Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет: “Информационные технологии и прикладная математика”
Кафедра: 806 “Вычислительная математика и программирование”

Курсовая работа

по курсу «Искусственный интеллект и Глубокое обучение»

Студент: Полей-Добронравова Амелия Вадимовна

Группа: М8О-407Б

Преподаватель: Вишняков

Дата: 26.12.2021

Итоговая оценка:

Подпись преподавателя:

Москва, 2021

Содержание

Введение

В данной работе прежде всего стоит обозначить основные понятия и сферы применения изучаемой технологии.

Машинное обучение - процесс, при котором в ходе решения большого количества сходных задач аналитическая система выявляет закономерности и обучается дальнейшему принятию решений без участия человека.

Основная идея МЛ - автоматизированный или ручной поиск закономерностей в типовых данных и выбор лучшего решения из представленных.

Глубокое обучение - метод машинного обучения, который предполагает построение искусственной нейронной сети.

Нейронная сеть - математическая модель, имитирующая работу человеческого мозга.

Классификация - отнесение объекта к одному из заранее определенных классов.

Регрессия - предсказание значения числовой функции на следующих шагах.

Детекция - определение области на фотографии, занимаемой искомым объектом.

Наиболее часто технологию машинного обучения используют в маркетинге. Например, Amazon использует его для того, чтобы показывать покупателям тот товар, который их должен заинтересовать. Это происходит на основе анализа данных о прошлых покупках и других пользователях.

Google и Яндекс также применяют в своей работе машинное обучение, чтобы показывать рекламу определенным пользователям.

Таким же образом устроены умные ленты в соцсетях. Facebook,Instagram, Twitter или TikTok исследуют интересы пользователей по всем известным о них данным: просмотр постов, лайки и комментарии, посещение пабликов и групп и др.

Также технология применяется в медицине и структурах безопасности. В медицине это предварительная диагностика и подбор индивидуального плана лечения на основе данных из истории болезни пациента. В сфере безопасности - системы распознавания лиц.

Области применения можно перечислять очень долго, и важность МЛ в современном мире - неотъемлемый факт.

Постановка задачи

На сегодняшний день любой активный пользователь интернета слушает в нём же музыку. Самые известные и удобные сервисы: Spotify, Яндекс Музыка, Youtube music. В каждом из них большой упор на рекомендации и составление тематических плейлистов. Конечно, подобрать индивидуальное предложение для каждого вручную невозможно. Для этого используют алгоритмы машинного обучения.

Даже внутри одного жанра можно найти похожие по мотивам композиции и именно их рекомендовать пользователю. Но для новичка это неподъемная задача.

Я поставила перед собой цель предсказывать жанр музыкальной композиции по её характеристикам, описанным в виде числовых или категориальных фичей. Это - задача классификации.

Описание данных

Мной был выбран датасет “music_genre”. Он доступен на Kaggle. Полный список жанров, включенных в CSV: Electronic, Anime, Jazz, Alternative, Country, Rap, Blues, Rock, Classical, Hip-Hop.

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50005 entries, 0 to 50004
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   instance_id       50000 non-null  float64
 1   artist_name       50000 non-null  object
 2   track_name        50000 non-null  object
 3   popularity        50000 non-null  float64
 4   acousticness      50000 non-null  float64
 5   danceability      50000 non-null  float64
 6   duration_ms       50000 non-null  float64
 7   energy            50000 non-null  float64
 8   instrumentalness  50000 non-null  float64
 9   key               50000 non-null  object
 10  liveness          50000 non-null  float64
 11  loudness          50000 non-null  float64
 12  mode              50000 non-null  object
 13  speechiness       50000 non-null  float64
 14  tempo             50000 non-null  object
 15  obtained_date     50000 non-null  object
 16  valence           50000 non-null  float64
 17  music_genre       50000 non-null  object
dtypes: float64(11), object(7)
memory usage: 6.9+ MB
```

Как видно из скриншота выше, в датасете 17 колонок.

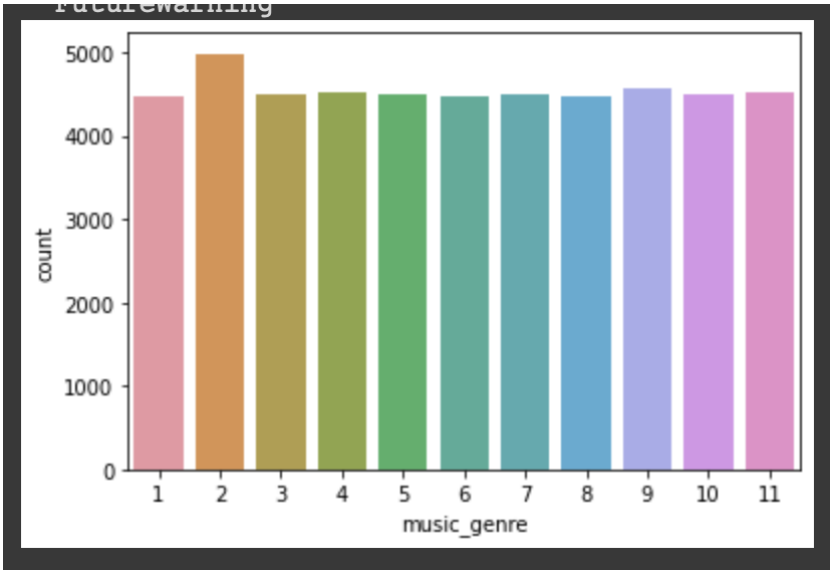
Числовые признаки

instance_id - уникальный id;
popularity - популярность;
acousticness - акустичность;
dancebility - танцевальность;
duration_ms - длительность в микросекундах;
energy - энергичность;
instumentalness - инструментальность;
liveness - оживленность;
loudness - громкость;
speechiness - количество речи;
valence - валентность.

Категориальные признаки

artist_name - имя исполнителей;
track_name - имя композиции;
key - тональность;
mode - лад;
tempo - темп;
obtained_date - дата получения;
music_genre - target, жанр.

Перед выбором этого датасета я проверила сбалансированность классов, проблемы в этом не возникло:



Описание алгоритма

Существуют в разы эффективнее алгоритмы для классификации, но самый известный и простой - **логистическая регрессия**. Так как в датасете представлено больше двух жанров, логистическая регрессия именно многоклассовая.

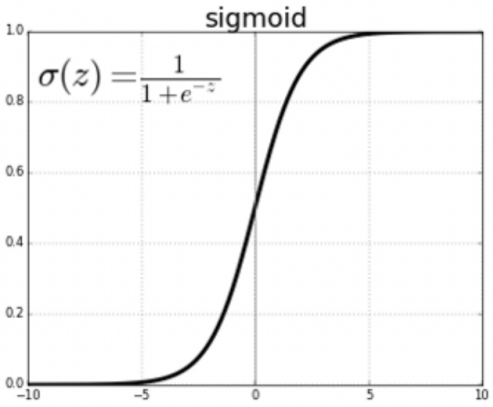
Многоклассовая логистическая регрессия в виде предсказания выдаёт вектор из n элементов, где n - количество классов, на которое обучалась модель. Каждый элемент этого вектора - вероятность принадлежности к i-ому классу. Итоговым ответом выбирается номер элемента, содержащего наибольшую вероятность (argmax).

Для обучения модели входное множество очищенных данных разделяют на две части: обучающее и тестовое множество. Обучающее подаётся на вход модели для выявления закономерностей. На тестовом смотрят, насколько хороши или плохи результаты предсказания на данных, которые модель не видела.

Процесс обучения логистической регрессии - алгоритм обратного распространения ошибки:

1. Задать всем начальным параметрам случайные значения.
2. Предсказать значение.
3. Посчитать ошибку с помощью функции ошибки. Если ошибка меньше выбранной точности, закончить.
4. Посчитать антиградиент и изменить веса и смещение.
5. Перейти к пункту 2.

Каким образом устроена именно модель. В начале - применение линейной функции $f(x) = w*x+b$,
где x - элемент входных данных;
w - веса подходящей размерности;
b - смещение.
Так как у меня задача для нескольких классов, веса и смещения у каждого класса свои, отличающиеся от других.
Далее - функции активации. Для каждого класса результат линейной функции используется для подсчета **логита**:



Далее вектор логитов передаётся функции softmax, высчитывающей вероятности:

$$f_i(x) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Очевидно, что она возвращает значения от 0 до 1.

Чтобы посчитать ошибку по всему вектору вероятностей, над таргетом(ответом для x) производят операцию one-hot-encoding:

id	color
1	red
2	blue
3	green
4	blue

One Hot Encoding

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

И помещают результаты softmax и значения target в какую-то функцию ошибки. Мной выбрана logloss:

$$\text{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y_i}) + (1-y_i)\log(1-\hat{y_i})\right]$$

После этого, идет изменение весов и смещения по антиградиенту. Для этого рассчитываются производные функций. Упрощение для комбинации softmax и logloss, использованное мной:

$$\frac{\partial \text{logloss}}{\partial w} = (a-y)x$$

И веса уменьшаются на это значение, умноженное на шаг обучения - маленькое число, обычно порядка 10^{-5} контролирующее скорость обучения. Смещение уменьшается на разность предсказаний и ответов, т.к. производная по смещению равна 1.

Обучение продолжается либо до выбранной точности обучения, либо максимальное число эпох. *Эпоха* - проведения процесса обучения целиком над обучающей выборкой.

Для улучшения показателей обучения я использовала *регуляризацию*: Существует много видов регуляризации, я использовала два из них - L1 и L2. L1-регуляризация способствует разреженности функции, когда лишь немногие факторы не равны нулю. L2-регуляризация способствует появлению малых весовых коэффициентов модели, но не способствует их точному равенству нулю.

L1-регуляризация реализует это путём отбора наиболее важных факторов, которые сильнее всего влияют на результат. Для простоты можете считать, что факторы с малой величиной влияния на конечный результат фактически «помогают» вам предсказывать лишь шум в наборе обучающих данных. L2-регуляризация предотвращает переобучения модели путём запрета на непропорционально большие весовые коэффициенты.

```
grad_l2 = self.l2_coef * wc * 2
grad_l1 = self.l1_coef * np.sign(wc)
res = grad_basic + grad_l1 + grad_l2
```

L1 = коэффициент * знак весов

L2 = коэффициент * веса * 2

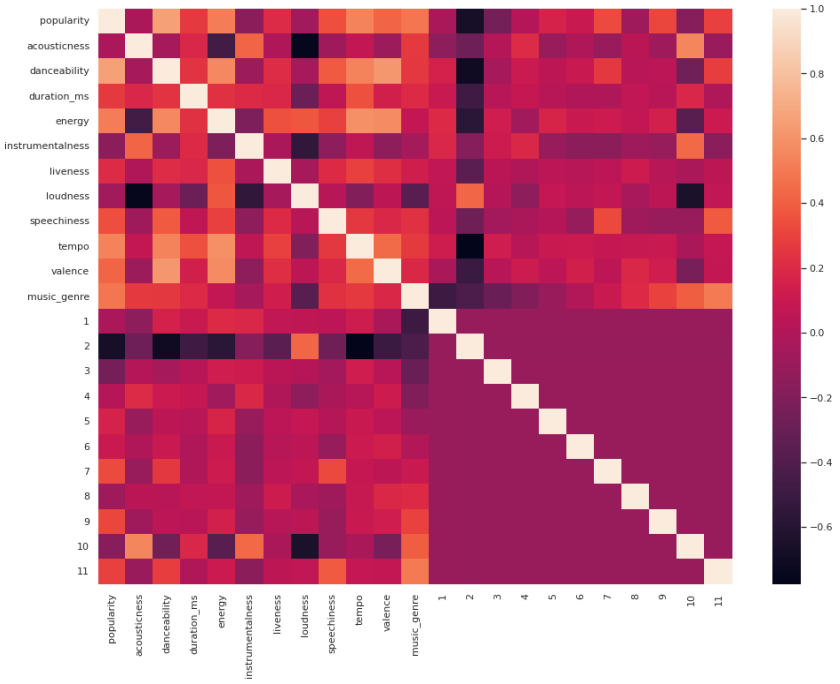
Все эти значения суммируются с общим рассчитанным градиентом и передаются на изменение весов.

Предобработка данных

До изучения свойств признаков я сразу удалила из данных id, имя артиста, название композиции, дату получения трека. Все эти признаки категориальные, они тяжелы, занимают много места и несут мало смысловой нагрузки.

Далее я заполнила все пропущенные ячейки нулями.

Изучила зависимость числовых признаков друг от друга и влияние их на целевой признак (матрица корреляции):

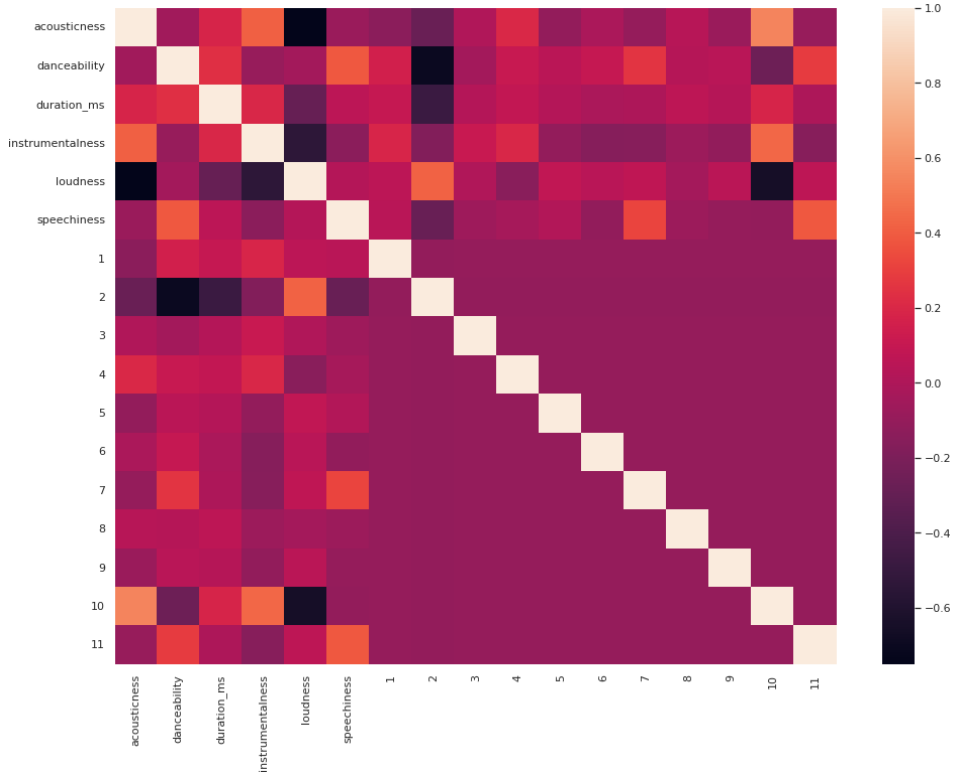


Я обнаружила, что нормальная корреляция видна сильнее всего у energy и loudness, выкинула energy.

Над категориальными признаками я провела one hot encoding.

Улучшения, проведенные в рамках курсовой:

С жанрами очень низкая корреляция у популярности, оживленности, валентности и темпа. Удалю их. Остались те признаки, у которых заметна корреляция:



Далее я нормализовала данные с помощью sklearn. Нормализование данных нужно потому, что алгоритмы машинного обучения, как правило, работают лучше или сходятся быстрее, когда данные имеют меньший масштаб.

Нормализация также делает процесс обучения менее чувствительным к масштабу функций. Это приводит к улучшению коэффициентов после тренировки.

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Реализация алгоритма

Код:

```
def logit(x, w, b):
   return x @ w + b

def softmax(a):
 a = np.array(a)
 return np.exp(a-max(a)) / np.sum(np.exp(a-max(a)))

def softmax_grad(s): #сумма по одному из измерений
```

```
    jacobian_m = np.diag(s)
    print(s.shape)
    for i in range(len(jacobian_m)):
        for j in range(len(jacobian_m)):
            print(s[i], s[i].dtype)
            if i == j:
                jacobian_m[i][j] = s[i] * (1-s[i])
            else:
                jacobian_m[i][j] = -s[i]*s[j]
    return jacobian_m.sum(axis = 1)

class LogisticRegression(object):
    def __init__(self, l1_coef, l2_coef):
        self.l1_coef = l1_coef
        self.l2_coef = l2_coef
        self.w = None
        self.b = None
        self.num_classes = None

    def fit(self, X, y, epochs=500, lr=1e-7, batch_size=64, flag = False):
        n, k = X.shape
        self.num_classes = 11
        self.b = np.random.random(self.num_classes)
        if self.w is None:
            np.random.seed(10)
            self.w = np.random.random((X.shape[1], self.num_classes))
        losses = []
        for i in range(epochs):
            l = []
            for X_batch, y_batch in generate_batches(X_train, y, batch_size):
                predictions = self.predict(X_batch)
                l.append(self.__loss(y_batch, predictions))
                self.w -= lr * self.get_grad(X_batch, y_batch, predictions)
                self.b -= lr * np.sum(predictions-y_batch)/len(y_batch) #свой
градиент
            losses.append(np.sum(l) / len(l))
            if flag:
              break
        return losses

    def get_grad(self, X_batch, y_batch, predictions):
        logs = logit(X_batch, self.w, self.b)
        soft = []
        for log, yy in zip(logs, y_batch):
          soft.append(softmax(log) - yy)
          #print(s.shape, yy.shape)
        grad_basic = np.dot(X_batch.T, np.array(soft))
```

```
        wc = np.copy(self.w)
        grad_l2 = self.l2_coef * wc * 2
        grad_l1 = self.l1_coef * np.sign(wc)
        res = grad_basic + grad_l1 + grad_l2
        return res

    def predict(self, X):
        logs = logit(X, self.w, self.b)
        answer = []
        for one in range(len(logs)):
          answer.append(softmax(logs[one]))
        return answer

    def predict_test(self, X):
      logs = logit(X, self.w, self.b)
      answer = []
      for log in logs.values:
        answer.append(np.argmax(softmax(log))+1)
      return answer
    def __loss(self, y, p): #logloss
        p = np.clip(p, 1e-10, 1 - 1e-10)
        return -np.sum(y * np.log(p) + (1 - y) * np.log(1 - p))/len(y)
```

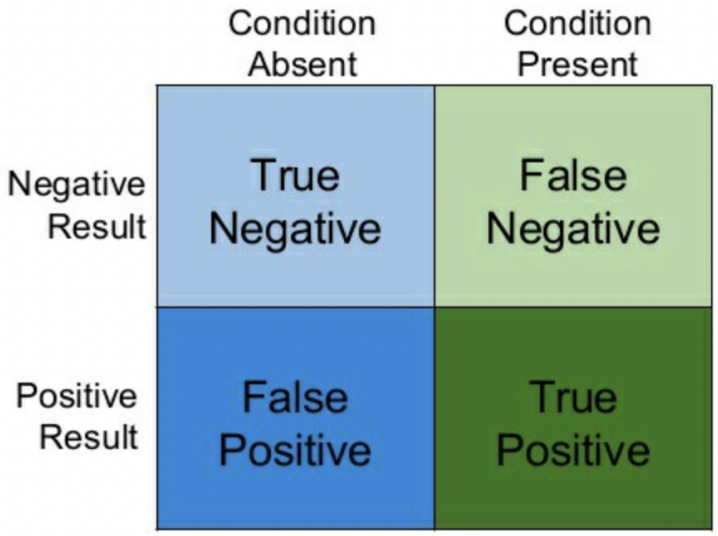
Пояснения:
__loss() - функция ошибки;
predict_test() - возвращает номера выигравших классов;
predict() - возвращает вектор вероятностей принадлежности к каждому классу;
get_grad() - подсчет градиента и регуляризаций;
fit() - функция обучения, возвращает loss на каждой эпохе.

Метрики качества

Как и в других аспектов, существует несколько видов метрик. Они отличаются для разных типов задач МЛ, и есть несколько вариантов внутри одной задачи. Рассмотрим метрики для классификации:
precision - доля истинного позитива во всех положительных прогнозах. Точность 1 означает, что у вас нет ложного срабатывания.
recall - доля истинно положительных результатов по всем фактическим положительным элементам.
Отзыв 1 означает, что у вас нет ложного отрицания.

f1_score как сренее гармоничество precision и recall отличается свойством: если одно из этих двух значений резко уменьшается, оценка f также уменьшается.
В данной задаче нет смысла отдавать предпочтение какой-либо из этих двух, лучше учитывать их обе

Для реализации f1_score мне нужно было реализовать каждую из этого списка, а также функцию для выделения количества:



Код этой функции ниже. Важен порядок передачи аргументов: в начале передается истинное значение, потом предсказанное.

```
[173] def compute_tp_tn_fn_fp(y_act, y_pred):
        tp = 0
        tn = 0
        fn = 0
        fp = 0
        num_classes = len(set(y_act))
        for i in range(num_classes):
          for h in range(len(y_act)):
            tp += (y_act[h] == i) & (y_pred[h] == i)
            tn += (y_act[h] != i) & (y_pred[h] != i)
            fn += (y_act[h] == i) & (y_pred[h] != i)
            fp += (y_act[h] != i) & (y_pred[h] == i)
        return tp, tn, fp, fn
```

Точность и отзыв:

```python
def compute_recall(tp, fn):
    '''
    Recall = TP /FN + TP

    '''
    return (tp  * 100)/ float( tp + fn+ 0.001)
```

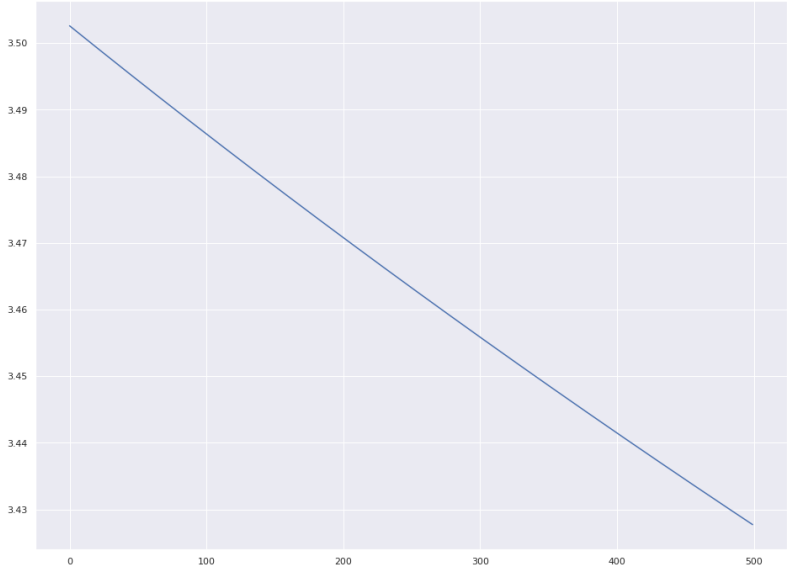
```python
def compute_precision(tp, fp):
    '''
    Precision = TP  / FP + TP

    '''
    return (tp  * 100)/ float( tp + fp + 0.001)
```

И наконец f1:

```python
# F1 = 2 * (precision * recall) / (precision + recall)
def compute_f1_score(y_true, y_pred): #for binary classification
    # calculates the F1 score
    tp, tn, fp, fn = compute_tp_tn_fn_fp(y_true, y_pred)
    precision = compute_precision(tp, fp)/100
    recall = compute_recall(tp, fn)/100
    f1_score = (2*precision*recall)/ (precision + recall + 0.001)
    return f1_score
```

Полученные результаты

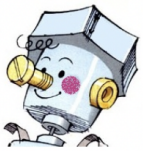
Итоговый вариант обучала 500 эпох. Коэффициент L1-регуляризации я задала 0.2, коэффициент L2-регуляризации решила поставить 0, то есть L2 вообще не применялась при обучении.
Изменение значения ошибки в течение обучения:



Выборка	Было	Стало	Было sklearn	Стало sklearn
Обучающая	0%	9%	21%	26%
Тестовая	0%	8,5%	20%	25%

При обучении модели sklearn я поставила максимально число эпох как и у своей модели - 500. Выбрала была l1 регуляризация, способ обучения - liblinear (зависит от выбора регуляризации).

Выводы



Для этой задачи стоило взять другую модель. Логистическая регрессия слишком простой алгоритм, используемый для изучения основ машинного обучения. Мой датасет оказался сложен, и здесь имеет смысл применять что-то другое, начиная от нейронных сетей и заканчивая бустингом.

Моя реализация заметно хуже библиотечной, и у меня есть предположение, что в библиотечной добавлены другие способы по улучшению результатов помимо регуляризации. Например, в документации обозначено: “В случае мультикласса алгоритм обучения использует схему «один против остальных» (OvR)”.

OvR - это эвристический метод использования алгоритмов двоичной классификации для классификации нескольких классов.
Он разделяет набор данных с несколькими классами на несколько задач двоичной классификации. Итоговые прогнозы делаются с использованием наиболее надежной модели.

Возможным недостатком этого подхода является то, что он требует создания одной модели для каждого класса. Например, для трех классов требуется три модели. Это может быть проблемой для больших наборов данных, медленных моделей или очень большого количества классов.

В случае же моего датасета это стало выигрышной стратегией. В случае OvR не используется softmax, что скорее всего и послужило причиной для такой разницы.