Московский государственный технический
университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Технологии машинного обучения»

Отчет по лабораторной работе №3

«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей.»

Выполнил:
студент группы ИУ5-62Б
Барышников Михаил

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

Москва, 2022 г.

Описание задания:

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
3. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.
4. Произведите подбор гиперпараметра K с использованием GridSearchCV и/или RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Желательно использование нескольких стратегий кросс-валидации.
5. Сравните метрики качества исходной и оптимальной моделей.

Лабораторая работа №3: Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей.

In [1]:
```
#Датасет содержит данные о кредитах на покупку электроники, которые были одобрены Tinkoff.ru.
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from warnings import simplefilter

simplefilter('ignore')
```

In [2]:
```
# записываем CSV-файл в объект DataFrame
data = pd.read_csv('credit_train.csv', encoding='cp1251', sep=';')
```

In [3]:
```
# смотрим на первые пять строк
data.head()
```

Out[3]:

	client_id	gender	age	marital_status	job_position	credit_sum	credit_month	tariff_id	score_shk	education	living_region	monthly_income	credit_count	overdue_credit_count	open_
0	1	M	NaN	NaN	UMN	59998,00	10	1.6	NaN	GRD	КРАСНОДАРСКИЙ КРАЙ	30000.0	1.0	1.0	
1	2	F	NaN	MAR	UMN	10889,00	6	1.1	NaN	NaN	МОСКВА	NaN	2.0	0.0	
2	3	M	32.0	MAR	SPC	10728,00	12	1.1	NaN	NaN	ОБЛ САРАТОВСКАЯ	NaN	5.0	0.0	
3	4	F	27.0	NaN	SPC	12009,09	12	1.1	NaN	NaN	ОБЛ ВОЛГОГРАДСКАЯ	NaN	2.0	0.0	
4	5	M	45.0	NaN	SPC	NaN	10	1.1	0,421385	SCH	ЧЕЛЯБИНСКАЯ ОБЛАСТЬ	NaN	1.0	0.0	

1) Обработка пропусков в данных

In [4]:
```
#проверяем типы данных и заполненность столбцов
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170746 entries, 0 to 170745
Data columns (total 15 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   client_id             170746 non-null  int64  
 1   gender                170746 non-null  object 
 2   age                   170743 non-null  float64
 3   marital_status        170743 non-null  object 
 4   job_position          170746 non-null  object 
 5   credit_sum            170744 non-null  object 
 6   credit_month          170746 non-null  int64  
 7   tariff_id             170746 non-null  float64
 8   score_shk             170739 non-null  object 
 9   education             170741 non-null  object 
 10  living_region         170554 non-null  object 
 11  monthly_income        170741 non-null  float64
 12  credit_count          161516 non-null  float64
 13  overdue_credit_count  161516 non-null  float64
 14  open_account_flg      170746 non-null  int64  
dtypes: float64(5), int64(3), object(7)
memory usage: 19.5+ MB
```

In [5]:
```
#удаляем столбец с номером клиента (так как он незначимый)
# и с регионом проживания (так как он нуждается в серьезной предобработке)
data.drop(['client_id', 'living_region'], axis=1, inplace=True)
```

In [6]:
```
# анализируем столбец marital_status, смотрим, какое значение в нем является самым частым
data['marital_status'].describe()
```

Out[6]:
```
count     170743
unique         5
top          MAR
freq       93954
Name: marital_status, dtype: object
```

In [7]:
```
# анализируем столбец education, смотрим, какое в нем самое частое значение
data['education'].describe()
```

Out[7]:
```
count     170741
unique         5
top          SCH
freq       87537
Name: education, dtype: object
```

In [8]:
```
# дозаполняем нечисловые столбцы с пропусками самыми часто встречающимися значениями
data['marital_status'].fillna('MAR', inplace=True)
data['education'].fillna('SCH', inplace=True)
```

In [9]:
```
# дозаполняем числовые столбцы с пропусками медианными значениями
data['age'].fillna(data['age'].median(), inplace=True)
data['credit_count'].fillna(data['credit_count'].median(), inplace=True)
data['overdue_credit_count'].fillna(data['overdue_credit_count'].median(), inplace=True)
```

In [10]:
```
#меняем в столбцах 'credit_sum', 'score_shk'  запятые на точки  и преобразуем их в числовой  формат
for i in ['credit_sum', 'score_shk']:
    data[i] = data[i].str.replace(',', '.').astype('float')
```

In [11]:
```
# дозаполняем ставшие теперь числовыми столбцы 'credit_sum', 'score_shk'   медианными значениями
data['score_shk'].fillna(data['score_shk'].median(), inplace=True)
data['monthly_income'].fillna(data['monthly_income'].median(), inplace=True)
data['credit_sum'].fillna(data['credit_sum'].median(), inplace=True)
```

In [12]:
```
# смотрим, что получилось
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170746 entries, 0 to 170745
Data columns (total 13 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   gender                170746 non-null  object 
 1   age                   170746 non-null  float64
 2   marital_status        170746 non-null  object 
 3   job_position          170746 non-null  object 
 4   credit_sum            170746 non-null  float64
 5   credit_month          170746 non-null  int64  
 6   tariff_id             170746 non-null  float64
 7   score_shk             170746 non-null  float64
 8   education             170746 non-null  object 
 9   monthly_income        170746 non-null  float64
 10  credit_count          170746 non-null  float64
 11  overdue_credit_count  170746 non-null  float64
 12  open_account_flg      170746 non-null  int64  
dtypes: float64(7), int64(2), object(4)
memory usage: 16.9+ MB
```

2) Кодирование категориальных признаков

In [13]:
```
category_cols = ['gender', 'job_position', 'education', 'marital_status']
```

In [14]:
```
print("Количество уникальных значений\n")
for col in category_cols:
    print(f'{col}: {data[col].unique().size}')
```

```
Количество уникальных значений

gender: 2
job_position: 18
education: 5
marital_status: 5
```

In [15]:
```
# кодируем нечисловые столбцы методом дамми-кодирования
data = pd.concat([data,
                  pd.get_dummies(data['gender'], prefix="gender"),
                  pd.get_dummies(data['job_position'], prefix="job_position"),
                  pd.get_dummies(data['education'], prefix="education"),
                  pd.get_dummies(data['marital_status'], prefix="marital_status")],
                 axis=1)
```

In [16]:
```
#удаляем старые нечисловые столбцы, вместо них уже появились новые числовые
data.drop(['gender','job_position','education','marital_status'], axis=1, inplace=True)
```

In [17]:
```
data.head()
```

Out[17]:

	age	credit_sum	credit_month	tariff_id	score_shk	monthly_income	credit_count	overdue_credit_count	open_account_flg	gender_F	...	education_ACD	education_GRD	education_PGR
0	34.0	59998.00	10	1.6	0.461599	30000.0	1.0	1.0	0	0	...	0	1	0
1	34.0	10889.00	6	1.1	0.461599	35000.0	2.0	0.0	0	1	...	0	0	0
2	32.0	10728.00	12	1.1	0.461599	35000.0	5.0	0.0	0	0	...	0	0	0
3	27.0	12009.09	12	1.1	0.461599	35000.0	2.0	0.0	0	1	...	0	0	0
4	45.0	21229.00	10	1.1	0.421385	35000.0	1.0	0.0	0	0	...	0	0	0

5 rows × 39 columns

3) Разделение выборки на обучающую и тестовую

In [18]:
```
data_sample = data.sample(n=20000)
y = data_sample['open_account_flg']
X = data_sample.drop('open_account_flg', axis=1)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=21)
```

4) Масштабирование данных

In [19]:
```
scaler = MinMaxScaler().fit(x_train)
x_train = pd.DataFrame(scaler.transform(x_train), columns=x_train.columns)
x_test = pd.DataFrame(scaler.transform(x_test), columns=x_train.columns)
x_train.describe()
```

Out[19]:

	age	credit_sum	credit_month	tariff_id	score_shk	monthly_income	credit_count	overdue_credit_count	gender_F	gender_M	...	education_ACD	education_(
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.0000	10000.000000	10000.000000	...	10000.00000	10000.00(
mean	0.350174	0.116259	0.241191	0.341563	0.491353	0.043287	0.116350	0.0223	0.526600	0.473400	...	0.00110	0.42(
std	0.198120	0.083180	0.107251	0.249714	0.130051	0.033850	0.097631	0.1037	0.499317	0.499317	...	0.03315	0.49(
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000	0.000000	0.000000	...	0.00000	0.00(
25%	0.188679	0.059753	0.212121	0.106383	0.397739	0.023929	0.055556	0.0000	0.000000	0.000000	...	0.00000	0.00(
50%	0.320755	0.091183	0.212121	0.340426	0.482866	0.036524	0.111111	0.0000	1.000000	0.000000	...	0.00000	0.00(
75%	0.471698	0.146176	0.272727	0.638298	0.578886	0.055416	0.166667	0.0000	1.000000	1.000000	...	0.00000	1.00(
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000	1.000000	1.000000	...	1.00000	1.00(

8 rows × 38 columns

5) Обучение KNN с произвольным k

In [20]:
```
def print_metrics(y_test, y_pred):
    print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
    print(f"Precision: {precision_score(y_test, y_pred)}")
    print(f"Recall: {recall_score(y_test, y_pred)}")
    print(f"F1-measure: {f1_score(y_test, y_pred)}")

def print_cv_result(cv_model, x_test, y_test):
```

```
    print(f'Оптимизация метрики {cv_model.scoring}: {cv_model.best_score_}')
    print(f'Лучший параметр: {cv_model.best_params_}')
    print('Метрики на тестовом наборе')
    print_metrics(y_test, cv_model.predict(x_test))
    print()
```

In [21]:
```
base_k = 10
base_knn = KNeighborsClassifier(n_neighbors=base_k)
base_knn.fit(x_train, y_train)
y_pred_base = base_knn.predict(x_test)
```

In [82]:
```
print(f'Test metrics for KNN with k={base_k}\n')
print_metrics(y_test, y_pred_base)
```

```
Test metrics for KNN with k=10

Accuracy: 0.8154
Precision: 0.42857142857142855
Recall: 0.04118616144975288
F1-measure: 0.0751503006012024
```

6) Кросс-валидация

In [85]:
```
metrics = ['accuracy', 'precision', 'recall', 'f1']
cv_values = [5, 10]

for cv in cv_values:
    print(f'Результаты кросс-валидации при cv={cv}\n')
    for metric in metrics:
        params = {'n_neighbors': range(1, 40)}
        knn_cv = RandomizedSearchCV(KNeighborsClassifier(), params, cv=cv, scoring=metric, n_jobs=-1)
        #knn_cv = GridSearchCV(KNeighborsClassifier(), params, cv=cv, scoring=metric, n_jobs=-1)
        knn_cv.fit(x_train, y_train)
        print_cv_result(knn_cv, x_test, y_test)
```

```
Результаты кросс-валидации при cv=5

Оптимизация метрики accuracy: 0.8278000000000001
Лучший параметр: {'n_neighbors': 37}
Метрики на тестовом наборе
Accuracy: 0.8178
Precision: 0.47058823529411764
Recall: 0.004393190554640308
F1-measure: 0.008705114254624592

Оптимизация метрики precision: 0.36774114774114774
Лучший параметр: {'n_neighbors': 18}
Метрики на тестовом наборе
Accuracy: 0.8174
Precision: 0.4666666666666667
Recall: 0.019220208676551345
F1-measure: 0.03691983122362869

Оптимизация метрики recall: 0.0796935385449861
Лучший параметр: {'n_neighbors': 7}
Метрики на тестовом наборе
Accuracy: 0.8077
Precision: 0.37799043062200954
Recall: 0.08676551345414607
F1-measure: 0.14113443501563197

Оптимизация метрики f1: 0.2416083143090169
Лучший параметр: {'n_neighbors': 1}
Метрики на тестовом наборе
Accuracy: 0.7348
Precision: 0.25428740390301596
Recall: 0.23613399231191654
F1-measure: 0.24487471526195898

Результаты кросс-валидации при cv=10

Оптимизация метрики accuracy: 0.8276
Лучший параметр: {'n_neighbors': 20}
Метрики на тестовом наборе
Accuracy: 0.8179
Precision: 0.5
Recall: 0.01757276221856123
F1-measure: 0.03395225464190982

Оптимизация метрики precision: 0.41375
Лучший параметр: {'n_neighbors': 17}
Метрики на тестовом наборе
Accuracy: 0.8171
Precision: 0.47368421052631576
Recall: 0.039538714991762765
F1-measure: 0.07298530157121133

Оптимизация метрики recall: 0.15418196654426766
Лучший параметр: {'n_neighbors': 3}
Метрики на тестовом наборе
Accuracy: 0.7831
Precision: 0.3213552361396304
Recall: 0.17188358045030203
F1-measure: 0.22397137745974957

Оптимизация метрики f1: 0.2000082832883067
Лучший параметр: {'n_neighbors': 3}
Метрики на тестовом наборе
Accuracy: 0.7831
Precision: 0.3213552361396304
Recall: 0.17188358045030203
F1-measure: 0.22397137745974957
```

In [86]:
```
for cv in cv_values:
    print(f'Результаты кросс-валидации при cv={cv}\n')
    for metric in metrics:
        params = {'n_neighbors': range(1, 40)}
        #knn_cv = RandomizedSearchCV(KNeighborsClassifier(), params, cv=cv, scoring=metric, n_jobs=-1)
        knn_cv = GridSearchCV(KNeighborsClassifier(), params, cv=cv, scoring=metric, n_jobs=-1)
        knn_cv.fit(x_train, y_train)
        print_cv_result(knn_cv, x_test, y_test)
```

```
Результаты кросс-валидации при cv=5

Оптимизация метрики accuracy: 0.8279
Лучший параметр: {'n_neighbors': 34}
Метрики на тестовом наборе
Accuracy: 0.8179
Precision: 0.5
Recall: 0.0032948929159802307
F1-measure: 0.006546644844517184

Оптимизация метрики precision: 0.44286393345216873
Лучший параметр: {'n_neighbors': 19}
Метрики на тестовом наборе
Accuracy: 0.8174
Precision: 0.4752475247524752
Recall: 0.026359143327841845
F1-measure: 0.04994797086368366

Оптимизация метрики recall: 0.2367516441792664
Лучший параметр: {'n_neighbors': 1}
Метрики на тестовом наборе
Accuracy: 0.7348
Precision: 0.25428740390301596
Recall: 0.23613399231191654
F1-measure: 0.24487471526195898

Оптимизация метрики f1: 0.2416083143090169
Лучший параметр: {'n_neighbors': 1}
Метрики на тестовом наборе
Accuracy: 0.7348
Precision: 0.25428740390301596
Recall: 0.23613399231191654
F1-measure: 0.24487471526195898

Результаты кросс-валидации при cv=10

Оптимизация метрики accuracy: 0.8281000000000001
Лучший параметр: {'n_neighbors': 32}
Метрики на тестовом наборе
Accuracy: 0.8179
Precision: 0.5
Recall: 0.003844041735310269
F1-measure: 0.007629427792915531

Оптимизация метрики precision: 0.465
Лучший параметр: {'n_neighbors': 20}
Метрики на тестовом наборе
Accuracy: 0.8179
Precision: 0.5
Recall: 0.01757276221856123
F1-measure: 0.03395225464190982

Оптимизация метрики recall: 0.2286311709506324
Лучший параметр: {'n_neighbors': 1}
Метрики на тестовом наборе
Accuracy: 0.7348
Precision: 0.25428740390301596
Recall: 0.23613399231191654
F1-measure: 0.24487471526195898

Оптимизация метрики f1: 0.23374373055765627
Лучший параметр: {'n_neighbors': 1}
Метрики на тестовом наборе
Accuracy: 0.7348
Precision: 0.25428740390301596
Recall: 0.23613399231191654
F1-measure: 0.24487471526195898
```

In [23]:
```
best_k = 1
y_pred_best = KNeighborsClassifier(n_neighbors=best_k).fit(x_train, y_train).predict(x_test)
```

7) Сравнение исходной и оптимальной моделей

In [24]:
```
print('Basic model\n')
print_metrics(y_test, y_pred_base)
print('_____________________')
print('\nOptimal model\n')
print_metrics(y_test, y_pred_best)
```

```
Basic model

Accuracy: 0.8201
Precision: 0.39037433155080214
Recall: 0.04152445961319681
F1-measure: 0.07506426735218508
_____________________

Optimal model

Accuracy: 0.7381
Precision: 0.24721080446271285
Recall: 0.23947667804323094
F1-measure: 0.24328228835596646
```

In [99]:
```
# анализируем зависимую переменную: какие значения она принимает и сколько раз
data['open_account_flg'].value_counts(dropna=False)
```

Out[99]:
```
0    140690
1     30056
Name: open_account_flg, dtype: int64
```

In [100…
```
# считаем, какая точность (доля правильных ответов) была бы у модели, если всем подряд пресказывать, что кредит они не выберут
d=140690/(140690+30056)
d
```

Out[100]:
```
0.823972450306303
```

In [ ]:
```

```