Лабораторная работа «Деревья решений»

Цель работы: получить практические навыки работы с методом деревьев решений на практических примерах с использованием языка программирования python

Теоретический материал

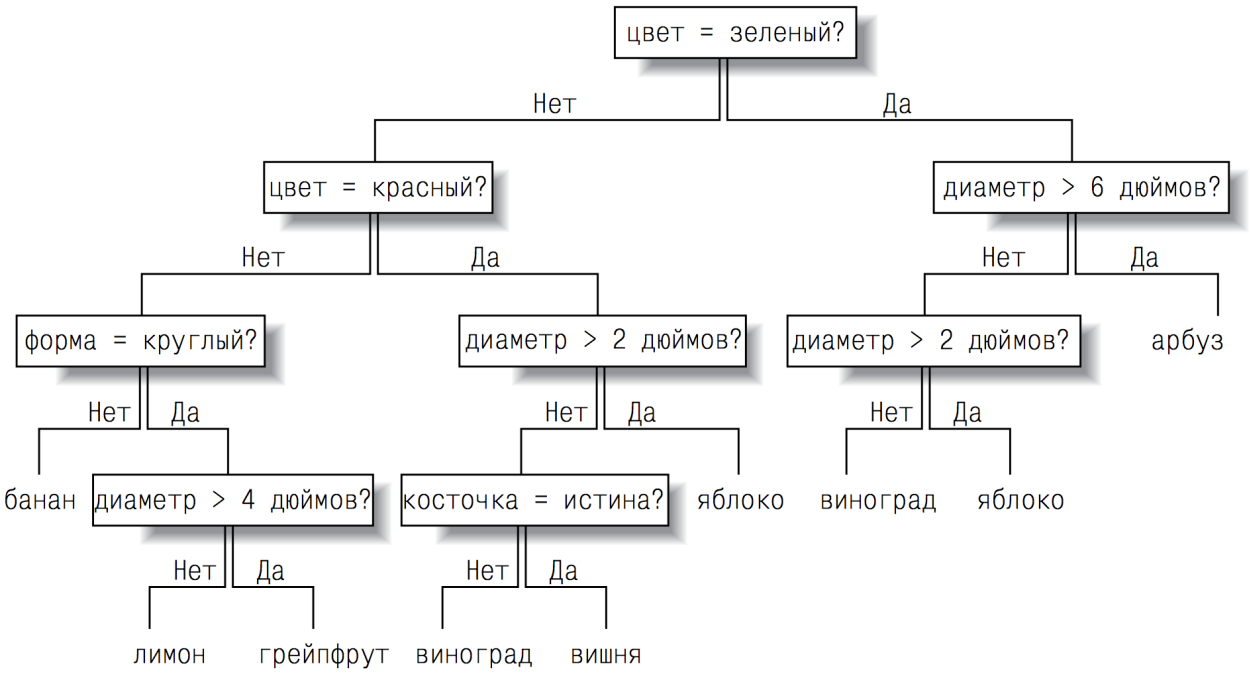
1. Деревья решений

Дерево решений - метод автоматического анализа данных для построения классификационных и регрессионных моделей, является как методом извлечения, так и одновременно методом представления данных.

Дерево решений представляет способ представления правил в иерархической, последовательной структуре, где каждому объекту соответствует единственный узел, дающий решение. Под правилом понимается логическая конструкция, представленная в виде "если ... то ...".

Таким образом, у дерева решений есть два вида отображения:

1) в виде графической структуры «дерево»
2) в виде списка правил «если …, то …»



Область применения деревья решений в настоящее время широка, но все задачи, решаемые этим аппаратом, могут быть объединены в следующие два класса:

Классификация: Деревья решений отлично справляются с задачами классификации, т.е. отнесения объектов к одному из заранее известных классов. Целевая переменная должна иметь дискретные значения.

Регрессия: Если целевая переменная имеет непрерывные значения, деревья решений позволяют установить зависимость целевой переменной от независимых(входных) переменных. Например, к этому классу относятся задачи численного прогнозирования(предсказания значений целевой переменной).

Алгоритмы построения дерева решений

Пусть нам задано некоторое обучающее множество T, содержащее объекты (примеры), каждый из которых характеризуется m атрибутами (атрибутами), причем один из них указывает на принадлежность объекта к определенному классу.

Идею построения деревьев решений из множества T, впервые высказанную Хантом, приведем по Р. Куинлену (R. Quinlan).

Пусть через $\{C_1, C_2, \ldots C_k\}$ обозначены классы (значения метки класса), тогда существуют 3 ситуации:

1. множество T содержит один или более примеров, относящихся к одному классу C_k. Тогда дерево решений для T – это лист, определяющий класс C_k;
2. множество T не содержит ни одного примера, т.е. пустое множество. Тогда это снова лист, и класс, ассоциированный с листом, выбирается из другого множества отличного от T, скажем, из множества, ассоциированного с родителем;
3. множество T содержит примеры, относящиеся к разным классам. В этом случае следует разбить множество T на некоторые подмножества. Для этого выбирается один из признаков, имеющий два и более отличных друг от друга значений $O_1, O_2, \ldots O_n$. T разбивается на подмножества $T_1, T_2, \ldots T_n$, где каждое подмножество T_i содержит все примеры, имеющие значение O_i для выбранного признака. Это процедура будет рекурсивно продолжаться до тех пор, пока конечное множество не будет состоять из примеров, относящихся к одному и тому же классу.

Вышеописанная процедура лежит в основе многих современных алгоритмов построения деревьев решений, этот метод известен еще под названием разделения и захвата (divide and conquer). Очевидно, что при использовании данной методики, построение дерева решений будет происходит сверху вниз.

Поскольку все объекты были заранее отнесены к известным нам классам, такой процесс построения дерева решений называется обучением с учителем (supervised learning). Процесс обучения также называют индуктивным обучением или индукцией деревьев (tree induction).

На сегодняшний день существует значительное число алгоритмов, реализующих деревья решений CART, C4.5, NewId, ITrule, CHAID, CN2 и т.д. Но наибольшее распространение и популярность получили следующие два:

CART (Classification and Regression Tree) – это алгоритм построения бинарного дерева решений – дихотомической классификационной модели. Каждый узел дерева при разбиении имеет только двух потомков. Как видно из названия алгоритма, решает задачи классификации и регрессии.

C4.5 – алгоритм построения дерева решений, количество потомков у узла не ограничено. Не умеет работать с непрерывным целевым полем, поэтому решает только задачи классификации.

Большинство из известных алгоритмов являются "жадными алгоритмами". Если один раз был выбран атрибут, и по нему было произведено разбиение на подмножества, то алгоритм не может вернуться назад и выбрать другой атрибут, который дал бы лучшее разбиение. И поэтому на этапе построения нельзя сказать даст ли выбранный атрибут, в конечном итоге, оптимальное разбиение.

Правила разбиения

Для построения дерева на каждом внутреннем узле необходимо найти такое условие (проверку), которое бы разбивало множество, ассоциированное с этим

узлом на подмножества. В качестве такой проверки должен быть выбран один из атрибутов. Общее правило для выбора атрибута можно сформулировать следующим образом: выбранный атрибут должен разбить множество так, чтобы получаемые в итоге подмножества состояли из объектов, принадлежащих к одному классу, или были максимально приближены к этому, т.е. количество объектов из других классов ("примесей") в каждом из этих множеств было как можно меньше.

Были разработаны различные критерии, но мы рассмотрим только два из них:

Теоретико-информационный критерий (энтропия)

Для выбора наиболее подходящего атрибута, предлагается следующий критерий:

$$\mathrm{Gain}(X) = \mathrm{Info}(T) - \mathrm{Info}_X(T)$$

где, Info(T) – энтропия множества T, а

$$\mathrm{Info}_X(T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} * Info(T_i)$$

Множества T_1, T_2, ... Tn получены при разбиении исходного множества T по проверке X. Выбирается атрибут, дающий максимальное значение по критерию (1).

Впервые эта мера была предложена Р. Куинленом в разработанном им алгоритме ID3.

Статистический критерий

Алгоритм CART использует так называемый индекс Gini (в честь итальянского экономиста Corrado Gini), который оценивает "расстояние" между распределениями классов.

$$Gini(c) = 1 - \sum_j p_j^2$$

Где c – текущий узел, а p_j – вероятность класса j в узле c.

CART был предложен Л.Брейманом (L.Breiman) и др.

Правила остановки

В дополнение к основному методу построения деревьев решений были предложены следующие правила:

1. Использование статистических методов для оценки целесообразности дальнейшего разбиения, так называемая "ранняя остановка" (prepruning). В конечном счете "ранняя остановка" процесса построения привлекательна в плане экономии времени обучения, но здесь уместно сделать одно важное предостережение: этот подход строит менее точные классификационные модели и поэтому ранняя остановка крайне нежелательна. Признанные авторитеты в этой области Л.Брейман и Р. Куинлен советуют буквально следующее: "Вместо остановки используйте отсечение».
2. Ограничить глубину дерева. Остановить дальнейшее построение, если разбиение ведет к дереву с глубиной превышающей заданное значение.
3. Разбиение должно быть нетривиальным, т.е. получившиеся в результате узлы должны содержать не менее заданного количества примеров.

Этот список эвристических правил можно продолжить, но на сегодняшний день не существует такого, которое бы имело большую практическую ценность. К этому вопросу следует подходить осторожно, так как многие из них применимы в каких-то частных случаях.

Правила отсечения

Очень часто алгоритмы построения деревьев решений дают сложные деревья, которые "переполнены данными", имеют много узлов и ветвей. Такие "ветвистые" деревья очень трудно понять. К тому же ветвистое дерево, имеющее много узлов, разбивает обучающее множество на все большее количество подмножеств, состоящих из все меньшего количества объектов.

Ценность правила, справедливого скажем для 2-3 объектов, крайне низка, и в целях анализа данных такое правило практически непригодно. Гораздо предпочтительнее иметь дерево, состоящее из малого количества узлов, которым бы соответствовало большое количество объектов из обучающей выборки. И тут возникает вопрос: а не построить ли все возможные варианты деревьев, соответствующие обучающему множеству, и из них выбрать дерево с наименьшей глубиной? К сожалению, это задача является NP-полной, это было показано Л. Хайфилем (L. Hyafill) и Р. Ривестом (R. Rivest), и, как известно, этот класс задач не имеет эффективных методов решения.

Для решения вышеописанной проблемы часто применяется так называемое отсечение ветвей (pruning).

Пусть под качеством дерева решений понимается отношение правильно классифицированных объектов при обучении к общему количеству объектов из обучающего множества, а под ошибкой – количество неправильно классифицированных. Предположим, что нам известен способ оценки ошибки дерева, ветвей и листьев. Тогда, возможно использовать следующее простое правило:

1. построить дерево;
2. отсечь или заменить поддеревом те ветви, которые не приведут к возрастанию ошибки.

В отличии от процесса построения, отсечение ветвей происходит снизу-вверх, двигаясь с листьев дерева, отмечая узлы как листья, либо заменяя их поддеревом.

Хотя отсечение не является панацеей, но в большинстве практических задач дает хорошие результаты, что позволяет говорить о правомерности использования подобной методики.

2. Разработка и тестирование моделей

В целом процесс решения задачи машинного обучения можно представить следующими этапами:

1. Определить решаемую задачу и представить ее в терминах машинного обучения
2. Получить и исследовать данные
3. Подготовить данные для алгоритмов машинного обучения
4. Разделить данные на обучающую и валидационную выборку
5. Попробовать различные модели на обучающей выборке и подобрать для них гиперпараметры
6. Проверить полученное решение на валидационной выборке и оформить его

Для решения поставленных задач в работе будут использоваться следующие библиотеки языка python:

- *pandas* – библиотека для работы с данными в табличном представлении
- *matplotlib* – библиотека для построения графиков и диаграмм
- *scikit-learn* – библиотека, реализующая методы и средства машинного обучения
- *jupiter notebook или Pycharm CE* – средства для написания/отладки кода

Также, для выполнения дополнительных заданий вам могут понадобиться библиотеки: *numpy* (эффективная работа с матрицами), *seaborn* (визуализация), *pydot* (сохранение структуры дерева в файл).

Рассмотрим этапы машинного обучения подробнее в контексте с используемыми библиотеками.

Определить решаемую задачу

Для решаемой задачи определить ее предметную область (медицина, финансы и т.д.) и выполнить содержательную постановку задачи. Например, разработать алгоритм прогноза стоимости дома в зависимости от его географических, строительных и других параметров.

После содержательной постановки можно формализовать задачу в терминах машинного обучения:

- определить класс задачи (классификация, регрессия, кластеризация, ранжирование и т.д.), в данной работе рассматриваются только задачи классификации
- выделить исходные (входные) признаки
- определить целевую переменную

Получить и исследовать данные

Загрузить данные с использованием pandas для их исследования. Для загрузки текстовых сепарабельных файлов можно использовать метод read_csv. Например:

```
data = pd.read_csv("путь до файла", sep=",", header=0)
```

где sep – это используемый разделитель в данных, header=0 – указывает, что в 0 строке указаны заголовки таблицы.

После загрузки данных, выполните следующие операции по изучению данных:

1) изучите каждую переменную и ее характеристики
   a) имя
   b) тип (вещественная, категориальная, текстовая и т.д.).
      Можно воспользоваться методом `describe()` в pandas.
   c) гистограмма и тип распределения переменной.
      Можно воспользоваться встроенной функцией `hist()` в pandas. Например:
      `data["feature"].hist()`
   d) % пропущенных значений (метод `isnull()` в pandas)
   e) для целевой переменной также нужно рассмотреть баланс классов (метод `value_counts()` в pandas)
2) визуализируйте данные, например: постройте диаграммы рассеяния для пар признаков, "ящики с уами" (box plot) для различных признаков и целевой пременной и т.д. Для визуализации можно использовать встроенные функции в pandas: `plot()` и `hist()`, библиотеку seaborn или matplotlib.
3) изучите корреляции между переменными (метод `corr()` в pandas)
4) подумайте, как бы вы могли решить задачу вручную.

Подготовить данные

В данной работе на этапе подготовки данных могут быть использованы следующие процедуры:

1) *восстановление пропусков.* Общая стратегия при заполнении пропусков – заполнить их такими значениями, чтобы алгоритм мог их идентифицировать как пропуски, например, значением, которое не встречается в обучающей выборке для данного признака. В pandas это можно сделать следующим образом:
   `data["feature"].fillna(0, inplace=True)`
   В данном случае мы заполнили все пропущенные значения нулями.
2) *кодирование категориальных переменных числовыми.* Для этого можно использовать `OneHotEncoder` (предпочтительнее) и `LabelEncoder` из библиотеки. `OneHotEncoder` для каждого значения признака создает новый

признак в данных, LabelEncoder просто присваивает каждому категориальному признаку числовому значение.

Пример использования OneHotEncoder:

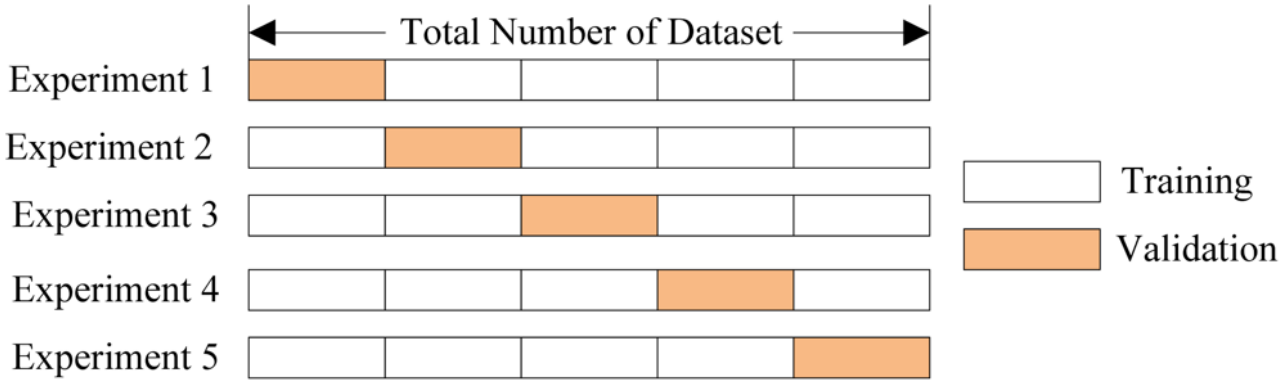
```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
enc.fit_transform([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])
```

Также на данном этапе можно исследовать и убирать выбросы из данных, придумывать новые признаки на основании имеющихся (feature engineering), балансировать классы в выборке и т.д., но в данной работе эти процедуры не рассматриваются.

Разбиение на обучающую и тестовую выборку

Главная задача обучаемых алгоритмов – их способность обобщаться, то есть хорошо работать на новых данных. Поскольку на новых данных мы сразу не можем проверить качество построенной модели, то надо пожертвовать небольшой порцией данных, чтоб на ней проверить качество модели. В работе будем использовать процедуру кросс-валидации или *K-fold кросс-валидации*.

В данном случае модель обучается K раз на (K-1) подвыборках исходной выборки (белый цвет), а проверяется на одной подвыборке (каждый раз на разной, оранжевый цвет). Получается K оценок качества модели, которые обычно усредняются, выдавая среднюю оценку качества классификации на кросс-валидации.



Кросс-валидация дает лучшую по сравнению с отложенной выборкой оценку качества модели на новых данных. Но кросс-валидация вычислительно дорогостоящая, если данных много.

Кросс-валидация – очень важная техника в машинном обучении (применяемая также в статистике и эконометрике), с ее помощью выбираются гиперпараметры моделей, сравниваются модели между собой, оценивается полезность новых признаков в задаче и т.д. Более подробно про кросс-валидацию можно почитать в источниках, приведенных в списке литературы.

В работе будем использовать реализацию процедуры кросс-валидации из библиотеки skLearn в виде функции cross_val_score. Пример использования:

```
cross_val_score(clf, X, y, cv=5)
```

где clf – обучаемый классификатор, X – исходные признаки, y – целевая переменная, cv=5 – разбиение на 5 фолдов.

Обучение и тестирование моделей, подбор гиперпараметров

С использованием процедуры кросс-валидации будем обучать деревья решений. В работе используется реализация деревьев решений для классификации DecisionTreeClassifier из библиотеки skLearn.

В качестве метрики качества разрабатываемых моделей в работе используется доля правильно классифицированных объектов (accuracy).

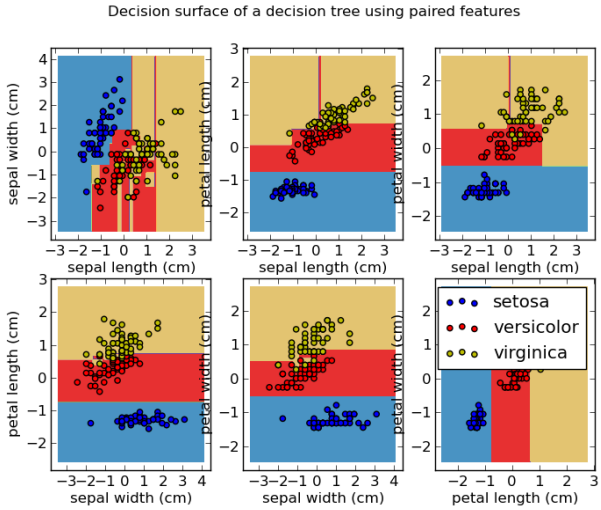
Пример обучения и тестирования дерева решений:

```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
```

```
clf = DecisionTreeClassifier(max_depth=10,max_features=3,
                             criterion=”gini”, random_state=0)
scores = cross_val_score(clf, data, target, cv=5)
print(scores.mean())
```

где max_depth – максимальная глубина дерева решений, max_features – максимальное количество признаков, используемое для выбора нового узла, criterion – критерий расщепления, random_state – фиксация стохастических параметров метода.

Также, структуру построенных деревьев можно вывести в файл pdf (необходима библиотека pydot), а для прогнозов дерева построить разделяющую поверхность, пример которой приведен на рисунке. С этими способами визуализации предлагается разобраться самостоятельно.



3. Настройка окружения

Необходимую сборку python (Anaconda) со всеми библиотеками для анализа данных и машинного обучения можно установить, скачав со страницы https://docs.continuum.io/anaconda/install/. При использовании библиотеки seaborn для визуализации, ее нужно будет поставить дополнительно.

Для написания и отладки кода можно использовать IDE Pycharm CE (https://www.jetbrains.com/pycharm/download/) или Jupiter notebook,которые позволяют запускать и отлаживать код в браузере (http://jupyter.org/).

4. Ход работы:

1) Изучить теоретический материал по методу деревьев решений.
2) Выполнить предварительный анализ структуры исходных данных согласно варианту. Для анализа необходимо использовать средства библиотеки *pandas*:
   a) Определить количество записей в выборке
   b) Описать исходные признаки, их шкалы, построить гистограммы
   c) Описать целевой признак, его варианты значений, построить гистограмму
   d) Определить класс задачи: классификация или регрессия
   e) Определить наличие пропусков в данных

* По желанию можно сделать различную визуализацию для исходных и прогнозируемых признаков (можно использовать библиотеки matplotlib, встроенные функции pandas, библиотеку seaborn).

3) Заполнить пропуски в данных при их наличии (подходы к заполнению пропусков описаны в теоретической части).
4) Закодировать категориальные признаки числовыми значениями (описано в теоретической части).
5) Провести серию экспериментов по построению и тестированию деревьев решений. Для валидации использовать схему разбиения выборки на 5 частей (фолдов) и функцию cross_val_score из библиотеки sklearn. В качестве метрик

качества использовать долю правильно классифицированных объектов (accuracy).

a) Зафиксируйте для всех экспериментов random_state=11, чтобы получать воспроизводимые результаты эксперименты. Если не зафиксировать значение параметра, то даже при одинаковых исходных данных всегда будут получаться разные результаты в силу стохастичности метода.
b) Постройте и протестируйте на кросс-валидации две модели с параметрами по умолчанию, отличающиеся критериями разбиения: gini и entropia (параметр criterion). Выберите лучший результат и зафиксируйте его для дальнейших экспериментов.
c) Проведите эксперименты по максимальному количеству признаков, используемых при построении дерева для выбора в конкретном узле: от 2 до количества исходных признаков в вашем варианте (параметр max_features).
d) Проведите эксперименты по оптимизации глубины дерева: от 2 до разумного значения согласно вашей задачи, например, 40 (параметр max_depth).

* По желанию можно оптимизировать и другие параметры дерева решений, отвечающие за количество элементов выборки в узле дерева.

e) * Для полученного оптимального дерева и дерева решений глубины 3 (остальные параметры произвольные) визуализируйте структуру дерева (используется метод `export_graphviz` библиотеки sklearn и утилита graphviz, нужно поставить отдельно).

6) Сформулируйте выводы по работе.

Структура отчета:

1. Титульный лист
2. Описание структуры исходных данных и задачи в терминах предметной области и машинного обучения
3. Результаты предварительного анализа и визуализации исходных признаков и целевой переменной
4. Результаты построения и тестирования деревьев решений
5. Выводы по работе
6. Листинг программы для построения и тестирования деревьев решений

Вопросы к защите:

1. Виды визуализации деревьев решений.
2. Алгоритм построения дерева решений.
3. Каким образом происходит выбор признака для разделения?
4. Каким образом происходит останов при построении дерева?
5. Зачем необходимо отсечение ветвей дерева?
6. Процедура кросс-валидации.
7. Восстановление пропусков и кодирование категориальных переменных.

Список литературы:

Про python:

1. Курс на Codeacademy https://www.codecademy.com/learn/learn-python
2. Курсы на Stepic: https://stepik.org/course/67/ (основы) и https://stepik.org/course/512 (применение)
3. Программирование на python, хорошие видео-лекции: https://compscicenter.ru/courses/python/2015-autumn/classes/

Про python и анализ данных:

4. Кросс-валидация (англ) https://sebastianraschka.com/blog/2016/model-evaluation-selection-part1.html

5. Кросс-валидация http://www.machinelearning.ru/wiki/index.php?title=Кросс-валидация
6. Энтропия и деревья решений https://habrahabr.ru/post/171759/
7. Деревья классификации и регрессии http://www.williamspublishing.com/PDF/978-5-8459-1170-4/part.pdf
8. Введение в pandas: анализ данных с помощью python https://khashtamov.com/ru/pandas-introduction/
9. Первичный анализ данных с pandas https://habrahabr.ru/company/ods/blog/322626/
10. Работа с Jupiter http://devpractice.ru/python-lesson-6-work-in-jupyter-notebook/