**Министерство науки и высшего образования Российской Федерации**
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***
***(КФ МГТУ им. Н.Э. Баумана)***

**ФАКУЛЬТЕТ** **ИУК «Информатика и управление»**

**КАФЕДРА** **ИУК5 «Информатика и вычислительная техника»**

# Лабораторная работа №3

## «Классификация, деревья решений и метод ближайших соседей. Линейные модели классификации и регрессии»

**ДИСЦИПЛИНА: «Проектирование интеллектуальных систем»**

Выполнил: студент гр. ИУК4-11М _____ ( Сафронов Н.С. )
(подпись) (Ф.И.О.)

Проверил: _____ ( Потапов А.Е. )
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2024

**Цель работы:** формирование практических навыков применения простейших методов классификации и регрессии, формирование практических навыков применения линейных моделей для интеллектуальных систем, формирование практических навыков работы с методами обучения на основе ансамбля классификаторов.

**Задачи:**

- Разработать решение задачи классификации методами решающих деревьев и ближайших соседей. Обосновать выбор правил и метрик. Выполнить задачу кросс-валидации. Рассмотреть решение задачи регрессии на полученных моделях.
- Разработать решение задачи линейной регрессии, разработать решение задачи логистической регрессии с линейным классификатором. Применить регуляризацию для моделей. Оценить качество решений.
- Формирование практических навыков работы с методами обучения на основе ансамбля классификаторов.

**Результаты выполнения работы**

**Часть 1.** Деревья решений в игрушечной задаче и на данных Adult репозитория UCI

```
df_train = {}
df_train['Внешность'] = ['приятная', 'приятная', 'приятная', 'отталкивающая',
                         'отталкивающая', 'отталкивающая', 'приятная']
df_train['Алкоголь_в_напитке'] = ['да', 'да', 'нет', 'нет', 'да', 'да', 'да']
df_train['Уровень_красноречия'] = ['высокий', 'низкий', 'средний', 'средний', 'низкий',
                                   'высокий', 'средний']
df_train['Потраченные_деньги'] = ['много', 'мало', 'много', 'мало', 'много',
                                  'много', 'много']
df_train['Поедет'] = LabelEncoder().fit_transform(['+', '-', '+', '-', '-', '+', '+'])

df_train = create_df(df_train, features)
df_train
```

| | Поедет | Внешность_отталкивающая | Внешность_приятная | Алкоголь_в_напитке_да | Алкоголь_в_напитке_нет | Уровень_красноречия_высокий | Уровень_красноречия_низкий | Уровень_красноречия_средний | Потраченные_деньги_мало | Потраченные_деньги_много |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 3 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 4 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 6 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |

**Рисунок 1** – Обучающая выборка

```
df_test
```

| | Алкоголь_в_напитке_да | Алкоголь_в_напитке_нет | Уровень_красноречия_высокий | Уровень_красноречия_средний | Потраченные_деньги_много | Внешность_приятная | Потраченные_деньги_мало | Внешность_отталкивающая |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

**Рисунок 2** – Тестовая выборка

**Вопрос 1.** Какова энтропия начальной системы ($S_0$)? Под состояниями системы понимаем значения признака "Поедет" – 0 или 1 (то есть всего 2 состояния).

$$S = -\sum_{i=1}^{N} p_i \, log_2 p_i$$

$$S_0 = -\frac{3}{7} * log_2 \frac{3}{7} - \frac{4}{7} * log_2 \frac{4}{7} = 0.985$$

**Вопрос 2.** Рассмотрим разбиение обучающей выборки по признаку «Внешность_приятная». Какова энтропия $S_1$ левой группы, тех, у кого внешность приятная, и правой группы – $S_2$? Каков прирост информации при данном разбиении (IG)?

$$S_1 = -\frac{1}{4} log_2 \frac{1}{4} - \frac{3}{4} log_2 \frac{3}{4} = 0.811$$

$$S_2 = -\frac{2}{3} log_2 \frac{2}{3} - \frac{1}{3} log_2 \frac{1}{3} = 0.918$$

$$IG = S_0 - \frac{4}{7} S_1 - \frac{3}{7} S_2 = 0.128$$

```
dt = DecisionTreeClassifier(criterion="entropy", random_state=17)
dt.fit(df_train, y)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=17)
```

**Рисунок 3** – Обучение классификатора дерева решений

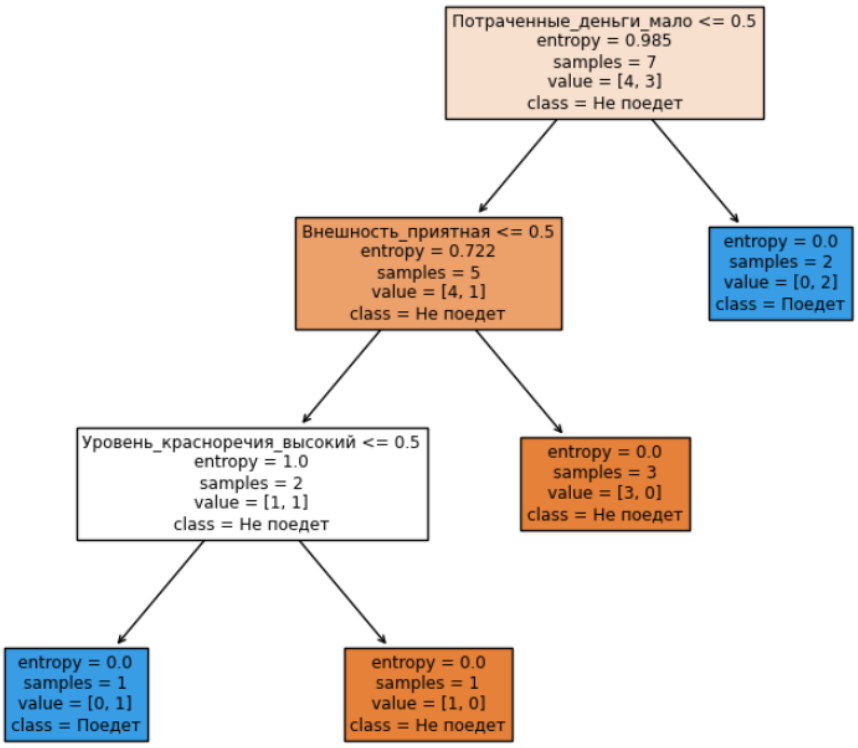


**Рисунок 4** – Полученное дерево решений

```python
from math import log


def entropy(a_list):
    lst = list(a_list)
    size = len(lst)
    entropy = 0
    set_elements = len(set(lst))
    if set_elements in [0, 1]:
        return 0
    for i in set(lst):
        occ = lst.count(i)
        entropy -= occ / size * log(occ / size, 2)
    return entropy
```

**Рисунок 5** – Функция расчёта энтропия Шеннона

```python
print(entropy(balls)) # 9 синих и 11 желтых
print(entropy(balls_left)) # 8 синих и 5 желтых
print(entropy(balls_right)) # 1 синий и 6 желтых
print(entropy([1,2,3,4,5,6])) # энтропия игральной кости с несмещенным центром тяжести
```

```
0.9927744539878084
0.961236604722876
0.5916727785823275
2.584962500721156
```

**Рисунок 6** – Проверка расчёта энтропия Шеннона

**Вопрос 3**. Чему равна энтропия состояния, заданного списком balls_left?

0.961

**Вопрос 4.** Чему равна энтропия игральной кости с несмещенным центром тяжести?

2.585

**Вопрос 5.** Каков прирост информации при разделении выборки на balls_left и balls_right?

```
def information_gain(root, left, right):
    return (
        entropy(root)
        - 1.0 * len(left) / len(root) * entropy(left)
        - 1.0 * len(right) / len(root) * entropy(right)
    )
```

```
print(information_gain(balls, balls_left, balls_right))
```

```
0.16088518841412436
```

**Рисунок 7** – Прирост информации при разделении выборки на balls_left и balls_right

```
def best_feature_to_split(X, y, feature_names):
    clf = information_gains(X, y)
    best_feat_id = clf.index(max(clf))
    best_feature = feature_names[best_feat_id]
    print(f"Лучший критерий для разделения: {best_feature}")

    x_left = X[X.iloc[:, best_feat_id] == 0]
    x_right = X[X.iloc[:, best_feat_id] == 1]
    print(f"Разделение: {len(x_left)} (лево) и {len(x_right)} (право)")

    y_left = y[X.iloc[:, best_feat_id] == 0]
    y_right = y[X.iloc[:, best_feat_id] == 1]
    entropy_left = entropy(y_left)
    entropy_right = entropy(y_right)
    print(f"Энтропия: {entropy_left} (лево) и {entropy_right} (право)")
    print("_" * 30 + "\n")
    if entropy_left != 0:
        print(f"Разделяем левую группу на {len(x_left)} выборки:")
        best_feature_to_split(x_left, y_left, feature_names)
    if entropy_right != 0:
        print(f"Разделяем правую группу на {len(x_right)} выборки:")
        best_feature_to_split(x_right, y_right, feature_names)
```

**Рисунок 8** – Алгоритм построения дерева за счет рекурсивного вызова

```
best_feature_to_split(df_train, y, df_train.columns)
```

```
Лучший критерий для разделения: Потраченные_деньги_мало
Разделение: 5 (лево) и 2 (право)
Энтропия: 0.7219280948873623 (лево) и 0 (право)
______________________________

Разделяем левую группу на 5 выборки:
Лучший критерий для разделения: Внешность_отталкивающая
Разделение: 3 (лево) и 2 (право)
Энтропия: 0 (лево) и 1.0 (право)
______________________________

Разделяем правую группу на 2 выборки:
Лучший критерий для разделения: Уровень_красноречия_высокий
Разделение: 1 (лево) и 1 (право)
Энтропия: 0 (лево) и 0 (право)
______________________________
```

**Рисунок 9** – Результат работы алгоритма построения дерева за счет рекурсивного вызова

```
data_train.tail()
```

| | Age | Workclass | fnlwgt | Education | Education_Num | Martial_Status | Occupation | Relationship | Race | Sex | Capital_Gain | Capital_Loss | Hours_per_week | Country | Target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **32556** | 27 | Private | 257302 | Assoc-acdm | 12 | Married-civ-spouse | Tech-support | Wife | White | Female | 0 | 0 | 38 | United-States | <=50K |
| **32557** | 40 | Private | 154374 | HS-grad | 9 | Married-civ-spouse | Machine-op-inspct | Husband | White | Male | 0 | 0 | 40 | United-States | >50K |
| **32558** | 58 | Private | 151910 | HS-grad | 9 | Widowed | Adm-clerical | Unmarried | White | Female | 0 | 0 | 40 | United-States | <=50K |
| **32559** | 22 | Private | 201490 | HS-grad | 9 | Never-married | Adm-clerical | Own-child | White | Male | 0 | 0 | 20 | United-States | <=50K |
| **32560** | 52 | Self-emp-inc | 287927 | HS-grad | 9 | Married-civ-spouse | Exec-managerial | Wife | White | Female | 15024 | 0 | 40 | United-States | >50K |

**Рисунок 10** – Обучающая выборка

```
data_test.tail()
```

| | Age | Workclass | fnlwgt | Education | Education_Num | Martial_Status | Occupation | Relationship | Race | Sex | Capital_Gain | Capital_Loss | Hours_per_week | Country | Target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **16277** | 39 | Private | 215419.0 | Bachelors | 13.0 | Divorced | Prof-specialty | Not-in-family | White | Female | 0.0 | 0.0 | 36.0 | United-States | <=50K. |
| **16278** | 64 | NaN | 321403.0 | HS-grad | 9.0 | Widowed | NaN | Other-relative | Black | Male | 0.0 | 0.0 | 40.0 | United-States | <=50K. |
| **16279** | 38 | Private | 374983.0 | Bachelors | 13.0 | Married-civ-spouse | Prof-specialty | Husband | White | Male | 0.0 | 0.0 | 50.0 | United-States | <=50K. |
| **16280** | 44 | Private | 83891.0 | Bachelors | 13.0 | Divorced | Adm-clerical | Own-child | Asian-Pac-Islander | Male | 5455.0 | 0.0 | 40.0 | United-States | <=50K. |
| **16281** | 35 | Self-emp-inc | 182148.0 | Bachelors | 13.0 | Married-civ-spouse | Exec-managerial | Husband | White | Male | 0.0 | 0.0 | 60.0 | United-States | >50K. |

**Рисунок 11** – Тестовая выборка

**Вопрос 6.** Какова доля правильных ответов дерева решений на тестовой выборке при максимальной глубине дерева = 3 и random_state = 17?

```
tree = DecisionTreeClassifier(max_depth=3, random_state=17)
tree.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=17)
```

```
tree_predictions = tree.predict(X_test[X_train.columns])
```

```
accuracy_score(y_test, tree_predictions)
```

```
0.8447884036607088
```

**Рисунок 12** – Доля верных ответов дерева решений на тестовой выборке

```
%%time
tree_params = {"max_depth": range(2, 11)}

locally_best_tree = GridSearchCV(
    DecisionTreeClassifier(random_state=17), tree_params, cv=5
)

locally_best_tree.fit(X_train, y_train)
```

```
CPU times: total: 3.56 s
Wall time: 7.05 s
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

```
print("Лучшие параметры:", locally_best_forest.best_params_)
print("Лучшая оценка:", locally_best_forest.best_score_)
```

```
Лучшие параметры: {'max_depth': 14, 'max_features': 45}
Лучшая оценка: 0.8619822161458556
```

**Рисунок 13** – Дерево решений с настройкой параметров

**Вопрос 7**. Какова доля правильных ответов дерева решений на тестовой выборке при максимальной глубине дерева = 9 и random_state = 17?

```
tuned_tree = DecisionTreeClassifier(max_depth=9, random_state=17)
tuned_tree.fit(X_train, y_train)
tuned_tree_predictions = tuned_tree.predict(X_test[X_train.columns])
accuracy_score(y_test, tuned_tree_predictions)
```

```
0.8579939807137154
```

**Рисунок 14** – Доля верных ответов дерева решений на тестовой выборке

```
rf = RandomForestClassifier(n_estimators=100, random_state=17)
rf.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=17)
```

```
%%time
cv_scores = cross_val_score(rf, X_train, y_train, cv=3)
```

```
CPU times: total: 6.67 s
Wall time: 11.1 s
```

```
cv_scores, cv_scores.mean()
```

```
(array([0.85194398, 0.85572139, 0.859578  ]), 0.8557477912289437)
```

**Рисунок 15** – Случайный лес без настройки параметров (опционально)

```
forest_predictions = rf.predict(X_test[X_train.columns])
```

```
accuracy_score(y_test, forest_predictions)
```

```
0.8525274860266568
```

**Рисунок 16** – Прогноз для тестовой выборки

```
forest_params = {"max_depth": range(10, 16), "max_features": range(5, 105, 20)}

locally_best_forest = GridSearchCV(
    RandomForestClassifier(n_estimators=10, random_state=17, n_jobs=-1),
    forest_params,
    cv=3,
    verbose=1,
)

locally_best_forest.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
```

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

**Рисунок 17** – Случайный лес с настройкой параметров (опционально)

```
print("Лучшие параметры:", locally_best_forest.best_params_)
print("Лучшая оценка:", locally_best_forest.best_score_)
```

```
Лучшие параметры: {'max_depth': 14, 'max_features': 45}
Лучшая оценка: 0.8619822161458556
```

```
tuned_forest_predictions = locally_best_forest.predict(X_test[X_train.columns])
accuracy_score(y_test, tuned_forest_predictions)
```

```
0.8631533689576807
```

**Рисунок 18** – Прогноз для тестовой выборки

**Часть 2.** Прогнозирование популярности статей на TechMedia (Хабр) с помощью линейных моделей

```
# при необходимости поменяйте путь к данным
train_df = pd.read_csv('../../data/howpop_train.csv')
test_df  = pd.read_csv('../../data/howpop_test.csv')
```

```
train_df.head(1).T
```

| | 0 |
|---|---|
| **url** | https://habrahabr.ru/post/18284/ |
| **domain** | habrahabr.ru |
| **post_id** | 18284 |
| **published** | 2008-01-01 18:19:00 |
| **author** | @Tapac |
| **flow** | develop |
| **polling** | False |
| **content_len** | 4305 |
| **title** | Новогодний подарок блоггерам — WordPress 2.3.2 |
| **comments** | 0 |
| **favs** | 0 |
| **views** | 236 |
| **votes_plus** | 0.0 |
| **votes_minus** | 0.0 |
| **views_lognorm** | -0.792687 |
| **favs_lognorm** | -1.344075 |
| **comments_lognorm** | -2.436871 |

**Рисунок 19** – Пример записи обучающей выборки

**Вопрос 1.** Есть ли в train_df признаки, корреляция между которыми больше 0.9? Обратите внимание, именно различные признаки - корреляция признака с самим собой естественно больше 0.9 :)

```
sd = train_df.corr()

def search(x):
    if x > 0.9 and x < 1:
        return '+'
    return '-'

for columns in train_df.corr():
    sd[columns] = train_df.corr()[columns].apply(search)


print(sd)
```

**Рисунок 20** – Алгоритм вычисления признаков с корреляцией больше 0.9

| | post_id | polling | content_len | comments | favs | views | votes_plus | \ |
|---|---|---|---|---|---|---|---|---|
| post_id | - | - | - | - | - | - | - | |
| polling | - | - | - | - | - | - | - | |
| content_len | - | - | - | - | - | - | - | |
| comments | - | - | - | - | - | - | - | |
| favs | - | - | - | - | - | - | - | |
| views | - | - | - | - | - | - | - | |
| votes_plus | - | - | - | - | - | - | - | |
| votes_minus | - | - | - | - | - | - | - | |
| views_lognorm | - | - | - | - | - | - | - | |
| favs_lognorm | - | - | - | - | - | - | - | |
| comments_lognorm | - | - | - | - | - | - | - | |

| | votes_minus | views_lognorm | favs_lognorm | comments_lognorm |
|---|---|---|---|---|
| post_id | - | - | - | - |
| polling | - | - | - | - |
| content_len | - | - | - | - |
| comments | - | - | - | - |
| favs | - | - | - | - |
| views | - | - | - | - |
| votes_plus | - | - | - | - |
| votes_minus | - | - | - | - |
| views_lognorm | - | - | - | - |
| favs_lognorm | - | - | - | - |
| comments_lognorm | - | - | - | - |

**Рисунок 21** – Признаки с корреляцией больше 0.9

**Ответ:** нет.

**Вопрос 2.** В каком году было больше всего публикаций? (Рассматриваем train_df)

- 2014
- 2015
- 2016
- 2017

```
train_df['published'] = pd.to_datetime(train_df.published, yearfirst = True)
print(train_df.published.dtype)
```

```
datetime64[ns]
```

```
train_df['year'] = [d.year for d in train_df.published]
train_df['month'] = [d.month for d in train_df.published]

train_df['dayofweek'] = [d.isoweekday() for d in train_df.published]
train_df['hour'] = [d.hour for d in train_df.published]
```

```
train_df.groupby('year').post_id.count().idxmax()
```

```
2015
```

**Рисунок 22** – Поиск года с максимальным числом публикаций

**Вопрос 3**. Какой размер у полученного словаря?

- 43789
- 50624

- 93895
- 74378

```
vectorizer_title = TfidfVectorizer(min_df=3, max_df=0.3, ngram_range=(1, 3))

X_train_title = vectorizer_title.fit_transform(X_train['title'])
X_valid_title = vectorizer_title.transform(X_valid['title'])
X_test_title = vectorizer_title.transform(X_test['title'])
```

```
len(vectorizer_title.vocabulary_)
```

```
50624
```

**Рисунок 23** – Размер словаря, полученного от TfidfVectorizer с word

**Вопрос 4.** Какой индекс у слова 'python'?

```
vectorizer_title.vocabulary_['python']
```

```
9065
```

**Рисунок 24** – Индекс слова python

**Вопрос 5.** Какой размер у полученного словаря?

- 218
- 510
- 125
- 981

```
vectorizer_title_ch =  TfidfVectorizer(analyzer='char')

X_train_title_ch = vectorizer_title_ch.fit_transform(X_train['title'])
X_valid_title_ch = vectorizer_title_ch.transform(X_valid['title'])
X_test_title_ch = vectorizer_title_ch.transform(X_test['title'])
```

```
len(vectorizer_title_ch.vocabulary_)
```

```
218
```

**Рисунок 25** – Размер словаря, полученного от TfidfVectorizer с char

**Вопрос 6.** Выберите верные утверждения:

- обе модели показывают одинаковый результат (среднеквадратичная ошибка отличается не больше чем на тысячные), регуляризация ничего не меняет;
- при alpha=0.1 модель переобучается;

- среднеквадратичная ошибка первой модели на тесте меньше;
- при alpha=1.0 у модели обобщающая способность лучше, чем у при alpha=0.1

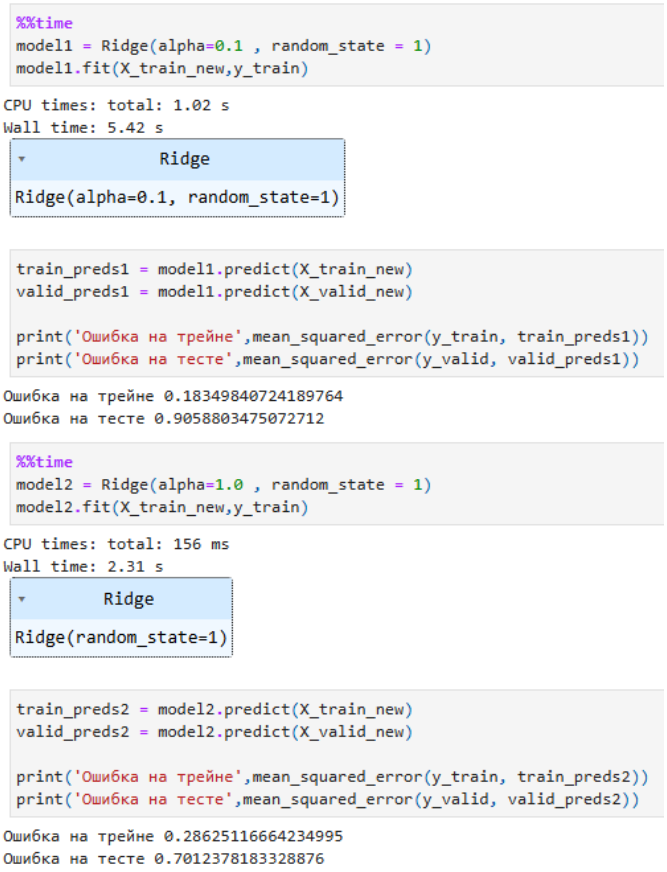


**Рисунок 26** – Решение задания 6

- Обе модели показывают одинаковый результат (среднеквадратичная ошибка отличается не больше чем на тысячные), регуляризация ничего не меняет.

*Неверно*: Значения ошибок отличаются на порядок больше, чем тысячные. Регуляризация оказывает заметное влияние.

- При alpha=0.1 модель переобучается.

*Верно*: При меньшем значении alpha ошибка на обучающей выборке значительно меньше (0.183), но на тестовой выборке ошибка заметно выше (0.905), что указывает на переобучение.

- Среднеквадратичная ошибка первой модели на тесте меньше.

*Неверно*: На тестовой выборке ошибка модели с alpha=0.1 выше (0.905 против 0.701).

- При alpha=1.0 у модели обобщающая способность лучше, чем при alpha=0.1.

*Верно*: Модель с alpha=1.0 показывает более сбалансированные ошибки на обучающей и тестовой выборках, что свидетельствует о лучшей обобщающей способности.

**Часть 3.** Логистическая регрессия и случайный лес в задаче кредитного скоринга

**Задание 1.** В зале суда есть 5 присяжных, каждый из них по отдельности с вероятностью 70% может правильно определить, виновен подсудимый или нет. С какой вероятностью они все вместе вынесут правильный вердикт, если решение принимается большинством голосов?

- 70.00%
- 83.20%
- 83.70%
- 87.50%

Для решения задачи применим **теорему жюри Кондорсе**, которая гласит, что вероятность того, что большинство жюри примет правильное решение, зависит от индивидуальной вероятности правильного решения каждым участником ($p$) и количества участников ($n$). Решение принимается большинством голосов ($k > n/2$).

$n = 5$ — количество присяжных.

$p = 0.7$ — вероятность, что один присяжный правильно решит.

Решение принимается большинством голосов ($k = 3,4,5$).

**Формула для вероятности правильного вердикта:**

$$\mu = \sum_{k=\lceil n/2 \rceil}^{n} \binom{n}{k} p^k (1-p)^{n-k}$$

где:

- $\lceil n/2 \rceil$ — минимальное количество голосов для большинства.
- $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ — биномиальный коэффициент.

$$\mu_3 = \binom{5}{3} \cdot p^3 \cdot (1-p)^2$$

$$\mu_3 = \frac{5!}{3! \cdot 2!} \cdot (0.7)^3 \cdot (0.3)^2 = 0.3087$$

$$\mu_4 = \binom{5}{4} \cdot p^4 \cdot (1-p)^1$$

$$\mu_4 = \frac{5!}{4! \cdot 1!} \cdot (0.7)^4 \cdot (0.3)^1 = 0.36015$$

$$\mu_5 = \binom{5}{5} \cdot p^5 \cdot (1-p)^0$$

$$\mu_5 = \frac{5!}{5! \cdot 0!} \cdot (0.7)^5 \cdot (0.3)^0 = 0.16807$$

$$\mu = \mu_3 + \mu_4 + \mu_5 = 0.837$$

**Задание 2.** Сделайте интервальную оценку среднего возраста (age) для клиентов, которые просрочили выплату кредита, с 90% "уверенностью". Используйте пример из статьи, поставьте np.random.seed(0), как это сделано в статье.

```
def get_bootstrap_samples(data, n_samples):
    indices = np.random.randint(0, len(data), (n_samples, len(data)))
    samples = data[indices]
    return samples


def stat_intervals(stat, alpha):
    boundaries = np.percentile(stat, [100 * alpha / 2.0, 100 * (1 - alpha / 2.0)])
    return boundaries


churn = data[data["SeriousDlqin2yrs"] == 1]["age"].values
np.random.seed(0)
churn_mean_scores = [np.mean(sample) for sample in get_bootstrap_samples(churn, 1000)]
print("Mean interval", stat_intervals(churn_mean_scores, 0.1))
```

```
Mean interval [45.71379414 46.12700479]
```

**Рисунок 27 -** Интервальная оценка среднего возраста (age) для клиентов, которые просрочили выплату кредита, с 90% "уверенностью"

**Задание 3.** Сделайте GridSearch с метрикой "roc-auc" по параметру C. Какое оптимальное значение параметра C?

```
grid_search = GridSearchCV(lr, parameters, n_jobs=-1, scoring="roc_auc", cv=skf)
grid_search = grid_search.fit(X, y)
grid_search.best_estimator_
```

```
LogisticRegression
LogisticRegression(C=0.001, class_weight='balanced', random_state=5)
```

**Рисунок 27** - Оптимальное значение параметра C

**Задание 4.** Можно ли считать лучшую модель устойчивой? (модель считаем устойчивой, если стандартное отклонение на валидации меньше 0.5%) Сохраните точность лучшей модели, она вам приходится для следующих заданий.

```
grid_search.cv_results_["std_test_score"][1]
```

```
0.008137559189742445
```

```
grid_search.best_score_
```

```
0.8089120626797153
```

**Рисунок 28 –** Значение стандартного отклонения на валидации и точность лучшей модели

**Задание 5.** Определите самый важный признак. Важность признака определяется абсолютным значением его коэффициента. Так же нужно нормализировать все признаки, чтобы можно их было корректно сравнить.

```
from sklearn.preprocessing import StandardScaler

lr = LogisticRegression(C=0.001, random_state=5, class_weight="balanced")
scal = StandardScaler()
lr.fit(scal.fit_transform(X), y)

pd.DataFrame(
    {"feat": independent_columns_names, "coef": lr.coef_.flatten().tolist()}
).sort_values(by="coef", ascending=False)
```

| | feat | coef |
|---|---|---|
| 1 | NumberOfTime30-59DaysPastDueNotWorse | 0.723427 |
| 3 | NumberOfTimes90DaysLate | 0.516788 |
| 4 | NumberOfTime60-89DaysPastDueNotWorse | 0.193558 |
| 6 | NumberOfDependents | 0.101443 |
| 2 | DebtRatio | -0.024096 |
| 5 | MonthlyIncome | -0.163146 |
| 0 | age | -0.416702 |

**Рисунок 29 –** Самые важные признаки

Самый важный признак - NumberOfTime30-59DaysPastDueNotWorse.

**Задание 6.** Посчитайте долю влияния DebtRatio на предсказание. (Воспользуйтесь функцией softmax)

```
print((np.exp(lr.coef_[0]) / np.sum(np.exp(lr.coef_[0])))[2])
```

```
0.11426375283065274
```

**Рисунок 30 –** Доля влияния DebtRatio на предсказание

**Задание 7.** Давайте посмотрим, как можно интерпретировать влияние наших признаков. Для этого заново оценим логистическую регрессию в абсолютных величинах. После этого посчитайте во сколько раз увеличатся шансы, что клиент не выплатит кредит, если увеличить возраст на 20 лет при всех остальных равных значениях признаков. (теоретический расчет можно посмотреть здесь)

```
lr = LogisticRegression(C=0.001, random_state=5, class_weight="balanced")
lr.fit(X, y)

pd.DataFrame(
    {"feat": independent_columns_names, "coef": lr.coef_.flatten().tolist()}
).sort_values(by="coef", ascending=False)
```

| | feat | coef |
|---|---|---|
| 1 | NumberOfTime30-59DaysPastDueNotWorse | 0.446830 |
| 3 | NumberOfTimes90DaysLate | 0.390379 |
| 4 | NumberOfTime60-89DaysPastDueNotWorse | 0.216038 |
| 6 | NumberOfDependents | 0.191725 |
| 2 | DebtRatio | -0.000006 |
| 5 | MonthlyIncome | -0.000011 |
| 0 | age | -0.013655 |

```
np.exp(lr.coef_[0][0] * 20)
```

```
0.7610094499540035
```

**Рисунок 31 –** Во сколько раз увеличатся шансы, что клиент не выплатит кредит, если увеличить возраст на 20 лет при всех остальных равных значениях признаков

**Задание 8.** На сколько точность лучшей модели случайного леса выше точности логистической регрессии на валидации?

```
%%time
rf_grid_search = GridSearchCV(
    rf, parameters, n_jobs=-1, scoring="roc_auc", cv=skf, verbose=True
)
rf_grid_search = rf_grid_search.fit(X, y)
print(rf_grid_search.best_score_ - grid_search.best_score_)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
0.026866475306627002
CPU times: total: 5.28 s
Wall time: 1min 29s
```

**Рисунок 32 –** На сколько точность лучшей модели случайного леса выше точности логистической регрессии на валидации

**Задание 9.** Определите какой признак имеет самое слабое влияние.

```
independent_columns_names[
    np.argmin(rf_grid_search.best_estimator_.feature_importances_)
]
```

```
'NumberOfDependents'
```

**Рисунок 33 –** Признак, имеющий самое слабое влияние

**Задание 10.** Какое наиболее существенное примущество логистической регрессии перед случайным лесом для нашей бизнес-задачи?

- меньше тратится времени для тренировки модели;
- меньше параметров для перебора;
- интепретируемость признаков;
- линейные свойства алгоритма.

С одной стороны, модель Random Forest лучше подходит для нашей проблемы кредитного скоринга. Ее производительность на 4% выше. Причиной такого результата является небольшое количество признаков и композиционное свойство случайных лесов.

С другой стороны, главное преимущество логистической регрессии заключается в том, что мы можем интерпретировать влияние признаков на результат модели.

**Задание 11.** Следующая задача обучить бэггинг классификатор (random_state=42). В качестве базовых классификаторов возьмите 100

логистических регрессий и на этот раз используйте не GridSearchCV, а RandomizedSearchCV. Так как перебирать все 54 варианта комбинаций долго, то поставьте максимальное число итераций 20 для RandomizedSearchCV. Также не забудьте передать параметр валидации cv и random_state=1. Какая лучшая точность получилась?

```
bg = BaggingClassifier(
    LogisticRegression(class_weight="balanced"),
    n_estimators=100,
    n_jobs=-1,
    random_state=42,
)
r_grid_search = RandomizedSearchCV(
    bg,
    parameters,
    n_jobs=-1,
    scoring="roc_auc",
    cv=skf,
    n_iter=20,
    random_state=1,
    verbose=True,
)
r_grid_search = r_grid_search.fit(X, y)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

```
r_grid_search.best_score_
```

```
0.8087951623513827
```

**Рисунок 34 –** Лучшая точность бэггинг классификатора

**Задача 12.** Дайте интерпретацию лучших параметров для бэггинга. Почему именно такие значения оказались лучшими?

- для бэггинга важно использовать как можно меньше признаков
- бэггинг лучше работает на небольших выборках
- меньше корреляция между одиночными моделями
- чем больше признаков, тем меньше теряется информации

Преимущество случайного леса в том, что деревья в композиции не сильно коррелируют. Аналогично, для бэггинга с логистической регрессией, чем слабее корреляция между отдельными моделями, тем выше точность. Поскольку в логистической регрессии почти нет случайности, нам приходится менять набор признаков, чтобы минимизировать корреляцию между нашими отдельными моделями.

**Вывод:** в ходе работы сформированы практические навыки применения методов классификации и регрессии, линейных моделей и ансамблей классификаторов для решения задач интеллектуальных систем.