```python
# ИУ5-65Б СОРОКИН АРТЕМ



from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
import random

from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split, GridSearchCV,
RandomizedSearchCV



iris = load_iris(return_X_y=False, as_frame=False)

print(type(iris))

# data.head(10)

data = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['target'])

print(data)

# СОЗДАНИ ПРОПУСКОВ


# ЧИСЛОВЫХ ПРИЗНАКОВ

for i in range(8):
    k = random.randint(1, 150)
    data['petal length (cm)'][k] = np.nan

# КАТЕГОРИАЛЬНЫЙ ПРИЗНАК
for i in range(8):
    k = random.randint(1, 150)
    data['target'][k] = np.nan



with pd.option_context('display.max_rows', None,
'display.max_columns', None):  # more options can be specified also
    print(data)

### ПРОПУСКИ ДАННЫХ
```

```
# Скопируем DataFrame чтобы попробовать несколько вариантов заполнения
пропусков
# В данном случае, если проанализировать пропуски, становится понятно,
что удобнее всего способ замены на значение
# из следующей записи или из предыдущей. Так как записи в наборе
данных более-менее отсортированные



# Попробуем сначала заполнить средним значением столбца
(категориальный признак так заполнять не разумно)


df = data.copy()

df_test = df.fillna(df.mean())['petal length (cm)']

print (df.isnull().sum())

with pd.option_context('display.max_rows', None,
'display.max_columns', None):
    print(df_test)


# Среднее значение 3.733099. Выпадает из списка и может нарушить
обучение модели в строках 27, 104, 122

### Заполнение пропусков значением из предыдущей записи
### Крайне удачный вариант, так как категориальные признаки в данном
случае отсортированы

df_test = df.fillna(method='bfill')
with pd.option_context('display.max_rows', None,
'display.max_columns', None):
    print(df_test)

# Получили практически исходный набор данных
```

Перейдем к обучению модели

```
iris.data.shape

x = iris.data
y = iris.target
```

Делим данные на тестовую выборку и тренировочную

```
#Делим данные на тренировочное и тестовое множество:

X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size =
0.2, random_state = 42)
print(X_train.shape, X_test.shape, Y_test.shape, Y_train.shape)
```

Будем использовать метод ближайших соседей, так как из данных видно, что классы имеют четко различимые границы

```
K_range = range(1, 11)
scores = {}
scores_list = []
for k in K_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, Y_train)
    Y_pred = knn.predict(X_test)
    scores[k] = metrics.accuracy_score(Y_test, Y_pred)
    scores_list.append(metrics.accuracy_score(Y_test, Y_pred))

print (scores)
```

Посмотрим какой подберет гиперпараметр встроенный метод KNeighborsClassifier

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x, y)

classes = {0:'setosa', 1:'versicolor', 2:'virginica'}

X_new = [[3,4,5,2],
        [5,4,2,2,]]

Y_pred = knn.predict(X_new)

print (classes [Y_pred[0]])
print (classes [Y_pred[1]])

#Создаем классификатор:

best_model = KNeighborsClassifier(
   n_neighbors=10,
   weights='distance',
   algorithm='auto',
   leaf_size=30,
   metric='euclidean',
   metric_params=None,
   n_jobs=4
)

best_model.fit(X_train, Y_train)
predicted = best_model.predict(X_test)

print('Evaluation:\n', metrics.classification_report(Y_test,
predicted))
```

Видим, что модель может классифицировать данные, причем как показывает со 100% точностью. Дело в том, что числовые признаки сильно отличаются друг от друга при смене класса - это видно из набора данных. В таких случаях метод К-ближайших соседей показывает наибольшую точность прогнозирования.

Заполнение пропусков значением из предыдущей записи

Крайне удачный вариант, так как категориальные признаки в данном случае отсортированы

```
df_test = df.fillna(method='bfill')
with pd.option_context('display.max_rows', None,
'display.max_columns', None):
    print(df_test)

# Получили практически исходный набор данных
```

```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal
width (cm)  \
0                  5.1               3.5                1.4
0.2
1                  4.9               3.0                1.4
0.2
2                  4.7               3.2                1.3
0.2
3                  4.6               3.1                1.5
0.2
4                  5.0               3.6                1.4
0.2
5                  5.4               3.9                1.7
0.4
6                  4.6               3.4                1.4
0.3
7                  5.0               3.4                1.5
0.2
8                  4.4               2.9                1.4
0.2
9                  4.9               3.1                1.5
0.1
10                 5.4               3.7                1.5
0.2
11                 4.8               3.4                1.6
0.2
12                 4.8               3.0                1.4
0.1
13                 4.3               3.0                1.1
0.1
14                 5.8               4.0                1.2
```

0.2			
15	5.7	4.4	1.5
0.4			
16	5.4	3.9	1.3
0.4			
17	5.1	3.5	1.4
0.3			
18	5.7	3.8	1.7
0.3			
19	5.1	3.8	1.5
0.3			
20	5.4	3.4	1.7
0.2			
21	5.1	3.7	1.5
0.4			
22	4.6	3.6	1.0
0.2			
23	5.1	3.3	1.7
0.5			
24	4.8	3.4	1.9
0.2			
25	5.0	3.0	1.6
0.2			
26	5.0	3.4	1.6
0.4			
27	5.2	3.5	1.5
0.2			
28	5.2	3.4	1.4
0.2			
29	4.7	3.2	1.6
0.2			
30	4.8	3.1	1.6
0.2			
31	5.4	3.4	1.5
0.4			
32	5.2	4.1	1.5
0.1			
33	5.5	4.2	1.4
0.2			
34	4.9	3.1	1.2
0.2			
35	5.0	3.2	1.2
0.2			
36	5.5	3.5	1.3
0.2			
37	4.9	3.6	1.4
0.1			
38	4.4	3.0	1.3
0.2			
39	5.1	3.4	1.5

0.2			
40	5.0	3.5	1.3
0.3			
41	4.5	2.3	1.3
0.3			
42	4.4	3.2	1.6
0.2			
43	5.0	3.5	1.6
0.6			
44	5.1	3.8	1.9
0.4			
45	4.8	3.0	1.4
0.3			
46	5.1	3.8	1.6
0.2			
47	4.6	3.2	1.4
0.2			
48	5.3	3.7	1.5
0.2			
49	5.0	3.3	1.4
0.2			
50	7.0	3.2	4.7
1.4			
51	6.4	3.2	4.5
1.5			
52	6.9	3.1	4.9
1.5			
53	5.5	2.3	4.0
1.3			
54	6.5	2.8	4.6
1.5			
55	5.7	2.8	4.5
1.3			
56	6.3	3.3	4.7
1.6			
57	4.9	2.4	3.3
1.0			
58	6.6	2.9	4.6
1.3			
59	5.2	2.7	3.9
1.4			
60	5.0	2.0	3.5
1.0			
61	5.9	3.0	4.2
1.5			
62	6.0	2.2	4.0
1.0			
63	6.1	2.9	4.7
1.4			
64	5.6	2.9	3.6

1.3			
65	6.7	3.1	4.4
1.4			
66	5.6	3.0	4.5
1.5			
67	5.8	2.7	4.1
1.0			
68	6.2	2.2	4.5
1.5			
69	5.6	2.5	3.9
1.1			
70	5.9	3.2	4.8
1.8			
71	6.1	2.8	4.0
1.3			
72	6.3	2.5	4.9
1.5			
73	6.1	2.8	4.7
1.2			
74	6.4	2.9	4.3
1.3			
75	6.6	3.0	4.4
1.4			
76	6.8	2.8	4.8
1.4			
77	6.7	3.0	5.0
1.7			
78	6.0	2.9	4.5
1.5			
79	5.7	2.6	3.5
1.0			
80	5.5	2.4	3.8
1.1			
81	5.5	2.4	3.7
1.0			
82	5.8	2.7	3.9
1.2			
83	6.0	2.7	5.1
1.6			
84	5.4	3.0	4.5
1.5			
85	6.0	3.4	4.5
1.6			
86	6.7	3.1	4.7
1.5			
87	6.3	2.3	4.4
1.3			
88	5.6	3.0	4.1
1.3			
89	5.5	2.5	4.0

1.3			
90	5.5	2.6	4.4
1.2			
91	6.1	3.0	4.6
1.4			
92	5.8	2.6	4.0
1.2			
93	5.0	2.3	3.3
1.0			
94	5.6	2.7	4.2
1.3			
95	5.7	3.0	4.2
1.2			
96	5.7	2.9	4.2
1.3			
97	6.2	2.9	4.3
1.3			
98	5.1	2.5	3.0
1.1			
99	5.7	2.8	4.1
1.3			
100	6.3	3.3	6.0
2.5			
101	5.8	2.7	5.1
1.9			
102	7.1	3.0	5.9
2.1			
103	6.3	2.9	5.6
1.8			
104	6.5	3.0	5.8
2.2			
105	7.6	3.0	6.6
2.1			
106	4.9	2.5	4.5
1.7			
107	7.3	2.9	6.3
1.8			
108	6.7	2.5	5.8
1.8			
109	7.2	3.6	6.1
2.5			
110	6.5	3.2	5.1
2.0			
111	6.4	2.7	5.3
1.9			
112	6.8	3.0	5.5
2.1			
113	5.7	2.5	5.0
2.0			
114	5.8	2.8	5.1

2.4			
115	6.4	3.2	5.3
2.3			
116	6.5	3.0	6.7
1.8			
117	7.7	3.8	6.7
2.2			
118	7.7	2.6	6.9
2.3			
119	6.0	2.2	5.0
1.5			
120	6.9	3.2	5.7
2.3			
121	5.6	2.8	4.9
2.0			
122	7.7	2.8	6.7
2.0			
123	6.3	2.7	4.9
1.8			
124	6.7	3.3	5.7
2.1			
125	7.2	3.2	6.0
1.8			
126	6.2	2.8	4.8
1.8			
127	6.1	3.0	4.9
1.8			
128	6.4	2.8	5.6
2.1			
129	7.2	3.0	5.8
1.6			
130	7.4	2.8	6.1
1.9			
131	7.9	3.8	6.4
2.0			
132	6.4	2.8	5.6
2.2			
133	6.3	2.8	5.1
1.5			
134	6.1	2.6	5.6
1.4			
135	7.7	3.0	5.6
2.3			
136	6.3	3.4	5.6
2.4			
137	6.4	3.1	5.5
1.8			
138	6.0	3.0	4.8
1.8			
139	6.9	3.1	5.4

2.1
140 6.7 3.1 5.6
2.4
141 6.9 3.1 5.1
2.3
142 5.8 2.7 5.1
1.9
143 6.8 3.2 5.7
2.3
144 6.7 3.3 5.7
2.5
145 6.7 3.0 5.0
2.3
146 6.3 2.5 5.0
1.9
147 6.5 3.0 5.4
2.0
148 6.2 3.4 5.4
2.3
149 5.9 3.0 5.1
1.8

	target
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
5	0.0
6	0.0
7	0.0
8	0.0
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0
15	0.0
16	0.0
17	0.0
18	0.0
19	0.0
20	0.0
21	0.0
22	0.0
23	0.0
24	0.0
25	0.0
26	0.0

27	0.0
28	0.0
29	0.0
30	0.0
31	0.0
32	0.0
33	0.0
34	0.0
35	0.0
36	0.0
37	0.0
38	0.0
39	0.0
40	0.0
41	0.0
42	0.0
43	0.0
44	0.0
45	0.0
46	0.0
47	0.0
48	0.0
49	0.0
50	1.0
51	1.0
52	1.0
53	1.0
54	1.0
55	1.0
56	1.0
57	1.0
58	1.0
59	1.0
60	1.0
61	1.0
62	1.0
63	1.0
64	1.0
65	1.0
66	1.0
67	1.0
68	1.0
69	1.0
70	1.0
71	1.0
72	1.0
73	1.0
74	1.0
75	1.0
76	1.0

77	1.0
78	1.0
79	1.0
80	1.0
81	1.0
82	1.0
83	1.0
84	1.0
85	1.0
86	1.0
87	1.0
88	1.0
89	1.0
90	1.0
91	1.0
92	1.0
93	1.0
94	1.0
95	1.0
96	1.0
97	1.0
98	1.0
99	1.0
100	2.0
101	2.0
102	2.0
103	2.0
104	2.0
105	2.0
106	2.0
107	2.0
108	2.0
109	2.0
110	2.0
111	2.0
112	2.0
113	2.0
114	2.0
115	2.0
116	2.0
117	2.0
118	2.0
119	2.0
120	2.0
121	2.0
122	2.0
123	2.0
124	2.0
125	2.0
126	2.0

```
127     2.0
128     2.0
129     2.0
130     2.0
131     2.0
132     2.0
133     2.0
134     2.0
135     2.0
136     2.0
137     2.0
138     2.0
139     2.0
140     2.0
141     2.0
142     2.0
143     2.0
144     2.0
145     2.0
146     2.0
147     2.0
148     2.0
149     2.0
```

Перейдем к обучению модели

```
iris.data.shape
```

```
(150, 4)
```

```
x = iris.data
y = iris.target
```

Делим данные на тестовую выборку и тренировочную

```
#Делим данные на тренировочное и тестовое множество:

X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size =
0.2, random_state = 42)
print(X_train.shape, X_test.shape, Y_test.shape, Y_train.shape)
```

```
(120, 4) (30, 4) (30,) (120,)
```

Будем использовать метод ближайших соседей, так как из данных видно, что классы имеют четко различимые границы

```
K_range = range(1, 11)
scores = {}
scores_list = []
for k in K_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, Y_train)
    Y_pred = knn.predict(X_test)
    scores[k] = metrics.accuracy_score(Y_test, Y_pred)
```

```
    scores_list.append(metrics.accuracy_score(Y_test, Y_pred))

print (scores)

{1: 1.0, 2: 1.0, 3: 1.0, 4: 1.0, 5: 1.0, 6: 1.0, 7:
0.9666666666666667, 8: 1.0, 9: 1.0, 10: 1.0}
```

Посмотрим какой подберет гиперпараметр встроенный метод KNeighborsClassifier

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x, y)

KNeighborsClassifier(n_neighbors=3)

classes = {0:'setosa', 1:'versicolor', 2:'virginica'}

X_new = [[3,4,5,2],
        [5,4,2,2,]]

Y_pred = knn.predict(X_new)

print (classes [Y_pred[0]])
print (classes [Y_pred[1]])

versicolor
setosa

#Создаем классификатор:

best_model = KNeighborsClassifier(
   n_neighbors=10,
   weights='distance',
   algorithm='auto',
   leaf_size=30,
   metric='euclidean',
   metric_params=None,
   n_jobs=4
)

best_model.fit(X_train, Y_train)
predicted = best_model.predict(X_test)

print('Evaluation:\n', metrics.classification_report(Y_test,
predicted))

Evaluation:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
```

```
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Видим, что модель может классифицировать данные, причем как показывает со 100% точностью. Дело в том, что числовые признаки сильно отличаются друг от друга при смене класса - это видно из набора данных. В таких случаях метод К-ближайших соседей показывает наибольшую точность прогнозирования.