Out[204]:
```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'frame': None,
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) dataset\n-----------------------------
---------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 569\n\n    :Number of Attributes: 30 n
umeric, predictive attributes and the class\n\n    :Attribute Information:\n        - radius (mean of distances from
center to points on the perimeter)\n        - texture (standard deviation of gray-scale values)\n        - perimeter
\n        - area\n        - smoothness (local variation in radius lengths)\n        - compactness (perimeter^2 / are
a - 1.0)\n        - concavity (severity of concave portions of the contour)\n        - concave points (number of con
cave portions of the contour)\n        - symmetry\n        - fractal dimension ("coastline approximation" - 1)\n\n
The mean, standard error, and "worst" or largest (mean of the three\n        worst/largest values) of these features
were computed for each image,\n        resulting in 30 features.  For instance, field 0 is Mean Radius, field\n
10 is Radius SE, field 20 is Worst Radius.\n\n        - class:\n                - WDBC-Malignant\n                -
WDBC-Benign\n\n    :Summary Statistics:\n\n    ===================================== ====== ======\n
Min    Max\n    ===================================== ====== ======\n    radius (mean):                        6.981
28.11\n    texture (mean):                       9.71   39.28\n    perimeter (mean):                     43.79  188.
5\n    area (mean):                          143.5  2501.0\n    smoothness (mean):                    0.053  0.163\n
compactness (mean):                   0.019  0.345\n    concavity (mean):                     0.0    0.427\n    conc
ave points (mean):                0.0    0.201\n    symmetry (mean):                      0.106  0.304\n    fractal
dimension (mean):             0.05   0.097\n    radius (standard error):              0.112  2.873\n    texture (sta
ndard error):             0.36   4.885\n    perimeter (standard error):           0.757  21.98\n    area (standard e
rror):                6.802  542.2\n    smoothness (standard error):          0.002  0.031\n    compactness (standar
d error):         0.002  0.135\n    concavity (standard error):           0.0    0.396\n    concave points (standard
error):     0.0    0.053\n    symmetry (standard error):            0.008  0.079\n    fractal dimension (standard e
rror):   0.001  0.03\n    radius (worst):                       7.93   36.04\n    texture (worst):
12.02  49.54\n    perimeter (worst):                    50.41  251.2\n    area (worst):                         185.
2  4254.0\n    smoothness (worst):                   0.071  0.223\n    compactness (worst):                  0.027
1.058\n    concavity (worst):                    0.0    1.252\n    concave points (worst):               0.0    0.29
1\n    symmetry (worst):                     0.156  0.664\n    fractal dimension (worst):            0.055  0.208\n
===================================== ====== ======\n\n    :Missing Attribute Values: None\n\n    :Class Distributio
n: 212 - Malignant, 357 - Benign\n\n    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian\n\n
:Donor: Nick Street\n\n    :Date: November, 1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) da
tasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of a fine needle\naspirate (FNA) of a
breast mass.  They describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating plane described
above was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nConstruction Via Linear P
rogramming." Proceedings of the 4th\nMidwest Artificial Intelligence and Cognitive Science Society,\npp. 97-101, 199
2], a classification method which uses linear\nprogramming to construct a decision tree.  Relevant features\nwere se
lected using an exhaustive search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual linear prog
ram used to obtain the separating plane\nin the 3-dimensional space is that described in:\n[K. P. Bennett and O. L.
Mangasarian: "Robust Linear\nProgramming Discrimination of Two Linearly Inseparable Sets",\nOptimization Methods and
Software 1, 1992, 23-34].\n\nThis database is also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\nc
d math-prog/cpo-dataset/machine-learn/WDBC/\n\n.. topic:: References\n\n   - W.N. Street, W.H. Wolberg and O.L. Mang
asarian. Nuclear feature extraction \n     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on \n
Electronic Imaging: Science and Technology, volume 1905, pages 861-870,\n     San Jose, CA, 1993.\n   - O.L. Mangasa
```

```
rian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and \n     prognosis via linear programming. Operations Research, 43(4), pages 570-577, \n     July-August 1995.\n   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques\n     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) \n     163-171.',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension'], dtype='<U23'),
 'filename': 'breast_cancer.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [205]:
```
train_input , test_input , train_target , test_target = train_test_split(cancer.data, cancer.target , random_state = 4
```
executed in 15ms, finished 11:57:24 2023-10-25

In [206]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(train_input)
```
executed in 13ms, finished 11:57:24 2023-10-25

Out[206]:
```
▾ StandardScaler
StandardScaler()
```

In [207]:
```
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
```
executed in 6ms, finished 11:57:24 2023-10-25

In [208]:
```
decisions = lr.decision_function(train_scaled[:,:5])
decisions
```
executed in 17ms, finished 11:57:24 2023-10-25

Out[208]:
```
array([[ 0.07042131,  5.29486531, -8.44359052, ...,  6.59081425,
         1.39492013, -2.67012852],
       [-3.8751889 , -4.37157959, 11.31746922, ...,  2.58559666,
        -9.40217833,  4.16025488],
       [-1.1130764 ,  2.08609317, -0.4867963 , ...,  0.13192612,
         1.92713095, -4.4936687 ],
       ...,
       [ 2.45467921,  5.18689785, -7.73469202, ...,  1.55121768,
         2.43239063, -5.60249733],
       [-2.31393068, -2.85766968,  7.74188303, ...,  3.46854077,
        -9.03706072,  3.65339436],
       [-4.079075  , -0.09600672,  7.38259626, ..., -3.22697538,
         1.59404047, -4.16571357]])
```

In [209]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(train_scaled , train_target)
```
executed in 22ms, finished 11:57:24 2023-10-25

Out[209]:
```
▾ LogisticRegression
LogisticRegression()
```

In [210]:
```
lr.score(train_scaled , train_target)
```
executed in 7ms, finished 11:57:24 2023-10-25

Out[210]: 0.9859154929577465

In [211]:
```
lr.score(test_scaled , test_target)
```
executed in 15ms, finished 11:57:24 2023-10-25

Out[211]: 0.9790209790209791

In [212]:
```
lr.predict(test_scaled[:5])
```
executed in 14ms, finished 11:57:24 2023-10-25

Out[212]: array([1, 0, 0, 1, 1])

In [213]:
```
test_scaled.shape
```
executed in 8ms, finished 11:57:24 2023-10-25

Out[213]:
```
(143, 30)
```

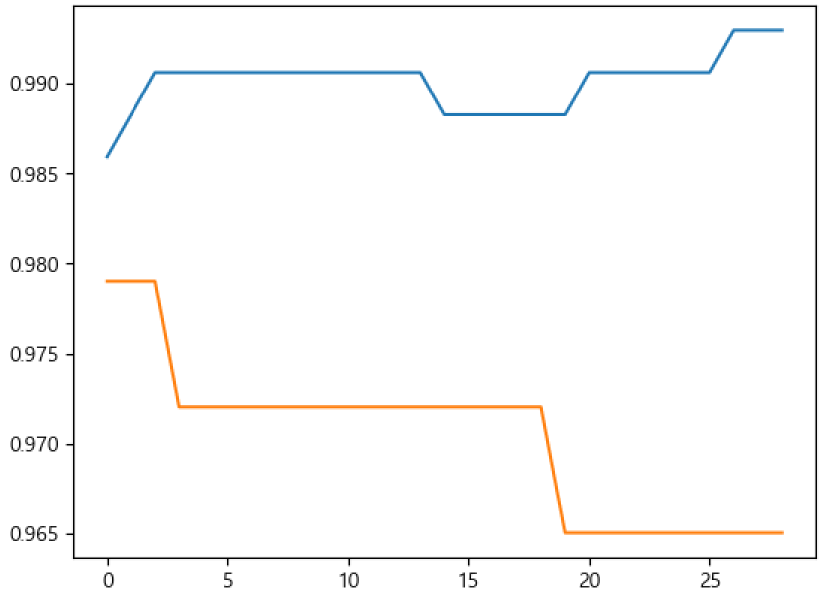
In [214]:
```
proba = lr.predict_proba(test_scaled[:5])
np.round(proba , 3)
```
executed in 13ms, finished 11:57:24 2023-10-25

Out[214]:
```
array([[0.124, 0.876],
       [1.   , 0.   ],
       [0.997, 0.003],
       [0.001, 0.999],
       [0.   , 1.   ]])
```

In [215]:
```
train_score = []
test_score = []
for i in range(1,30):
    lr = LogisticRegression(C = i)
    lr.fit(train_scaled , train_target)
    train_score.append(lr.score(train_scaled , train_target))
    test_score.append(lr.score(test_scaled , test_target))
```
executed in 371ms, finished 11:57:24 2023-10-25

In [216]:
```
plt.plot(train_score)
plt.plot(test_score)
```
executed in 263ms, finished 11:57:24 2023-10-25

Out[216]:
```
[<matplotlib.lines.Line2D at 0x2bc84bef190>]
```



In [217]:
```
pd.DataFrame(cancer.data , columns = cancer.feature_names)
```
executed in 35ms, finished 11:57:24 2023-10-25

Out[217]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	p
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	

569 rows × 30 columns

In [218]:
```
cancer.feature_names[:10]
```
executed in 19ms, finished 11:57:24 2023-10-25

Out[218]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension'],
      dtype='<U23')
```

In [219]:
```
cancer.data[:,:10]
```
executed in 13ms, finished 11:57:24 2023-10-25

Out[219]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 1.471e-01, 2.419e-01,
        7.871e-02],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 7.017e-02, 1.812e-01,
        5.667e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 1.279e-01, 2.069e-01,
        5.999e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 5.302e-02, 1.590e-01,
        5.648e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 1.520e-01, 2.397e-01,
        7.016e-02],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 1.587e-01,
        5.884e-02]])
```

In [220]:
```
train_input , test_input , train_target , test_target = train_test_split(cancer.data[:,:10], cancer.target , random_st
```
executed in 10ms, finished 11:57:24 2023-10-25

In [221]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(train_input)
```
executed in 9ms, finished 11:57:24 2023-10-25

Out[221]:
```
▾ StandardScaler
StandardScaler()
```

In [222]:
```
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
```
executed in 16ms, finished 11:57:24 2023-10-25

In [223]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C = 6)
lr.fit(train_scaled , train_target)
```
executed in 27ms, finished 11:57:25 2023-10-25

Out[223]:
```
▾ LogisticRegression
LogisticRegression(C=6)
```

In [224]:
```
print(lr.score(train_scaled , train_target))
print(lr.score(test_scaled , test_target))
```
executed in 14ms, finished 11:57:25 2023-10-25

```
0.9436619718309859
0.9440559440559441
```

In [225]:
```
result = lr.predict(test_scaled)
from sklearn.metrics import classification_report
print(classification_report(result , test_target))
```
executed in 19ms, finished 11:57:25 2023-10-25

```
              precision    recall  f1-score   support

           0       0.94      0.91      0.93        56
           1       0.94      0.97      0.95        87

    accuracy                           0.94       143
   macro avg       0.94      0.94      0.94       143
weighted avg       0.94      0.94      0.94       143
```

In [ ]:

In [226]:
```
from sklearn.neighbors import KNeighborsClassifier
kn = KNeighborsClassifier()
kn.fit(train_scaled , train_target)
```
executed in 18ms, finished 11:57:25 2023-10-25

Out[226]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

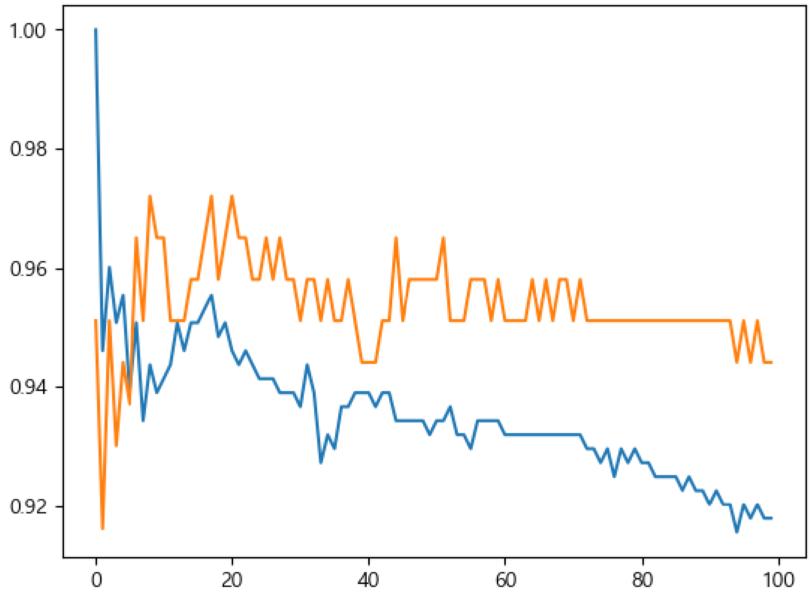
In [227]:
```
len(train_scaled)
```
executed in 16ms, finished 11:57:25 2023-10-25

Out[227]: 426

In [228]:
```
train_score = []
test_score = []
for i in range(1,101):
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(train_scaled , train_target)
    train_score.append(knn.score(train_scaled , train_target))
    test_score.append(knn.score(test_scaled , test_target))
```
executed in 1.80s, finished 11:57:26 2023-10-25

In [229]:
```
plt.plot(train_score)
plt.plot(test_score)
```
executed in 127ms, finished 11:57:27 2023-10-25

Out[229]: [<matplotlib.lines.Line2D at 0x2bc830126a0>]



In [230]:
```
for i in range(len(train_score)):
    if train_score[i]<test_score[i]:
        print(i)
        break
```
executed in 4ms, finished 11:57:27 2023-10-25

6

In [231]:
```
from sklearn.neighbors import KNeighborsClassifier
kn = KNeighborsClassifier(n_neighbors = 5)
kn.fit(train_scaled , train_target)
```
executed in 16ms, finished 11:57:27 2023-10-25

Out[231]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [232]:
```
print(kn.score(train_scaled , train_target))
print(kn.score(test_scaled , test_target))
```
executed in 26ms, finished 11:57:27 2023-10-25

```
0.9553990610328639
0.9440559440559441
```

In [233]:
```
from sklearn.datasets import load_iris
iris = load_iris()
```
executed in 14ms, finished 11:57:27 2023-10-25

In [234]:
```
iris
```
executed in 20ms, finished 11:57:27 2023-10-25

Out[234]:
```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
```

In [235]:
```
train_input , test_input , train_target , test_target = train_test_split(iris.data, iris.target , random_state = 42)
```
executed in 11ms, finished 11:57:27 2023-10-25

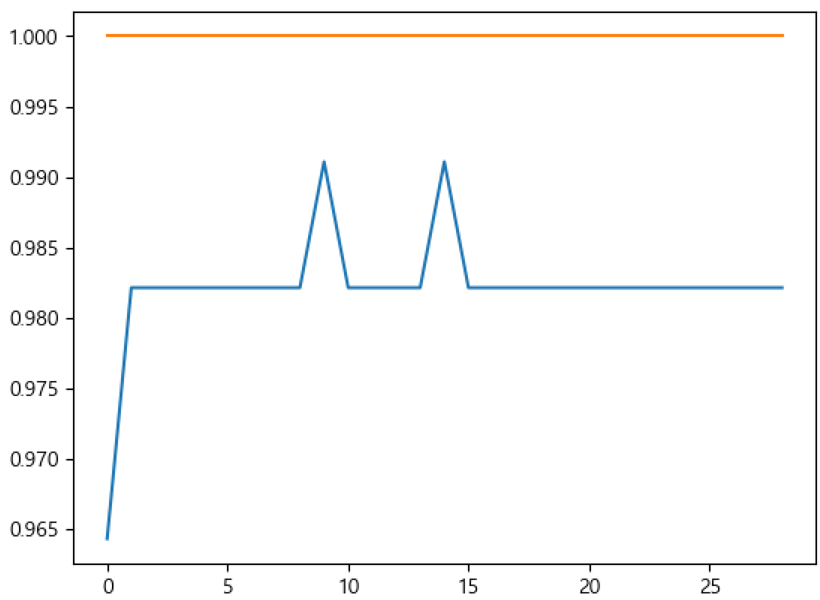
In [236]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(train_input , train_target)
print(lr.score(train_input , train_target))
print(lr.score(test_input , test_target))
```
executed in 20ms, finished 11:57:27 2023-10-25

```
0.9642857142857143
1.0
```

In [237]:
```
train_score = []
test_score = []
for i in range(1,30):
    lr = LogisticRegression(C = i)
    lr.fit(train_input , train_target)
    train_score.append(lr.score(train_input , train_target))
    test_score.append(lr.score(test_input , test_target))
```
executed in 344ms, finished 11:57:27 2023-10-25

In [238]:
```
plt.plot(train_score)
plt.plot(test_score)
```
executed in 129ms, finished 11:57:27 2023-10-25

Out[238]: [<matplotlib.lines.Line2D at 0x2bc82b475b0>]



In [239]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C = 10)
lr.fit(train_input , train_target)
print(lr.score(train_input , train_target))
print(lr.score(test_input , test_target))
```
executed in 20ms, finished 11:57:27 2023-10-25

```
0.9910714285714286
1.0
```

In [240]:
```
np.round(lr.predict_proba(train_input) , 3)
```
executed in 15ms, finished 11:57:27 2023-10-25

Out[240]:
```
array([[0.997, 0.003, 0.   ],
       [0.998, 0.002, 0.   ],
       [0.   , 0.036, 0.964],
       [0.001, 0.961, 0.038],
       [0.   , 0.962, 0.038],
       [0.998, 0.002, 0.   ],
       [0.994, 0.006, 0.   ],
       [0.004, 0.995, 0.001],
       [0.   , 0.496, 0.504],
       [0.   , 0.054, 0.946],
       [0.   , 0.992, 0.007],
       [0.   , 0.001, 0.999],
       [0.001, 0.991, 0.008],
       [0.   , 0.   , 1.   ],
       [0.001, 0.908, 0.092],
       [0.996, 0.004, 0.   ],
       [0.   , 0.   , 1.   ],
       [0.001, 0.998, 0.001],
       [0.994, 0.006, 0.   ],
```

In [241]:
```
decisions = lr.decision_function(train_input)
decisions
```
executed in 18ms, finished 11:57:27 2023-10-25

Out[241]:
```
array([[ 12.30534073,   6.49706033, -18.80240106],
       [ 13.01663648,   6.89327622, -19.9099127 ],
       [ -7.89444059,   2.30269426,   5.59174633],
       [ -3.57225913,   3.40023233,   0.1720268 ],
       [ -5.07740319,   4.15170857,   0.92569462],
       [ 12.82031872,   6.55558949, -19.37590821],
       [ 11.95354448,   6.84701483, -18.80055931],
       [ -1.09875917,   4.33362381,  -3.23486465],
       [ -7.16806645,   3.57625482,   3.59181163],
       [ -8.75562338,   2.94667037,   5.80895301],
       [ -3.84484923,   4.36656271,  -0.52171348],
       [-11.71768411,   2.13449051,   9.58319361],
       [ -2.72431581,   3.77819383,  -1.05387802],
       [-14.41227508,   3.03730925,  11.37496583],
       [ -4.06001417,   3.17599955,   0.88401463],
       [ 12.06535383,   6.58233002, -18.64768385],
       [-15.0856491 ,   3.26829161,  11.81735749],
       [ -2.22926363,   4.48499912,  -2.2557355 ],
       [ 11.82536693,   6.66759971, -18.49296664],
```

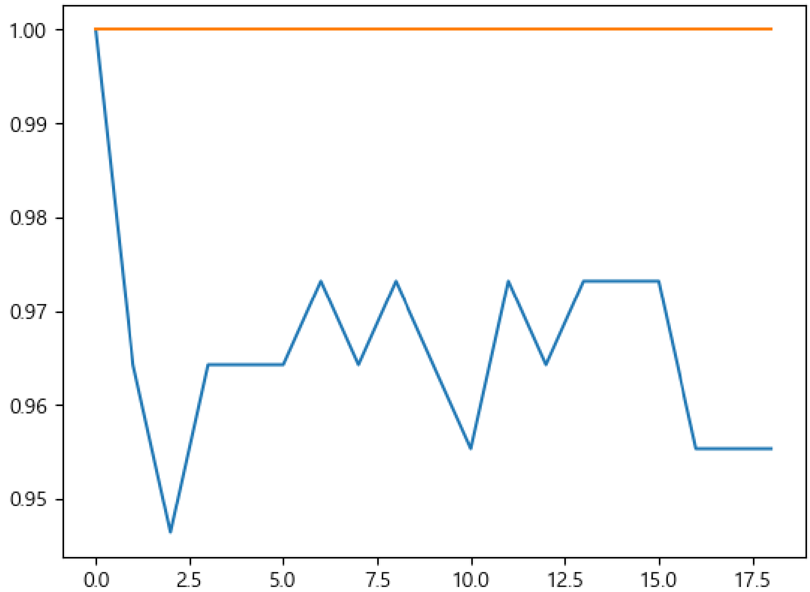
In [242]:
```
from sklearn.neighbors import KNeighborsClassifier
kn = KNeighborsClassifier(n_neighbors = 3)
kn.fit(train_input , train_target)
print(kn.score(train_input , train_target))
print(kn.score(test_input , test_target))
```
executed in 13ms, finished 11:57:27 2023-10-25

```
0.9464285714285714
1.0
```

In [243]:
```
train_score = []
test_score = []
for i in range(1,20):
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(train_input , train_target)
    train_score.append(knn.score(train_input , train_target))
    test_score.append(knn.score(test_input , test_target))
```
executed in 112ms, finished 11:57:27 2023-10-25

In [244]:
```
plt.plot(train_score)
plt.plot(test_score)
```
executed in 132ms, finished 11:57:27 2023-10-25

Out[244]: [<matplotlib.lines.Line2D at 0x2bc83b97550>]



In [245]:
```
from sklearn.neighbors import KNeighborsClassifier
kn = KNeighborsClassifier(n_neighbors = 7)
kn.fit(train_input , train_target)
print(kn.score(train_input , train_target))
print(kn.score(test_input , test_target))
```
executed in 18ms, finished 11:57:27 2023-10-25

```
0.9732142857142857
1.0
```

In [246]:
```
result = kn.predict(test_input)
from sklearn.metrics import classification_report
print(classification_report(result , test_target))
```
executed in 15ms, finished 11:57:27 2023-10-25

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38
```

In [247]:
```
iris.target_names[result]
```
executed in 15ms, finished 11:57:27 2023-10-25

Out[247]:
```
array(['versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor',
       'setosa', 'versicolor', 'virginica', 'versicolor', 'versicolor',
       'virginica', 'setosa', 'setosa', 'setosa', 'setosa', 'versicolor',
       'virginica', 'versicolor', 'versicolor', 'virginica', 'setosa',
       'virginica', 'setosa', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'setosa', 'setosa', 'setosa', 'setosa',
       'versicolor', 'setosa', 'setosa', 'virginica', 'versicolor',
       'setosa'], dtype='<U10')
```

In [ ]: