МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э.Баумана

Отчет по лабораторной работе №4
по курсу «Технологии машинного обучения»

Линейные модели, SVM и деревья решений.

Подготовил
Ионов С.А.
ИУ5-62Б

1) Описание задания

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
4. Обучите следующие модели:
    - одну из линейных моделей;
    - SVM;
    - дерево решений.
5. Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей

Дополнительные задания:

- Проведите эксперименты с важностью признаков в дереве решений.
- Визуализируйте дерево решений.

2) Текст программы

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
import numpy as np
from typing import Dict, Tuple
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import plot_roc_curve, plot_precision_recall_curve
from sklearn.model_selection import cross_validate, KFold, StratifiedKFold
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from scipy.stats import uniform, randint
from sklearn.model_selection import validation_curve, learning_curve
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier, export_graphviz

import graphviz
import pydotplus
```

0) Подготовка

```
data = pd.read_csv('data/BankChurners.csv', sep=',')
data.drop(columns=data.columns[[data.shape[1]-2, data.shape[1]-1]], inplace=True)
data.head()
data.info()
data.duplicated().sum()
col_obj = data.dtypes[data.dtypes==object].index.values.tolist()
for i in enumerate(col_obj):
    uniq_obj = data[i[1]].unique()
    print(f'{i[0]+1}. {i[1]}: {uniq_obj} | КОЛ-ВО: {len(uniq_obj)}')
le = LabelEncoder()
data['Gender'] = le.fit_transform(data['Gender'])
for i in range(2):
    print(f'{i}: {le.inverse_transform([i]).tolist()[0]}')
mapping = {'Existing Customer' : 0, 'Attrited Customer' : 1}
data['Attrition_Flag'] = data['Attrition_Flag'].apply(lambda x: mapping[x])
data_copy = data
data_copy = data_copy.apply(LabelEncoder().fit_transform)
col_num = data.dtypes[data.dtypes!=object].index.values.tolist()
col_num.remove('Attrition_Flag')
col_num.remove('Gender')
se = StandardScaler()
data[col_num] = se.fit_transform(data[col_num])
data = pd.get_dummies(data, drop_first=True)
plt.figure(figsize=(10,10))
g = sns.heatmap(data_copy.corr())
data_copy.corr()['Avg_Open_To_Buy'].sort_values(ascending=False)
data_copy.corr()['Credit_Limit'].sort_values(ascending=False)
data_copy.corr()['Total_Trans_Amt'].sort_values(ascending=False)
data_copy.corr()['Total_Trans_Ct'].sort_values(ascending=False)
data_copy.corr()['Attrition_Flag'].sort_values(ascending=False)

dataD_copy = data_copy.drop(columns=['Avg_Open_To_Buy'])
dataD = data.drop(columns=['Avg_Open_To_Buy'])
data_copy.drop(columns=['Avg_Open_To_Buy', 'Total_Trans_Amt'], inplace=True)
data.drop(columns=['Avg_Open_To_Buy', 'Total_Trans_Amt'], inplace=True)

plt.figure(figsize=(10,10))
g = sns.heatmap(data_copy.corr())

def class_proportions(array: np.ndarray) -> Dict[int, Tuple[int, float]]:
    """
    Вычисляет пропорции классов
    array - массив, содержащий метки классов
    """
    labels, counts = np.unique(array, return_counts=True)
    counts_perc = counts/array.size
    res = dict()
```

```
    for label, count2 in zip(labels, zip(counts, counts_perc)):
        res[label] = count2
    return res

def print_class_proportions(array: np.ndarray):
    """
    Вывод пропорций классов
    """
    proportions = class_proportions(array)
    if len(proportions)>0:
        print('Метка \t Количество \t Процент встречаемости')
    for i in proportions:
        val, val_perc = proportions[i]
        val_perc_100 = round(val_perc * 100, 2)
        print('{} \t {} \t \t {}%'.format(i, val, val_perc_100))

from imblearn.under_sampling import RandomUnderSampler

TEST_SIZE = 0.3
RANDOM_STATE = 0
SPLITS_DEFAULT = 5
CROSS_VAL = StratifiedKFold(n_splits=SPLITS_DEFAULT)

rus = RandomUnderSampler(random_state=RANDOM_STATE)
data_X, data_y = rus.fit_resample(data.drop(columns='Attrition_Flag'),  data['Attrition_Flag'])
dataD_X, dataD_y = rus.fit_resample(dataD.drop(columns='Attrition_Flag'),
dataD['Attrition_Flag'])

data_X_train, data_X_test, data_y_train, data_y_test = train_test_split(data_X, \
                                                    data_y, test_size=TEST_SIZE, \
                                                    random_state=RANDOM_STATE)
dataD_X_train, dataD_X_test, dataD_y_train, dataD_y_test = train_test_split(dataD_X, \
                                                    dataD_y, test_size=TEST_SIZE, \
                                                    random_state=RANDOM_STATE)

print_class_proportions(data_y_train)
print_class_proportions(data_y_test)

def print_metrics(X_train, Y_train, X_test, Y_test, clf):
    clf.fit(X_train, Y_train)
    target = clf.predict(X_test)
    ret = balanced_accuracy_score(Y_test, target)
    print(f'Сбалансированная оценка: {ret}')
    plot_roc_curve(clf, X_test, Y_test)
    plt.show()
    print(f'Матрица ошибок:\n {confusion_matrix(Y_test, target)}')
    print(classification_report(Y_test, target, target_names=['Existing Customer', 'Attrited
Customer']))
```

```
    return ret

def print_gridResults(grid, p):
    print(f'Подобранный параметр: {grid.best_params_}')
    print(f'Оценка при подобранном параметре: {grid.best_score_}')
    return [grid.best_params_[p], grid.best_score_]

def get_optimum(clf, dist, n_iter, data_X_train, data_y_train, scoring='balanced_accuracy'):
    Random_grid = RandomizedSearchCV(clf, dist, n_iter=n_iter, \
                                     scoring=scoring, cv=CROSS_VAL, random_state=RANDOM_STATE)
    Random_grid.fit(data_X_train, data_y_train)
    res = print_gridResults(Random_grid, list(dist.keys())[0])
    return res
```

1) Logistic Regression

```
LogReg_param = {"C":uniform(loc=0, scale=100)}  # равномерное распределение между 0 и 100
linearD = get_optimum(LogisticRegression(max_iter=10000), LogReg_param, 20, dataD_X_train, dataD_y_train)
LRd = print_metrics(dataD_X_train, dataD_y_train, dataD_X_test, dataD_y_test, LogisticRegression(C=linearD[0], max_iter=10000))

linear = get_optimum(LogisticRegression(max_iter=10000), LogReg_param, 20, data_X_train, data_y_train)
LR = print_metrics(data_X_train, data_y_train, data_X_test, data_y_test, LogisticRegression(C=linear[0], max_iter=10000))
```

2) SVM

```
SVM_param = {"C":uniform(loc=0, scale=100)}  # равномерное распределение между 0 и 100
svmD = get_optimum(SVC(random_state=RANDOM_STATE), SVM_param, 3, dataD_X_train, dataD_y_train)
SVMd = print_metrics(dataD_X_train, dataD_y_train, dataD_X_test, dataD_y_test, SVC(random_state=RANDOM_STATE, C=svmD[0]))

svm = get_optimum(SVC(random_state=RANDOM_STATE), SVM_param, 3, data_X_train, data_y_train)
SVM = print_metrics(data_X_train, data_y_train, data_X_test, data_y_test, SVC(random_state=RANDOM_STATE, C=svm[0]))
```

3) Дерево решений

```
dataL_X, dataL_y = rus.fit_resample(data_copy.drop(columns='Attrition_Flag'), data_copy['Attrition_Flag'])
dataLD_X, dataLD_y = rus.fit_resample(dataD_copy.drop(columns='Attrition_Flag'), dataD_copy['Attrition_Flag'])
dataL_X_train, dataL_X_test, dataL_y_train, dataL_y_test = train_test_split(dataL_X, dataL_y, test_size=TEST_SIZE, \
                                                    random_state=RANDOM_STATE)
```

```
dataLD_X_train, dataLD_X_test, dataLD_y_train, dataLD_y_test = train_test_split(dataLD_X,
dataLD_y, test_size=TEST_SIZE, \
                                                    random_state=RANDOM_STATE)
Tree_param = {"max_depth":randint(1, 10)}  # целочисленное распределение между 1 и 10

treeD = get_optimum(DecisionTreeClassifier(random_state=RANDOM_STATE), Tree_param,
10, dataLD_X_train, dataLD_y_train)
DTd = print_metrics(dataLD_X_train, dataLD_y_train, dataLD_X_test, dataLD_y_test,
DecisionTreeClassifier(max_depth=treeD[0], \
                                                        random_state=RANDOM_STATE))
tr_clfD = DecisionTreeClassifier(max_depth=treeD[0], random_state=RANDOM_STATE)
tr_clfD.fit(dataLD_X_train, dataLD_y_train)
dot_data = export_graphviz(tr_clfD, out_file=None,
                           feature_names=dataLD_X_train.columns.tolist(),
                           class_names=['Existing Customer', 'Attrited Customer'],
                           filled=True, rounded=True, special_characters=True)
graph = graphviz.Source(dot_data)
graph
def show_feature_importance(importance, col_names):
   data =
pd.DataFrame({'feature_names':np.array(col_names),'feature_importance':np.array(importance
)})
   data.sort_values(by=['feature_importance'], ascending=False,inplace=True)

   plt.figure(figsize=(10,7))
   sns.barplot(x=data['feature_importance'], y=data['feature_names'])
   plt.title('Feature importance using DecisionTreeClassifier')
   plt.xlabel('importance')
   plt.ylabel('name')
show_feature_importance(tr_clfD.feature_importances_, dataLD_X_train.columns.tolist())

tree = get_optimum(DecisionTreeClassifier(random_state=RANDOM_STATE), Tree_param,
10, dataL_X_train, dataL_y_train)
DT = print_metrics(dataL_X_train, dataL_y_train, dataL_X_test, dataL_y_test,
DecisionTreeClassifier(max_depth=tree[0], \
                                                        random_state=RANDOM_STATE))
tr_clf = DecisionTreeClassifier(max_depth=tree[0], random_state=RANDOM_STATE)
tr_clf.fit(dataL_X_train, dataL_y_train)
dot_data = export_graphviz(tr_clf, out_file=None,
                           feature_names=dataL_X_train.columns.tolist(),
                           class_names=['Existing Customer', 'Attrited Customer'],
                           filled=True, rounded=True, special_characters=True)
graph = graphviz.Source(dot_data)
graph
show_feature_importance(tr_clf.feature_importances_, dataL_X_train.columns.tolist())
```

4) Кривые обучения

```
def plot_learning_curve(data_X, data_y, clf):
```

```
    train_sizes, train_scores, test_scores = learning_curve(estimator=clf, X=data_X, y=data_y,
                                                    train_sizes=np.linspace(0.1, 1.0, 10), cv=5)
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)
    plt.figure(figsize=(7,5))
    plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5,
label=f'тренировочная верность')
    plt.fill_between(train_sizes, train_mean+train_std, train_mean-train_std, alpha=0.15,
color='blue')
    plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5,
            label=f'проверочная верность')
    plt.fill_between(train_sizes, test_mean+test_std, test_mean-test_std, alpha=0.15,
color='green')
    plt.grid()
    plt.legend(loc='upper right')
    plt.xlabel('Число тренировочных образцов')
    plt.ylabel('Верность')
    plt.show()

plot_learning_curve(dataLD_X_train, dataLD_y_train, tr_clfD)
plot_learning_curve(dataL_X_train, dataL_y_train, tr_clf)



lst_label_cv = ['LR', 'SVM', 'DT']
dc_score = {'Удален 1 признак':[LRd, SVMd, DTd],\
          'Удалено 2 признака':[LR, SVM, DT]}

pd.DataFrame(dc_score, index=lst_label_cv)

plot_roc_curve(LogisticRegression(C=linearD[0], max_iter=10000).fit(dataD_X_train,
dataD_y_train), \
              dataD_X_test, dataD_y_test)
plot_roc_curve(SVC(random_state=RANDOM_STATE, C=svmD[0]).fit(dataD_X_train,
dataD_y_train), \
              dataD_X_test, dataD_y_test)
plot_roc_curve(DecisionTreeClassifier(max_depth=treeD[0],
random_state=RANDOM_STATE).fit(dataLD_X_train, \
              dataLD_y_train), dataLD_X_test, dataLD_y_test)

plot_roc_curve(LogisticRegression(C=linear[0], max_iter=10000).fit(data_X_train,
data_y_train), \
              data_X_test, data_y_test)
plot_roc_curve(SVC(random_state=RANDOM_STATE, C=svm[0]).fit(data_X_train,
data_y_train), data_X_test, data_y_test)
plot_roc_curve(DecisionTreeClassifier(max_depth=tree[0],
random_state=RANDOM_STATE).fit(dataL_X_train, \
```

dataL_y_train), dataL_X_test, dataL_y_test)

3) Экранные формы с примерами выполнения программы

1. Подготовка

- Для лабораторной работы был выбран датасет [Credit Card customers](). Будем решать задачу классификации для предсказания оттока клиентов в банке, выдающем кредитные карты. Первые 5 строк и несколько столбцов набора данных имеют вид:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book
0	768805383	Existing Customer	45	M	3	High School	Married	$60K – 80K	Blue	39
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	44
2	713982108	Existing Customer	51	M	3	Graduate	Married	$80K – 120K	Blue	36
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	34
4	709106358	Existing Customer	40	M	3	Uneducated	Married	$60K – 80K	Blue	21

5 rows × 21 columns

- Состав признаков в датасете:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   CLIENTNUM                 10127 non-null  int64
 1   Attrition_Flag            10127 non-null  object
 2   Customer_Age              10127 non-null  int64
 3   Gender                    10127 non-null  object
 4   Dependent_count           10127 non-null  int64
 5   Education_Level           10127 non-null  object
 6   Marital_Status            10127 non-null  object
 7   Income_Category           10127 non-null  object
 8   Card_Category             10127 non-null  object
 9   Months_on_book            10127 non-null  int64
 10  Total_Relationship_Count  10127 non-null  int64
 11  Months_Inactive_12_mon    10127 non-null  int64
 12  Contacts_Count_12_mon     10127 non-null  int64
 13  Credit_Limit              10127 non-null  float64
 14  Total_Revolving_Bal       10127 non-null  int64
 15  Avg_Open_To_Buy           10127 non-null  float64
 16  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 17  Total_Trans_Amt           10127 non-null  int64
 18  Total_Trans_Ct            10127 non-null  int64
 19  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 20  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: float64(5), int64(10), object(6)
memory usage: 1.6+ MB
```

- Уникальные значения категориальных признаков:

```
1. Attrition_Flag: ['Existing Customer' 'Attrited Customer'] | КОЛ-ВО: 2
2. Gender: ['M' 'F'] | КОЛ-ВО: 2
3. Education_Level: ['High School' 'Graduate' 'Uneducated' 'Unknown' 'College' 'Post-Graduate'
 'Doctorate'] | КОЛ-ВО: 7
4. Marital_Status: ['Married' 'Single' 'Unknown' 'Divorced'] | КОЛ-ВО: 4
5. Income_Category: ['$60K - $80K' 'Less than $40K' '$80K - $120K' '$40K - $60K' '$120K +'
 'Unknown'] | КОЛ-ВО: 6
6. Card_Category: ['Blue' 'Gold' 'Silver' 'Platinum'] | КОЛ-ВО: 4
```

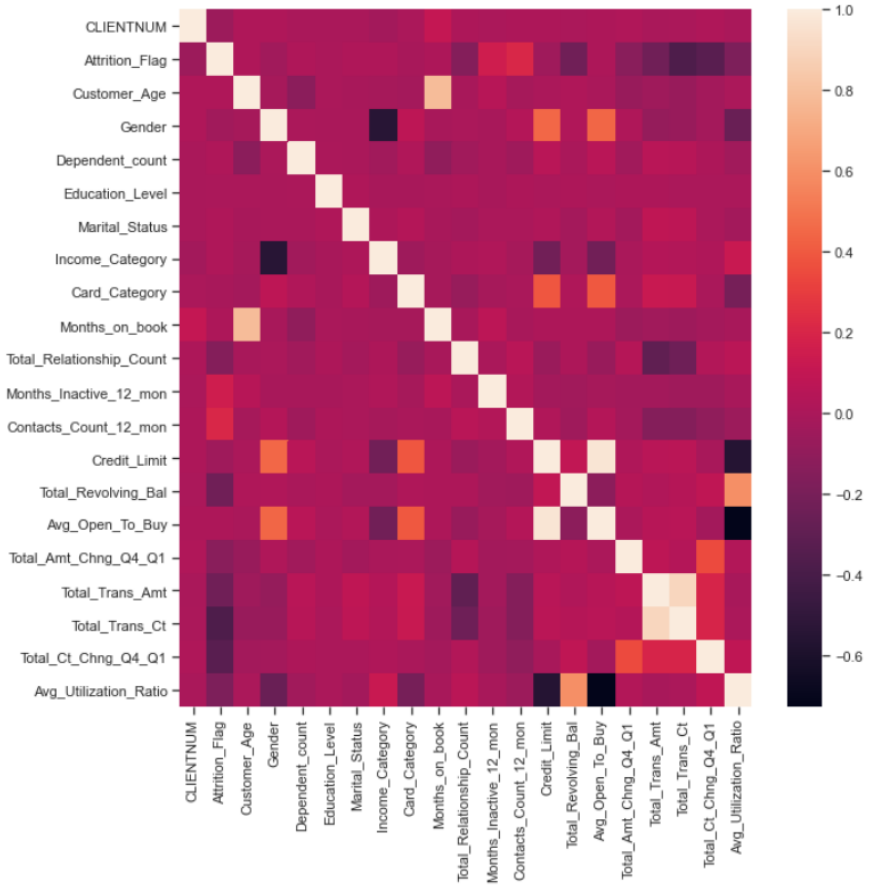
- Целевая переменная:

Нашей целевой переменной является **Attrition_Flag**: *Existing Customer* - ещё клиент, *Attrited Customer* - ушедший клиент

- Далее была произведена проверка наличия случайных ошибок в категориях датасета:

```
1. Attrition_Flag: ['Existing Customer' 'Attrited Customer'] | КОЛ-ВО: 2
2. Gender: ['M' 'F'] | КОЛ-ВО: 2
3. Education_Level: ['High School' 'Graduate' 'Uneducated' 'Unknown' 'College' 'Post-Graduate'
 'Doctorate'] | КОЛ-ВО: 7
4. Marital_Status: ['Married' 'Single' 'Unknown' 'Divorced'] | КОЛ-ВО: 4
5. Income_Category: ['$60K - $80K' 'Less than $40K' '$80K - $120K' '$40K - $60K' '$120K +'
 'Unknown'] | КОЛ-ВО: 6
6. Card_Category: ['Blue' 'Gold' 'Silver' 'Platinum'] | КОЛ-ВО: 4
```

- Нашей целевой переменной является “Attrition_Flag”: Existing Customer - ещё клиент, Attrited Customer - ушедший клиент. Кодируем: 0 – Existing Customer, 1 – Attrited Customer.
- Далее копируем DataFrame с идеей применения метода «Label Encoding», после чего использования его в классификаторе Decision Tree. В другом DataFrame для всех категориальных признаков был применен метод «one-hot encoding» для того, чтобы использовать в классификаторах Logistic Regression и SVM.
- Также выполняем масштабирование признаков второго DataFrame методом «Standard Scaler» с идеей получения более хороших оценок accuracy в SVM.
- Из корреляционной матрицы видим сильную корреляцию между отдельными признаками: «Credit_Limit» и «Avg_Open_To_Buy», а также "Total_Trans_Amt" и "Total_Trans_Ct".



- Корреляция признаков с целевым:

```
Attrition_Flag              1.000000
Contacts_Count_12_mon       0.204491
Months_Inactive_12_mon      0.152449
Dependent_count             0.018991
Marital_Status              0.018597
Customer_Age                0.018227
Income_Category             0.017584
Months_on_book              0.013687
Avg_Open_To_Buy             0.013000
Education_Level             0.005551
Card_Category              -0.006038
Gender                     -0.037272
Credit_Limit               -0.043096
CLIENTNUM                  -0.059369
Total_Amt_Chng_Q4_Q1       -0.128713
Total_Relationship_Count   -0.150005
Avg_Utilization_Ratio      -0.177846
Total_Revolving_Bal        -0.231116
Total_Trans_Amt            -0.231827
Total_Ct_Chng_Q4_Q1        -0.324012
Total_Trans_Ct             -0.371429
Name: Attrition_Flag, dtype: float64
```

- В дальнейшем в результате анализа корреляции коллинеарных признаков с целевым признаком были приняты следующие решения:

- Среди двух сильно коррелирующих признаков "Credit_Limit" (кредитный лимит по кредитной карте) и "Avg_Open_To_Buy" (Разница между кредитным лимитом, присвоенным счету держателя карты, и текущим остатком на счете (в среднем за последние 12 месяцев)) **был удален признак "Avg_Open_To_Buy"**, так как он сильнее коррелирует с другими признаками объектов и слабее коррелирует с целевым признаком
- Первый случай (удален 1 признак): среди двух сильно коррелирующих признаков "Total_Trans_Amt" (общая сумма транзакций за 12 мес.) и "Total_Trans_Ct" (общее количество транзакций за 12 месяцев) **был удален признак "Total_Trans_Amt"**, так как он сильнее коррелирует с остальными признаками объектов и слабее коррелирует с целевым признаком.
- Второй случай (удалено 2 признака): среди двух сильно коррелирующих признаков "Total_Trans_Amt" (общая сумма транзакций за 12 мес.) и "Total_Trans_Ct" (общее количество транзакций за 12 месяцев) **были оставлены оба**, так как они оба достаточно сильно коррелируют с целевым признаком.

- Было замечено, что целевой признак имеет несбалансированные классы. С целью устранения этой проблемы был применен метод «under sampling» так, что пропорции классов приобрели следующий вид:

```
print_class_proportions(data_y_train)
```

```
Метка    Количество      Процент встречаемости
0        1113            48.88%
1        1164            51.12%
```

```
print_class_proportions(data_y_test)
```

```
Метка    Количество      Процент встречаемости
0        514             52.61%
1        463             47.39%
```

2. Logistic Regression (удален 1 признак)

- Выборка была поделена на обучающую (на ней будет выполнен подбор гиперпараметра C) и отложенную (на ней будет замерена

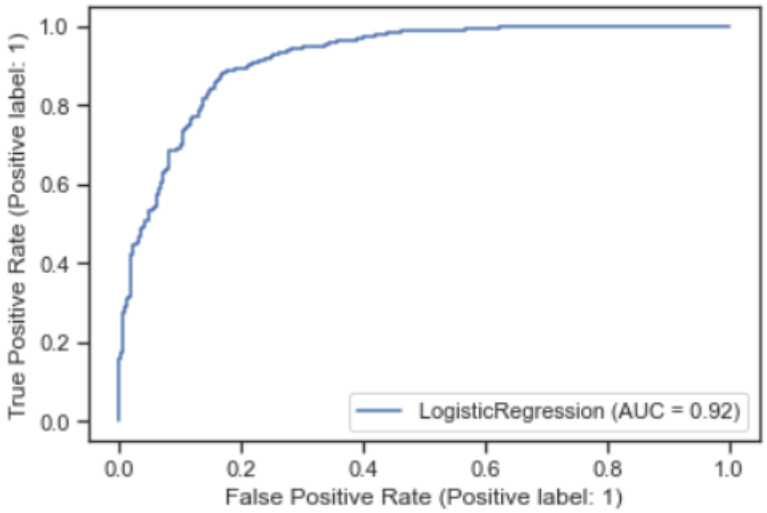
итоговая accuracy при полученном гиперпараметре C после его подбора с помощью решетчатого поиска и кросс-валидации) в отношении 80% против 20%

- При использовании кросс-валидации и Randomized Grid Search был подобран гиперпараметр и получена оценка:

```
Подобранный параметр: {'C': 71.51893663724195}
Оценка при подобранном параметре: 0.8443274807428306
```

- После итоговой оценки по оставленной выборке:

```
Сбалансированная оценка: 0.8512261431536838
```



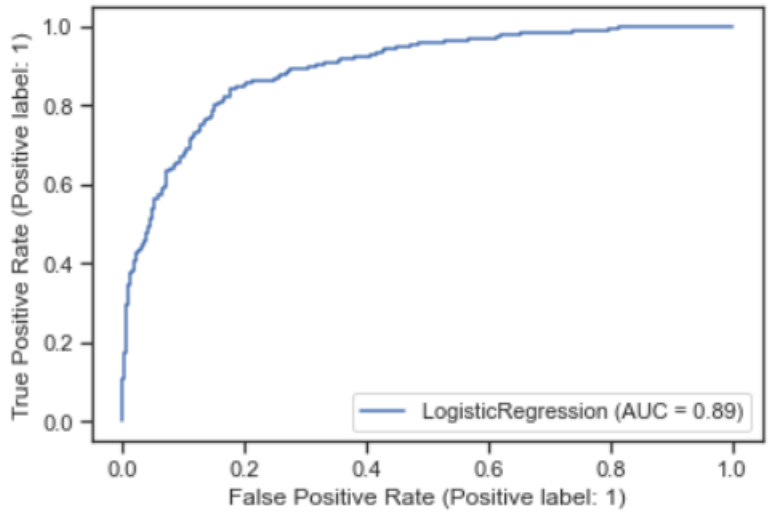

```
Матрица ошибок:
 [[431  83]
 [ 63 400]]
                  precision    recall  f1-score   support

Existing Customer       0.87      0.84      0.86       514
Attrited Customer       0.83      0.86      0.85       463

         accuracy                           0.85       977
        macro avg       0.85      0.85      0.85       977
     weighted avg       0.85      0.85      0.85       977
```

3. Logistic Regression (удалено 2 признака)

- Аналогично получаем следующие результаты при удалении двух признаков:

```
Подобранный параметр: {'C': 38.34415188257777}
Оценка при подобранном параметре: 0.8222731019052523
```

```
Сбалансированная оценка: 0.824978359707877
```




```
Матрица ошибок:
 [[424  90]
 [ 81 382]]
                   precision    recall  f1-score   support

Existing Customer       0.84      0.82      0.83       514
Attrited Customer       0.81      0.83      0.82       463

         accuracy                           0.82       977
        macro avg       0.82      0.82      0.82       977
     weighted avg       0.83      0.82      0.83       977
```

Видно, что оценка ухудшилась.

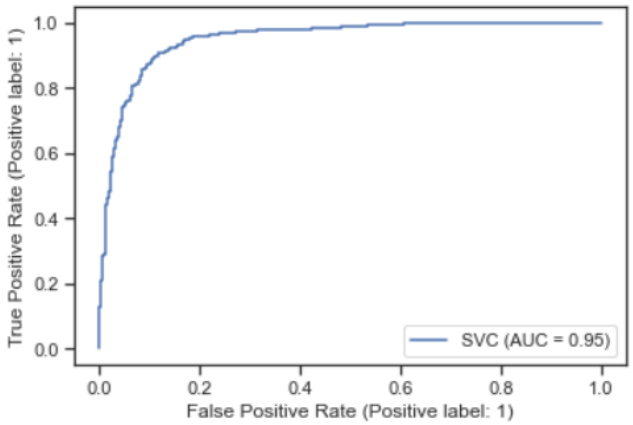
4. SVM (удален 1 признак)

- После подбора гиперпараметра, отвечающего за регуляризацию модели, рандомным решетчатым поиском с применением кросс-валидации получаем следующий результат:

```
Подобранный параметр: {'C': 54.88135039273247}
Оценка при подобранном параметре: 0.869575441397712
```

- После итоговой оценки по оставленной выборке:

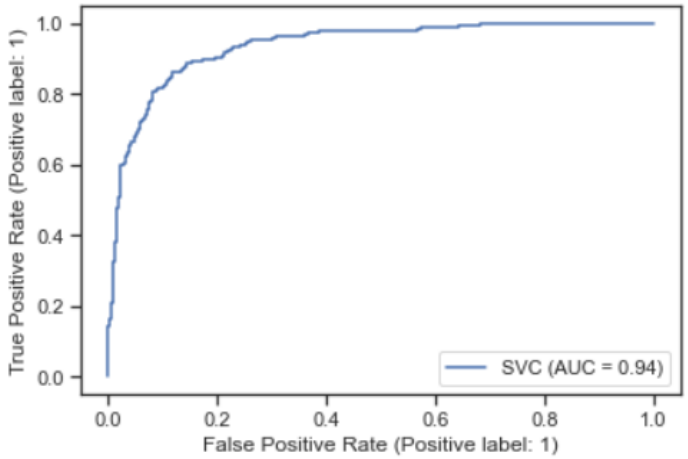
```
Сбалансированная оценка: 0.893145279895118
```




```
Матрица ошибок:
 [[453  61]
 [ 44 419]]
                   precision    recall  f1-score   support

Existing Customer       0.91      0.88      0.90       514
Attrited Customer       0.87      0.90      0.89       463

         accuracy                           0.89       977
        macro avg       0.89      0.89      0.89       977
     weighted avg       0.89      0.89      0.89       977
```

5. SVM (удалено 2 признака)

```
Подобранный параметр: {'C': 54.88135039273247}
Оценка при подобранном параметре: 0.8471342397891911

Сбалансированная оценка: 0.8680761570202788
```




```
Матрица ошибок:
 [[435  79]
 [ 51 412]]
                   precision    recall  f1-score   support

Existing Customer       0.90      0.85      0.87       514
Attrited Customer       0.84      0.89      0.86       463

         accuracy                           0.87       977
        macro avg       0.87      0.87      0.87       977
     weighted avg       0.87      0.87      0.87       977
```

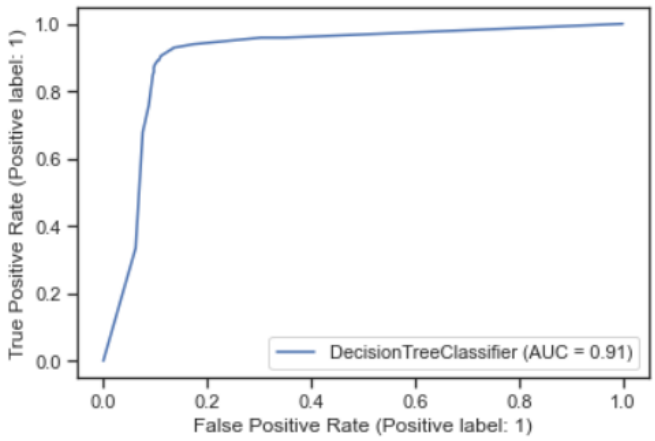
6. Дерево решений (удален 1 признак)

- Подбор гиперпараметра «максимальной глубины дерева» дает следующий результат:

```
Подобранный параметр: {'max_depth': 8}
Оценка при подобранном параметре: 0.9002867748031624
```

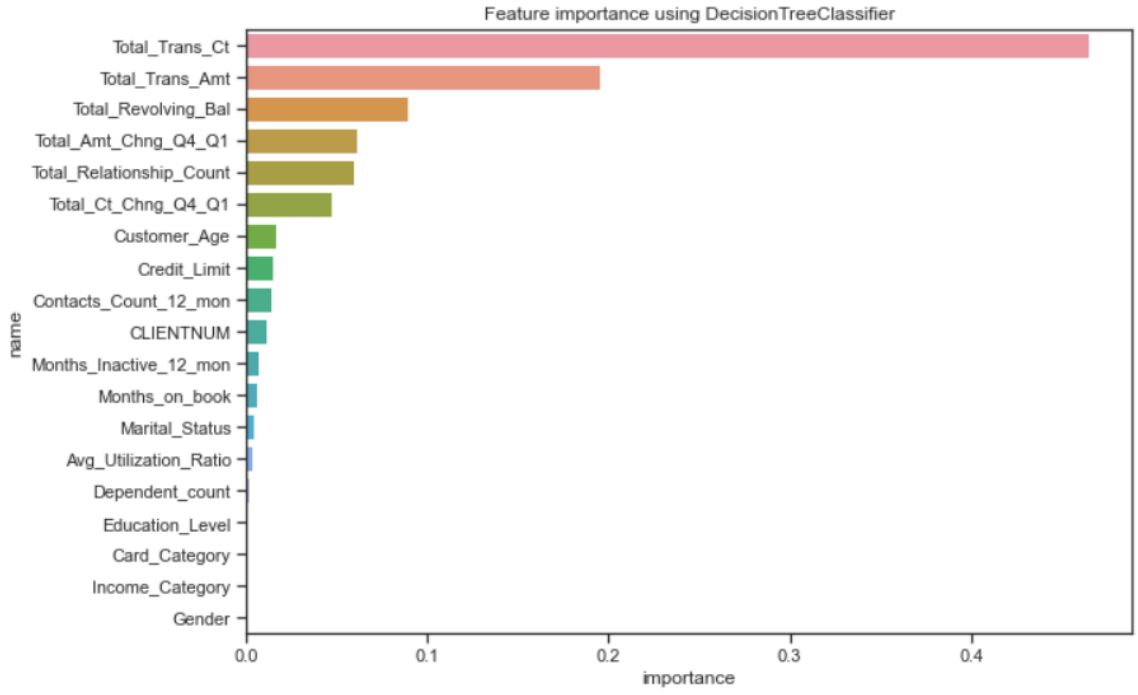
- После итоговой оценки по оставленной выборке:

```
Сбалансированная оценка: 0.8970363304787756
```




```
Матрица ошибок:
 [[457  57]
 [ 44 419]]
                   precision    recall  f1-score   support

Existing Customer       0.91      0.89      0.90       514
Attrited Customer       0.88      0.90      0.89       463

         accuracy                           0.90       977
        macro avg       0.90      0.90      0.90       977
     weighted avg       0.90      0.90      0.90       977
```

- Получена следующая важность признаков:



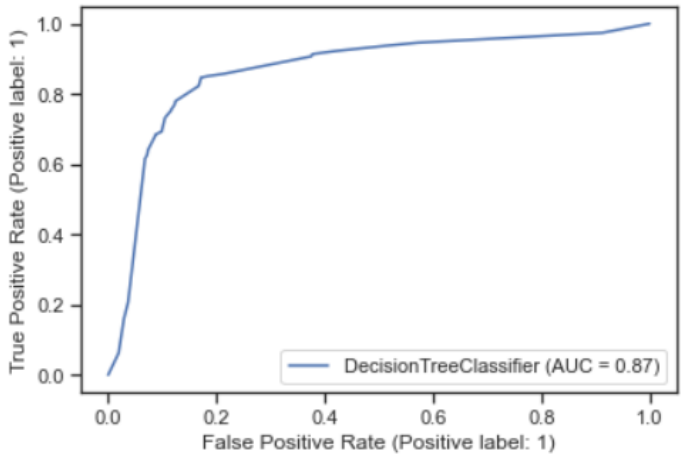
7. Дерево решений (удалено 2 признака)

- Подбор гиперпараметра «максимальной глубины дерева» дает следующий результат:

```
Подобранный параметр: {'max_depth': 6}
Оценка при подобранном параметре: 0.8439906922693083
```

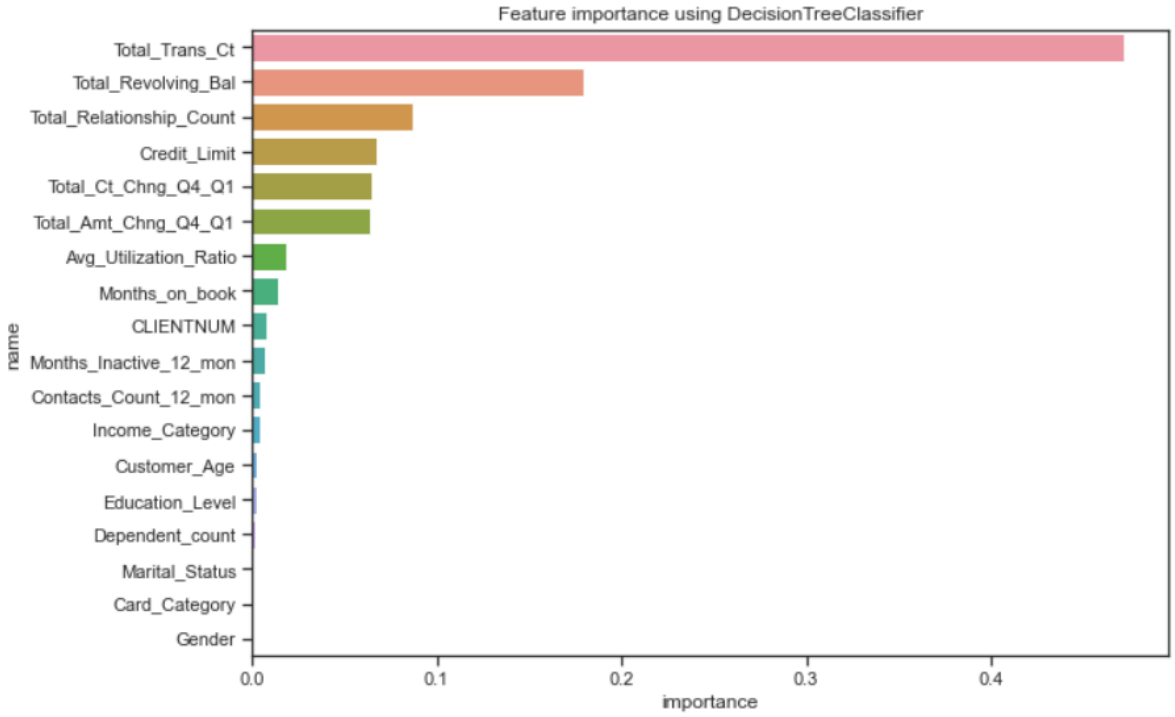
- После итоговой оценки по оставленной выборке:

```
Сбалансированная оценка: 0.8231651973678682
```



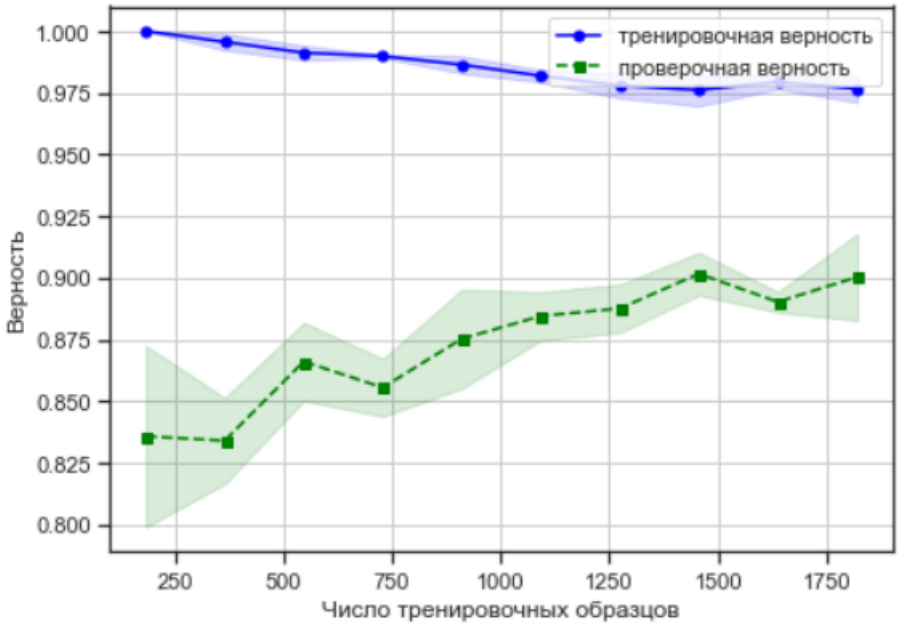

```
Матрица ошибок:
 [[451  63]
 [107 356]]
                  precision    recall  f1-score   support

Existing Customer       0.81      0.88      0.84       514
Attrited Customer       0.85      0.77      0.81       463

         accuracy                           0.83       977
        macro avg       0.83      0.82      0.82       977
     weighted avg       0.83      0.83      0.83       977
```

- Получена следующая важность признаков:

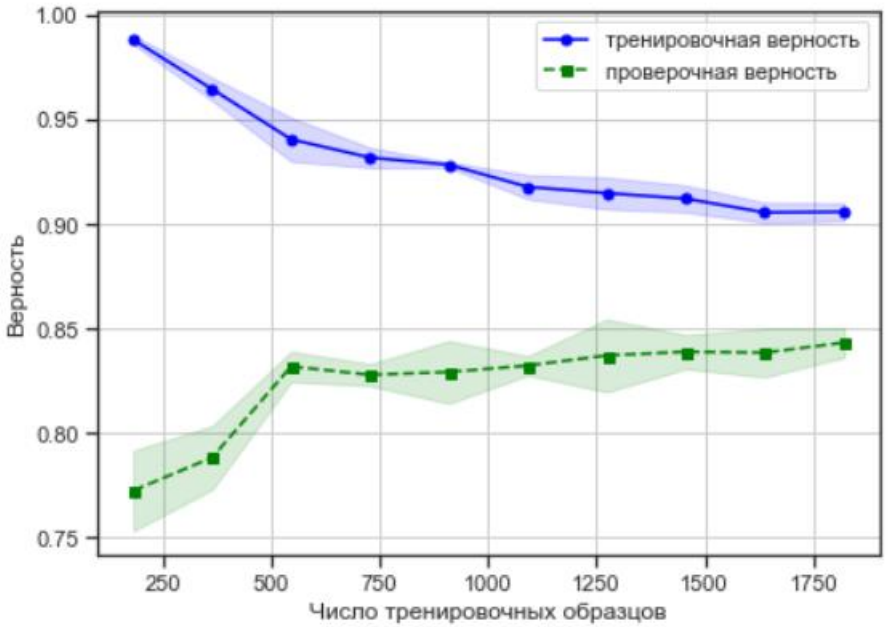


8. Кривые обучения для Decision Tree Classifier

- Кривая обучения с удалением одного признака:



- Кривая обучения с удалением двух признаков:

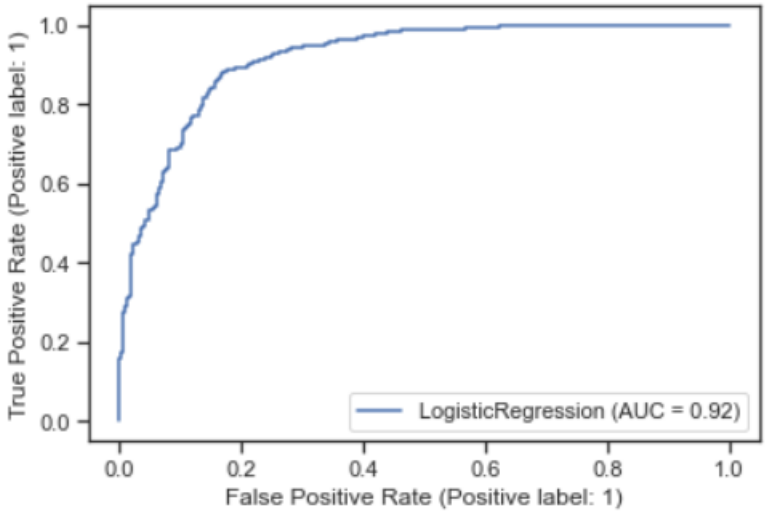


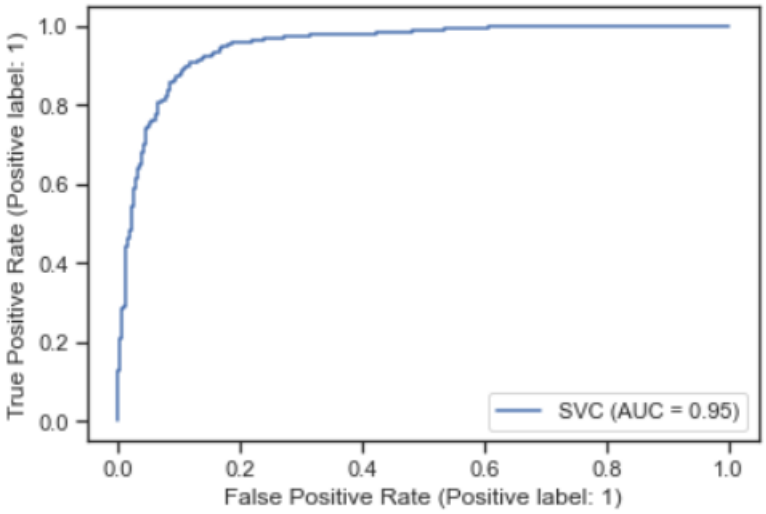
9. Итоги

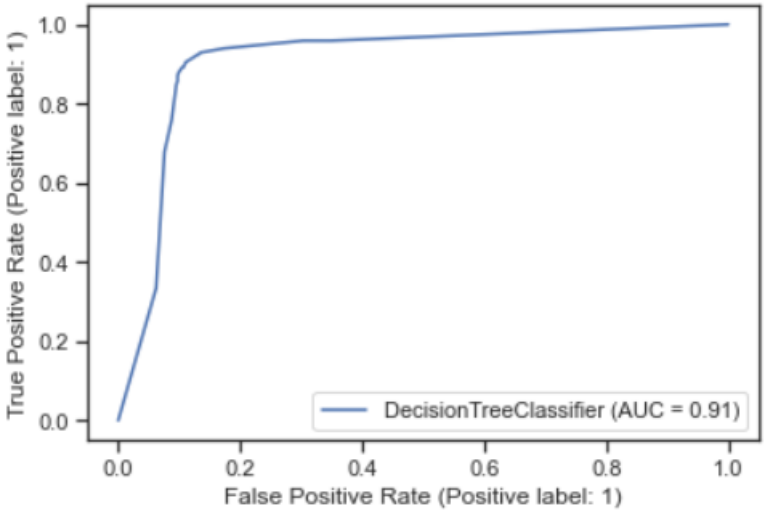
- Метрика «balanced_accuracy»:

	Удален 1 признак	Удалено 2 признака
LR	0.851226	0.824978
SVM	0.893145	0.868076
DT	0.897036	0.823165

- Метрика «ROC-кривая» для LR, SVM, DT соответственно (удален 1 признак):







- Метрика «ROC-кривая» для LR, SVM, DT соответственно (удалено 2 признака):