Прогнозирование выдачи кредита

```
import pandas as pd
import numpy as np
```

Загрузка данных и предобработка

```
data = pd.read_csv("crx.data")
```

```
data.head()
```

```
   b  30.83      0  u  g  w  v  1.25  t t.1  01  f g.1  00202  0.1  +
0  a  58.67  4.460  u  g  q  h  3.04  t   t   6  f   g  00043  560  +
1  a  24.50  0.500  u  g  q  h  1.50  t   f   0  f   g  00280  824  +
2  b  27.83  1.540  u  g  w  v  3.75  t   t   5  t   g  00100    3  +
3  b  20.17  5.625  u  g  w  v  1.71  t   f   0  f   s  00120    0  +
4  b  32.08  4.000  u  g  m  v  2.50  t   f   0  t   g  00360    0  +
```

```
data.shape
```

```
(689, 16)
```

```
data['b'].unique()
```

```
array(['a', 'b', '?'], dtype=object)
```

```
data.replace('?', np.nan, inplace=True)
```

```
data['b'].unique()
```

```
array(['a', 'b', nan], dtype=object)
```

```
data.isnull().sum()
```

```
b        12
30.83    12
0         0
u         6
g         6
w         9
v         9
1.25      0
t         0
t.1       0
01        0
f         0
g.1       0
00202    13
0.1       0
+         0
dtype: int64
```

```
total_count = data.shape[0]
print('Всего строк: {}'.format(total_count))
```

```
Всего строк: 689
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['+'] = le.fit_transform(data['+'])
```

```
data.head()
```

```
   b  30.83      0  u  g  w  v  1.25  t t.1  01  f g.1  00202  0.1  +
0  a  58.67  4.460  u  g  q  h  3.04  t   t   6  f   g  00043  560  0
1  a  24.50  0.500  u  g  q  h  1.50  t   f   0  f   g  00280  824  0
2  b  27.83  1.540  u  g  w  v  3.75  t   t   5  t   g  00100    3  0
3  b  20.17  5.625  u  g  w  v  1.71  t   f   0  f   s  00120    0  0
4  b  32.08  4.000  u  g  m  v  2.50  t   f   0  t   g  00360    0  0
```

```
target = data['+']
np.unique(target)
```

```
array([0, 1])
```

```
data = data.drop(columns='+')
data.head()
```

```
   b  30.83      0  u  g  w  v  1.25  t t.1  01  f g.1  00202  0.1
0  a  58.67  4.460  u  g  q  h  3.04  t   t   6  f   g  00043  560
1  a  24.50  0.500  u  g  q  h  1.50  t   f   0  f   g  00280  824
2  b  27.83  1.540  u  g  w  v  3.75  t   t   5  t   g  00100    3
3  b  20.17  5.625  u  g  w  v  1.71  t   f   0  f   s  00120    0
4  b  32.08  4.000  u  g  m  v  2.50  t   f   0  t   g  00360    0
```

Обрабатываем пропуски

```
# Выберем числовые колонки с пропущенными значениями
# Цикл по колонкам датасета
num_cols = []
for col in data.columns:
    # Количество пустых значений
    temp_null_count = data[data[col].isnull()].shape[0]
    dt = str(data[col].dtype)
    if temp_null_count>0 and (dt=='float64' or dt=='int64'):
        num_cols.append(col)
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
        print('Колонка {}. Тип данных {}. Количество пустых значений
{}, {}%.'.format(col, dt, temp_null_count, temp_perc))

# Выберем категориальные колонки с пропущенными значениями
# Цикл по колонкам датасета
cat_cols = []
for col in data.columns:
```

```
    # Количество пустых значений
    temp_null_count = data[data[col].isnull()].shape[0]
    dt = str(data[col].dtype)
    if temp_null_count>0 and (dt=='object'):
        cat_cols.append(col)
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
        print('Колонка {}. Тип данных {}. Количество пустых значений
{}, {}%.'.format(col, dt, temp_null_count, temp_perc))
```

```
Колонка b. Тип данных object. Количество пустых значений 12, 1.74%.
Колонка 30.83. Тип данных object. Количество пустых значений 12,
1.74%.
Колонка u. Тип данных object. Количество пустых значений 6, 0.87%.
Колонка g. Тип данных object. Количество пустых значений 6, 0.87%.
Колонка w. Тип данных object. Количество пустых значений 9, 1.31%.
Колонка v. Тип данных object. Количество пустых значений 9, 1.31%.
Колонка 00202. Тип данных object. Количество пустых значений 13,
1.89%.
```

```
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
imp1 = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
data['b'] = imp1.fit_transform(data[['b']])

data['30.83'] = imp1.fit_transform(data[['30.83']])
data['u'] = imp1.fit_transform(data[['u']])
data['g'] = imp1.fit_transform(data[['g']])
data['w'] = imp1.fit_transform(data[['w']])
data['v'] = imp1.fit_transform(data[['v']])
data['00202'] = imp1.fit_transform(data[['00202']])
data.isnull().sum()
```

```
b        0
30.83    0
0        0
u        0
g        0
w        0
v        0
1.25     0
t        0
t.1      0
01       0
f        0
g.1      0
00202    0
0.1      0
dtype: int64
```

Кодируем категориальные признаки

```
data.dtypes
```

```
b         object
30.83     object
0        float64
u         object
g         object
w         object
v         object
1.25     float64
t         object
t.1       object
01         int64
f         object
g.1       object
00202     object
0.1        int64
dtype: object
```

```
data['b'] = le.fit_transform(data['b'])
data['30.83'] = le.fit_transform(data['30.83'])
data['u'] = le.fit_transform(data['u'])
data['g'] = le.fit_transform(data['g'])
data['w'] = le.fit_transform(data['w'])
data['v'] = le.fit_transform(data['v'])
data['t'] = le.fit_transform(data['t'])
data['t.1'] = le.fit_transform(data['t.1'])
data['g.1'] = le.fit_transform(data['g.1'])
data['f'] = le.fit_transform(data['f'])
data['00202'] = le.fit_transform(data['00202'])
data.head()
```

	b	30.83	0	u	g	w	v	1.25	t	t.1	01	f	g.1	00202	0.1
0	0	327	4.460	1	0	10	3	3.04	1	1	6	0	0	11	560
1	0	89	0.500	1	0	10	3	1.50	1	0	0	0	0	95	824
2	1	125	1.540	1	0	12	7	3.75	1	1	5	1	0	31	3
3	1	43	5.625	1	0	12	7	1.71	1	0	0	0	2	37	0
4	1	167	4.000	1	0	9	7	2.50	1	0	0	1	0	114	0

Масштабируем числовые данные

```
from sklearn.preprocessing import MinMaxScaler
sc1 = MinMaxScaler()
data['0'] = sc1.fit_transform(data[['0']])
data['1.25'] = sc1.fit_transform(data[['1.25']])
data['01'] = sc1.fit_transform(data[['01']])
data['0.1'] = sc1.fit_transform(data[['0.1']])
data.head()
```

```
   b  30.83         0  u  g   w  v      1.25  t  t.1        01  f  g.1
00202  \
0  0    327  0.159286  1  0  10  3  0.106667  1    1  0.089552  0    0
11
1  0     89  0.017857  1  0  10  3  0.052632  1    0  0.000000  0    0
95
2  1    125  0.055000  1  0  12  7  0.131579  1    1  0.074627  1    0
31
3  1     43  0.200893  1  0  12  7  0.060000  1    0  0.000000  0    2
37
4  1    167  0.142857  1  0   9  7  0.087719  1    0  0.000000  1    0
114

       0.1
0  0.00560
1  0.00824
2  0.00003
3  0.00000
4  0.00000
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR,
NuSVR, LinearSVR
from sklearn import svm
from scipy.stats import uniform as sp_randFloat
from scipy.stats import randint as sp_randInt
from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score,
classification_report
from sklearn.metrics import confusion_matrix
res = {}
res_1 = {}
```

Делим выборку на обучающую и тестовую

```
from sklearn.model_selection import train_test_split
data_X_train, data_X_test, data_y_train, data_y_test =
train_test_split(
    data, target, test_size=0.2, random_state=1)
```

```
data_X_train.shape, data_y_train.shape
```

```
((551, 15), (551,))
```

```
data_X_test.shape, data_y_test.shape
```

```
((138, 15), (138,))
```

```
np.unique(target)
```

```
array([0, 1])
```

Обучение модели

Обучаем модель с помощью следующих методов:

1. Логистическая регрессия
2. Дерево решений
3. Случайный лес
4. Метод опорных векторов
5. Градиентного бустинга

Логистическая регрессия

```
model = LogisticRegression()
parameters = {"C": np.logspace(-4, 2, 20),
                  "solver"    : ['newton-cg', 'lbfgs', 'liblinear'],
                  "class_weight"    : ['balanced', None]
                 }
randm_1 = RandomizedSearchCV(estimator=model, param_distributions =
parameters, scoring='f1',
                                cv = 2, n_iter = 10, n_jobs=-1)
randm_1.fit(data_X_train, data_y_train)
print('Лучшие параметры:', randm_1.best_params_)
print('F-мера на перекрестной проверке:', randm_1.best_score_)
print('F-мера логистической регрессии на тестовом наборе:',
randm_1.score(data_X_test, data_y_test))
res['логистической регрессии'] = randm_1.score(data_X_test,
data_y_test)
```

```
Лучшие параметры: {'solver': 'liblinear', 'class_weight': None, 'C':
11.288378916846883}
F-мера на перекрестной проверке: 0.868476430976431
F-мера логистической регрессии на тестовом наборе: 0.8671328671328671
```

```
model = LogisticRegression()
parameters = {"C": np.logspace(-4, 2, 20),
                  "solver"    : ['newton-cg', 'lbfgs', 'liblinear'],
                  "class_weight"    : ['balanced', None]
                 }
randm_1_1 = RandomizedSearchCV(estimator=model, param_distributions =
parameters, scoring='accuracy',
                                cv = 2, n_iter = 10, n_jobs=-1)
```

```
randm_1_1.fit(data_X_train, data_y_train)
print('Лучшие параметры:', randm_1_1.best_params_)
print('Accuracy на перекрестной проверке:', randm_1_1.best_score_)
print('Accuracy логистической регрессии на тестовом наборе:',
randm_1_1.score(data_X_test, data_y_test))
res_1['логистической регрессии'] = randm_1_1.score(data_X_test,
data_y_test)
```

```
Лучшие параметры: {'solver': 'newton-cg', 'class_weight': 'balanced',
'C': 2.6366508987303554}
Accuracy на перекрестной проверке: 0.8566271409749671
Accuracy логистической регрессии на тестовом наборе:
0.8623188405797102
```

Дерево решений

```
model = DecisionTreeClassifier()
parameters = {"min_samples_split": range(2, 200, 5),
                  "criterion"    : ['gini', 'entropy'],
                  "class_weight" : ['balanced', None],
                  "max_depth"    : range(1, 35),
              "max_features" : ['auto', None, 'log2']
                 }
randm_2 = RandomizedSearchCV(estimator=model, param_distributions =
parameters, scoring='f1',
                             cv = 2, n_iter = 10, n_jobs=-1)
randm_2.fit(data_X_train, data_y_train)
print('Лучшие параметры:', randm_2.best_params_)
print('F-мера на перекрестной проверке:', randm_2.best_score_)
print('F-мера Дерево решений на тестовом наборе:',
randm_2.score(data_X_test, data_y_test))
res['Дерево решений'] = randm_2.score(data_X_test, data_y_test)
```

```
Лучшие параметры: {'min_samples_split': 22, 'max_features': None,
'max_depth': 28, 'criterion': 'entropy', 'class_weight': 'balanced'}
F-мера на перекрестной проверке: 0.8559441483504383
F-мера Дерево решений на тестовом наборе: 0.8258064516129032
```

```
model = DecisionTreeClassifier()
parameters = {"min_samples_split": range(2, 200, 5),
                  "criterion"    : ['gini', 'entropy'],
                  "class_weight" : ['balanced', None],
                  "max_depth"    : range(1, 35),
              "max_features" : ['auto', None, 'log2']
                 }
randm_2_1 = RandomizedSearchCV(estimator=model, param_distributions =
parameters, scoring='accuracy',
                             cv = 2, n_iter = 10, n_jobs=-1)
randm_2_1.fit(data_X_train, data_y_train)
print('Лучшие параметры:', randm_2_1.best_params_)
```

```
print('Accuracy на перекрестной проверке:', randm_2_1.best_score_)
print('Accuracy Дерево решений на тестовом наборе:',
randm_2_1.score(data_X_test, data_y_test))
res_1['Дерево решений'] = randm_2_1.score(data_X_test, data_y_test)
```

```
Лучшие параметры: {'min_samples_split': 117, 'max_features': 'auto',
'max_depth': 15, 'criterion': 'entropy', 'class_weight': 'balanced'}
Accuracy на перекрестной проверке: 0.847562582345191
Accuracy Дерево решений на тестовом наборе: 0.8260869565217391
```

Метод опорных векторов

```
model = SVC()
parameters = {"C": np.logspace(-4, 2, 20),
                  "gamma"    : ['scale', 'auto'],
                  "class_weight" : ['balanced', None],
                  "kernel"    : ['linear', 'rbf', 'poly', 'sigmoid']
                 }
randm_3 = RandomizedSearchCV(estimator=model, param_distributions =
parameters, scoring='f1',
                             cv = 2, n_iter = 10, n_jobs=-1)
randm_3.fit(data_X_train, data_y_train)
print('Лучшие параметры:', randm_3.best_params_)
print('Accuracy на перекрестной проверке:', randm_3.best_score_)
print('Accuracy метода опорных векторов на тестовом наборе:',
randm_3.score(data_X_test, data_y_test))
res['Метод опорных векторов'] = randm_3.score(data_X_test,
data_y_test)
```

```
Лучшие параметры: {'kernel': 'linear', 'gamma': 'scale',
'class_weight': 'balanced', 'C': 0.03359818286283781}
Accuracy на перекрестной проверке: 0.8580807461038165
Accuracy метода опорных векторов на тестовом наборе:
0.8652482269503547
```

```
model = SVC()
parameters = {"C": np.logspace(-4, 2, 20),
                  "gamma"    : ['scale', 'auto'],
                  "class_weight" : ['balanced', None],
                  "kernel"    : ['linear', 'rbf', 'poly', 'sigmoid']
                 }
randm_3_1 = RandomizedSearchCV(estimator=model, param_distributions =
parameters, scoring='accuracy',
                             cv = 2, n_iter = 10, n_jobs=-1)
randm_3_1.fit(data_X_train, data_y_train)
print('Лучшие параметры:', randm_3_1.best_params_)
print('Accuracy на перекрестной проверке:', randm_3_1.best_score_)
print('Accuracy метода опорных векторов на тестовом наборе:',
randm_3_1.score(data_X_test, data_y_test))
```

```
res_1['Метод опорных векторов'] = randm_3_1.score(data_X_test,
data_y_test)
```

```
Лучшие параметры: {'kernel': 'linear', 'gamma': 'auto',
'class_weight': 'balanced', 'C': 48.32930238571752}
Accuracy на перекрестной проверке: 0.8566337285902503
Accuracy метода опорных векторов на тестовом наборе: 0.855072463768116
```

Случайный лес

```
model = RandomForestClassifier()
parameters = {"min_samples_split": range(2, 100, 5),
                  "criterion"    : ['gini', 'entropy'],
                  "n_estimators" : range(50, 200, 5),
                  "max_depth"    : range(1, 35),
              "class_weight" : ['balanced', None],
              "max_features": ['auto', None, 'log2']
                 }
randm_4 = RandomizedSearchCV(estimator=model, param_distributions =
parameters, scoring='f1',
                             cv = 2, n_iter = 10, n_jobs=-1)
randm_4.fit(data_X_train, data_y_train)
print('Лучшие параметры:', randm_4.best_params_)
print('F-мера на перекрестной проверке:', randm_4.best_score_)
print('F-мера случайного леса на тестовом наборе:',
randm_4.score(data_X_test, data_y_test))
res['Случайный лес'] = randm_4.score(data_X_test, data_y_test)
```

```
Лучшие параметры: {'n_estimators': 195, 'min_samples_split': 2,
'max_features': 'log2', 'max_depth': 7, 'criterion': 'entropy',
'class_weight': 'balanced'}
F-мера на перекрестной проверке: 0.8818713687888513
F-мера случайного леса на тестовом наборе: 0.8783783783783783
```

```
model = RandomForestClassifier()
parameters = {"min_samples_split": range(2, 100, 5),
                  "criterion"    : ['gini', 'entropy'],
                  "n_estimators" : range(50, 200, 5),
                  "max_depth"    : range(1, 35),
              "class_weight" : ['balanced', None],
              "max_features": ['auto', None, 'log2']
                 }
randm_4_1 = RandomizedSearchCV(estimator=model, param_distributions =
parameters, scoring='accuracy',
                             cv = 2, n_iter = 10, n_jobs=-1)
randm_4_1.fit(data_X_train, data_y_train)
print('Лучшие параметры:', randm_4_1.best_params_)
print('Accuracy на перекрестной проверке:', randm_4_1.best_score_)
print('Accuracy случайного леса на тестовом наборе:',
```

```
randm_4_1.score(data_X_test, data_y_test))
res_1['Случайный лес'] = randm_4_1.score(data_X_test, data_y_test)
```

```
Лучшие параметры: {'n_estimators': 55, 'min_samples_split': 7,
'max_features': 'auto', 'max_depth': 19, 'criterion': 'gini',
'class_weight': 'balanced'}
Accuracy на перекрестной проверке: 0.867536231884058
Accuracy случайного леса на тестовом наборе: 0.8985507246376812
```

Градиентный бустинг

```
model = GradientBoostingClassifier()
parameters = {"learning_rate": sp_randFloat(),
                  "subsample"    : sp_randFloat(),
                  "n_estimators" : sp_randInt(100, 1000),
                  "max_depth"    : sp_randInt(4, 10)
                 }
randm_5 = RandomizedSearchCV(estimator=model, param_distributions =
parameters, scoring='f1',
                                cv = 2, n_iter = 10, n_jobs=-1)
randm_5.fit(data_X_train, data_y_train)
print('Лучшие параметры:', randm_5.best_params_)
print('F-мера на перекрестной проверке:', randm_5.best_score_)
print('F-мера градиентного бустинга на тестовом наборе:',
randm_5.score(data_X_test, data_y_test))
res['Градиентный бустинг'] = randm_5.score(data_X_test, data_y_test)
```

```
Лучшие параметры: {'learning_rate': 0.01743355607035113, 'max_depth':
8, 'n_estimators': 348, 'subsample': 0.012059409372783492}
F-мера на перекрестной проверке: 0.8706095424720168
F-мера градиентного бустинга на тестовом наборе: 0.8652482269503547
```

```
model = GradientBoostingClassifier()
parameters = {"learning_rate": sp_randFloat(),
                  "subsample"    : sp_randFloat(),
                  "n_estimators" : sp_randInt(100, 1000),
                  "max_depth"    : sp_randInt(4, 10)
                 }
randm_5_1 = RandomizedSearchCV(estimator=model, param_distributions =
parameters, scoring='accuracy',
                                cv = 2, n_iter = 10, n_jobs=-1)
randm_5_1.fit(data_X_train, data_y_train)
print('Лучшие параметры:', randm_5_1.best_params_)
print('Accuracy на перекрестной проверке:', randm_5_1.best_score_)
print('Accuracy градиентного бустинга на тестовом наборе:',
randm_5_1.score(data_X_test, data_y_test))
res_1['Градиентный бустинг'] = randm_5_1.score(data_X_test,
data_y_test)
```

```
Лучшие параметры: {'learning_rate': 0.2833726466380505, 'max_depth':
8, 'n_estimators': 252, 'subsample': 0.5031796276113675}
```

```
Accuracy на перекрестной проверке: 0.8584584980237154
Accuracy градиентного бустинга на тестовом наборе: 0.8768115942028986
```

Вывод результатов

```
print('Для F-меры:')
res
```

```
Для F-меры:

{'логистической регрессии': 0.8671328671328671,
 'Дерево решений': 0.8258064516129032,
 'Случайный лес': 0.8783783783783783,
 'Градиентный бустинг': 0.8652482269503547,
 'Метод опорных векторов': 0.8652482269503547}
```

```
print('Для accuracy:')
res_1
```

```
Для accuracy:

{'логистической регрессии': 0.8623188405797102,
 'Дерево решений': 0.8260869565217391,
 'Метод опорных векторов': 0.855072463768116,
 'Случайный лес': 0.8985507246376812,
 'Градиентный бустинг': 0.8768115942028986}
```

```
from sklearn.metrics import confusion_matrix
y_pred = randm_4.predict(data_X_test)
confusion_matrix(data_y_test, y_pred)
```

```
array([[55,  7],
       [11, 65]], dtype=int64)
```

```
from sklearn.metrics import confusion_matrix
y_pred = randm_4_1.predict(data_X_test)
confusion_matrix(data_y_test, y_pred)
```

```
array([[57,  5],
       [ 9, 67]], dtype=int64)
```