Лабораторная работа «Деревья решений»

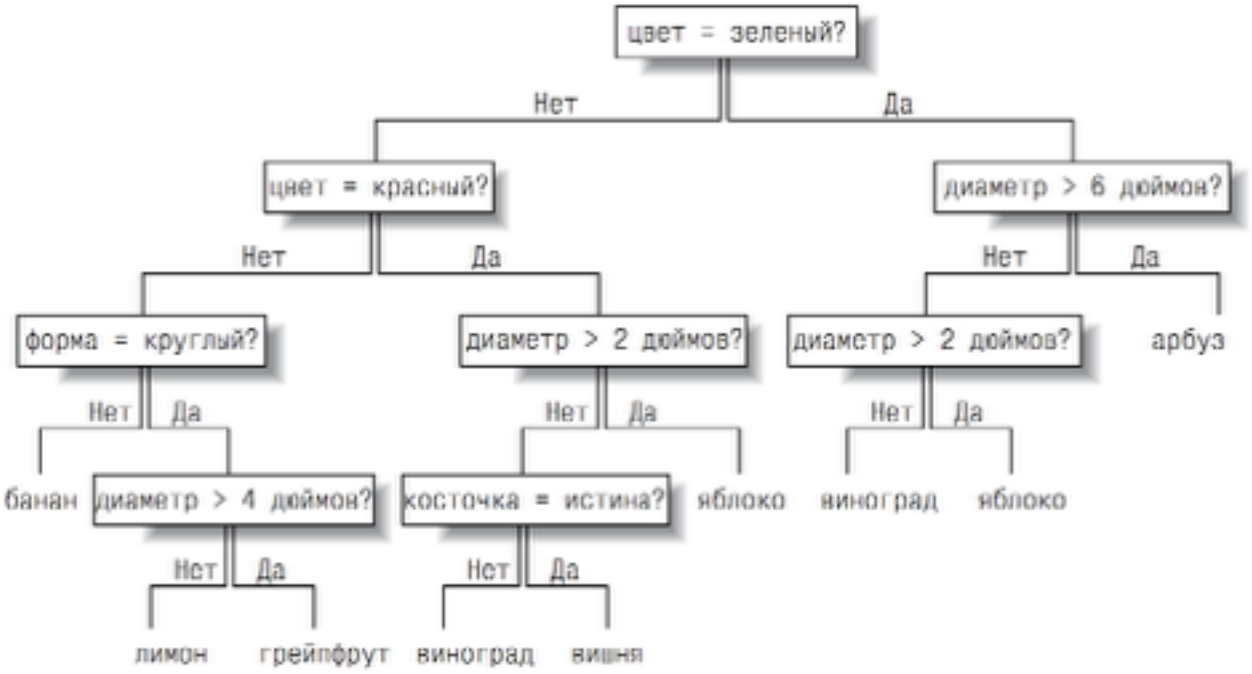
Цель работы: получить практические навыки работы с методом деревьев решений на практических примерах с использованием языка программирования python

Теоретический материал

1. Основные понятия

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).

2. 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 $\{C_1, C_2, ... C_k\}$ 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, ... O_n. T разбивается на подмножества T_1, T_2, ... 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.

3. 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
Алгоритм C4.5, усовершенствованная версия алгоритма ID3 (Iterative Dichotomizer), использует теоретико-информационный подход. 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 P. Куинленом в разработанном им алгоритме ID3. Кроме вышеупомянутого алгоритма C4.5, есть еще целый класс алгоритмов, которые используют этот критерий выбора атрибута.

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.

4. Правило 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 P. 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.

4. 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 точностью (распознавания) 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.

Настройка окружения

1. Установить Python версии 2.7.x
2. Устанавливаем среду разработки Pycharm CE
3. Устанавливаем пакет методов интеллекутального анализа данных scikitLearn (http://scikit-learn.org)

Деревья решений в scikitLearn

DecisionTreeClassifier is a class capable of performing multi-class classification on a dataset.
As other classifiers, **DecisionTreeClassifier** take as input two arrays: an array X of size [n_samples, n_features] holding the training samples, and an array Y of integer values, size [n_samples], holding the class labels for the training samples:

```
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
```

After being fitted, the model can then be used to predict new values:

```
>>> clf.predict([[2., 2.]])
array([1])
```

DecisionTreeClassifier is capable of both binary (where the labels are [-1, 1]) classification and multiclass (where the labels are [0, ..., K-1]) classification. Using the Iris dataset, we can construct a tree as follows:

```
>>> from sklearn.datasets import load_iris
>>> from sklearn import tree
>>> iris = load_iris()
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(iris.data, iris.target)
```

Once trained, we can export the tree in Graphviz format using the **export_graphviz** exporter. Below is an example export of a tree trained on the entire iris dataset:

```
>>> from sklearn.externals.six import StringIO
>>> with open("iris.dot", 'w') as f:
...     f = tree.export_graphviz(clf, out_file=f)
```

Then we can use Graphviz's dot tool to create a PDF file (or any other supported file type): dot -Tpdf iris.dot -o iris.pdf.

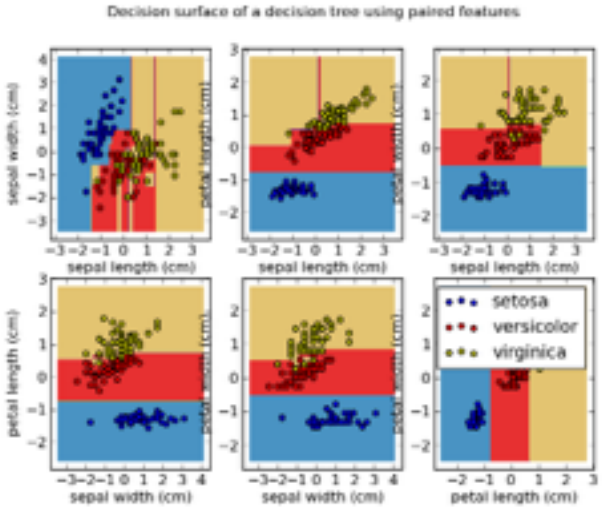
```
>>> import os
>>> os.unlink('iris.dot')
```

Alternatively, if we have Python module pydot installed, we can generate a PDF file (or any other supported file type) directly in Python:

```
>>> from sklearn.externals.six import StringIO
>>> import pydot
>>> dot_data = StringIO.StringIO()
>>> tree.export_graphviz(clf, out_file=dot_data)
>>> graph = pydot.graph_from_dot_data(dot_data.getvalue())
>>> graph.write_pdf("iris.pdf")
```

After being fitted, the model can then be used to predict new values:

```
>>> clf.predict(iris.data[0, :])
array([0])
```



Ход работы:

1) Прочитать теоретическую часть метод дерева решений
2) Для своего варианта описать структуру исходных данных:
   - общие характеристики массива данных: предметная область, количество записей
   - входные параметры: названия и типы
   - выходной класс: название и значения

3) Провести серию экспериментов с построением и тестированием деревьев решений, переразбивая исходное множество данных, заданное в варианте, следующим образом:

Номер эксперимента	Размер обучающей выборки	Размер тестовой выборки
1	60 %	40 %
2	70 %	30 %
3	80 %	20 %
4	90 %	10 %

5) Для заданных в варианте экземпляров спрогнозировать выходной класс.
6) Сформулировать выводы по использованию деревьев решений для исходной задачи.

Структура исходных данных:

Массив данных представлен файлом в формате CSV. Данные содержат следующие сведения о человеке для каждой записи:
age: continuous.
workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
fnlwgt: continuous.
education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
education-num: continuous.
marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex: Female, Male.
capital-gain: continuous.
capital-loss: continuous.
hours-per-week: continuous.
native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

Salary: >50K, <=50K.

Эти атрибуты описывают отдельного взрослого человека (каждая строчка в файле данных). В качестве выходной (класса) может быть только одна из переменных, она указана в варианте задания, остальные переменные тогда являются входными. В варианте задания также указан массив экземпляров, который необходимо использовать для обучения и тестирования дерева решений, а также экземпляры, которые необходимо использовать для прогнозирования выходных классов.

Структура отчета:

1. Титульный лист
2. Описание структуры исходных данных и задачи в терминах предметной области
3. Результаты построения и тестирования деревьев решений
4. Выводы по использованию метода
5. 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. Требования к исходным данным при построении дерева.
5. Каким образом происходит останов при построении дерева?
6. Зачем необходимо отсечение ветвей дерева?

Приложение

Пример программы построения и обучения дерева решений

```
import numpy as np
import pylab as pl

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

# Parameters
n_classes = 3
plot_colors = "bry"
plot_step = 0.02

# Load data
iris = load_iris()

for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                [1, 2], [1, 3], [2, 3]]):
    # We only take the two corresponding features
    X = iris.data[:, pair]
    y = iris.target

    # Shuffle
    idx = np.arange(X.shape[0])
    np.random.seed(13)
    np.random.shuffle(idx)
    X = X[idx]
    y = y[idx]

    # Standardize
    mean = X.mean(axis=0)
    std = X.std(axis=0)
    X = (X - mean) / std

    # Train
    clf = DecisionTreeClassifier().fit(X, y)

    # Plot the decision boundary
    pl.subplot(2, 3, pairidx + 1)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                         np.arange(y_min, y_max, plot_step))

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
```

```
    cs = pl.contourf(xx, yy, Z, cmap=pl.cm.Paired)

    pl.xlabel(iris.feature_names[pair[0]])
    pl.ylabel(iris.feature_names[pair[1]])
    pl.axis("tight")

    # Plot the training points
    for i, color in zip(range(n_classes), plot_colors):
        idx = np.where(y == i)
        pl.scatter(X[idx, 0], X[idx, 1], c=color,
label=iris.target_names[i],
                   cmap=pl.cm.Paired)

    pl.axis("tight")

pl.suptitle("Decision surface of a decision tree using paired features")
pl.legend()
pl.show()
```