Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

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

Студент: Кузьмичев А. Н.
Преподаватель: Вишняков Б.В.

Дата: 25.12.2021
Оценка:
Подпись:

Москва, 2021

1 Ознакомительная часть

Введение:

Постановка задачи:
Выбрать задачу (классификация или регрессия), датасет и метрику качества. Выбранные данные необходимо визуализировать и проанализировать. После этого выполнить препроцессинг. Затем реализовать алгоритм классификации, проверить качество обучения, сравнить с моделью из sklearn.

Описание данных:

Данные были получены из изображений, взятых с подлинных и поддельных образцов, похожих на банкноты. Для оцифровки использовалась промышленная камера, обычно используемая для проверки отпечатков. Конечные изображения имеют размер 400x400 пикселей. Благодаря объективу и расстоянию до исследуемого объекта были получены изображения в оттенках серого с разрешением около 660 dpi.

Информация об атрибутах:

1. Дисперсия изображения с вейвлет-преобразованием (непрерывная)
2. Асимметрия изображения с вейвлет-преобразованием (непрерывная)
3. Эксцесс изображения с вейвлет-преобразованием (непрерывный)
4. Энтропия изображения (непрерывная)
5. Класс (целое число)

Вариант:
Логистическая регрессия. Будем предсказывать класс (является ли купюра фейком, или нет).

2 Теоретическая часть

Логистическая регрессия

Логистическая регрессия - частный случай обобщенной линейной регрессии. Предполагается, что зависимая переменная принимает два значения и имеет биномиальное распределение. На практике логистическая регрессия используется для решения задач классификации с линейно-разделяемыми классами. Логистическая регрессия применяется для прогнозирования вероятности возникновения некоторого события по значениям множества признаков. Для этого вводится зависимая переменная y, принимающая значения 0 и 1 и множество независимых переменных $x_1, \dots, x_n$ на основе значений которых требуется вычислить вероятность принятия того или иного значения зависимой переменной.

Итак, пусть объекты задаются n числовыми признаками $f_j: X \to R, j = 1 \dots n$ и пространство признаковых описаний в таком случае $X = R^n$. Пусть Y $-$ конечное множество меток классов и задана обучающая выборка пар «объект-ответ» $X^m = \{(x_1, y_1), \dots (x_m, y_m)\}$.

Рассмотрим случай двух классов: $\boldsymbol{Y} = \{\boldsymbol{-1}, \boldsymbol{+1}\}$. В логистической регрессии строится линейный алгоритм классификации $a: X \to Y$ вида

$$\mathrm{a}(x, w) = sign\left(\sum_{j=1}^{n} w_j f_j(x) - w\_0\right) = sign < x, w >$$

где w_j – вес j-го признака, w_0 – порог принятия решения, $w = (w_0, \dots, w_n)$ – вектор весов, $< x, w >$ - скалярное произведение признакового описания объекта на вектор весов. Предполагается, что искусственно введён нулевой признак: $f_{0(x)} - 1$.

Задача обучения линейного классификатора заключается в том, чтобы по выборке X^m настроить вектор весов w. В логистической регрессии для этого решается задача минимизации эмпирического риска с функцией потерь специального вида:

$$Q(w) = \sum_{i=1}^{m} ln(1 + exp(-y_i < x_i, w >)) \to min\_w$$

После того, как решение **w** найдено, становится возможным не только вычислять классификацию $a(x) = sign(x, w)$ для произвольного объекта **x**, но и оценивать апостериорные вероятности его принадлежности классам:

$$P = \{y|x\} = \sigma(y < x, w >), y \in Y$$

где $\sigma(z) = \frac{1}{1+e^{-1}}$ - сигмоидная функция.

3 Практическая часть

Предобработка данных

Загрузим датасет

	0	1	2	3	4
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

Взглянем на размер данных и наличие пустых элементов

```
[ ] print("Dataframe shape:", notes.shape)
    notes.isnull().sum()
```

```
Dataframe shape: (1372, 5)
0    0
1    0
2    0
3    0
4    0
dtype: int64
```

А затем – на корреляцию между особенностями

```
plt.subplots(figsize=(7, 5))
notes_corr = notes.iloc[:, :-1].corr().abs()
sns.heatmap(notes_corr, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3875a10d50>
```

Исключим первый столбец, так как он имеет высокую корреляцию с двумя другими

```
[ ]  notes = notes.drop(
         columns=notes.columns[1], axis=1
     )
```

Разобьем выборку на тестовую и обучающую

```
from sklearn.model_selection import train_test_split


X = notes.iloc[:, :-1]
y = notes.iloc[:, -1]

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()

X_train, X_test, y_train, y_test = train_test_split(
    ss.fit_transform(X), y,
    test_size=0.3, random_state=10
)
```

Реализация алгоритма

```
class LogisticRegression:
    def __init__(self, *, reg_param=1.0, lr=0.01, max_iter=100):
        self._a  = reg_param
        self._lr = lr
        self._max_iter = max_iter
        self._feat_count = 0
        self._weights = None
        self._bias = 0

    def fit(self, feats, labels):
        if self._weights is None:
            self._feat_count = feats.shape[1]
            self._weights = np.zeros(self._feat_count)
        elif self._feat_count != feats.shape[1]:
            err = f"Feature count does not match previous count {self._fea
t_count}"
            raise ValueError(err)

        costs      = np.zeros((self._max_iter))
        precisions = np.zeros((self._max_iter))
        recalls    = np.zeros((self._max_iter))

        for i in range(self._max_iter):
            pred = self._predict(feats)
```

```
            costs[i] = self._cost(pred, labels)
            precisions[i] = precision(pred.round().astype(int), labels)
            recalls[i] = recall(pred.round().astype(int), labels)

            sample_count = len(labels)
            norm_coef = self._a
            coef = np.mean(pred - labels)

            dw = np.dot(feats.T, pred - labels) / sample_count + norm_coef
 * self._weights
            db = coef + norm_coef * self._bias

            self._weights -= dw * self._lr
            self._bias    -= db * self._lr

        return {
            "cost": costs,
            "precision": precisions,
            "recall": recalls,
            "max_iter": self._max_iter,
        }

    def predict(self, X):
        return self._predict(X).round()

    def _predict(self, feats):
        weights = self._weights
        bias    = self._bias
        return 1 / (1 + np.exp(-( np.dot(feats, weights) + bias)))

    def _cost(self, preds, labels):
        return -1 * np.mean(
            np.multiply(labels, np.log(preds)) \
            + np.multiply(1 - labels, np.log(1 - preds))
        )
```

Метрики

При обучении будут использоваться стандартные метрики:

- **Precision** (точность) - количество правильно классифицированных положительных предметов из выбранных для классификации:

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

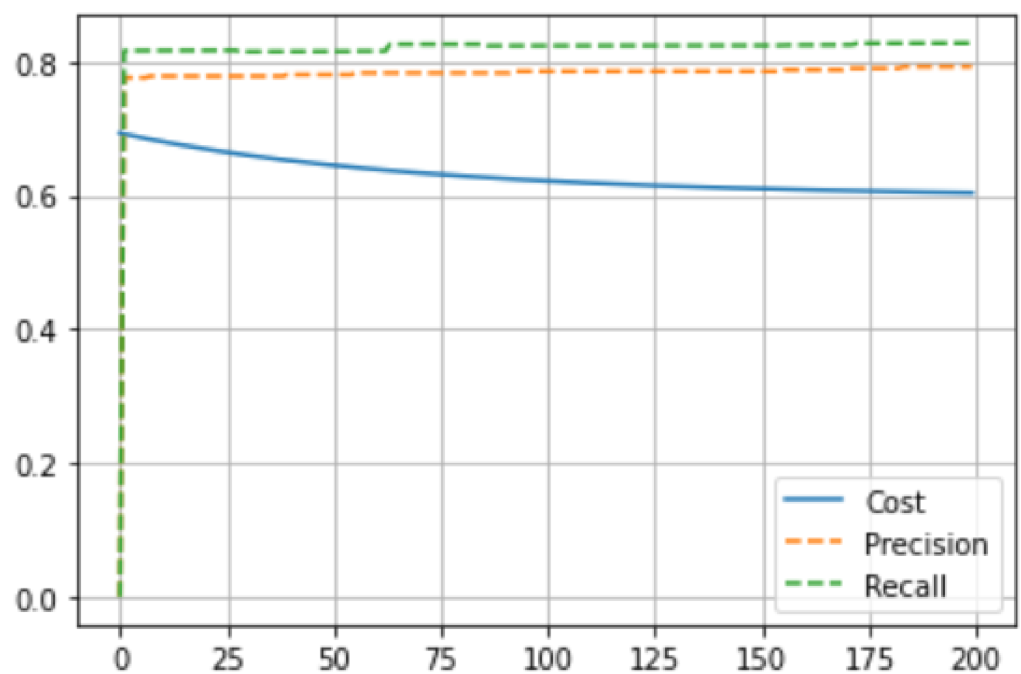
- **Recall** (полнота) - количество правильно классифицированных положительных предметов из всех возможных (т.е. множество правильно и неправильно классифицированных предметов):

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

TP - количество корректно классифицированных положительных предметов, FP - количество некорректно классифицированных отрицательных предметов, TP - количество некорректно классифицированных положительных предметов.

Можно также использовать **accuracy** (количество правильно классифицированных предметов из всех доступных), но её точность малозначима, так как классов неравное количество.

Результаты



	Алгоритм SKL	Алгоритм 1 (4 лр)	Алгоритм 2 (КП)
Recall	0.9015544041450777	0.8258426966292135	0.8305084745762712

Для оптимизации алгоритма обнулил reg_param, тем самым убрав L2 нормализацию.