Классификация. Наивная байесовская классификация. Деревья решений для классификации

Сергей Лисицын

lisitsyn.s.o@gmail.com

29 марта 2011 г.

Задача классификации
Classification

- $\sim$ задача распознавания образов
- Имеется некоторое пространство объектов
- Объекты разделены каким-либо образом на классы – непересекающиеся множества схожих объектов
- Известны классы некоторых из объектов (прецеденты)
- Решение задачи – алгоритм, позволяющий определить класс для любого элемента пространства объектов на основе имеющихся прецедентов
- В некоторых случаях необходимо определение вероятностей классов

Общая задача классификации

Пусть имеется некоторое множество объектов $\mathcal{X}$, конечное множество классов $\mathcal{C}$ и определено отображение

$$f : \mathcal{X} \to \mathcal{C},$$

причём известно, что некоторым элементам $X \subset \mathcal{X}$ соответствуют некоторые классы из множества $\mathcal{C}$. Множество

$$\{X, f(X)\} = \{(x_i, f(x_i))\}_{i=1}^{N}$$

называется обучающей выборкой.

Задача классификации состоит в нахождении функции $\hat{f}$, аппроксимирующей f на всех элементах $\mathcal{X}$, которая позволит любой объект из $\mathcal{X}$ отнести к некоторому классу из $\mathcal{C}$.

Для чего это нужно?

- Распознавание изображений
- Поддержка решений в банковской сфере
- Дифференциальная диагностика
- Классификация документов
- Анализ геостатистической информации
- Другие задачи, требующие «узнавания объектов»

Review: классификация

- Качество классификации, принцип минимизации эмпирического риска
- Обобщающая способность алгоритма
- Кросс-валидация (скользящий контроль) как практическая оценка обобщающей способности
- Переобучение и недообучение

Принципы MLE и MAP

Maximum likelihood estimation (MLE), maximum a posteriori (MAP)

- Метод максимального правдоподобия (MLE): условная вероятность данных при гипотезе максимизируется

$$\hat{h} = \arg\max_h P(\mathcal{D}|h)$$

- Принцип максимальной апостериорной вероятности (MAP): условная вероятность гипотезы при данных максимизируется

$$\hat{h} = \arg\max_h P(h|\mathcal{D})$$

Принцип максимальной апостериорной вероятности для классификации

MAP classification

- В случае классификации гипотеза – принадлежность классифицируемого объекта некоторому классу
- Оптимальным классификатором с точки зрения MAP и ERM является

$$\hat{f}(x) = \arg\max_{c \in \mathcal{C}} P(c|x)$$

- Саму функцию вероятности $P(c|x)$ найти по обучающей выборке не представляется возможным, приходится находить её каким-то иным образом

Байесовская классификация
Bayes classifier

- Используя теорему Байеса для классификации можно строить классификатор следующим образом:

$$\hat{f}(x) = \arg\max_{c \in \mathcal{C}} \frac{P(c)P(X = x|c)}{P(X = x)} = \arg\max_{c \in \mathcal{C}} P(c)P(X = x|c)$$

- Вероятность объекта $P(X = x)$ не зависит от класса, поэтому её даже не нужно вычислять
- Таким образом, задача обучения баейсовского классификатора сводится к определению функций $P(c)$ и $P(X = x|c)$

Наивный байесовский классификатор
Naïve Bayes classifier

- Если допустить, что все признаки являются независимыми в совокупности (что довольно **наивно**), то задача существенно упрощается: вероятность факторизуется

$$P(X = x|c) = P(X_1 = x_1|c)P(X_2 = x_2|c) \cdots P(X_n = x_n|c),$$

тогда классификатор будет иметь вид

$$\hat{f}(x) = \arg\max_{c \in C} P(c) \prod_i P(X_i = x_i|c)$$

- Такое допущение, несмотря на свою грубость, позволяет построить классификатор существенно проще и достичь при этом хорошей точности классификации

Вычисление оценок вероятностей классификатора

- Априорная вероятность вычисляется как частота класса среди элементов обучающей выборки

$$\hat{P}(c) = \frac{\sum_{e \in \mathcal{X}} [f(e) = c]}{\sum_{e \in \mathcal{X}} 1},$$

- Вероятности значений атрибута в классе находятся как частота элементов с таким значением среди всех элементов некоторого класса

$$\hat{P}(X_i = x_i | c) = \frac{\sum_{e \in \mathcal{X}} [f(e) = c][e_i = x_i]}{\sum_{e \in \mathcal{X}} [f(e) = c]}$$

- На практике некоторые вероятности будут обнуляться, что приведёт к обнулению всей вероятности класса – проблема решается добавлением параметра ε:

$$P(X_i = x_i | c) = P(X_i = x_i | c) + \varepsilon$$

Обучение NB классификатора

Naïve bayes

Вход: множество классов $\mathcal{C}$, множество $\{(x^i, \alpha(x^i)\}_i$
Выход: параметры классификатора

$$\hat{f}(x) = \arg\max_{c \in \mathcal{C}} \hat{P}(c) \prod_k \hat{P}(X_k = x_k | c)$$

1. для всех x из обучающей выборки
   увеличить соответствующее количество объектов класса $f(x)$; для всех атрибутов x_k увеличить суммы в выражениях для $\hat{P}(X_k = x_k | c)$
2. для всех классов c
   вычислить априорные вероятности $\hat{P}(c)$, вычислить соответствующие каждому атрибуту $\hat{P}(X_k = x_k | c)$

```
class NB_classifier(Classifier):
    def __init__(self,reg_par):
      self.probs = \
      defaultdict(lambda: defaultdict(lambda: defaultdict(float)))
      self.c_probs, self.reg_par = defaultdict(float), reg_par
    def train(self,trainset):
      for (x,c) in trainset:
          self.c_probs[c] += 1
          for index,attribute in enumerate(x):
              self.probs[c][index][attribute] += 1
      for c in self.c_probs.keys():
          for index in self.probs[c].keys():
              for attribute in self.probs[c][index]:
                  self.probs[c][index][attribute] /= self.c_probs[c]
          self.c_probs[c] /= len(trainset)
      self.probs, self.c_probs = dict(self.probs), dict(self.c_probs)
    def classify(self,x):
      scores = self.c_probs.copy()
      for c in scores.keys():
          for index,attribute in enumerate(x):
              scores[c] *= \
              (self.probs[c][index][attribute]+self.reg_par)
      return max(scores,key=scores.get),scores
```

Классификация текстов

- Наивный байесовский классификатор считается одним из лучших классификаторов текстовых документов
- Естественно, что предварительно необходимо привести все слова текстового документа к своей начальной форме
- Вместо оценки вероятностей $P(X_i = x_i|c)$ иногда оценивается вероятность $P(x_i|c)$ появления слова x_i из документа x среди экземпляров класса c
- Прежде всего, слова текста нужно преобразовать в начальные формы (для русского языка эту задачу решает mystem компании Яндекс)

Классификация новостных заголовков

Происшествия	**Спорт**
В Саратове гаишник съел деньги	Сборная России снова сыграла вничью
Пьяный водитель врезался в билборд	Спортсмен съел хлеб с допингом
В Курске убит криминальный авторитет	Сборная Чехии проиграла товарищеский матч
Депутат Хакасии стрелял в ребенка	В Кубке Гагарина определились полуфиналисты
В Москве убили трех человек	Команда Норвегии выиграла биатлонную эстафету
В Москве милиционеры ранили дебошира	Обладателем кубка Стэнли стал Детроит

Классификация новостных заголовков

- Обучающая выборка слишком мала для реальных данных, но даёт понятие о том, как работает алгоритм
- Правильно классифицируется:
  - Обладатель оружия стрелял в человека
  - Сборная Ирландии снова одержала победу
  - Депутат думы задержан с поличным
  - ...
- Ошибочно классифицируется:
  - В Москве прошёл матч ветеранов
  - Спортсмен угрожал нападением с ножом
  - Команда по футболу устроила дебош
  - ...
- Приводит классификатор в уныние:
  - НАИВНЫЙ БАЙЕСОВСК? KLASSNФNKATOP ДЛЯ TEKCTOB
  - БЕЗНОГNМ

Гауссовый наивный байесовский классификатор
Gaussian Naïve Bayes (GNB)

- В случае, если атрибуты непрерывны, оценка их вероятности затрудняется, равенство конкретному числу вообще имеет нулевую вероятность
- Вероятность $P(X_i = x_i|c)$ оценивается с помощью гауссовской плотности

$$P(X_i = x_i|c) = \underbrace{\frac{1}{\sigma(i,c)\sqrt{2\pi}} \exp\left\{-\frac{\left(x_i - \mu(i,c)\right)^2}{2\sigma^2(i,c)}\right\}}_{\mathcal{N}(x_i,\mu(i,c),\sigma(i,c))}$$

- В процессе обучения тогда необходимо будет найти статистики $\hat{\mu}(i,c)$ и $\hat{\sigma}(i,c)$, оценивающие соответствующие параметры распределений

Нахождение параметров распределений

Статистики $\hat{\mu}(i,c)$ и $\hat{\sigma}(i,c)$ находятся по формулам:

$$\hat{\mu}(i,c) = \underbrace{\frac{1}{\sum_{x \in \mathcal{X}} [\alpha(x) = c]}}_{\substack{\text{обратное количество} \\ \text{экземпляров класса}}} \underbrace{\sum_{x \in \mathcal{X}} x_i \cdot [f(x) = c]}_{\substack{\text{сумма i-ых атрибутов} \\ \text{элементов класса}}},$$

то есть имеет смысл среднего значение атрибута в классе, а

$$\hat{\sigma}(i,c) = \underbrace{\frac{1}{\sum_{x \in \mathcal{X}} [f(x) = c] - 1}}_{\substack{\text{обратное количество} \\ \text{экземпляров класса}}} \underbrace{\sum_{x \in \mathcal{X}} \left(x_i - \mu(i,c)\right)^2 \cdot [f(x) = c]}_{\substack{\text{сумма квадратов отклонений} \\ \text{i-ых атрибутов элементов класса}}},$$

имеет смысл среднеквадратичного отклонения значения атрибута в классе

Обучение GNB классификатора

Вход: множество классов $\mathcal{C}$, множество $\{(x^i, \alpha(x^i)\}_i$
Выход: параметры классификатора

$$\hat{f}(x) = \arg\max_{c \in \mathcal{C}} \hat{P}(c) \prod_k \mathcal{N}\left(x_k, \hat{\mu}(k,c), \hat{\sigma}(k,c)\right)$$

1. для всех x из обучающей выборки
   увеличить соответствующее количество объектов класса $f(x)$; для всех атрибутов x_k увеличить суммы в выражениях для $\mu(k,c)$
2. для каждого класса c
   вычислить $\hat{P}(c)$ как отношение количества объектов класса c к общему количеству объектов; для каждого k-го атрибута вычислить $\mu(k,c)$, разделив на количество объектов класса c
3. для всех атрибутов объектов x
   увеличить на $(x_k - \mu(k,c))^2$ соответствующие им суммы в $\sigma(k,c)$
4. для каждого класса c
   для каждого k-го атрибута вычислить $\sigma(k,c)$, разделив на количество объектов класса c за вычетом одного

```
class GNB_classifier(Classifier):
    def __init__(self,dimension):
        self.mu,self.sigma,self.apriori = \
        (defaultdict(lambda: numpy.zeros(dimension)),
        defaultdict(lambda: numpy.zeros(dimension))),{}
    def train(self,trainset):
        class_count = defaultdict(int)
        for (x,c) in trainset:
            class_count[c] += 1
            self.mu[c]+= x
        for c in self.mu.keys():
            self.mu[c] /= class_count[c]
            self.apriori[c] = float(class_count[c])/len(trainset)
        for (x,c) in trainset:
            self.sigma[c] = (self.mu[c]-x)**2
        for c in self.mu.keys():
            self.sigma[c] /= ((class_count[c])-1)
    def classify(self,x):
        rates = {}
        for c in self.apriori.keys():
            rates[c] = self.apriori[c]*numpy.exp(
       -((x-self.mu[c])**2)/(2*self.sigma[c]**2)).prod()
        return max(rates,key=rates.get)
```

Классификация текстов на основе TF-IDF и классификатора GNB

- Коэффициентом TF (term frequency) называется частота слова w в документе d:

$$TF(w,d) = \frac{n(w)}{\sum_{w \in d} n(w)}$$

- Коэффициентом IDF (inversed document frequency) называется доля документов среди D, в которых встречается слово w:

$$IDF(w) = \log \frac{\sum_{d \in D} 1}{\sum_{d \in D} [w \in d]}$$

- TF-IDF представляет любой документ d из k различных слов как вектор численных значений $TF(w_i, d) \cdot IDF(w_i), i = 0, \ldots, k$

Эффективность

- Наивный байесовский подход хорошо подходит для классификации текстов
- Уже обученный классификатор достаточно трудно «дообучивать» и проще переобучить его на дополненных данных
- Для вычисления всех параметров достаточно два раза «обойти» все объекты обучающей выборки, то есть обучение происходит за $O(n)$
- Сама классификация происходит достаточно быстро, требует хранения $k(2n+1)$ чисел в памяти, k – количество классов; $(2n+1)$ – количество параметров для каждого класса
- Наивный (не гауссовый) байесовский классификатор требует большого количества памяти при разнородных выборках

What to know?

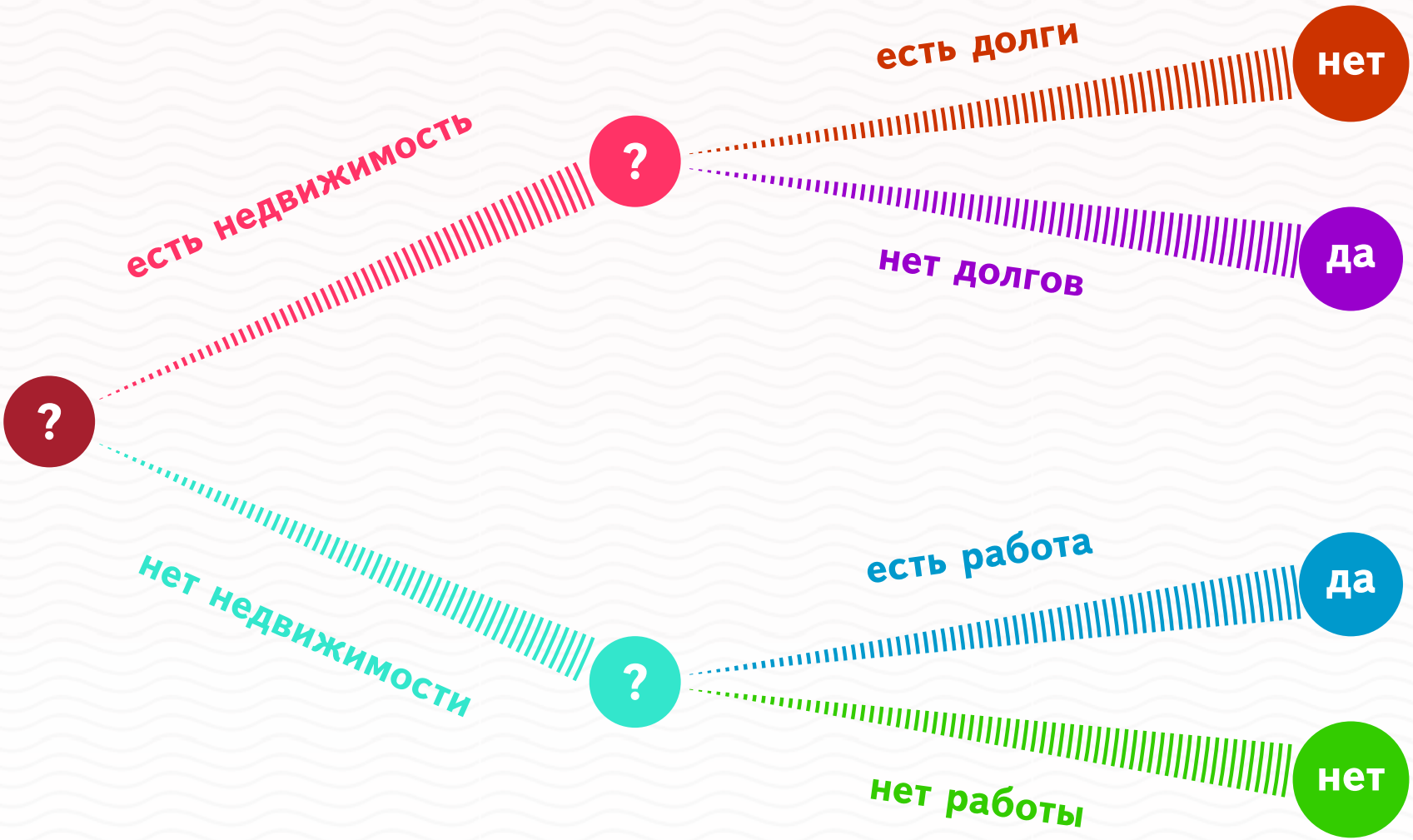
- Байесовская классификация, наивная байесовская классификация
- Классификация коротких новостных заголовков с помощью наивного байесовского классификатора
- Гауссовский наивный байесовский классификатор как альтернатива для непрерывных признаков
- Классификация документов с помощью представления TF-IDF и гауссовского наивного байесовского классификатора

Деревья решений
Decision Tree (DT)

- Дерево решений – дерево, в нетерминальных узлах которого расположены некоторые предикаты, исходящие из этого узла рёбра характеризуют значения предиката; в терминальных узлах находятся значения функции
- Значение функции находится последовательным переходом по ребрам, соответствующим значениям предиката в узле, до тех пор, пока не будет достигнут терминальный узел
- Любое решающее дерево может быть представлено как решающий список с помощью ДНФ, полученных обходом дерева

Выдавать ли кредит?

Простой пример дерева решений



Деревья для классификации

- В случае классификации значения функции – классы
- Высокая скорость классификации
- Обработка одновременно категориальных и численных данных
- Высокая интерпретируемость – классификацию построенным деревом может производить даже человек
- Достаточно строгая статистическая основа алгоритмов построения деревьев
- Как правило чрезмерно усложняются на классификации векторов классификации
- Задача построения дерева всегда сводится к нахождению предикатов для нетерминальных узлов и наилучшему их размещению в дереве

Предикаты объектов обучающей выборки

- Для построения дерева классификации необходимо выбрать некоторые предикаты, способные разделить выборку на непересекающиеся составляющие
- Примеры таких предикатов
  - $\varphi_{=a_i}(x) = [x_i = a_i]$ – предикат равенства
  - $\varphi_{>a_i}(x) = [x_i > a_i]$ – предикат неравенства
  - другие неравенства, составные неравенства нескольких атрибутов, ...
- Наиболее часто встречаются предикаты равенства, но в случае вещественных переменных они бессмысленны
- Предикаты неравенства же прежде всего должны быть найдены из обучающей выборки

Бинаризация предикатов

- Вещественные атрибуты (та же ситуация, как и с GNB) бесполезно оценивать предикатами равенства – дерево будет слишком сложным и на тестовых выборках с вероятностью 1 будет отказываться от классификации
- Для простоты можно строить предикат $\varphi_i(x) = [x_i > \overline{x}_i]$, где $\overline{x}_i$ – среднее значение атрибута x_i в обучающей выборке
- В случае с категориальными признаками объектов строятся многозначными – по количеству различных значений признака
- Дальше будем рассматривать деревья для чисто категориальных данных (в таком случае признаки объекта сами характеризуют свои предикаты)

Алгоритм ID3

Iterative Dichotomizer 3

- Описан Россом Квинланом в 1976 году
- Определил повышенный интерес к деревьям классификации и регрессии
- Имеется множество алгоритмов, развивающих ID3 (C4.5, C5.0, etc)
- Строит дерево на основе энтропийной «ценности» каждого из предикатов: в более высокие по иерархии узлы устанавливаются предикаты с максимальным приростом информации

Описание алгоритма ID3

Вход: обучающая выборка T – пары (объект-ответ), множество предикатов $\mathcal{B}$ на обучающей выборке
Выход: дерево решений

1. Создать корень дерева
2. Если множество предикатов $\mathcal{B}$ состоит из одного предиката – поместить в корень класс, характерный для большинства объектов выборки
3. Если в обучающей выборке встречается только один класс – поместить в корень этот класс
4. Найти наиболее информативный предикат $\beta \in \mathcal{B}$
5. Для каждого возможного значения b_i предиката β:
   - Рекурсивно запустить алгоритм для подмножества $\{x \in T | \beta(x) = b_i\}$ и множества предикатов $\mathcal{B} \setminus \beta$

```
class ID3_classifier(Classifier):
    def __init__(self):
        self.tree = {}

    def train(self, trainset):
        if not trainset: return None
        attributes = trainset[0].keys()
        for x in trainset:
            if not x.keys() == attributes:
                return None
        attributes.remove('class')
        self.tree = create_DT(trainset, attributes)

    def classify(self, x):
        return traverse_classify(self.tree, x)
```

```
def create_DT(trainset,attributes):
        classes = [x['class'] for x in trainset]
        default = majority_class(trainset)
        if len(attributes) == 1:
            return default
        if classes.count(classes[0]) == len(classes):
            return classes[0]
        else:
            best = best_attribute(trainset,attributes)
            tree = {best : {}}
            for value in set(map(lambda x: x[best], trainset)):
                subtree = create_DT(
                    [x for x in trainset if x[best] == value],
                    [attr for attr in attributes if attr != best])
                tree[best][value] = subtree
            return tree
def traverse_classify(tree, x):
        if type(tree) == type("string"):
            return tree
        else:
            attr = tree.keys()[0]
            subtree = tree[attr][x[attr]]
        return traverse_classify(x, subtree)
```

Информационная энтропия

- Если $\mathcal{D}$ разбивается на два класса из $\mathcal{C}$ (m и n объектов)

$$H(\mathcal{D},\mathcal{C}) = -\frac{m}{m+n}\log_2\frac{m}{m+n} - \frac{n}{m+n}\log_2\frac{n}{m+n}$$

- Обобщая на s классов из $\mathcal{C}$ с количествами объектов m_i $(i = 1,\ldots,s), \sum_i m_i = n$, формула имеет вид

$$H(\mathcal{D},\mathcal{C}) = -\sum_{i=1}^{s}\frac{m_i}{n}\log\frac{m_i}{n}$$

- Наилучший предикат $\mathcal{F}_{best}$ с возможными значениями $(f_1, f_2, \ldots)$ выбирается из прироста информации (information gain): $\mathcal{F}_{best} = \arg\max_{\mathcal{F}} Gain(\mathcal{D},\mathcal{F})$,

$$Gain(\mathcal{D},\mathcal{F}) = \underbrace{H(\mathcal{D},\mathcal{C})}_{\text{до разбиения}} - \underbrace{\sum_i \frac{|\mathcal{D}_i|}{|\mathcal{D}|}H(\mathcal{D}_i,\mathcal{C})}_{\text{результат разбиения}},\quad \mathcal{D}_i = \{x \in \mathcal{D} | \mathcal{F}(x) = f_i\}$$

Реализация: энтропия и прирости информации

```
1  def entropy(data, target_attr):
2      freqs = defaultdict(int)
3      data_entropy = 0.0
4      for x in data:
5          freqs[x[target_attr]] += 1
6      for freq in freqs.values():
7          data_entropy += \
8          (-float(freq)/len(data)) * math.log(float(freq)/len(data), 2)
9      return data_entropy
10
11 def gain(data, attr, target_attr):
12     freqs = defaultdict(int)
13     subset_entropy = 0.0
14     for x in data:
15         freqs[x[attr]] += 1
16     for val in freqs.keys():
17         prob = freqs[val] / sum(freqs.values())
18         subset = [record for record in data if record[attr] == val]
19         subset_entropy += prob * entropy(subset, target_attr)
20     return (entropy(data, target_attr) - subset_entropy)
```

Реализация: поиск информативного признака и самого частого класса

```
1 def best_attribute(data, attributes):
2     attr_w_gains = \
3     map(lambda attr: (attr, gain(data, attr)), attributes)
4     return max(attr_w_gains, key=lambda x: x[1])[0]
5
6
7 def majority_class(data):
8     classes = [x['class'] for x in data]
9     return max(set(classes),key=classes.count)
```

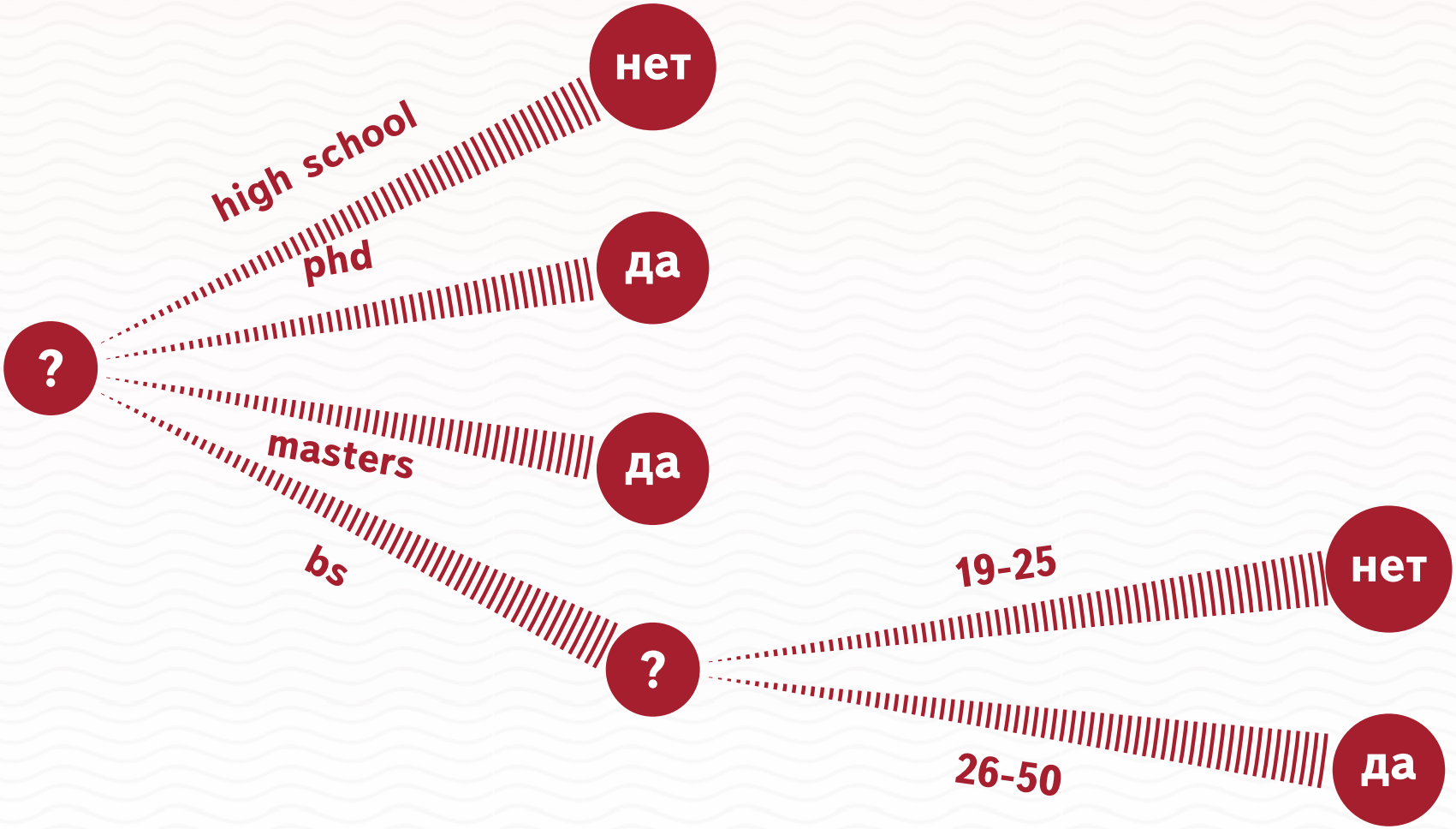
Деревья для задачи кредитной классификации

- Упрощённая задача кредитного скоринга, на основе прошлых данных принимается решение выдавать ли новому клиенту кредит
- Обучающая выборка содержит всего 6 предыдущих клиентов

Work	Education	Age	?
-	high school	19-25	no
+	msc degree	19-25	yes
-	bs degree	19-25	no
+	bs degree	26-50	yes
-	phd	26-50	yes
+	phd	19-25	yes

- В результате обработки, алгоритм строит дерево для классификации

Построенное дерево для кредитной классификации



Эффективность

- В случае признака с сильно неоднородными значениями разбиение по соответствующему ему предикату будет являться наиболее информативным, но помещать такой предикат в корень (или близко к корню) бесполезно
- Существуют иные способы оценки предикатов (Gain Ratio, Gini Index, ...), превосходящие энтропийную оценку
- Деревья решений удобны для категориальных признаков и смешанных пространств объектов (как с категориальными, так и вещественными признаками), для чисто вещественных признаков деревья не так удобны
- Классификация выполняется простым обходом дерева – достаточно быстро
- Построение дерева требует дорогостоящих вычислений энтропии и прироста информации

Проблема оверфиттинга и принцип минимальной длины описания

Minimum Description Length (MDL)

- Чем больше дерево, тем более оно подвержено оверфиттингу
- Дерево, учитывающее все возможные признаки является как правило переобученным
- Принцип минимальной длины описания является аналогом «бритвы Оккама» для обучаемых алгоритмов
- В случае с деревом классификации MDL формулируется как минимизация суммы функции потерь на обучающей выборке (риска) и количества его узлов N

$$\sum_{x \in \mathcal{X}} [f(x) \neq \hat{f}(x)] + N \to \min$$

What to know?

- Дерево решений, дерево решений для классификации
- Алгоритм ID3
- Способы определения предикатов на множестве объектов, их бинаризация и слияние
- Энтропийная оценка предикатов как отношение порядка на их множестве
- MDL принцип, его смысл для дерева классификации

Источники

1. Mitchell T., "Bayesian Classifiers, Conditional Independence and Naïve Bayes"
2. Mitchell T., "Gaussian Naïve Bayes, and Logistic Regression"
3. Воронцов К.В., «Байесовская классификация, непараметрические методы»
4. Quinlan J., "Induction of Decision Trees"
5. Николенко С.И., «Деревья принятия решений»
6. Xing E., "Introduction to ML and Decision Trees"
7. Воронцов К.В., «Лекции по логическим алгоритмам классификации»