**Московский государственный технический университет им. Н.Э.Баумана**

Утверждаю:

Гапанюк Ю.Е.

"__"_____2019 г.

# Курсовая работа по курсу “Технологии машинного обучения” “Классификация”

Вариант №18

Пояснительная записка
(вид документа)

писчая бумага
(вид носителя)

13
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-63 _____

Федорова А.А.

"__"_____2019 г.

Москва – 2019

# Содержание

# Задание

В данной курсовой работе необходимо предпринять следующие шаги:

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.

5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9. Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется постройение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

Приведенная схема исследования является рекомендуемой. Возможно выполнение курсовой работы на нестандартную тему, которая должна быть предварительно согласована с ответственным за прием курсовой работы.

# Введение

Курсовая работа – самостоятельная часть учебной дисциплины «Технологии машинного обучения» – учебная и практическая исследовательская студенческая работа, направленная на решение комплексной задачи машинного обучения. Результатом курсовой работы является отчет, содержащий описания моделей, тексты программ и результаты экспериментов.

Курсовая работа опирается на знания, умения и владения, полученные студентом в рамках лекций и лабораторных работ по дисциплине.

В рамках данной курсовой работы необходимо применить навыки, полученные в течение курса «Технологии машинного обучения», и обосновать полученные результаты.

# Основная часть

## *Постановка задачи*

Сейчас у каждого более-менее известного банковского продукта есть свое приложение, либо сайт. С помощью данных сайтов и приложений можно грамотно выстраивать рекламу внутри своего продукта. Для того, чтобы реклама предлагалась действительно грамотно, необходимо понимать, какие у нас есть пользователи, и что может заинтересовать каждого из них.

Чтобы разобраться, что может заинтересовать пользователя, можно воспользоваться какими-то паттернами поведения всех когорт пользователей нашего приложения, а потом направлено для каждой когорты предлагать определенную рекламу. Для этого необходимо собирать определенную информацию о каждом пользователе. Например, это могут быть такие данные, как: возраст, пол, город проживания, семейное положение, профессия и многое другое. Но и этого недостаточно. Так как предполагается, что в приложении имеется доступ к пользовательским действиям, мы можем знать, какими функциями нашего продукта пользуется еще наш клиент. В таком случае можем смотреть, какие еще функции выбирают клиенты, которые используют те или иные функции, и предлагать их другим клиентам направлено.

Пока что у меня нет доступа к реальным данным, поэтому я взяла дотаяет, с помощью которого в теории можно решить данную задачу.

## *Описание выбранного датасета*

**Для выполнения курсовой работы я выбрала датасет "Black Friday"**
https://www.kaggle.com/mehdidag/black-friday
Набор данных представляет собой образец транзакций, совершенных в розничном магазине. В частности, здесь проблема заключается в проблеме регрессии, когда мы пытаемся предсказать зависимую переменную (сумму покупки) с помощью информации, содержащейся в других переменных. Так как данные предоставлены " Analytics Vidhya" - часть из них представляет закодированные данные.
Также в этом датасете может быть решена проблема классификации. Такая задача мне показалась интереснее и у меня уже есть идеи, куда эту модель можно использовать с пользой.
Данный датасет содержит следующие поля:
User_ID - уникальный идентификатор каждого покупателя (в модели не используется)
Product_ID - уникальный идентификатор каждого товара (в модели не используется)
Gender - пол покупателя (M/F)
Age - возраст покупателя - дается в виде промежутков возрастов
Occupation - код рода деятельности каждого покупателя
City_Category - категория города, в котором проживает покупатель (данные также закодированы)
Stay_In_Current_City_Years - сколько лет покупатель прожил в данном городе (делится на промежутки 0, 1, 2, 3, 4+ лет)

Marital_Status- семейное положение (0/1)
Product_Category_(2,3) - категория, которой принадлежит товар. Один и тот же товар может принадлежать сразу к нескольким категориям.(В моем случае под этим подразумеваются функции продукта, которыми пользуется наш клиент)
Purchase - стоимость покупки в долларах
Product_Category_1 - категория товара. Я выбрала ее целевым признаком. В моем случае это можно рассмотреть, как id рекламы, которую мы хотим предсказать.

## Текст программы с примерами выполнения программы:

```
#Импорт библиотек
from google.colab import drive
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from prettytable import PrettyTable
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score, recall_score
```

```
#Монтирую гугл диск, чтобы взять оттуда датасет
drive.mount("/content/gdrive", force_remount=True)
```

```
#Загружаю данные с гугл диска
data = pd.read_csv('/content/gdrive/My Drive/BlackFriday.csv', sep=",")
```

```
data.head(5)
```

| _ID | Gender | Age | Occupation | City_Category | Stay_In_Current_City_Years | Marital_Status | Product_Category_1 | Product_Category_2 |
|---|---|---|---|---|---|---|---|---|
| 9042 | F | 0-17 | 10 | A | 2 | 0 | 3 | NaN |
| 3942 | F | 0-17 | 10 | A | 2 | 0 | 1 | 6.0 |
| 7842 | F | 0-17 | 10 | A | 2 | 0 | 12 | NaN |
| 5442 | F | 0-17 | 10 | A | 2 | 0 | 12 | 14.0 |

```
data.shape
```

```
(537577, 12)
```

```
data.dtypes
```

```
User_ID                       int64
Product_ID                   object
Gender                       object
Age                          object
Occupation                    int64
City_Category                object
Stay_In_Current_City_Years   object
Marital_Status                int64
Product_Category_1            int64
Product_Category_2          float64
Product_Category_3          float64
Purchase                      int64
```

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

```
User_ID                          0
Product_ID                       0
Gender                           0
Age                              0
Occupation                       0
City_Category                    0
Stay_In_Current_City_Years       0
Marital_Status                   0
Product_Category_1               0
Product_Category_2          166986
Product_Category_3          373299
Purchase                         0
dtype: int64
```

```
data['Product_Category_3'] = imp.fit_transform(data[['Product_Category_3']])
data.isnull().sum()
```

```
User_ID                     0
Product_ID                  0
Gender                      0
Age                         0
Occupation                  0
City_Category               0
Stay_In_Current_City_Years  0
Marital_Status              0
Product_Category_1          0
Product_Category_2          0
Product_Category_3          0
Purchase                    0
```

```
print(data['Gender'].unique(), data['Age'].unique(), data['City_Category'].unique(),
data['Stay_In_Current_City_Years'].unique())
```

```
['F' 'M'] ['0-17' '55+' '26-35' '46-50' '51-55' '36-45' '18-25'] ['A' 'C' 'B'] ['2' '4+' '3' '1' '0']
```

```
le = LabelEncoder()
data['Gender'] = le.fit_transform(data[['Gender']])
data['Age'] = le.fit_transform(data[['Age']])
data['City_Category'] = le.fit_transform(data[['City_Category']])
data['Stay_In_Current_City_Years'] = le.fit_transform(data[['Stay_In_Current_City_Years']])
```

```
print(data['Gender'].unique(), data['Age'].unique(), data['City_Category'].unique(),
data['Stay_In_Current_City_Years'].unique())
```

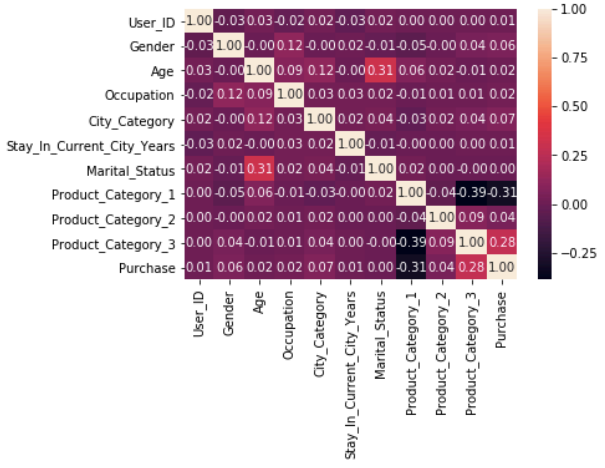
```
[0 1] [0 6 2 4 5 3 1] [0 2 1] [2 4 3 1 0]
```

```
data.dtypes
```

```
User_ID                      int64
Product_ID                  object
Gender                       int64
Age                          int64
Occupation                   int64
City_Category                int64
Stay_In_Current_City_Years   int64
Marital_Status               int64
Product_Category_1           int64
Product_Category_2         float64
Product_Category_3         float64
Purchase                     int64
```

```
sns.heatmap(data.corr(method='pearson'), annot=True, fmt='.2f', square=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd735abc208>
```



В итоге мной были выбраны следующие фичи для последующей работы с ними:

```
cols_x = ['Gender', 'Age', 'Occupation', 'City_Category',
          'Stay_In_Current_City_Years', 'Marital_Status',
          'Product_Category_2', 'Product_Category_3', 'Purchase', ]
# целевой признак
col_y = 'Product_Category_1'
#разделение данных
X_train, X_test, y_train, y_test = train_test_split(data[cols_x][0:50000], data[col_y][0:50000], test_size = 0.5, random_state = 11)
N_train, _ = X_train.shape
N_test,  _ = X_test.shape
print (N_train, N_test)
```

```
25000 25000
```

**Функция перевода предсказанного признака в бинарный**

```
def convert_target_to_binary(array:np.ndarray, target:int) -> np.ndarray:
    # Если целевой признак совпадает с указанным, то 1 иначе 0
    res = [1 if x==target else 0 for x in array]
    return res
```

```
# Если целевой признак == 8,
# то будем считать этот случай 1 в бинарном признаке
bin_y_test = convert_target_to_binary(y_test, 8)
list(zip(y_test, bin_y_test))[0:15]
```

```
[(5, 0),
 (2, 0),
 (5, 0),
 (11, 0),
 (1, 0),
 (1, 0),
 (6, 0),
 (11, 0),
 (8, 1),
 (4, 0),
 (4, 0),
 (16, 0),
 (1, 0),
 (8, 1),
 (8, 1)]
```

# Модели без подобранных параметров

## K-ближайших соседей

```
KNeighborsClassifierObj = KNeighborsClassifier(n_neighbors = 3)
```

```
KNeighborsClassifierObj.fit(X_train, y_train)
target_knei = KNeighborsClassifierObj.predict(X_test)
```

```
# Конвертация предсказанных признаков
bin_target_knei = convert_target_to_binary(target_knei, 8)
```

```
ac_knei = accuracy_score(y_test, target_knei)
bas_knei = balanced_accuracy_score(bin_y_test, bin_target_knei)
rec_knei = recall_score(bin_y_test, bin_target_knei)
print('''accuracy_score: {0}
balanced_accuracy_score: {1}
recall_score: {2}'''.format(ac_knei, bas_knei, rec_knei))
```

```
accuracy_score: 0.87892
balanced_accuracy_score: 0.9446752419437592
recall_score: 0.9306682116647355
```

## Градиентный спуск

```
grad = SGDClassifier().fit(X_train, y_train)
target_grad = grad.predict(X_test)
```

```
# Конвертация предсказанных признаков
bin_target_grad = convert_target_to_binary(target_grad, 8)
```

```
ac_grad = accuracy_score(y_test, target_grad)
bas_grad = balanced_accuracy_score(bin_y_test, bin_target_grad)
rec_grad = recall_score(bin_y_test, bin_target_grad)
print('''accuracy_score: {0}
balanced_accuracy_score: {1}
recall_score: {2}'''.format(ac_grad, bas_grad, rec_grad))
```

```
accuracy_score: 0.04544
balanced_accuracy_score: 0.4987363880755799
recall_score: 0.028196214754731556
```

### Дерево принятия решений

```
[ ]  dtcl = DecisionTreeClassifier().fit(X_train, y_train)
     target_dtc = dtcl.predict(X_test)
```

```
[ ]  # Конвертация предсказанных признаков
     bin_target_dtc = convert_target_to_binary(target_dtc, 8)
```

```
[ ]  ac_dtc = accuracy_score(y_test, target_dtc)
     bas_dtc = balanced_accuracy_score(bin_y_test, bin_target_dtc)
     rec_dtc = recall_score(bin_y_test, bin_target_dtc)
     print('''accuracy_score: {0}
     balanced_accuracy_score: {1}
     recall_score: {2}'''.format(ac_dtc, bas_dtc, rec_dtc))
```

```
accuracy_score: 0.90616
balanced_accuracy_score: 0.9408931650827164
recall_score: 0.9036307454615682
```

### Модель бэггинга

```
[ ]  bag = BaggingClassifier(DecisionTreeClassifier(random_state=1), n_estimators=10).fit(X_train, y_train)
     target_bag = bag.predict(X_test)
```

```
[ ]  # Конвертация предсказанных признаков
     bin_target_bag = convert_target_to_binary(target_bag, 8)
```

```
[ ]  ac_bag = accuracy_score(y_test, target_bag)
     bas_bag = balanced_accuracy_score(bin_y_test, bin_target_bag)
     rec_bag = recall_score(bin_y_test, bin_target_bag)
     print('''accuracy_score: {0}
     balanced_accuracy_score: {1}
     recall_score: {2}'''.format(ac_bag, bas_bag, rec_bag))
```

```
accuracy_score: 0.92584
balanced_accuracy_score: 0.9647756009558848
recall_score: 0.9544225569718038
```

### Случайный лес

```
[ ]  forest = RandomForestClassifier(random_state=1, max_features = 2).fit(X_train, y_train)
     target_forest = forest.predict(X_test)
```

```
[ ]  # Конвертация предсказанных признаков
     bin_target_forest = convert_target_to_binary(target_forest, 8)
```

```
[ ]  ac_forest = accuracy_score(y_test, target_forest)
     bas_forest = balanced_accuracy_score(bin_y_test, bin_target_forest)
     rec_forest = recall_score(bin_y_test, bin_target_forest)
     print('''accuracy_score: {0}
     balanced_accuracy_score: {1}
     recall_score: {2}'''.format(ac_forest, bas_forest, rec_forest))
```

```
accuracy_score: 0.79264
balanced_accuracy_score: 0.8888389205450331
recall_score: 0.8346852066434917
```

## Подбор гиперпараметров с использованием GridSearchCV и кросс-валидации

### K- ближайших соседей

```
[ ]  n_range_knei = np.array(range(1,30,1))
     tuned_parameters_knei = [{'n_neighbors': n_range_knei}]
     tuned_parameters_knei
```

```
[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])}]
```

Из-за большого объема данных, подбор параметров проходит не на всем датасете, а только на его части

```
knei_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters_knei, cv=KFold(n_splits=5), scoring= 'accuracy')
knei_gs.fit(data[cols_x][0:10000], data[col_y][0:10000])
```

```
knei_gs.best_params_
```

```
{'n_neighbors': 5}
```

**Обучение с подобранным параметром**

```
knei_gs.best_estimator_.fit(X_train, y_train)
target_knei_gs = knei_gs.best_estimator_.predict(X_test)
```

**Проверка качества модели**

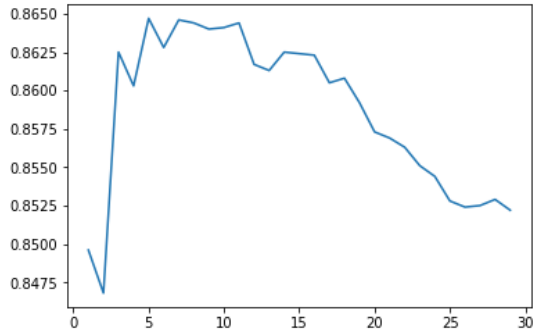
```
# Конвертация предсказанных признаков
bin_target_knei_gs = convert_target_to_binary(target_knei_gs, 8)
```

```
ac_knei_gs = accuracy_score(y_test, target_knei_gs)
bas_knei_gs = balanced_accuracy_score(bin_y_test, bin_target_knei_gs)
rec_knei_gs = recall_score(bin_y_test, bin_target_knei_gs)
print('''accuracy_score          С gs: {0}, без gs: {3}
balanced_accuracy_score С gs: {1}, без gs: {4}
recall_score            С gs: {2}, без gs: {5}'''.format(round(ac_knei_gs,3),
round(bas_knei_gs, 3), round(rec_knei_gs, 3), round(ac_knei, 3),
round(bas_knei, 3), round(rec_knei, 3)))
```

```
accuracy_score          С gs: 0.878, без gs: 0.879
balanced_accuracy_score С gs: 0.952, без gs: 0.945
recall_score            С gs: 0.949, без gs: 0.931
```

```
# Изменение качества на обучающей выборке
plt.plot(n_range_knei, knei_gs.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7fd73034bcf8>]
```




## Градиентный спуск

```
n_range_grad = np.array(np.arange(0.0001,1,0.05))
tuned_parameters = [{'alpha': n_range_grad}]
tuned_parameters
```

```
[{'alpha': array([1.000e-04, 5.010e-02, 1.001e-01, 1.501e-01, 2.001e-01, 2.501e-01,
        3.001e-01, 3.501e-01, 4.001e-01, 4.501e-01, 5.001e-01, 5.501e-01,
        6.001e-01, 6.501e-01, 7.001e-01, 7.501e-01, 8.001e-01, 8.501e-01,
        9.001e-01, 9.501e-01])}]
```

```
grad_gs = GridSearchCV(SGDClassifier(), tuned_parameters, cv=KFold(n_splits=5), scoring='accuracy')
grad_gs.fit(data[cols_x][0:1000], data[col_y][0:1000])
grad_gs.best_params_
```

```
{'alpha': 0.10010000000000001}
```

**Обучение с подобранным параметром**

```
grad_gs.best_estimator_.fit(X_train, y_train)
target_grad_gs = grad_gs.best_estimator_.predict(X_test)
```

#### Проверка качества модели

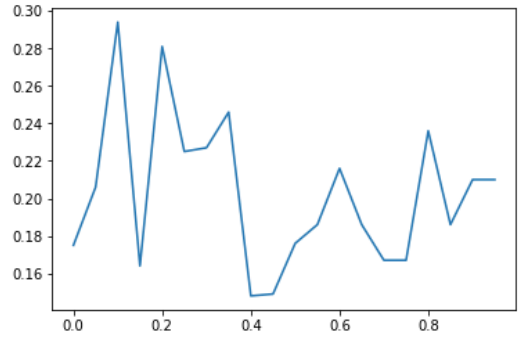
```
[ ]  # Конвертация предсказанных признаков
     bin_target_grad_gs = convert_target_to_binary(target_grad_gs, 8)
```

```
[ ]  ac_grad_gs = accuracy_score(y_test, target_grad_gs)
     bas_grad_gs = balanced_accuracy_score(bin_y_test, bin_target_grad_gs)
     rec_grad_gs = recall_score(bin_y_test, bin_target_grad_gs)
     print('''accuracy_score          C gs: {0}, без gs: {3}
     balanced_accuracy_score C gs: {1}, без gs: {4}
     recall_score            C gs: {2}, без gs: {5}'''.format(round(ac_grad_gs,3),
     round(bas_grad_gs, 3), round(rec_grad_gs, 3), round(ac_grad, 3),
     round(bas_grad, 3), round(rec_grad, 3)))
```

```
accuracy_score          C gs: 0.363, без gs: 0.045
balanced_accuracy_score C gs: 0.5, без gs: 0.499
recall_score            C gs: 0.0, без gs: 0.028
```

```
[ ]  # Изменение качества на обучающей выборке
     plt.plot(n_range_grad, grad_gs.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7fd73018f0b8>]
```




## Дерево принятия решений

```
[ ]  n_range_dtc = np.array(range(1,1000,30))
     tuned_parameters_dtc = [{'max_depth': n_range_dtc}]
     tuned_parameters_dtc
```

```
[{'max_depth': array([  1,  31,  61,  91, 121, 151, 181, 211, 241, 271, 301, 331, 361,
         391, 421, 451, 481, 511, 541, 571, 601, 631, 661, 691, 721, 751,
         781, 811, 841, 871, 901, 931, 961, 991])}]
```

```
[ ]  dtc_gs = GridSearchCV(DecisionTreeClassifier(), tuned_parameters_dtc, cv=KFold(n_splits=10), scoring= 'accuracy')
     dtc_gs.fit(data[cols_x][0:100000], data[col_y][0:100000])
     dtc_gs.best_params_
```

```
{'max_depth': 991}
```

#### Обучение с подобранным параметром

```
[ ]  dtc_gs.best_estimator_.fit(X_train, y_train)
     target_dtc_gs = dtc_gs.best_estimator_.predict(X_test)
```

#### Проверка качества модели

```
[ ]  # Конвертация предсказанных признаков
     bin_target_dtc_gs = convert_target_to_binary(target_dtc_gs, 8)
```

```
[ ]  ac_dtc_gs = accuracy_score(y_test, target_dtc_gs)
     bas_dtc_gs = balanced_accuracy_score(bin_y_test, bin_target_dtc_gs)
     rec_dtc_gs = recall_score(bin_y_test, bin_target_dtc_gs)
     print('''accuracy_score          C gs: {0}, без gs: {3}
     balanced_accuracy_score C gs: {1}, без gs: {4}
     recall_score            C gs: {2}, без gs: {5}'''.format(round(ac_dtc_gs,3),
     round(bas_dtc_gs, 3), round(rec_dtc_gs, 3), round(ac_dtc, 3),
     round(bas_dtc, 3), round(rec_dtc, 3)))
```

```
accuracy_score          C gs: 0.908, без gs: 0.906
balanced_accuracy_score C gs: 0.94, без gs: 0.941
recall_score            C gs: 0.902, без gs: 0.904
```

## Случайный лес

```
[ ]  n_range_forest = np.array(range(1,1000,30))
     tuned_parameters_forest = [{'max_depth': n_range_forest}]
     tuned_parameters_forest
```

```
[{'max_depth': array([  1,  31,  61,  91, 121, 151, 181, 211, 241, 271, 301, 331, 361,
         391, 421, 451, 481, 511, 541, 571, 601, 631, 661, 691, 721, 751,
         781, 811, 841, 871, 901, 931, 961, 991])}]
```

```
[ ]  forest_gs = GridSearchCV(RandomForestClassifier(), tuned_parameters_forest, cv=KFold(n_splits=5), scoring='accuracy')
     forest_gs.fit(data[cols_x][0:10000], data[col_y][0:10000])
     forest_gs.best_params_
```

### Обучение с подобранным параметром

```
[ ]  forest_gs.best_estimator_.fit(X_train, y_train)
     target_forest_gs = bag_gs.best_estimator_.predict(X_test)
```

### Проверка качества модели

CODE TEXT

```
[ ]  # Конвертация предсказанных признаков
     bin_target_forest_gs = convert_target_to_binary(target_forest_gs, 6)
```

```
[ ]  ac_forest_gs = accuracy_score(y_test, target_forest_gs)
     bas_forest_gs = balanced_accuracy_score(bin_y_test, bin_target_forest_gs)
     rec_forest_gs = recall_score(bin_y_test, bin_target_forest_gs)
     print('''accuracy_score          C gs: {0}, без gs: {3}
     balanced_accuracy_score C gs: {1}, без gs: {4}
     recall_score            C gs: {2}, без gs: {5}'''.format(round(ac_forest_gs,3),
     round(bas_forest_gs, 3), round(rec_forest_gs, 3), round(ac_forest, 3),
     round(bas_forest, 3), round(rec_forest, 3)))
```

```
accuracy_score          C gs: 0.932, без gs: 0.793
balanced_accuracy_score C gs: 0.477, без gs: 0.889
recall_score            C gs: 0.0, без gs: 0.835
```

### Проверка качества модели

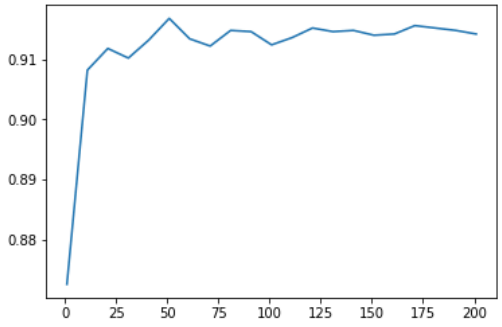
```
[ ]  # Конвертация предсказанных признаков
     bin_target_bag_gs = convert_target_to_binary(target_bag_gs, 6)
```

```
[ ]  ac_bag_gs = accuracy_score(y_test, target_bag_gs)
     bas_bag_gs = balanced_accuracy_score(bin_y_test, bin_target_bag_gs)
     rec_bag_gs = recall_score(bin_y_test, bin_target_bag_gs)
     print('''accuracy_score          C gs: {0}, без gs: {3}
     balanced_accuracy_score C gs: {1}, без gs: {4}
     recall_score            C gs: {2}, без gs: {5}'''.format(round(ac_bag_gs,3),
     round(bas_bag_gs, 3), round(rec_bag_gs, 3), round(ac_bag, 3),
     round(bas_bag, 3), round(rec_bag, 3)))
```

```
accuracy_score          C gs: 0.932, без gs: 0.926
balanced_accuracy_score C gs: 0.477, без gs: 0.965
recall_score            C gs: 0.0, без gs: 0.954
```

```
[ ]  # Изменение качества на обучающей выборке
     plt.plot(n_range_bag, bag_gs.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7fd7300a1f98>]
```



```
x = PrettyTable()
x.field_names = ["Model", "accuracy_score", "balanced_accuracy_score", "recall_score"]
```

```
x.add_row(["К-соседей GS", round(ac_knei_gs,3), round(bas_knei_gs, 3),
           round(rec_knei_gs, 3)])
x.add_row(["К-соседей без GS", round(ac_knei, 3), round(bas_knei, 3),
           round(rec_knei, 3)])
x.add_row(["Градиентный спуск GS", round(ac_grad_gs,3), round(bas_grad_gs, 3),
           round(rec_grad_gs, 3)])
x.add_row(["Градиентный спуск без GS", round(ac_grad, 3), round(bas_grad, 3),
              round(rec_grad, 3)])
x.add_row(["Дерево принятия решений GS", round(ac_dtc_gs,3), round(bas_dtc_gs, 3),
           round(rec_dtc_gs, 3)])
x.add_row(["Дерево принятия решений без GS", round(ac_dtc, 3),
          round(bas_dtc, 3), round(rec_dtc, 3)])
x.add_row(["Модель бэггинга GS", round(ac_bag_gs,3), round(bas_bag_gs, 3),
           round(rec_bag_gs, 3)])
x.add_row(["Модель бэггинга без GS", round(ac_bag, 3), round(bas_bag, 3),
           round(rec_bag, 3)])
x.add_row(["Случайный лес GS", round(ac_forest_gs,3), round(bas_forest_gs, 3),
           round(rec_forest_gs, 3)])
x.add_row(["Случайный лес без GS", round(ac_forest, 3),
round(bas_forest, 3), round(rec_forest, 3)])
```

```
print(x)
```

```
+--------------------------------+--------------+------------------------+--------------+
|             Model              | accuracy_score | balanced_accuracy_score | recall_score |
+--------------------------------+--------------+------------------------+--------------+
|          К-соседей GS          |     0.878    |          0.952         |    0.949     |
|        к-соседей без GS        |     0.879    |          0.945         |    0.931     |
|      Градиентный спуск GS      |     0.363    |           0.5          |     0.0      |
|   Градиентный спуск без GS     |     0.045    |          0.499         |    0.028     |
|   Дерево принятия решений GS   |     0.908    |          0.94          |    0.902     |
| Дерево принятия решений без GS |     0.906    |          0.941         |    0.904     |
|       Модель бэггинга GS       |     0.932    |          0.477         |     0.0      |
|     Модель бэггинга без GS     |     0.926    |          0.965         |    0.954     |
|        Случайный лес GS        |     0.932    |          0.477         |     0.0      |
|      Случайный лес без GS      |     0.793    |          0.889         |    0.835     |
+--------------------------------+--------------+------------------------+--------------+
```

## Заключение

Таким образом, с помощью машинного обучения можно направленно рекламировать функции своего продукта и получать от этого максимальный прирост пользователей различных функций.

Наилучший результат показала модель бэггинга. В результате некоторых доработок данная модель может быть вполне использована на практике.

## Список литературы


1. Лекции Гапанюка Ю.Е. [Электронный ресурс]. – Электрон. дан. - URL: https://github.com/ugapanyuk/ml_course/wiki/COURSE_TMO (дата обращения: 20.05.2019)
2. Black Friday Data Set [Электронный ресурс]. – Электрон. дан. - URL: https://www.kaggle.com/mehdidag/black-friday (дата обращения: 20.05.2019)
3. Stackoverflow [Электронный ресурс]. – Электрон. дан. - URL: https://stackoverflow.com (дата обращения: 20.05.2019)
4. Scikit-learn [Электронный ресурс]. – Электрон. дан. - URL: https://scikit-learn.org/stable/# (дата обращения: 20.05.2019)