train,y_train)
overfitted_test_score=overfitted_knn.score(X_test,y_test)

underfitted_train_score=underfitted_knn.score(X_train,y_train)
underfitted_test_score=underfitted_knn.score(X_test,y_test)
print("Overfitted KNN scores:\n Train score= " + str(overfitted_train_score) + "\n Test score= " + str(overfitte
print("Underfitted KNN scores:\n Train score= " + str(underfitted_train_score) + "\n Test score= " + str(underf
```

```
Overfitted KNN scores:
 Train score= 0.6342465753424658
 Test score= 0.0027397260273972603
Underfitted KNN scores:
 Train score= 0.0410958904109589
 Test score= 0.0027397260273972603
```



In [21]:
```python
parameters = { 'n_neighbors' : range(1, 50, 2),
               'weights' : ['uniform','distance'],
               'metric' : ['minkowski','euclidean','manhattan']}
grid_search = GridSearchCV(KNeighborsClassifier(), parameters, verbose = 1, cv=3, n_jobs = -1)
grid_results = grid_search.fit(X_train, y_train)
print("Best: %f using %s" % (grid_results.best_score_, grid_results.best_params_))
```

```
Fitting 3 folds for each of 150 candidates, totalling 450 fits
C:\Users\Sener\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:676: UserWarning: The least popula
ted class in y has only 1 members, which is less than n_splits=3.
  warnings.warn(
Best: 0.010951 using {'metric': 'minkowski', 'n_neighbors': 43, 'weights': 'uniform'}
```

GridSearchCV()

In [22]:
```python
from sklearn.datasets import load_diabetes
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, train_test_split
from sklearn.ensemble import RandomForestRegressor
import numpy as np
```

In [23]:
```python
#Hier hebben we gebruik gemaakt van een reshape om de shape van de arrays te veranderen om de grid_search te ku
X_test = np.array(X_test).reshape(-1, 1)
X_train = np.array(X_train).reshape(-1, 1)
```

In [24]:
```python
#De grid search wordt hier uitgevoerd om de beste waardes voor de hyperparameters te bepalen
grid_search = GridSearchCV(RandomForestRegressor(random_state=0),
                           {'n_estimators':np.arange(5,100,5),
                            'max_features':np.arange(0.1,1.0,0.05),
                            'max_depth': [1, 2, 3, 4, 5],
                           },cv=5, scoring="r2",verbose=1,n_jobs=-1)
grid_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 1710 candidates, totalling 8550 fits
```

Out[24]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=0), n_jobs=-1,
             param_grid={'max_depth': [1, 2, 3, 4, 5],
                         'max_features': array([0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55, 0.6
,
       0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95]),
                         'n_estimators': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,
85,
       90, 95])},
             scoring='r2', verbose=1)
```

In [25]:
```python
print(" Results van Random Search " )
print("\n Beste schatter over de gezochte parameters:\n", grid_search.best_estimator_)
print("\n Beste score over alle gezochte parameters parameters:\n", grid_search.best_score_)
print("\n Beste parameters over alle gezochte parameters:\n", grid_search.best_params_)
```

```
 Results van Random Search

 Beste schatter over de gezochte parameters:
 RandomForestRegressor(max_depth=3, max_features=0.1, n_estimators=70,
                       random_state=0)

 Beste score over alle gezochte parameters parameters:
 0.39362183869436185

 Beste parameters over alle gezochte parameters:
 {'max_depth': 3, 'max_features': 0.1, 'n_estimators': 70}
```

Het model testen

Hier testen we het model met de beste parameters verkregen vanuit de grid seatch op de test dataset

In [26]:
```python
def algoritme(x_data):
    X_train, X_test, y_train, y_test = train_test_split(X2,y2 ,test_size=0.2, random_state = 1)
    rfreg = RandomForestRegressor(n_estimators = 40, max_features = 0.1, max_depth = 2).fit(X_train,y_train)
    scores = cross_val_score(rfreg, X_test, y_test, cv=5)
    return scores.mean()
```

Classificatie model

Week 1 lassificatie

In [27]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
```

In [28]:
```python
df = pd.read_csv('data.csv')
#Dataset inladen om aan de slag te gaan met de data
```

In [29]:
```python
df.head()
```

Out[29]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 33 columns

In [30]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [31]:
```python
df.describe()
```

Out[31]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	co points_
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.0
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.0
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.0
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.0
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.0
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.0
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.0
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.2

8 rows × 32 columns

In [32]:
```python
df.isnull().sum()
```

Out[32]:
```
id                          0
diagnosis                   0
radius_mean                 0
texture_mean                0
perimeter_mean              0
area_mean                   0
smoothness_mean             0
compactness_mean            0
concavity_mean              0
concave points_mean         0
symmetry_mean               0
fractal_dimension_mean      0
radius_se                   0
texture_se                  0
perimeter_se                0
area_se                     0
smoothness_se               0
compactness_se              0
concavity_se                0
concave points_se           0
symmetry_se                 0
fractal_dimension_se        0
radius_worst                0
texture_worst               0
perimeter_worst             0
area_worst                  0
smoothness_worst            0
compactness_worst           0
concavity_worst             0
concave points_worst        0
symmetry_worst              0
fractal_dimension_worst     0
Unnamed: 32               569
dtype: int64
```

In [33]:
```python
#De bovenstaande stappen hebben we gedaan, om de data te verkennen. Eerst hebben we gekeken wat voor data het i
```

In [34]:
```python
#Dropping 32 columns that contain NaN values
df.drop('Unnamed: 32', axis = 1, inplace = True)
```

In [35]:
```python
df
```

Out[35]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800
...	...	...	...	...	...	...	...	...	...
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000

569 rows × 32 columns

In [36]:
```python
df.corr()
```

Out[36]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_m
id	1.000000	0.074626	0.099770	0.073159	0.096893	-0.012968	0.000096	0.050
radius_mean	0.074626	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676
texture_mean	0.099770	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302
perimeter_mean	0.073159	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716
area_mean	0.096893	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685
smoothness_mean	-0.012968	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521
compactness_mean	0.000096	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883
concavity_mean	0.050080	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000
concave points_mean	0.044158	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921
symmetry_mean	-0.022114	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500
fractal_dimension_mean	-0.052511	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336
radius_se	0.143048	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.631
texture_se	-0.007526	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.076
perimeter_se	0.137331	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.660
area_se	0.177742	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.617
smoothness_se	0.096781	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.098
compactness_se	0.033961	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.670
concavity_se	0.055239	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.691
concave points_se	0.078768	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.683
symmetry_se	-0.017306	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.178
fractal_dimension_se	0.025725	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.449
radius_worst	0.082405	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.688
texture_worst	0.064720	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.299
perimeter_worst	0.079986	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.729
area_worst	0.107187	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.675
smoothness_worst	0.010338	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	0.448
compactness_worst	-0.002968	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	0.754
concavity_worst	0.023203	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	0.884
concave points_worst	0.035174	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	0.861
symmetry_worst	-0.044224	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	0.409
fractal_dimension_worst	-0.029866	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	0.514

31 rows × 31 columns

In [37]:
```python
#In deze stap hebben we gekeken waar de verbanden lagen in de verschillende gegevens. Door deze stap hebben we
```

In [38]:
```python
# Getting Features

x = df.drop(columns = 'diagnosis')

# Getting Predicting Value
y = df['diagnosis']
```

In [39]:
```python
#train_test_splitting of the dataset
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=0)
```

In [40]:
```python
print(len(x_train))
```

```
455
```

In [41]:
```python
print(len(x_test))
```

```
114
```

In [42]:
```python
print(len(y_train))
```

```
455
```

In [43]:
```python
print(len(y_test))
```

```
114
```

In [44]:
```python
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[44]:
```
RandomForestClassifier()
```

In [45]:
```python
y_pred=rfc.predict(x_test)
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,mean_squared_error
print(classification_report(y_test,y_pred))
print("Training Score: ",rfc.score(x_train,y_train)*100)
```

```
              precision    recall  f1-score   support

           B       0.97      0.97      0.97        67
           M       0.96      0.96      0.96        47

    accuracy                           0.96       114
   macro avg       0.96      0.96      0.96       114
weighted avg       0.96      0.96      0.96       114

Training Score:  100.0
```

In [46]:
```python
print(accuracy_score(y_test,y_pred)*100)
```

```
96.49122807017544
```

In [47]:
```python
print(confusion_matrix(y_test,y_pred))
```

```
[[65  2]
 [ 2 45]]
```

Week 2 Classificatie

In [48]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
```

In [49]:
```python
# Het train en test verschil met betrekking op de grootte van de dataset, bepaal je zelf. We hebben gekozen om
# Dit betekent automatisch dat de train ste bestaat uit 80%.
# De prestatiemetrics wijzen uit dat er weldegelijk verschillen zijn met betrekking op de 2 verschillende sets.
```

In [50]:
```python
df2 = pd.read_csv('data.csv')
```

In [51]:
```python
###  Independent And dependent features
X=df.iloc[:,2:]
y=df.iloc[:,1]
```

In [52]:
```python
X=X.dropna(axis=1)
```

In [53]:
```python
y.value_counts()
```

Out[53]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [54]:
```python
#Na het verwijderen van de NaN waardes, hebben we geconcludeerd dat de gegevens van B/M correct zijn
```

In [55]:
```python
from sklearn import datasets
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import plot_roc_curve, plot_confusion_matrix
from sklearn.naive_bayes import GaussianNB
```

In [56]:
```python
cls1 = RandomForestClassifier()
cls1.fit(x_train,y_train)
```

Out[56]:
```
RandomForestClassifier()
```

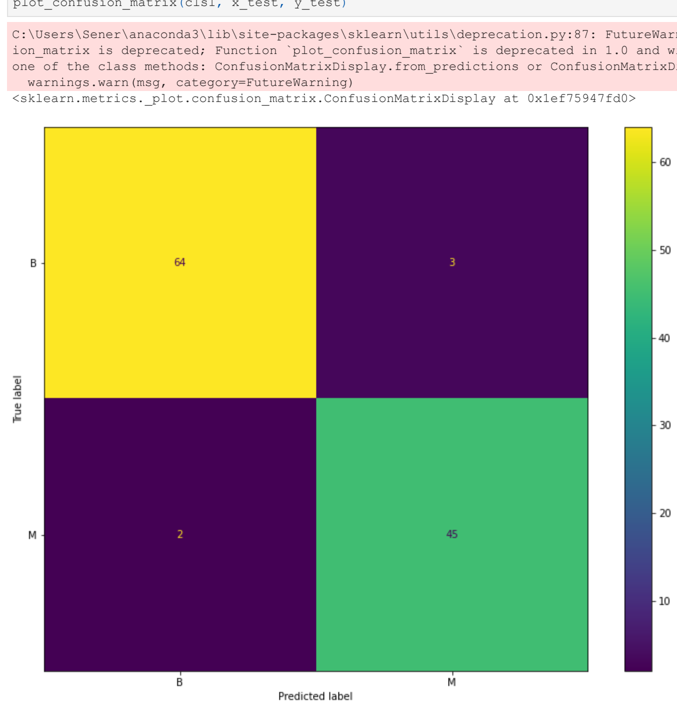
In [57]:
```
#We hebben alle benodigde packages geimporteerd om de matrixen te kunnen maken
```

Prestatiematrix

In [58]:
```
#Prestatiematrix
plot_confusion_matrix(cls1, x_test, y_test)
```

```
C:\Users\Sener\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[58]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1ef75947fd0>
```



In [59]:
```
#Prestatiematrix
plot_confusion_matrix(cls1, x_train, y_train)
```

```
C:\Users\Sener\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[59]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1ef6f450ac0>
```

In [60]:
```
#Uit de verschillende prestatiemaxtrixen, is te zien dat er een duidelijk verschil is tussen de aantal waardes
#Zichtbaar is, dat de testdata beschikt over meerdere fouten ten opzichte van de train data. Dat is te zien in
#Dit is opvallend, omdat de train data over meer gegevens beschikt, maar minder waardes in de paarse vlakken he
#We hebben voor deze prestatiematrix gekozen (random forrest), omdat we al onderzocht hadden dat een random for
#Een RFC kan goed voorspellen of het schatten van kansen op gebeurtenissen, dat is precies wat we doen in deze
#Een nadeel is dat het erg ingewikkeld kan zijn doordat er gebruik wordt gemaakt van 'meerdere bomen'.
```

In [61]:
```
print(confusion_matrix(y_test,y_pred))
```

```
[[65  2]
 [ 2 45]]
```

In [62]:
```
#K-FOLD analyse
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
result = model.score(X_test, y_test)
print(result)
```

```
0.9035087719298246
```

In [63]:
```
from sklearn.model_selection import KFold
model=DecisionTreeClassifier()
kfold_validation=KFold(10)
```

In [64]:
```
import numpy as np
from sklearn.model_selection import cross_val_score
results=cross_val_score(model,X,y,cv=kfold_validation)
print(results)
print(np.mean(results))
```

```
[0.94736842 0.9122807  0.87719298 0.94736842 0.89473684 0.96491228
 0.9122807  0.92982456 0.96491228 0.94642857]
0.9297305764411028
```

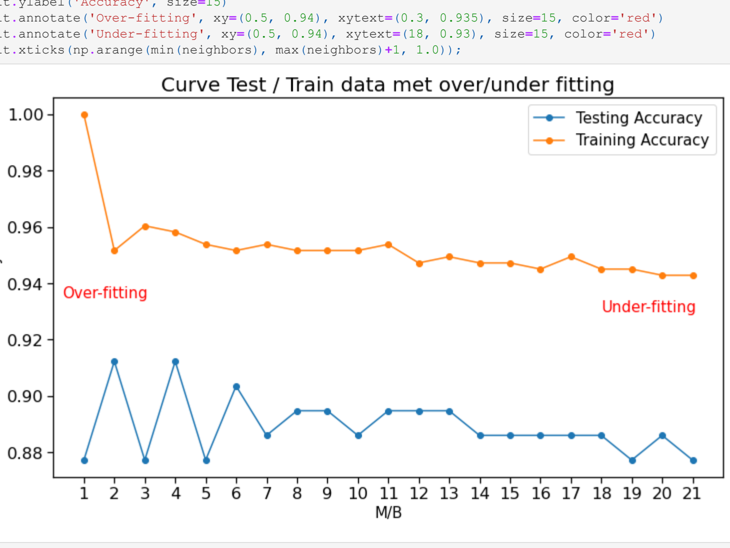
In [65]:
```
#Voor de K-fold analyse hebben we voor K de waarde 10 gegeven, omdat we deze waarde vaak terug zagen komen bij
#Er is een duidelijk verschil te zien in de test en train data met dezelfde waarde voor K.
#Testwaarde = 90,6%, Trainwaarde = 93,7%
```

Underfit en overfit model

In [66]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)

#Hier laten we de zien hoe precies het model is
print('Accuracy KNN(1): ', knn.score(X_test, y_test))
```

```
Accuracy KNN(1):  0.8771929824561403
```

In [67]:
```
neighbors = np.arange(1, 22)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop verschillende waardes van K
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)

plt.figure(figsize=(12,7))
sns.set_context('notebook', font_scale=1.5)
plt.title('Curve Test / Train data met over/under fitting', size=20)
plt.plot(neighbors, test_accuracy, marker ='o', label = 'Testing Accuracy')
plt.plot(neighbors, train_accuracy, marker ='o', label = 'Training Accuracy')
plt.legend(prop={'size':15})
plt.xlabel('M/B', size=15)
plt.ylabel('Accuracy', size=15)
plt.annotate('Over-fitting', xy=(0.5, 0.94), xytext=(0.3, 0.935), size=15, color='red')
plt.annotate('Under-fitting', xy=(0.5, 0.94), xytext=(18, 0.93), size=15, color='red')
plt.xticks(np.arange(min(neighbors), max(neighbors)+1, 1.0));
```



In [68]:
```
#In dit model is weergegeven dat er een significant verschil is tussen de test set en de train set
#Dit gaat over hoe precies de set is.
#Het model is onderfit, omdat het model geen gegevens leert en het patroon niet extraheert.
```

RandomizedSearchCV()

In [69]:
```
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
```

In [70]:
```
rf_grid = {"n_estimators":np.arange(10, 1000, 20),
           "max_depth":[None, 3, 5, 10],
           "max_features":["sqrt", "log2", "auto"],
           "min_samples_split":np.arange(2, 20, 2),
           "min_samples_leaf":np.arange(1, 20, 2)}
```

In [71]:
```
random_rf = RandomizedSearchCV(RandomForestClassifier(),
                               param_distributions=rf_grid,
                               cv=5,
                               n_iter=20,
                               n_jobs=-1,
                               random_state=0)

random_rf.fit(X_train, y_train)


print(f"Train score: {random_rf.score(X_train, y_train)}")
print(f"Test score: {random_rf.score(X_test, y_test)}")
```

```
Train score: 0.9868131868131869
Test score: 0.9385964912280702
```

In [72]:
```
#Er is gekozen voor deze parameters, omdat het verschil in test en train data nu goed wordt weergegeven.
#Hier is de score per set af te lezen, er is een significant verschil in beide sets, met een percentageverschil
```

In [73]:
```
random_rf.best_params_
```

Out[73]:
```
{'n_estimators': 70,
 'min_samples_split': 8,
 'min_samples_leaf': 3,
 'max_features': 'log2',
 'max_depth': 5}
```

In [74]:
```
# Use parameters for the best model
best_rf = RandomForestClassifier()
best_rf.set_params(**random_rf.best_params_)

# Fit best model
best_rf.fit(x_train, y_train)

# Score best model
best_rf.score(x_test, y_test)
```

Out[74]:
```
0.6403508771929824
```

In [75]:
```
#In de regels hierboven, worden de beste parameters voor het betreffende model in het model gestopt, om zo een
```