МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
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: Тояков А. О.
Преподаватель: Вишняков Б. В.
Группа: М8о-407б-18
Дата:
Оценка:
Подпись:

Москва, 2021

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Для начала обозначим определение предмета: машинное обучение (англ. *machine learning*, ML) — класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение за счёт применения решений множества сходных задач.

Подобные технологии в современном мире применяются повсеместно. Например, в интернете нативная реклама может использовать алгоритмы машинного обучения для того, чтобы рекомендовать пользователям товары в зависимости от их вкусов и предпочтений. ML применяется и в более серьёзных задачах. На основании результатов обучившихся моделей, врачи могут предсказывать ту или иную болезнь у человека. Конечно, пока компьютер не может заменить специалиста, но даже в нынешнее время точность предсказаний довольно высока. Главное хорошо подготовить данные, настроить гиперпараметры и подобрать правильный алгоритм.

ПОСТАНОВКА ЗАДАЧИ

По заданию было необходимо выбрать какой-нибудь датасет и произвести над ним некоторые манипуляции:

1) Разметить данные (устранить выбросы, произвести фичеинжиниринг и тд.).
2) Посчитать разные статистики, посмотреть на корреляцию.
3) Выбрать метрику качества и реализовать её.
4) Реализовать и обучить логистическую регрессию или линейную регрессию.
5) Оценить качество модели на обучающей и тестовой выборках.
6) Построить графики ошибок
7) Выполнить пункты 4-6 с помощью библиотеки sklearn.

ОПИСАНИЕ ДАННЫХ

Я выбрал датасет Iris, который является довольно популярным при изучении ML. Он включает 5 числовых фичей и 1 категориальную. Относительно последней мы и будем производить предсказание.

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Выбросов в данных не обнаружилось, как и лишних столбцов, поэтому было необходимо лишь закодировать столбец Species.

ОПИСАНИЕ АЛГОРИТМА

Есть много алгоритмов классификации, которые работаю лучше, чем логистическая регрессия. Однако она довольно проста в реализации, поэтому я выбрал её.

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.

ПРЕДОБРАБОТКА ДАННЫХ

Как я писал ранее, датасет почти не нуждался в обработке. Я лишь провёл one hot encoding над категориальным признаком Species.

РЕАЛИЗАЦИЯ АЛГОРИТМА

```
def generate_batches(X, y, batch_size):
    assert len(X) == len(y)
    np.random.seed(42)
    X = np.array(X)
    y = np.array(y)
    perm = np.random.permutation(len(X))
    for i in np.arange(len(X)//batch_size):
        ind = perm[i * batch_size : (i + 1) * batch_size]
        yield X[ind], np.eye(3)[y[ind]]
```

```
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=400, lr=1e-7, batch_size=64, flag = False):
        n, k = X.shape
        self.num_classes = 3
        self.b = np.random.random(self.num_classes)
        if self.w is None:
            np.random.seed(2)
            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)
            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)
    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)
```

МЕТРИКИ КАЧЕСТВА

В задачах классификации мы имеем несколько основных метрик, которые строятся на основании матрицы ошибок:

	$y = 1$	$y = 0$
$\hat{y} = 1$	True Positive (TP)	False Positive (FP)
$\hat{y} = 0$	False Negative (FN)	True Negative (TN)

Интуитивно понятной, очевидной и почти неиспользуемой метрикой является accuracy — доля правильных ответов алгоритма:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Для оценки качества работы алгоритма на каждом из классов по отдельности введем метрики precision (точность) и recall (полнота).

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм.

Существует несколько различных способов объединить precision и recall в агрегированный критерий качества. F-мера — среднее гармоническое precision и recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

Код:

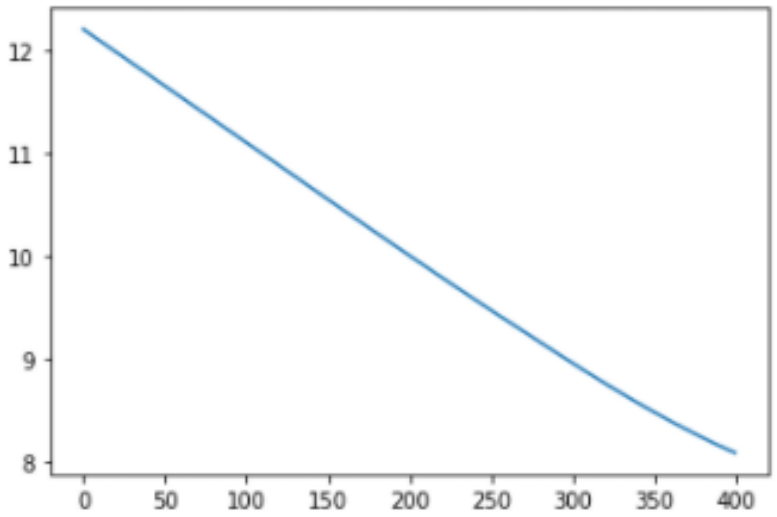
```
def compute_f1_score(y_true, y_pred):
    tp, tn, fp, fn = compute_tp_tn_fn_fp(y_true, y_pred)
    precision = compute_precision(tp, fp) / 100
    recall = compute_recall(tp, fn) / 100
    f1_score = (2*precision*recall) / (precision + recall + 0.001)
    return f1_score
```

Среднее гармоническое f1 обладает важным свойством — оно близко к нулю, если хотя бы один из аргументов близок к нулю. Именно поэтому оно является более предпочтительным, чем среднее арифметическое. f2 острее реагирует на recall, а f0.5 чувствительнее к точности. В данных условиях лучше учитывать обе метрики, а не отдавать предпочтение какой-то одной.

ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

Количество эпох было = 400. Коэффициент регуляризации l1 = 0.2, а l2 не использовался вовсе.

График изменения функции ошибки:



Однако результаты меня не порадовали. Как бы я не менял гиперпараметры, метрики выдавали 0. Обучив модель с помощью линейной регрессии из библиотеки sklearn, я получил хорошие результаты, но в своей реализации повторить их не смог.

ВЫВОДЫ

После выполнения курсового проекта я не был удовлетворён результатом. Моя реализация не отрабатывает хорошо в отличие от библиотеки sklearn. В чём проблема я так и не понял, т. к. до дедлайна оставалось очень мало времени. Возможно ошибка в разделении данных с помощью train_test_split(). Возможно какие-то гиперпараметры были подобраны не верно или проблема в самом алгоритме.

Так или иначе я узнал немного нового о машинном обучении, а также закрепил уже имеющиеся знания о метриках и алгоритмах ML.

В будущем я не планирую заниматься машинным обучением, однако, если придётся столкнуться с этой областью, то у меня уже будут базовые знания и разобраться будет легче.