МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Курсовой проект
по курсу «Методы, средства и технологии мультимедиа»

Выполнил: М. А. Инютин
Группа: М8О-407Б-19
Преподаватель: Б. В. Вишняков

Москва, 2022

1 Задача и набор данных

1 Выбор набора данных

Я выбрал задачу классификации и набор данных «Политическая кухня» [1] из олимпиады «Я — профессионал» по направлению «Искуственный интеллект (бакалавриат)». В рамках олимпиады я участвовал в другом направлении, однако мне интересно в спокойном режиме решить задачу классификации.

2 Описание

Задача предоставлена партнером олимпиады — Федеральным исследовательским центром "Информатика и управление"РАН

На некотором несуществующем интернет-ресурсе «Политическая кухня» популярностью пользуются два вида видеороликов: про политику и про кулинарию. При этом под роликами про консервативную политику комментарии оставляют только консерваторы, про либеральную — только либералы. Кулинарные видео комментируют только кулинары. Иногда на ресурсе "Политическая кухня"происходит сбой, и комментарии перепутываются (кулинарный комментарий попадает под политическое видео, либеральный — под консервативное видео и т.п.), тогда необходимо по комментарию определить, кто его оставил (консерватор, либерал или кулинар) и перенести в соответствующий раздел. Доступа к самим текстам комментариев у команды "Политической кухни"нет, но все тексты прошли обработку лингвистическим анализатором и каждый представлен набором численных признаков.

Перед вами стоит задача разработать алгоритм машинного обучения, предсказывающий кем был написан комментарий: консерватором, либералом или кулинаром.

3 Формат ввода

Тренировочная выборка `Train.csv` представляет собой csv-таблицу со столбцами-признаками и столбцом целевой переменной `target`.

Описание признаков обучающих данных:

- `comments_count` — общее количество комментариев под видео, для которого создан комментарий,
- `replies_count` — общее количество ответов на комментарии под видео,
- `both_count` — общее количество сообщений под видео,
- `sentence_count` — общее количество предложений под видео,
- `word_count` — общее количество слов под видео,
- `target` - метка кем был оставлен комментарий (0 - либерал, 1 - кулинар, 2 - консерватор),
- остальные столбцы — признаки, полученные с помощью лингвистического анализатора.

2 Препроцессинг

1 Выбросы

В наборе данных есть достаточно много выбросов, поэтому для каждого признака я вычисляю среднее значение μ_i и дисперсию σ_i:

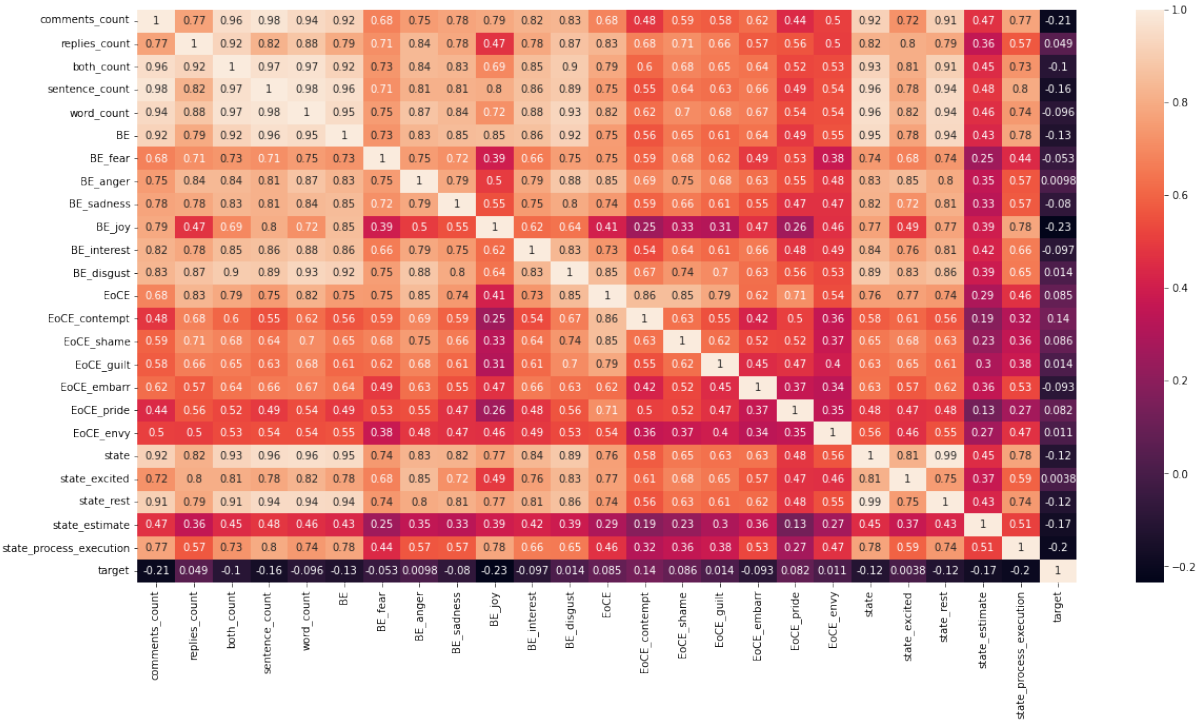
```
COEF_OUTLIERS = 3

col_val = dict()

for col in ds:
      col_np = ds[col].to_numpy()
      mu = np.mean(col_np)
      sigma = np.std(col_np)
      col_val[col] = (mu - COEF_OUTLIERS * sigma, mu + COEF_OUTLIERS * sigma)
```

Оставляю только те строки, в которых каждый признак попадает в промежуток от $\mu_i - 3 \cdot \sigma_i$ до $\mu_i + 3 \cdot \sigma_i$. При попытке удалять выбросы самостоятельно я терял около $1/5$ набора данных, то есть пользователей с очень больним количеством сообщений оказалось много и они реальны, а не получены из-за ошибок.

```
for key in col_val.keys():
      l, r = col_val[key]
      ds = ds.loc[l <= ds[key]]
      ds = ds.loc[ds[key] <= r]
```

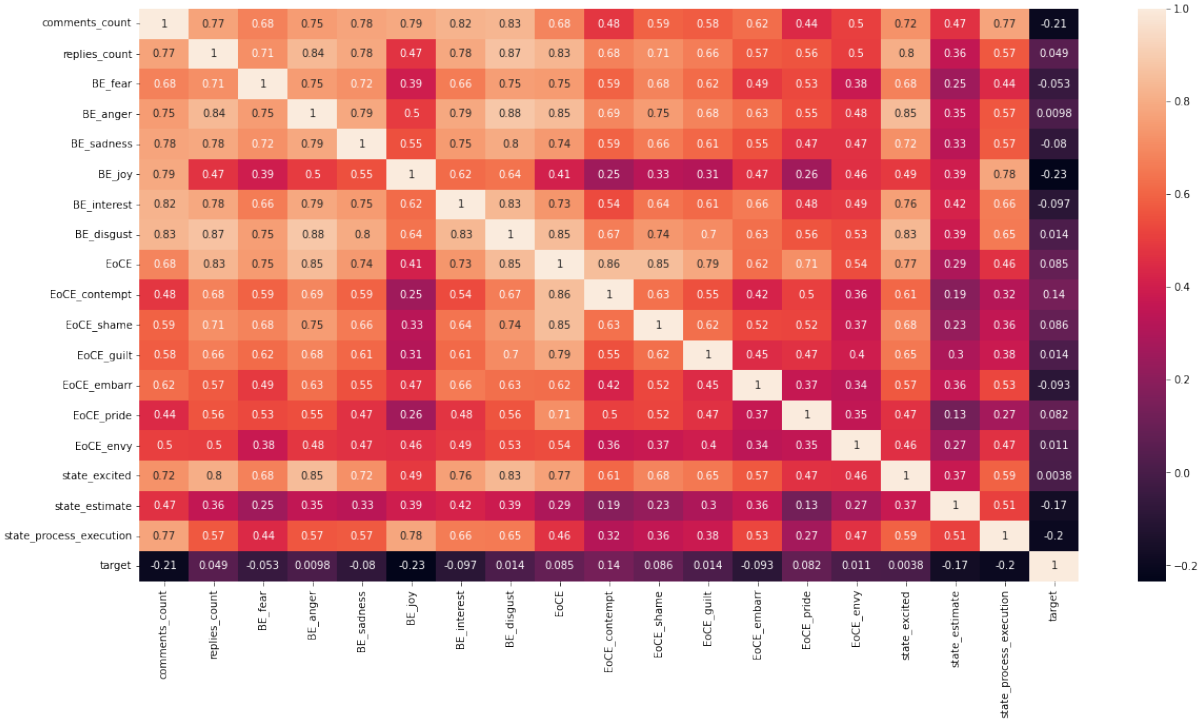
2 Коррелированные признаки

Построю матрицу корреляции для всех признаков:



Видно, что некоторые признаки почти полностью линейно зависят от других. Это не удивительно: чем больше слов написал пользователь, тем больше предложений и самих комментариев. Удаляю все признаки, для которых коэффициент корреляции $R \geq 0.9$: `both_count`, `sentence_count`, `word_cound`, `BE`, `state`, `state_rest`.

Теперь матрица корреляции выглядит так:



Классы `target` сильно не сбалансированы, поэтому я случайно удаляю классы 1 и 2 так, чтобы всех классов было поровну:

```
COEF_UNDER = 1
CLASSES = 2

ds_0 = ds[ds["target"] == 0]
ds_other = ds[ds["target"] != 0]
down_ds_other = ds_other.sample(n=len(ds_0) * CLASSES * COEF_UNDER, random_state=1)
ds = pd.concat([down_ds_other, ds_0])
```

Нормализую все признаки, чтобы они были в промежутке $[0, 1]$.

```
X = normalize(X, norm="max", axis=0)
```

Разбиваю данные на обучающую и тестовую выборку в соотношении 80 к 20.

```
train_X, test_X, train_y, test_y = train_test_split(
      X, y, train_size=0.8, random_state=1, shuffle=True
)
```

3 Алгоритм

1 Выбор алгоритма

Я выбрал дерево решений — логический алгоритм классификации, решающий задачи классификации и регрессии. Представляет собой объединение логических условий в структуру дерева.

2 Реализация алгоритма

2.1 Вершина дерева

Описываю класс вершины дерева, в которой хранится предикат $X_{ind} > value$. Если вершина является терминальной, то $value$ хранит вероятности для класса.

```
1  class TreeData:
2      def __init__(self, value=None, ind=None, leaf=True):
3          self.value = value
4          self.ind = ind
5          self.leaf = leaf
6
7      def is_leaf(self):
8          return self.leaf
9
10     def decide(self, X):
11         if self.is_leaf():
12             return None
13         else:
14             return True if X[self.ind] > self.value else False
15
16     def predict(self, X):
17         if self.is_leaf():
18             return value
19         else:
20             return None
```

2.2 Граф

Для удобного представления дерева реализую граф, в который легко можно добавить новые вершины. Каждая вершина хранит словарь переходов `data`, что позволяет при случае использовать этот класс не только для бинарного дерева, но и для дерева общего вида.

```
class Graph:
      def __init__(self):
          self.data = []
          self.info = []
          self.size = 0
          self.add()

      def can_go(self, u, c):
          return c in self.data[u]

      def go(self, u, c):
          return self.data[u][c]

      def set_go(self, u, c, v):
          self.data[u][c] = v

      def get_tree_data(self, u):
          return self.info[u]

      def set_tree_data(self, u, tree_data):
          self.info[u] = tree_data

      def add(self):
          self.data.append(dict())
          self.info.append(TreeData())
          self.size += 1
          return self.size - 1

      def is_leaf(self, u):
          return self.info[u].is_leaf()
```

2.3 Decision Tree

Описываю сам класс дерева принятия решений. Функция `decide_rec` нужна для спуска по дереву. `get_split_ids` возвращает индексы признаков, по которым будет происходит деление — случайные индексы или индексы всех признаков. `calc_cnt` подсчитывает количество каждого класса в узле, а `calc_p` вычисляет вероятности классов. Функция `crit` вычисляет метрики разбиения данных. Поддерживаются три метрики, как и в `DecisionTreeClassifier` [2].

```
class DecisionTree:
      def __init__(self, classes, max_depth, min_samples_split, splitter, criterion):
          self.tree = Graph()
          self.classes = classes
          self.max_depth = max_depth
          self.min_samples_split = min_samples_split
          self.splitter = splitter
          self.criterion = criterion
          self.data = []
          self.INF = 1e18

      def decide(self, X):
          return self.decide_rec(0, X)

      def decide_rec(self, u, X):
          u_data = self.tree.get_tree_data(u)
          if self.tree.is_leaf(u):
              return u_data.value
          else:
              dec = u_data.decide(X)
              return self.decide_rec(self.tree.go(u, dec), X)

      def make_leaf(self, u, value):
          u_data = TreeData(value=value, leaf=True)
          self.tree.set_tree_data(u, u_data)

      def rnd_ids(self, n):
          res = set()
          while len(res) * len(res) < n:
              rnd_num = randint(0, n - 1)
              while rnd_num in res:
                  rnd_num = randint(0, n - 1)
              res.add(rnd_num)
          return np.array([elem for elem in res])

      def get_split_ids(self):
          if self.splitter == "random":
              return self.rnd_ids(self.d)
          if self.splitter == "best":
              return range(self.d)
```

```
41
42      def build(self, X, y):
43          self.X = X
44          self.y = y
45          self.n = X.shape[0]
46          self.d = X.shape[1]
47          ids = np.arange(self.n)
48          self.build_rec(0, ids, 1)
49
50      def calc_cnt(self, ids):
51          cnt = np.zeros(self.classes)
52          for j in ids:
53              cnt[self.y[j]] += 1
54          uniq = 0
55          for el in cnt:
56              if (el < 1):
57                  uniq += 1
58          return cnt, uniq
59
60      def calc_p(self, cnt):
61          n = 0
62          for el in cnt:
63              n += el
64          return cnt / n
65
66      def entropy(self, cnt, p):
67          res = 0
68          for i, elem in enumerate(p):
69              if elem > 0:
70                  res += elem * np.log(elem)
71          return -res
72
73      def gini(self, cnt, p):
74          res = 0
75          for elem in p:
76              res += elem * (1 - elem)
77          return res
78
79      def log_loss(self, cnt, p):
80          res = 0
81          for i, elem in enumerate(p):
82              if 0 < elem < 1:
83                  res += elem * np.log(elem) + (1 - elem) * np.log(1 - elem)
84          return -res
85
86      def crit(self, cnt, p):
87          if self.criterion == "entropy":
88              return self.entropy(cnt, p)
89          if self.criterion == "gini":
```

```
            return self.gini(cnt, p)
        if self.criterion == "log_loss":
            return self.log_loss(cnt, p)

    def build_rec(self, u, ids, h):
        n = len(ids)
        cnt, uniq = self.calc_cnt(ids)
        p = self.calc_p(cnt)
        stop1 = n < self.min_samples_split
        stop2 = False if self.max_depth == None else h > self.max_depth
        stop3 = uniq == 1
        if stop1 or stop2 or stop3:
            self.make_leaf(u, p)
            return
        u_data = self.tree.get_tree_data(u)
        split_ids = self.get_split_ids()
        loss = self.INF
        res = (-1, -1)
        for i in split_ids:
            tmp = sorted([(self.X[j][i], self.y[j]) for j in ids])
            size_l, size_r = 0, n
            cnt_l, cnt_r = np.zeros(self.classes), [el for el in cnt]
            for j in range(self.n):
                while size_l < n and leq(tmp[size_l][0], tmp[j][0]):
                    elem_y = tmp[size_l][1]
                    cnt_l[elem_y] += 1
                    cnt_r[elem_y] -= 1
                    size_l += 1
                size_r = n - size_l
                if size_l == 0 or size_r == 0:
                    continue
                p_l, p_r = self.calc_p(cnt_l), self.calc_p(cnt_r)
                loss_l = self.crit(cnt_l, p_l)
                loss_r = self.crit(cnt_r, p_r)
                split_loss = (size_l * loss_l + size_r * loss_r) / n
                if split_loss < loss:
                    loss = split_loss
                    res = (i, tmp[j][0])
        if res == (-1, -1):
            self.make_leaf(u, p)
            return
        u_data = TreeData(value=res[1], ind=res[0], leaf=False)
        self.tree.set_tree_data(u, u_data)
        l, r = [], []
        for j in ids:
            if u_data.decide(self.X[j]):
                l.append(j)
            else:
                r.append(j)
```

```
139             ul = self.tree.add()
140             ur = self.tree.add()
141             self.tree.set_go(u, True, ul)
142             self.tree.set_go(u, False, ur)
143             self.build_rec(ul, np.array(l), h + 1)
144             self.build_rec(ur, np.array(r), h + 1)
```

2.4 Классификатор

Сам классификатор унаследован от `ClassifierMixin` и `BaseEstimator`, как описано в [3] и [4]. Это нужно для интеграции с `sklearn` и кроссвалидации.

```
class MyDecisionTreeClassifier(ClassifierMixin, BaseEstimator):
      def __init__(
          self,
          classes=2,
          max_depth=None,
          min_samples_split=2,
          splitter="best",
          criterion="gini",
      ):
          self.classes = classes
          self.max_depth = max_depth
          self.min_samples_split = min_samples_split
          self.splitter = splitter
          self.criterion = criterion
          self.tree = DecisionTree(
              classes, max_depth, min_samples_split, splitter, criterion
          )

      def fit(self, X, y):
          # Check that X and y have correct shape
          X, y = check_X_y(X, y)
          # Store the classes seen during fit
          self.classes_ = unique_labels(y)

          self.X_ = X
          self.y_ = y
          self.tree.build(X, y)
          # Return the classifier
          return self

      def predict(self, X):
          # Check is fit had been called
          check_is_fitted(self, ["X_", "y_"])

          # Input validation
          X = check_array(X)

          p = []
          for elem in X:
              z = self.tree.decide(elem)
              p.append(np.argmax(z))
          return np.array(p)
```

3 Описание алгоритма

3.1 Классификация

Дерево решений — это бинарное дерево, в каждом узле которого хранится предикат. Если значение компоненты вектора больше значения, записанного в узле, то переходим в правое поддерево, иначе в левое. Так каждый входной вектор спускается от корня до листа и мы определяем вероятность принадлежности к каждому классу.

3.2 Построение

Дерево строится рекурсивно. Изначально есть вся обучающая выборка. Перебираем каждый признак, по которому возможно разделить выборку на две, сортируем по этому признаку объекты выборки. Теперь перебираем за один проход величину, которую запишем в узел: все объекты с меньшим или равным значение пойдут налево, все объекты с большим пойдут направо. Предполагая, что при таком делении мы создадим два листа, вычисляем ошибку рабиения. Чем она меньше, тем лучше разбиение. После перебора всех признаков и значений, разбиваем выборку и строим дерево рекурсивно для левого и правого поддеревьев.

4 Результаты

1 Выбор метрики качества

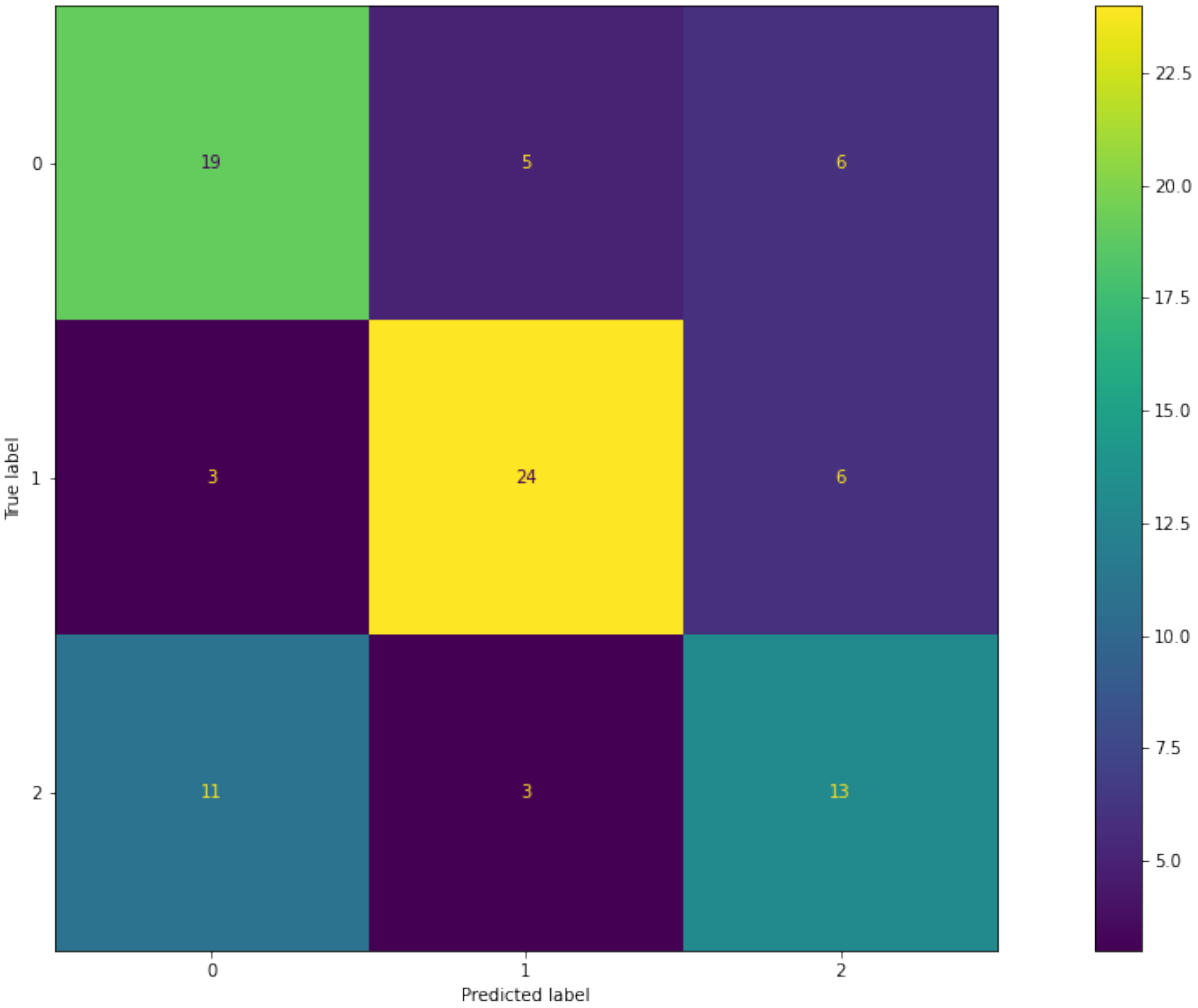
Для оценки результата использую метрики `accuracy` — отношени верно классифицированных объектов к общему количеству объектов выборки. Для подбора параметров я использую кроссвалидацию по следующим параметрам:

```
params = {
    "dtc__max_depth": [None, 32, 16, 8],
    "dtc__min_samples_split": [2, 4, 8],
    "dtc__splitter": ["best", "random"],
    "dtc__criterion": ["gini", "entropy", "log_loss"],
}
```

Так как данных достаточно мало, то кроссвалидация разбивает выборку только на две части:

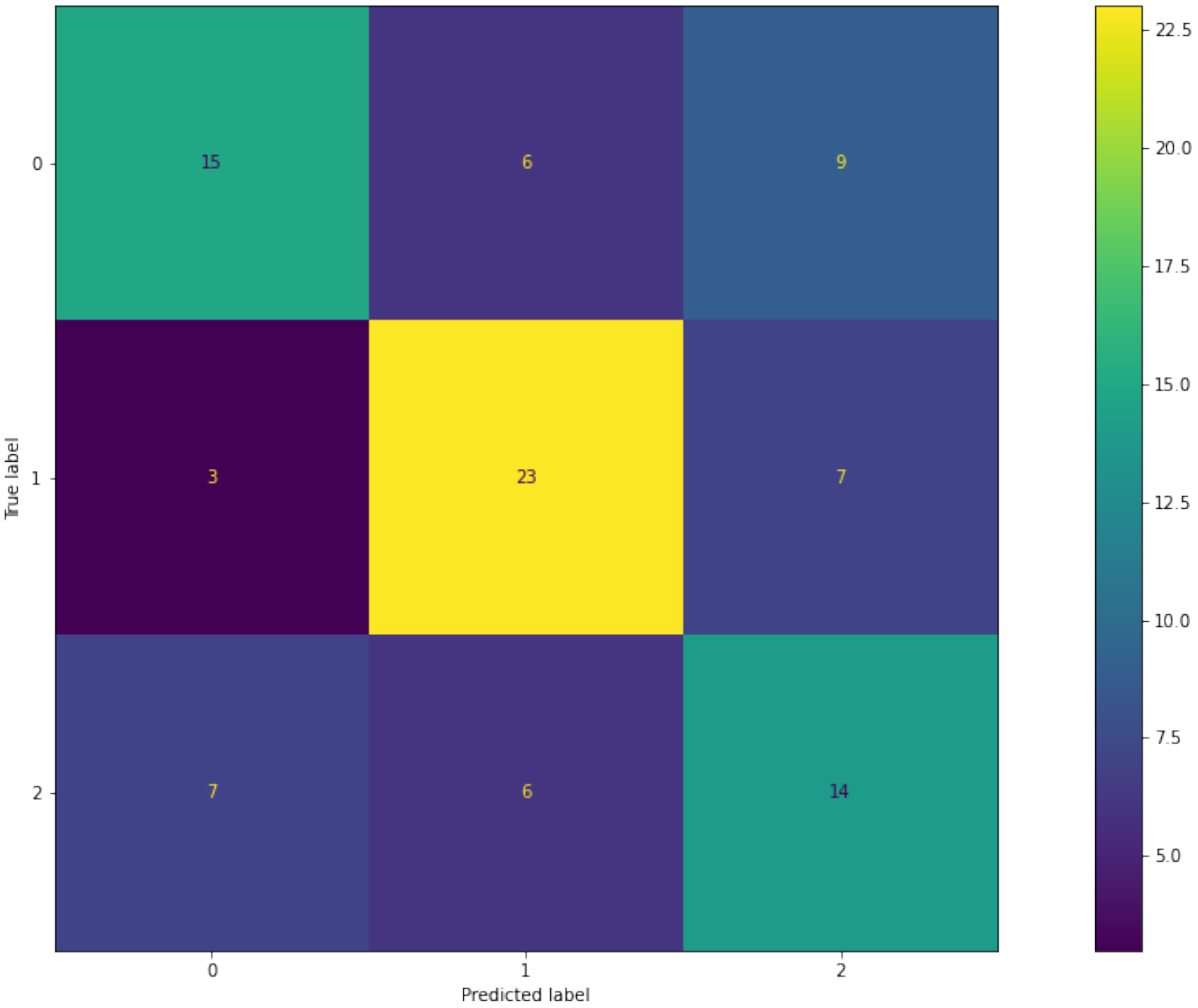
```
gscv_my_dtc = GridSearchCV(
      Pipeline([("dtc", MyDecisionTreeClassifier(classes=3))]),
      param_grid=params,
      cv=2,
      scoring="accuracy",
)
gscv_my_dtc.fit(train_X, train_y)
```

2 Точность на тестовой выборке




```
Лучшие гиперпараметры модели: {
'dtc__criterion': 'gini',
'dtc__max_depth': None,
'dtc__min_samples_split': 2,
'dtc__splitter': 'best'
}
Лучший счёт модели: 0.5377565752306823
Метрика Accuracy: 0.6222222222222222
```

3 Дерево решений из sklearn




```
Лучшие гиперпараметры модели: {
'dtc__criterion': 'entropy',
'dtc__max_depth': 8,
'dtc__min_samples_split': 4,
'dtc__splitter': 'best'
}
Лучший счёт модели: 0.6133638817400038
Метрика Accuracy: 0.5777777777777777
```

5 Выводы

При препроцессинге набора данных я столкнулся почти со всеми задачами, о которых я знал, но не сталкивался на практике: выбросы, сильно коррелированные признаки и перевес классов. Ручная обработка дала не очень хороший результат, поэтому я исследовал и применил методы обработки данных.

В лабораторной работе я реализовал дерево решений для регрессии, однако в нём были небольшие ошибки, и я хотел сделать реализацию более гибкой, поэтому выбрал дерево решений для классификации.

Результаты показали, что моё дерево решений показывает примерно ту же точность, что и дерево из `sklearn`, при кроссвалидации на разных общих параметрах.

Очень понравилось работать с набором данных не с `Kaggle`, где много сгенерированных данных. Я ощутил сложность обработки данных, приближенных к реальным, понял, насколько трудно бывает достичь точности даже в 50%.

Список литературы

[1] *Политическая кухня*
URL: https://disk.yandex.ru/d/9XuZIF8IIAx2fw
(дата обращения: 26.12.2022)

[2] *sklearn.tree.DecisionTreeClassifier*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
(дата обращения: 26.12.2022)

[3] *Developing scikit-learn estimators*
URL: https://scikit-learn.org/stable/developers/develop.html#
(дата обращения: 26.12.2022)

[4] *project-template/_template.py at master · scikit-learn-contrib*
URL: https://github.com/scikit-learn-contrib/project-template/blob/master/skltemplate/_template.py
(дата обращения: 26.12.2022)