СИИ Модуль 2 Отчет

Отчет по Модулю 2 курса "Систем искусственного интерфейса"

Подготовил Тюрин Иван Николаевич
группа P33102
Преподаватель Авдюшина Анна Евгеньевна

Лабораторная работа 1. Метод линейной регрессии

Введение

Выдается датасет по варианту и нужно научиться предсказывать некоторые численные (непрерывные) величиныпо данному датасету. Для этого нужно

- предобработать данные и проанализировать их,
- разработать модель линейной регрессии,
- разработать предобработчик входных данных (скалер),
- применить разработанные модели и скалеры,
- проанализировать результаты.

Описание метода

Суть метода линейной регрессии в том, чтобы апроксимировать некоторую эмпирическую зависимость линейной функцией лучшим образом.

Для этого нужно использовать некоторую метрику — удобнее всего подходит Метод наименьших квадратов, т.е. сумма квадратов отклонения теоретической зависимости от эмпирической по каждой точке.

Метрика минимизируется с помощью разных способов: итеративным умножением матриц, градиентным спуском, градиентным стохастических спуском и пр.
Также можно для управления получаемыми весами можно использовать методы регуляризации.

Псевдокод метода

```
class MyLinearRegression:
    def fit(X, y) -> Unit = save_weights((X^T • X)^{-1} • X^T • y)
    def predict(X) -> Array = X • get_weights()
```

Результаты выполнения

Результата оказались неплохими с учтом того, что явной линейной зависимости между признаками не наблюдается. Выходит результат очень зависит от исследуемой выборки. Так же стоит опасаться появления так называемой "авторегрессии" или "автокорелляции", т.е. ситуации когда признак начинает коррелировать с производным от него искусственно созданным признаком.

Примеры использования метода

Метод может быть полезен, когда в исследуемой выборке наблюдается отчетливая линейная зависимостью между признаками. Сферы применения:

- предсказаняи физических величин
- простая *аналитика*

Лабораторная работа 2. Метод k-ближайших соседей (k-NN)

Введение

Дан некоторый датасет, нужно научиться классифицировать элементы выборки по некоторому признаку, т.е. предсказывать некоторый категориальный признак. Для этого нужно

- обработать датасет и проанализировать его,
- реализовать модель k-ближайших соседей
- реализовать вычисления метрик (матрица ошибок)
- проверить работу классификатора

Описание метода

Метод заключается в том, что для рассматриваемого элемента определяется ближайшие к нему другие, ранее рассмотренные (при обучении) элементы по какой-то определенной метрике, например, L2 норме.

Псевдокод метода

```
def most_common(l: list) = max(set(l), key=l.count)
def euclidean(point, data) = sqrt(sum((point - data)^{2}))

class KNeighborsClassifier:
    def fit(X, y) = saveX(X), savey(y)

    def predict(X) -> Array:
        for i, point in enumerate(X):
            distances = metric(point, getX())
            neighbors[i] = most_common(sorted(distances))
        return neighbors

    def evaluate(X, y) -> float:
        y_pred = predict(X)
        accuracy = sum(y_pred == y) / len(y)
        return accuracy
```

Результаты выполнения

Метод вполне хорошо работает несмотря даже на слабую разделяемость данных визуально. Величина метрики ROC AUC не превыщает 0.7 на данном датасете.

Примеры использования метода

Метод может использоваться тогда, когда элементы выборки хорошо сгруппированы, то есть в общем выполняется условие малого удаления элементов одного класса друг от друга (по выбранной метрике). Сферы применения:

- медицина,
- психология,

- политика возможно,
- производство электроники.

Лабораторная работа 3. Деревья решений

Введение

Требуется научиться строить дерево решений и предсказывать бинарный класс элементов выборки по искуственно введенному признаку. Для этого нужно:

- обработать датасет и проанализировать его,
- ввести искусственный бинарный категориальный признак на основе имеющегося категориального (или численного дискретного),
- выбрать признаки которые будем использовать для построения дерева,
- реализовать классы использующиеся для построения структуры дерева,
- реализовать модель строющую решающее дерево и ищущую в ней класс для иследуемого элемента,
- реализовать функции для вычисления метрик модели,
- проверить качество полученной модели,
- проанализировать результаты.

Описание метода

Метод заключается в том, что мы для полученной выборки выбираем признак лучше всего разделяеющий ее: наибольший прирост информации (наименьшая взвешенная энтропия). Далее для узла в определенным признаком строим поддеревья для всех возможных значений этого признака в полученной выборке; процесс рекурсивный и каждого узла нового уровня выборка становится меньше. Энтропия же вычисляется через вероятности обнаружения определенного признака в подвыборке.

Псевдокод метода

```
class DTNode:
    INNER = 'inner'
    LEAF = 'leaf'

class DTInnerNode(DTNode):
    def __init__(feature: F, threasholds: list[F], children: list[DTNode], gain: float) -> Unit =
settype(DTNode.INNER)

class DTLeafNode(DTNode):
    def __init__(value: T) -> Unit = settype(DTNode.LEAF), setvalue(value)
```

```
class DecisionTreeClassifier:
    def __init__(min_samples: int > 0, max_depth: int > 0): Unit =
        set_min_samples(min_samples),
        set_max_depth(max_depth)

    def fit(X, y) -> Unit = set_tree_root(_build_tree(X, y))

    def predict(X) -> Array = [_predict(x, get_root()) for x in X.rows]

    def _build_tree(X, y) -> DTNode = _build_sub_tree(X, y, 1)

    def _build_sub_tree(X, y, curr_depth) -> DTNode:
        if len(X) > get_min_samples() and curr_depth < get_max_depth():
            feature, thresholds, children_datasets, gain = _best_split(X, y)
            if gain > 0: # gain can be 0 in worst case
```

```
                children = [_build_sub_tree(X_, y_, curr_depth + 1) for X_, y_ in children_datasets]
                return DTInnerNode(feature, thresholds, children, gain)
        return DTLeafNode(_leaf_value(y))

    def _leaf_value(self, y: pd.Series) = max(y.value_counts())

    def _best_split(self, X, y) -> tuple[F, list[V[F]], float, list[pd.DataFrame]]:
        best_split = None

        for f in X.features:
            thresholds = unique(f.values)
            children_datasets = [X[map], y[map] for t in thresholds if map := (X[f].value == t)]
            if len(children_datasets) < 2: continue
            info_gain = _info_gain(y, [y for X, y in children_datasets])
            best_split = max(best_split, new_split(f, info_gain, children_datasets, thresholds))
        return best_split

    def _info_gain(y_parent, y_children) -> float:
        parent_entropy = _entropy(y_parent)
        weighted_entropy = 0
        for y_child in y_children:
            weight = len(y_child) / len(y_parent)
            weighted_entropy += weight * _entropy(y_child)
        return parent_entropy - weighted_entropy

    def _entropy(y) -> float:
        prob = count_unique(y) / len(y)
        entropies = -prob * np.log2(prob)
        return sum(entropies)

    def _predict(x, node: DTNode) -> Target:
        if node.nodetype == DTNode.LEAF: return node.value
        else: # node.nodetype = DTNode.INNER
        for i, t in enumerate(node.threasholds):
            if check_val == t:
                return _predict(x, node.children[i])
        return _predict(x, random.choice(node.children))
```

Результаты выполнения

Разработанная модель умеет достаточно хорошо классифицировать, но результат сильно зависит от выбора признаков по которым строится дерево. Иногда бывает, что в тренировочной выборке не представлены значения признака из тестовой выборки, в таких случаях модель делает случайное предсказание; стоит делать обучающие выборки более репрезентативными.

Примеры использования метода

Метод решающих деревьев сам по себе очень приметивен и склонен к переобучению без дополнительных ограничений. Дерево которое строится в данном задании особенно склонно переобучаться, т.к. численные признаки используются как категориальные и количество признаков довольно мало. Тем не менне при использовании нескольких решающих деревьев можно получит значительные результаты и решать некоторые другие задачи: случайный лес или бустинг.
В таких случаях можно получать показатели лучше чем некоторые другие модели и при этом затрачивая сравнительно небольшие ресурсы.
Бустинг применяется сейчас повсеместно, почти никто не строит одиночные деревья при решении бизнес задач таких, например, как:

- кредитование,
- ...

Лабораторная работа 4. Логистическая регрессия

Введение

В задании требуется проанализировать уже рассмотренный в предыдущей лабораторной работе датасет, по которому нужно так же научиться классифицировать элементы выборки по определенному признаку с использованием логистической регрессии. Для этого нужно:

- обработать полученный датасет (в том числе устранить невалидные 0), проанализировать этот датасет,
- реализовать модель логистической регрессии (с градиентным спуском),
- протестировать модель по выбранным метрикам,
- проанализировать результаты.

Описание метода

Суть метода состоит в том, чтобы найти вероятность принадлежности элемента к одному из классов (0, 1). В отличии от обычной регрессии мы теперь считаем, что функция теперь выдает некоторую нелинейную вероятность принадлежности элемента классу 1. Вероятность определяется с помощью функции активации — сигмоиды. Метрика по которой мы оптимизируем наше предсказание вычисляется как логистический лосс, т.к. из функции активации.

Псевдокод метода

Предсказания:

$$y_{pred}(x, w) = \frac{1}{1 + e^{-\langle x, w \rangle}}$$

Лосс (LogLoss):

$$L(w) = -y \log y_{pred} - (1 - y) log (1 - y_{pred})$$

Градиент:

$$\frac{\partial L}{\partial w} = \left(-\frac{y}{y_{pred}} + \frac{1 - y}{1 - y_{pred}} \right) \frac{\partial y_{pred}}{\partial w}$$

$$\frac{\partial y_{pred}}{\partial w} = \frac{-1}{(1 + e^{-\langle x, w \rangle})^2} e^{-\langle x, w \rangle}(-x) = y_{pred}(1 - y_{pred})x$$

$$\frac{\partial L}{\partial w} = (y_{pred} - y)x$$

```
def sigmoid(h) = 1.0 / (1 + exp(-h))
class LogisticRegression:
    def fit(X, y, max_iter: int > 0, lr: float > 0) -> Unit:
        for _ in range(max_iter):
            grad = ((X^{T}) • (sigmoid(X^{T} • get_weights())) - y) / len(y)
            set_weights(get_weights() - grad * lr)
    def predict_proba(X): Array = sigmoid(X • get_weights())
    def predict(X, threshold: float=0.5): Array = predict_proba(X) >= threshold
```

Результаты выполнения

Разработанная модель справляется довольно хорошо, учитывая то, что элементы выборки слаборазделимы видуально. Качество модели зависит от количества итераций обучения, но судя по опыту

увеличивать число итераций обучения не особо имеет смысл после определенного числа, т.к. после этого прирост качества остается незначительным.

Примеры использования метода

Метод логистической регрессии может быть полезен в любом случае, когда требуется получить вероятность в бинарной классификации. Возможно он лучше других годится тогда, когда нет возможности выстроить какие-то осмысленные связи между признаками, как в решающих деревьях.

Сравнение методов

Сравнительный анализ методов

Рассмотренные методы машинного обучения отличаются задачами для которых их применяют и соответственно результатами которые они выдают при обучении. Реализованные модели, в принципе, одинакого ресурсозатратные. На данных выборках они все показывают сопостовимые результаты с библиотечными аналогами и эти результаты далеки от идеальных, тем не менне довольно хороши. Кроме того редко используются исключительно модель подобной реализации в задании: чаще используются разлиные оптимизации и композиции алгоритмов.

Примеры лучшего использования каждого метода

Метод линейной регрессии будет хорош при предсказании некоторой линейной зависимости, о существовании заранее известно, например, из законов физики. Поэтому их стоит использовать там, где уже есть хорошие математические модели: физика,

Методы классификации будет удобно использовать там, где есть хорошая разделимость данных на группы похожих (визуальная различимость на графиках). То есть в статистике, контроле качества продукции и пр.

Заключение

В результате выполнения лабораторных работ были изучены различные методы машинного обучения и их применения. Для некоторых из них реализованы модели в виде кода на Python в соответствии с заданием. Так же был получен опыт в обработке данных для последующего применения их для обучения моделе, в том числе устранение невалидных значений и добавление искусственных признаков.

Приложения

Весь код используемый при выполнении лабораторных работ представлен в личном репозитории на платформе GitHub:

- https://github.com/e1turin/itmo-ai-systems