Машинное обучение: задание 2

Составитель: Виктор Кантор

1 Теоретические задачи

30% баллов за задание

1.1 Ответы в листьях регрессионного дерева

Какая стратегия поведения в листьях регрессионного дерева приводит к меньшему матожиданию ошибки по MSE: отвечать средним значением таргета на объектах обучающей выборки, попавших в лист, или отвечать таргетом для случайного объекта из листа (считая все объекты равновероятными)?

1.2 Линейные модели в деревьях

Одна из частых идей — попытаться улучшить регрессионное дерево, выдавая вместо константных ответов в листьях ответ линейной регрессии, обученной на объектах из этого листа. Как правило такая стратегия не дает никакого ощутимого выигрыша. Попробуйте объяснить, почему? Как стоит модифицировать построение разбиений в дереве по MSE, чтобы при разбиении получались множества, на которых линейные модели должны работать неплохо?

1.3 Unsupervised decision tree

Unsupervised решающие деревья можно было бы применить для кластеризации выборки или оценки плотности, но проблема построения таких деревьев заключается в введении меры информативности. В одной статье предлагался следующий подход — оценивать энтропию множества S по формуле:

$$H(S) = \frac{1}{2} \ln \left((2\pi e)^n |\Sigma| \right)$$

Здесь Σ — оцененная по множеству матрица ковариаций. Т.е. не имея других сведений, в предложенном подходе мы по умолчанию считаем, что скопления точек можно приближенно считать распределенными нормально. Убедитесь, что это выражение в самом деле задает энтропию многомерного нормального распределения.

2 Применение решающего дерева

20% баллов за задание, оценочное время выполнения 30 минут + установка GraphViz

Постройте решающее дерево из sklearn на датасете german credit data из UCI репозитория и визуализируйте его. Попробуйте проинтерпретировать первые несколько разбиений, изучив описание признаков. Постройте графики зависимости качества на кросс-валидации и на обучающей выборке от глубины дерева

3 Реализация решающего дерева (опциональная часть)

50% баллов за задание, оценочное время выполнения 3-4 часа

В этом задании предлагается использовать датасет boston из sklearn.datasets. Оставьте последние 25% объектов для контроля качества, разделив X и y на X_train, y_train и X_test, y_test.

Реализуйте свой класс DecisionTree, имеющий методы fit и predict, позволяющие соответственно обучить решающее дерево по матрице признаков X_train и ответам y_train, а затем спрогнозировать ответы на тестовой выборке X_test. Оцените качество работы вашего дерева на тестовой выборке.

Рекомендации по реализации дерева:

1. Обучение дерева можно реализовать простым жадным рекурсивным алгоритмом — каждый раз выбирайте наилучшее разбиение (номер признака и порог по нему) по уместному на ваш взгляд критерию из рассмотренных на первой лекции о решающих деревьях (MSE, gini, энтропийный критерий, ошибка классификации)

2. Выбор наилучшего разбиения можно сделать простым перебором по признакам и порогам.

3. Пороги можно перебирать из заранее заданного множества порогов на обучающей выборке - например, взяв все пороги между принимаемыми значениями координат, либо взяв случайный набор порогов, либо взяв пороги по квантилям значений каждого признака (посчитать квантили будет несложно с помощью scipy). Если возможных порогов будет слишком много, выбор наилучшего разбиения может оказаться слишком долгой операцией.

4. Сделайте возможным передавать в конструктор класса ограничение по глубине дерева и заканчивайте построение дерева при достижении этого ограничения.

5. Можно реализовать отдельный класс для решающего правила вида «k-ый признак меньше порога» и отдельный класс для дерева. Также вам предстоит подумать, как хранить разбиения внутри дерева, чтобы их было удобно использовать.

6. Какие-то из решений вы можете подсмотреть в чужих реализациях дерева, но от вас не требуется написать применимую на практике библиотеку - только максимально простую демонстрацию того, как строится и применяется решающее дерево.

Примечание: на случай, если эта задача покажется большой, — среди студентов ФИВТ МФТИ были примеры ее выполнения за одну пару с написанием в процессе хорошего, продуманного, понятного и задокументированного кода.