K-Nearest Neighbors

In [1]:
```
import os
import sys

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

from mlxtend.plotting import plot_decision_regions
from sklearn.datasets import load_breast_cancer, load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier as KNC, kneighbors_graph
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

%matplotlib inline
```

In [2]:
```
f1_values_set1 = np.random.randint(low=12, high=17,size=50)
f2_values_set1 = np.random.randint(low=20, high=25,size=50)

f1_values_set2 = np.random.randint(low=2, high=7,size=50)
f2_values_set2 = np.random.randint(low=10, high=15,size=50)
```

In [3]:
```
dummy_dataset1 = pd.DataFrame({'f1':f1_values_set1,
                               'f2':f2_values_set1})

dummy_dataset2 = pd.DataFrame({'f1':f1_values_set2,
                               'f2':f2_values_set2})

dummy_data = pd.concat([dummy_dataset1,dummy_dataset2],axis=0).reset_index(drop=True
```

In [4]:
```
dummy_data['label'] = dummy_data['f1'].apply(lambda val: 1 if val > 7 else 0)
```

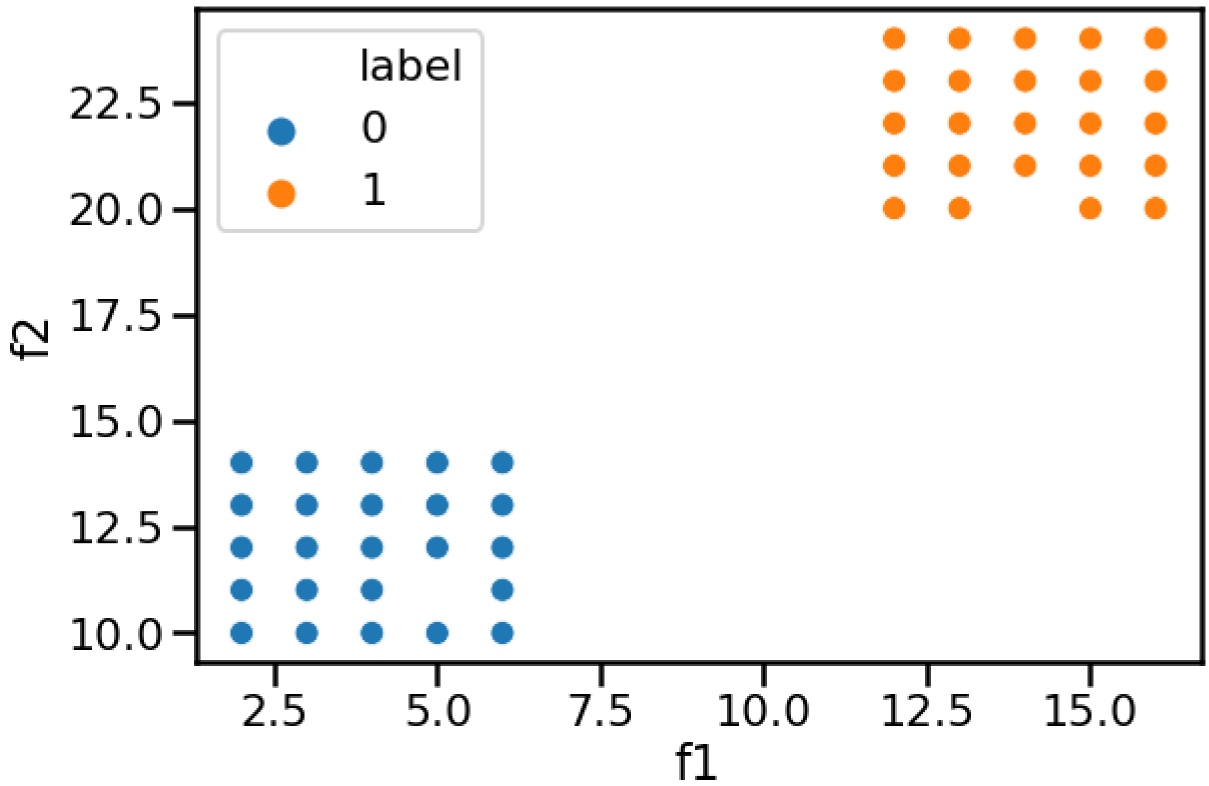
In [5]:
```
dummy_data.head()
```

Out[5]:

	f1	f2	label
0	12	22	1
1	13	23	1
2	12	20	1
3	12	21	1
4	16	24	1

In [6]:
```
sns.set_context(context='poster')
plt.figure(figsize=(9,6))
sns.scatterplot(x='f1',y='f2',hue='label',data=dummy_data)
```

Out[6]: <AxesSubplot:xlabel='f1', ylabel='f2'>



In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(dummy_data.iloc[:,0:-1],dummy_da
```

In [8]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[8]: ((60, 2), (40, 2), (60,), (40,))

K=1

In [9]:
```
knc = KNC(n_neighbors=1)
```

In [10]:
```
knc_model = knc.fit(X_train,y_train)
```

In [11]:
```
y_predict = knc.predict(X_test)
```

In [12]:
```
accuracy_score(y_test,y_predict), precision_score(y_test,y_predict), recall_score(y_
```

Out[12]: (1.0, 1.0, 1.0)

K=1, Weighted KNN with distance metric as Canberra

In [13]:
```
knc2 = KNC(n_neighbors=1, weights='distance',algorithm='ball_tree',metric=scipy.spat
```

In [14]:
```
knc_model2 = knc2.fit(X_train,y_train)
```

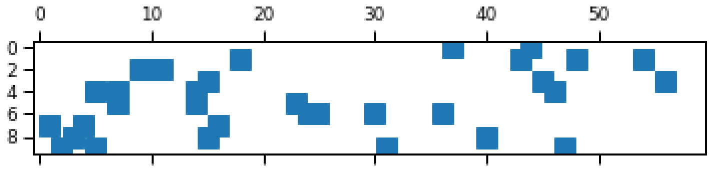
In [15]:
```
y_predict2 = knc_model2.predict(X_test)
```

In [16]:
```
accuracy_score(y_test,y_predict2), precision_score(y_test,y_predict2), recall_score(
```

Out[16]: (1.0, 1.0, 1.0)

In [17]:
```
# pd.DataFrame(knc2.kneighbors_graph(X=X_train,n_neighbors=1,mode='connectivity').to
sns.set_context('paper')
```

```
plt.spy(knc2.kneighbors_graph(X=X_train.iloc[0:10,],n_neighbors=5,mode='distance'))
plt.show()
```



VIF Calculation

In [102…

```
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
```

In [103…

```
df.head()
```

Out[103…

	carat	depth	price
0	0.23	61.5	326
1	0.21	59.8	326
2	0.23	56.9	327
3	0.29	62.4	334
4	0.31	63.3	335

In [104…

```
df.values
```

Out[104…

```
array([[2.300e-01, 6.150e+01, 3.260e+02],
       [2.100e-01, 5.980e+01, 3.260e+02],
       [2.300e-01, 5.690e+01, 3.270e+02],
       ...,
       [7.000e-01, 6.020e+01, 2.822e+03],
       [7.000e-01, 6.070e+01, 2.822e+03],
       [9.000e-01, 6.400e+01, 2.822e+03]])
```

In [350…

```
vif_df = pd.DataFrame()
vif_df['Values'] = [vif(df.values,i) for i in range(len(df.columns))]
vif_df['Features'] = df.columns
```

In [351…

```
vif_df
```

Out[351…

	Values	Features
0	60.169063	carat
1	11.015691	depth
2	37.304660	price

In [110…

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
```

In [120…

```
brst_can = cancer_df.copy(deep=True)
brst_can.head()
```

Out[120…

mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	di

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	di
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 31 columns

In [121…
```
brst_can = pd.DataFrame(ss.fit_transform(brst_can.iloc[:,0:-1]),columns=cancer_df.il
```

In [124…
```
brst_can.head()
```

Out[124…

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	n symm
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.532475	2.21
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.023846	0.548144	0.00
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.363478	2.037231	0.93
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.451707	2.86
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.428493	-0.00

5 rows × 30 columns

In [125…
```
cancer_vif = pd.DataFrame()
cancer_vif['Values'] = [vif(brst_can.values,i) for i in range(len(brst_can.columns))
cancer_vif['Features'] = brst_can.columns
```

In [126…
```
cancer_vif
```

Out[126…

	Values	Features
0	3806.115296	mean radius
1	11.884048	mean texture
2	3786.400419	mean perimeter
3	347.878657	mean area
4	8.194282	mean smoothness
5	50.505168	mean compactness
6	70.767720	mean concavity
7	60.041733	mean concave points
8	4.220656	mean symmetry
9	15.756977	mean fractal dimension

	Values	Features
10	75.462027	radius error
11	4.205423	texture error
12	70.359695	perimeter error
13	41.163091	area error
14	4.027923	smoothness error
15	15.366324	compactness error
16	15.694833	concavity error
17	11.520796	concave points error
18	5.175426	symmetry error
19	9.717987	fractal dimension error
20	799.105946	worst radius
21	18.569966	worst texture
22	405.023336	worst perimeter
23	337.221924	worst area
24	10.923061	worst smoothness
25	36.982755	worst compactness
26	31.970723	worst concavity
27	36.763714	worst concave points
28	9.520570	worst symmetry
29	18.861533	worst fractal dimension

KNN on Breast Cancer Dataset

In [18]:
```
cancer = load_breast_cancer()
```

In [19]:
```
cancer_df = pd.concat([pd.DataFrame(cancer.data,columns=cancer.feature_names),pd.Dat
cancer_df
```

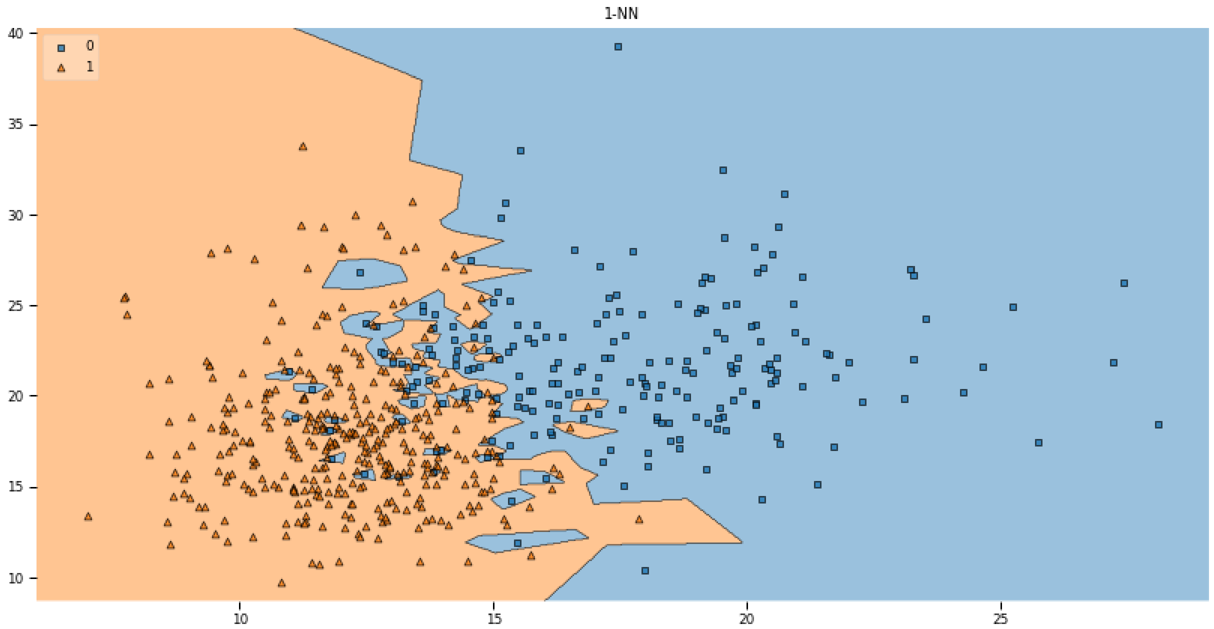
Out[19]:

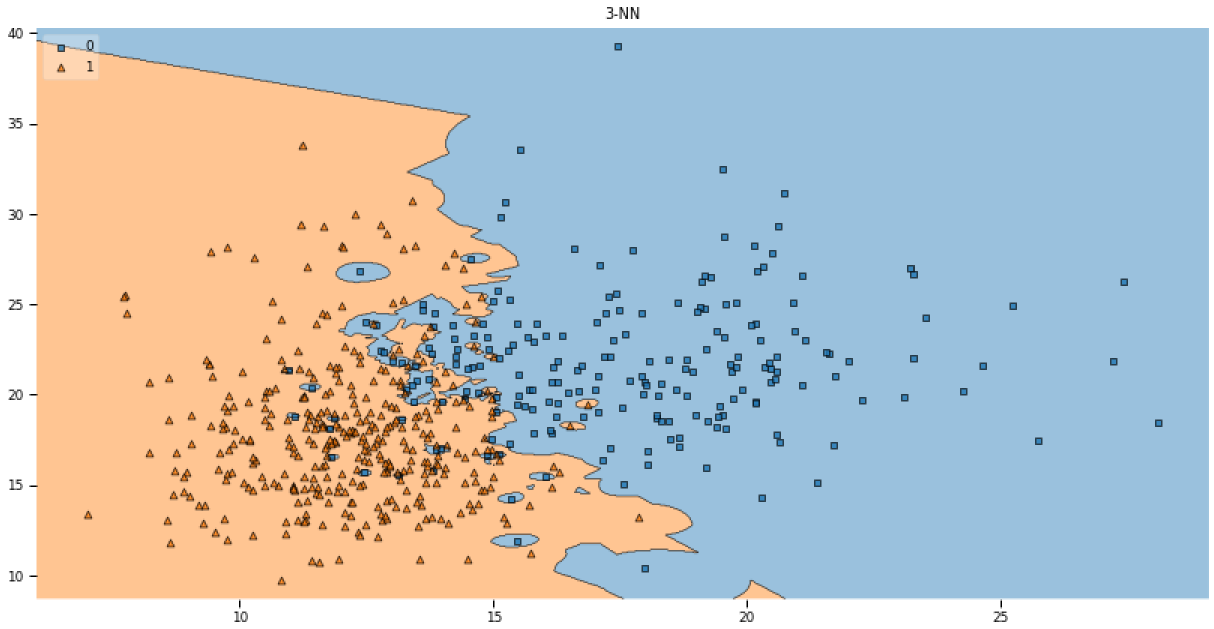
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809
...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752

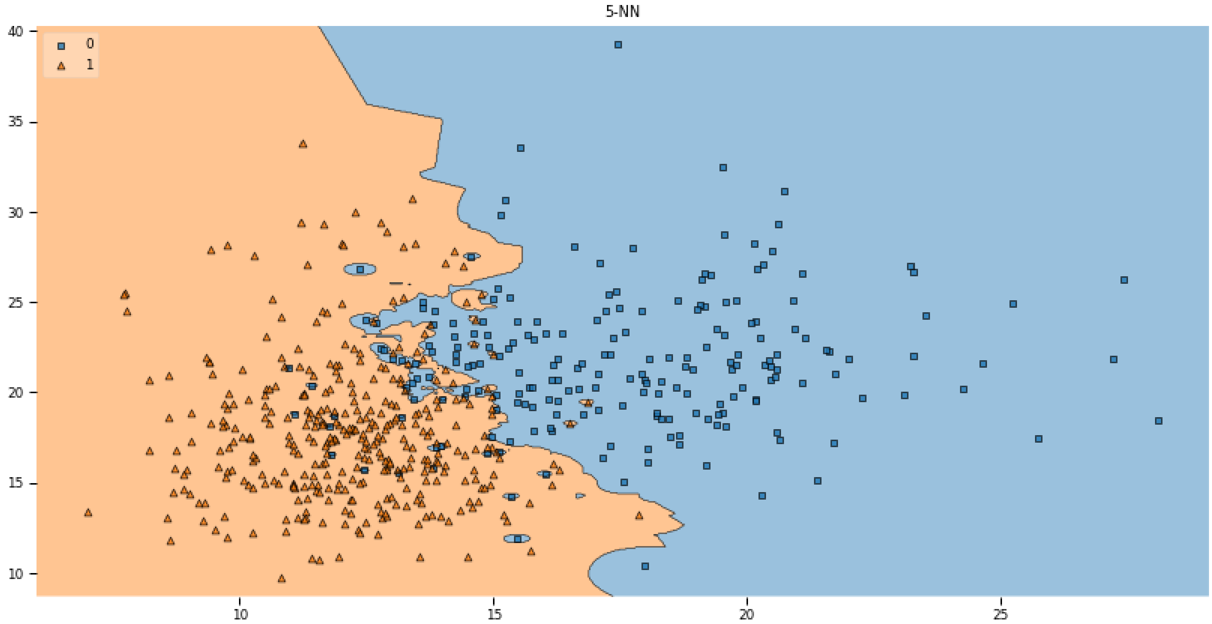
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587

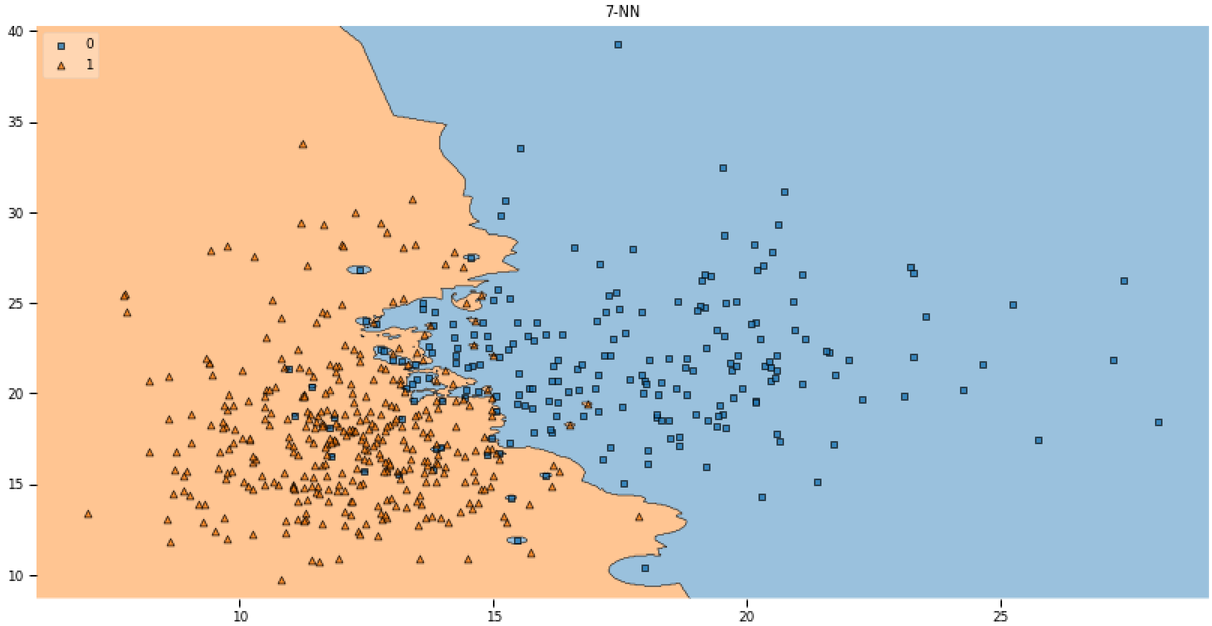
569 rows × 31 columns

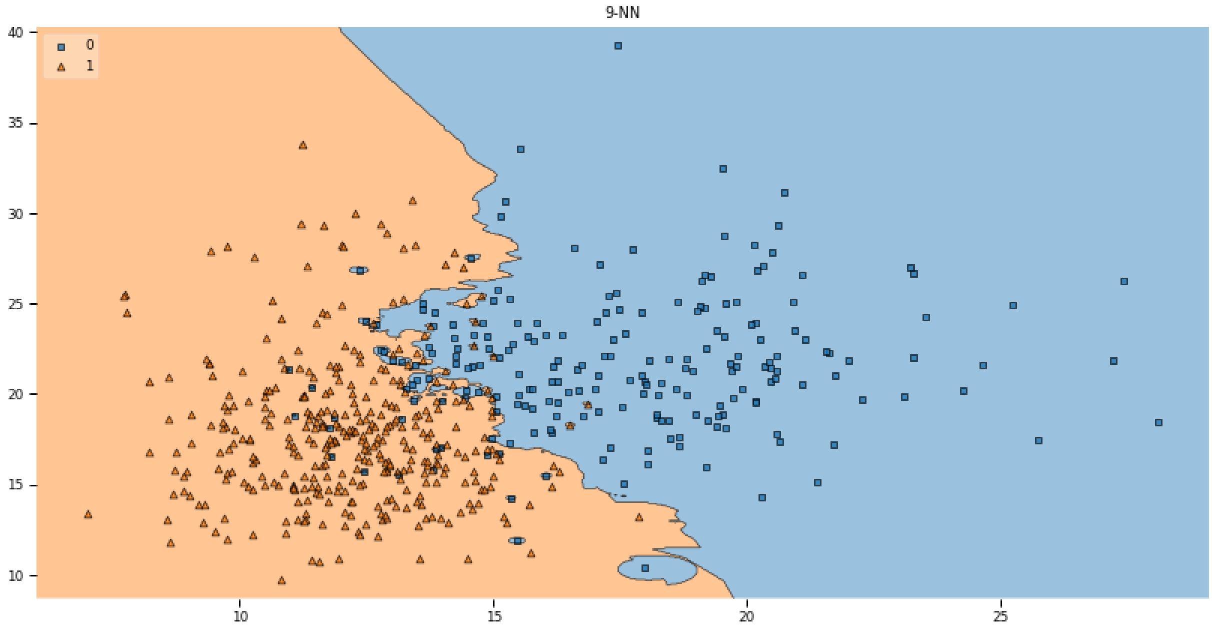
In [49]:
```
for k in range(1,21)[::2]:
    knn_dec_reg = KNC(n_neighbors=k,weights='distance',algorithm='kd_tree',leaf_size
    knn_dec_reg.fit(cancer_df.iloc[:,0:2],y=cancer_df['Label'])
    with plt.style.context('seaborn-bright'):
        plt.figure(figsize=(14,7))
        plot_decision_regions(X=cancer_df.iloc[:,0:2].values,y=cancer_df['Label'].va
        plt.title("{}-NN".format(k))
        plt.show()
```

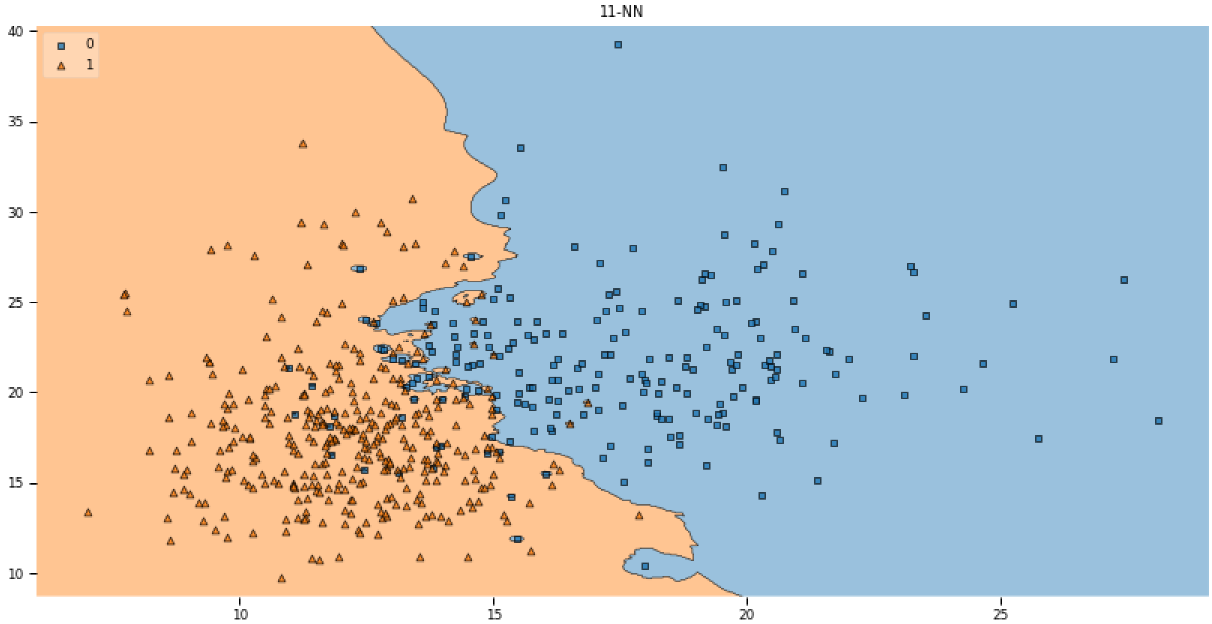


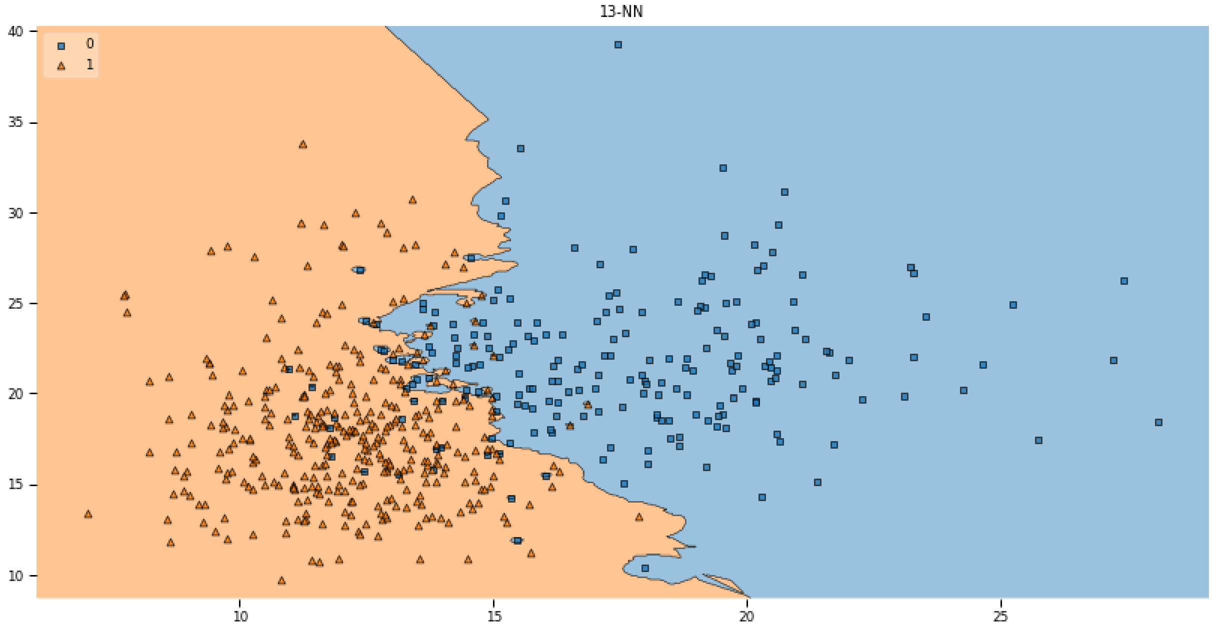


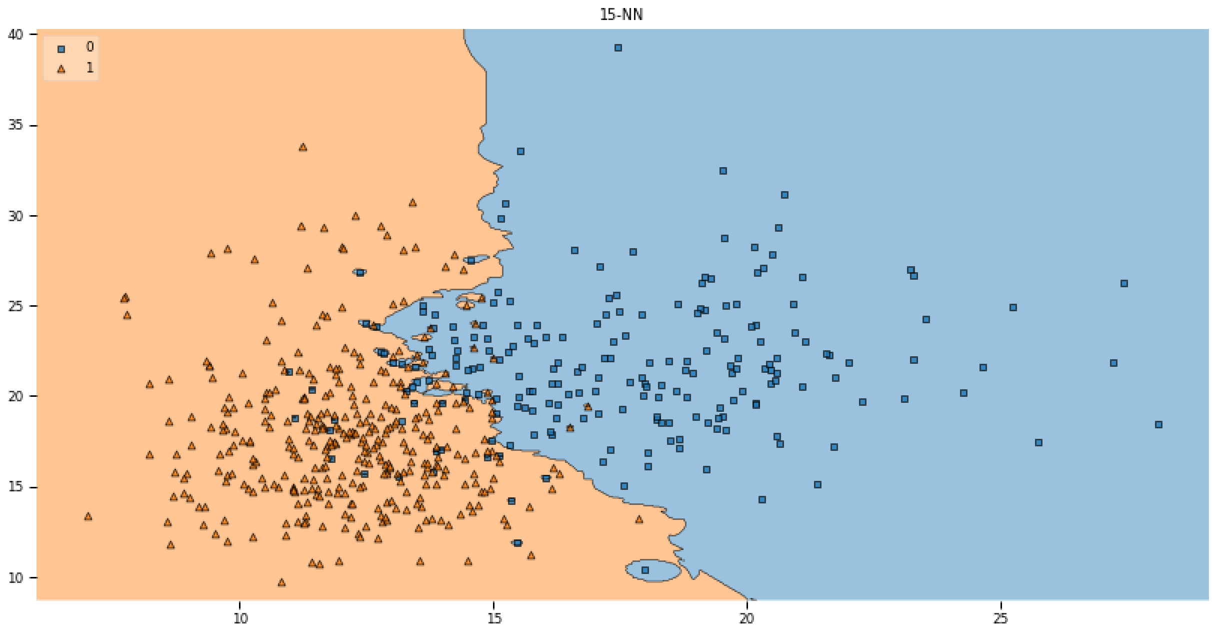

5-NN
0
1
40
35
30
25
20
15
10
10
15
20
25

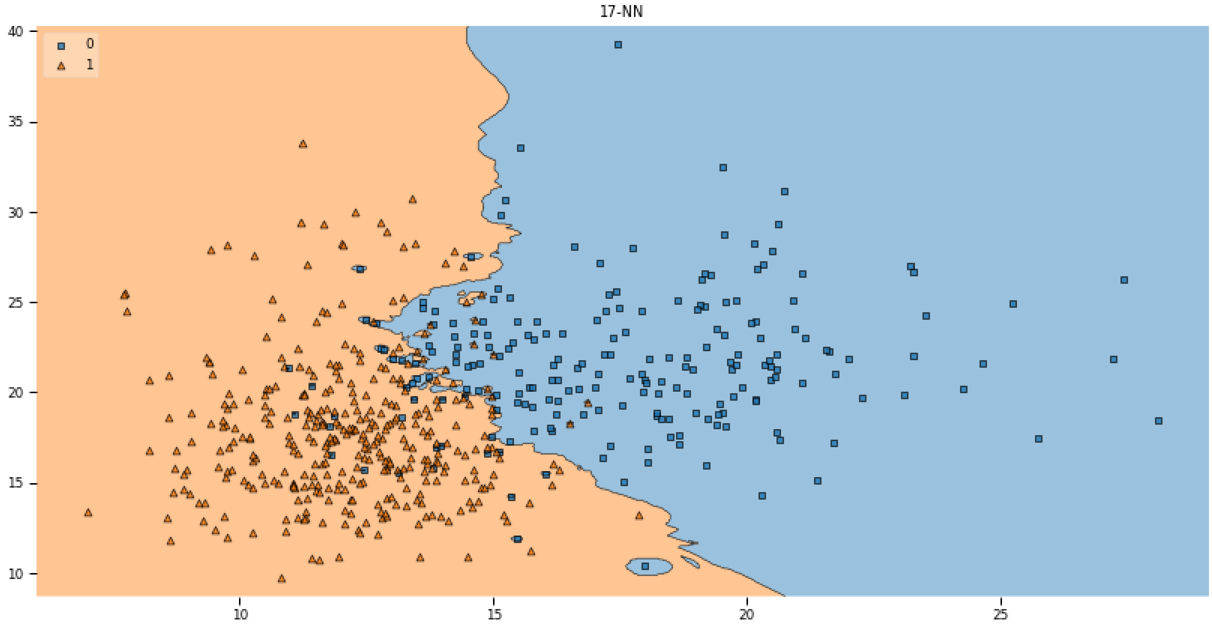

7-NN
0
1
40
35
30
25
20
15
10
10
15
20
25

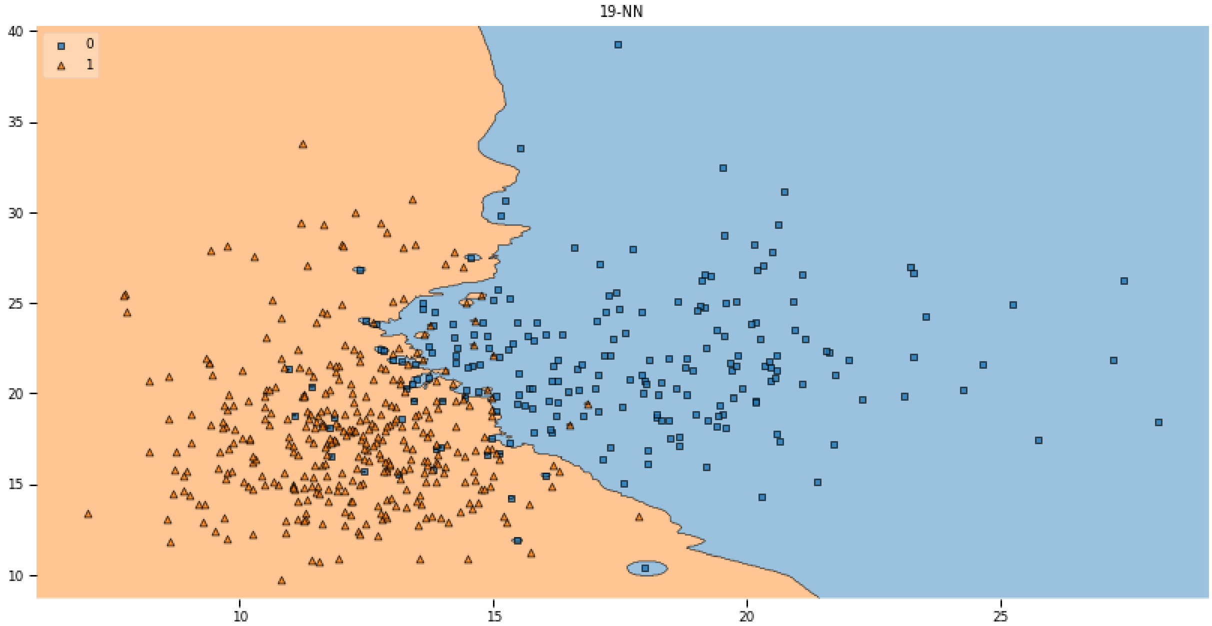

9-NN
0
1
40
35
30
25
20
15
10
10
15
20
25


11-NN
0
1
40
35
30
25
20
15
10
10
15
20
25


13-NN
0
1
40
35
30
25
20
15
10
10
15
20
25


15-NN
0
1
40
35
30
25
20
15
10
10
15
20
25





Only TRAIN and TEST

In [36]:
```
X_train, X_test, y_train, y_test = train_test_split(cancer_df.iloc[:,0:-1],cancer_df
```

In [21]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[21]: ((341, 30), (228, 30), (341,), (228,))

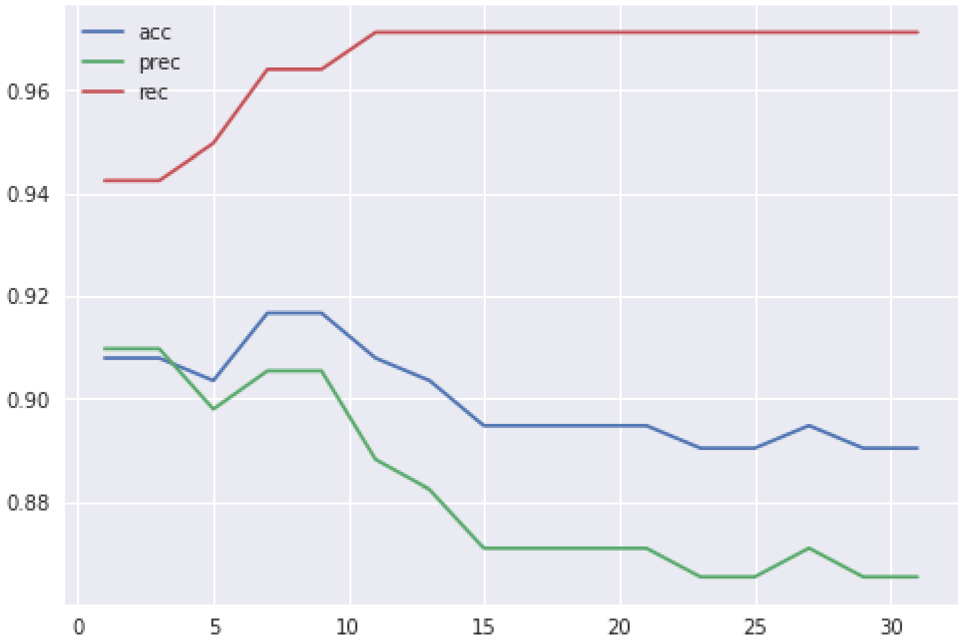
In [22]:
```
def filt(val):
    if val%2 != 0:
        return val
```

In [23]:
```
neighbors = list(filter(filt, [val for val in range(0,32)]))
neighbors
```

Out[23]: [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31]

In [24]:
```
acc_scr = []
rec_scr = []
prec_scr = []
```

```
for neighbor in neighbors:
    knn_model = KNC(n_neighbors=neighbor)
    knn_model.fit(X_train, y_train)
    knn_y_predict = knn_model.predict(X_test)
    acc_scr.append(accuracy_score(y_test,knn_y_predict))
    rec_scr.append(recall_score(y_test,knn_y_predict))
    prec_scr.append(precision_score(y_test,knn_y_predict))

with plt.style.context('seaborn'):
    sns.lineplot(x=neighbors,y=acc_scr,label='acc')
    sns.lineplot(x=neighbors,y=prec_scr,label='prec')
    sns.lineplot(x=neighbors,y=rec_scr,label='rec')
```



TRAIN, CV and TEST

In [25]:
```
X1, X_test, y1, y_test = train_test_split(cancer_df.iloc[:,0:-1],cancer_df['Label'],
```

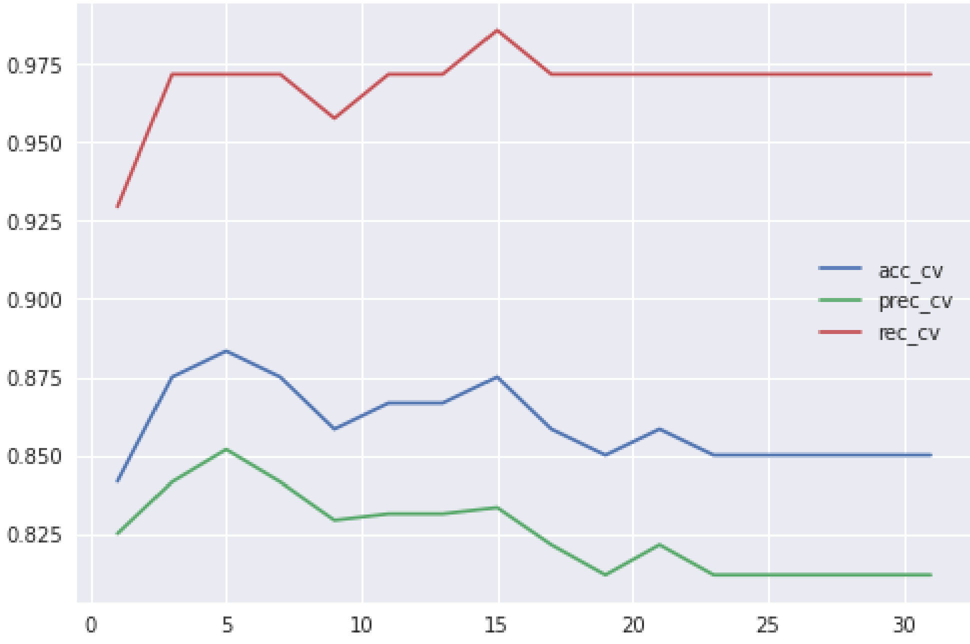
In [26]:
```
X_train, X_cv, y_train, y_cv = train_test_split(X1,y1,test_size=0.30,random_state=42
```

In [27]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape, X_cv.shape, y_cv.shape
```

Out[27]:
```
((278, 30), (171, 30), (278,), (171,), (120, 30), (120,))
```

CV performance metrics

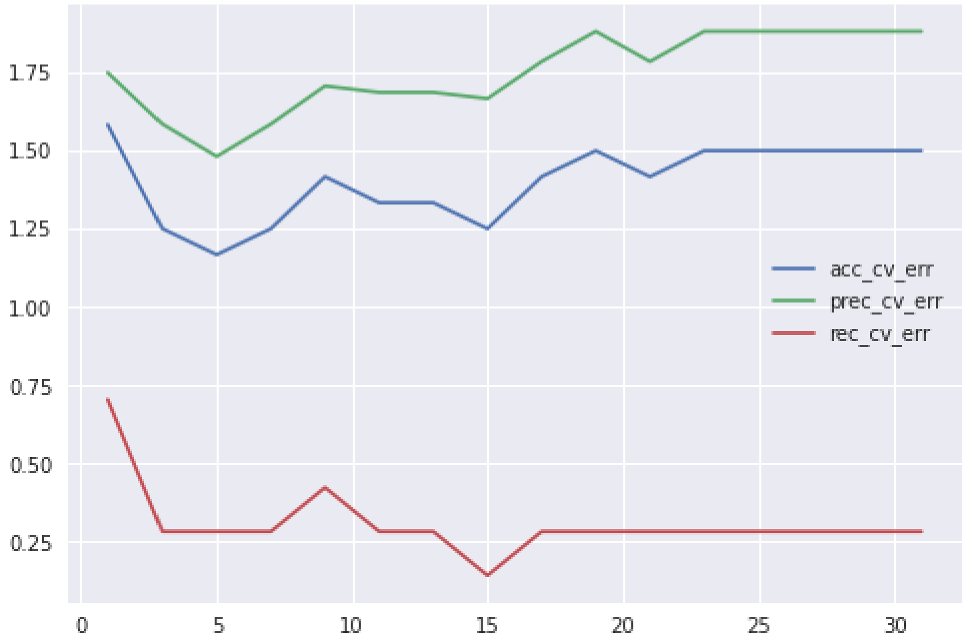
In [28]:
```
acc_scr_cv = []
rec_scr_cv = []
prec_scr_cv = []

for neighbor in neighbors:
    knn_model = KNC(n_neighbors=neighbor,weights='distance',algorithm='kd_tree',leaf
    knn_model.fit(X_train, y_train)
    y_cv_pred = knn_model.predict(X_cv)
    acc_scr_cv.append(accuracy_score(y_cv,y_cv_pred))
    rec_scr_cv.append(recall_score(y_cv,y_cv_pred))
    prec_scr_cv.append(precision_score(y_cv,y_cv_pred))

with plt.style.context('seaborn'):
    sns.lineplot(x=neighbors,y=acc_scr_cv,label='acc_cv')
    sns.lineplot(x=neighbors,y=prec_scr_cv,label='prec_cv')
    sns.lineplot(x=neighbors,y=rec_scr_cv,label='rec_cv')
```



CV Error

In [29]:
```
acc_scr_cv_err = [(1 - val)*10 for val in acc_scr_cv]
prec_scr_cv_err = [(1 - val)*10 for val in prec_scr_cv]
rec_scr_cv_err = [(1 - val)*10 for val in rec_scr_cv]
```

In [30]:
```
with plt.style.context('seaborn'):
    sns.lineplot(x=neighbors,y=acc_scr_cv_err,label='acc_cv_err')
    sns.lineplot(x=neighbors,y=prec_scr_cv_err,label='prec_cv_err')
    sns.lineplot(x=neighbors,y=rec_scr_cv_err,label='rec_cv_err')
```



TEST set performance metrics

In [31]:
```
knn_cancer_model = KNC(n_neighbors=neighbor,weights='distance',algorithm='kd_tree',l
knn_cancer_model.fit(X_train,y_train)
knn_cancer_ypred = knn_cancer_model.predict(X_test)
```

In [32]:
```
accuracy_score(y_test,knn_cancer_ypred), precision_score(y_test,knn_cancer_ypred), r
```

Out[32]:
```
(0.9415204678362573, 0.9152542372881356, 1.0)
```

Predicting Class Probability

In [33]:
```
print(knn_cancer_model.predict_proba(X_test))
```

```
[[0.10727533 0.89272467]
 [1.         0.        ]
 [0.90987962 0.09012038]
 [0.01882241 0.98117759]
 [0.01403691 0.98596309]
 [1.         0.        ]
 [1.         0.        ]
 [0.96872456 0.03127544]
 [0.02335353 0.97664647]
 [0.12278102 0.87721898]
 [0.03116392 0.96883608]
 [0.98430656 0.01569344]
 [0.12262872 0.87737128]
 [0.57084654 0.42915346]
 [0.02223204 0.97776796]
 [1.         0.        ]
 [0.11867577 0.88132423]
 [0.         1.        ]
 [0.         1.        ]
 [1.         0.        ]
 [0.28898917 0.71101083]
 [0.0501302  0.9498698 ]
 [1.         0.        ]
 [0.         1.        ]
 [0.         1.        ]
 [0.0243354  0.9756646 ]
 [0.02509231 0.97490769]
 [0.         1.        ]
 [0.         1.        ]
 [1.         0.        ]
 [0.         1.        ]
 [0.01898134 0.98101866]
 [0.         1.        ]
 [0.19632132 0.80367868]
 [0.         1.        ]
 [0.01658878 0.98341122]
 [0.92674356 0.07325644]
 [0.03880247 0.96119753]
 [1.         0.        ]
 [0.15321879 0.84678121]
 [0.         1.        ]
 [0.80984493 0.19015507]
 [0.         1.        ]
 [0.02596695 0.97403305]
 [0.         1.        ]
 [0.05303708 0.94696292]
 [0.         1.        ]
 [0.         1.        ]
 [0.05255506 0.94744494]
 [0.01350674 0.98649326]
 [1.         0.        ]
 [1.         0.        ]
 [0.04146001 0.95853999]
 [0.05088521 0.94911479]
 [0.         1.        ]
 [0.09863321 0.90136679]
 [0.         1.        ]
 [1.         0.        ]
 [0.20776767 0.79223233]
 [0.01592945 0.98407055]
 [0.02672009 0.97327991]
 [1.         0.        ]
 [1.         0.        ]
 [0.0593718  0.9406282 ]
 [0.01424098 0.98575902]
```

```
 [0.03313167 0.96686833]
 [0.9668277  0.0331723 ]
 [1.         0.        ]
 [0.         1.        ]
 [0.04307508 0.95692492]
 [0.25743538 0.74256462]
 [0.98509099 0.01490901]
 [0.05410487 0.94589513]
 [0.56849597 0.43150403]
 [0.         1.        ]
 [0.05490436 0.94509564]
 [0.11694131 0.88305869]
 [0.21835167 0.78164833]
 [0.0187632  0.9812368 ]
 [0.04211452 0.95788548]
 [0.95263292 0.04736708]
 [0.05722965 0.94277035]
 [0.16149533 0.83850467]
 [1.         0.        ]
 [1.         0.        ]
 [0.4904376  0.5095624 ]
 [0.23756747 0.76243253]
 [1.         0.        ]
 [0.         1.        ]
 [0.01563645 0.98436355]
 [0.05989591 0.94010409]
 [0.19446449 0.80553551]
 [0.25130616 0.74869384]
 [0.01627946 0.98372054]
 [0.         1.        ]
 [0.         1.        ]
 [1.         0.        ]
 [0.87381559 0.12618441]
 [0.01393219 0.98606781]
 [1.         0.        ]
 [0.96537113 0.03462887]
 [0.         1.        ]
 [1.         0.        ]
 [1.         0.        ]
 [0.13782438 0.86217562]
 [0.06321966 0.93678034]
 [0.0183746  0.9816254 ]
 [1.         0.        ]
 [0.11049027 0.88950973]
 [0.16877216 0.83122784]
 [0.9697765  0.0302235 ]
 [0.01915967 0.98084033]
 [0.08137518 0.91862482]
 [1.         0.        ]
 [0.         1.        ]
 [1.         0.        ]
 [0.         1.        ]
 [0.05219805 0.94780195]
 [0.         1.        ]
 [0.57208783 0.42791217]
 [0.09643647 0.90356353]
 [0.02692151 0.97307849]
 [0.04566413 0.95433587]
 [1.         0.        ]
 [0.04912859 0.95087141]
 [1.         0.        ]
 [0.84654748 0.15345252]
 [0.         1.        ]
 [0.05269405 0.94730595]
 [1.         0.        ]
 [1.         0.        ]
 [0.4721521  0.5278479 ]
 [0.19556569 0.80443431]
 [0.         1.        ]
```

```
 [0.48016319 0.51983681]
 [0.7242847  0.2757153 ]
 [0.08481298 0.91518702]
 [0.         1.        ]
 [0.13835626 0.86164374]
 [0.75779242 0.24220758]
 [0.04599738 0.95400262]
 [1.         0.        ]
 [0.         1.        ]
 [0.         1.        ]
 [0.43795616 0.56204384]
 [0.02604189 0.97395811]
 [1.         0.        ]
 [1.         0.        ]
 [0.68907434 0.31092566]
 [0.11228702 0.88771298]
 [0.78476206 0.21523794]
 [0.02011579 0.97988421]
 [0.         1.        ]
 [0.11952399 0.88047601]
 [0.0162412  0.9837588 ]
 [1.         0.        ]
 [1.         0.        ]
 [0.         1.        ]
 [0.14078314 0.85921686]
 [0.         1.        ]
 [0.         1.        ]
 [0.         1.        ]
 [0.0233792  0.9766208 ]
 [0.         1.        ]
 [0.50005405 0.49994595]
 [0.02265209 0.97734791]
 [0.         1.        ]
 [0.18478532 0.81521468]
 [0.         1.        ]
 [0.19417011 0.80582989]
 [0.0505732  0.9494268 ]]
```

Locality Sensitive Hashing

In [34]:
```
import numpy as np

class HashTable:
    def __init__(self, hash_size, inp_dimensions):
        self.hash_size = hash_size
        self.inp_dimensions = inp_dimensions
        self.hash_table = dict()
        self.projections = np.random.randn(self.hash_size, inp_dimensions)

    def generate_hash(self, inp_vector):
        bools = (np.dot(inp_vector, self.projections.T) > 0).astype('int')
        return ''.join(bools.astype('str'))

    def __setitem__(self, inp_vec, label):
        hash_value = self.generate_hash(inp_vec)
        self.hash_table[hash_value] = self.hash_table\
            .get(hash_value, list()) + [label]

    def __getitem__(self, inp_vec):
        hash_value = self.generate_hash(inp_vec)
        return self.hash_table.get(hash_value, [])

hash_table = HashTable(hash_size=4, inp_dimensions=20)
```

In [35]:
```
class LSH:
```

```
    def __init__(self, num_tables, hash_size, inp_dimensions):
        self.num_tables = num_tables
        self.hash_size = hash_size
        self.inp_dimensions = inp_dimensions
        self.hash_tables = list()
        for i in range(self.num_tables):
            self.hash_tables.append(HashTable(self.hash_size, self.inp_dimensions))

    def __setitem__(self, inp_vec, label):
        for table in self.hash_tables:
            table[inp_vec] = label

    def __getitem__(self, inp_vec):
        results = list()
        for table in self.hash_tables:
            results.extend(table[inp_vec])
        return list(set(results))
```

In [ ]:

In [ ]:

In [ ]: