МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)

Институт информационных технологий математики и механники

ОТЧЁТ

Лабораторная работа №2

по курсу

«Современные компьютерные технологии»

Выполнил:
студент группы 381503м-2
Соврасов В. В.

Нижний Новгород
2016

Содержание

Постановка задачи

Требуется решить задачу классификации ирисов [1] с помощью нечёткой модели TSK0 [2], а затем оценить качество полученного решения. Построение модели должно включать в себя следующие этапы:

- обработка входных данных;
- структурная идентификация модели;
- параметрическая оптимизация модели;
- верификация.

Адекватным результатом считается получение не менее 92% верных предсказаний классификатора на этапе верификации.

Метод решения задачи

Рассмотрим формальную постановку задачи классификации ирисов: требуется аппроксимировать зависимость $f : X \to Y$, где X — векторное пространство R^4, а Y — множество $\{1, 2, 3\}$, каждому элементу которого соответствует вид ирисов. Значения функции f известны на конечном множестве элементов $D = \{x(t) : x(t) \in R^4, 1 \leqslant t \leqslant N\}$, которое яляется обучающей выборкой.

Модель TSK0 решает задачу регрессии, поэтому будем считать, что если на векторе x^* модель выдала ответ y^*, то $\tilde{y} = \lceil y^* - 0.5 \rceil$ — ответ в задаче классификации. Таким образом, значения выходной переменной модели, лежащие в отрезке $[i - 0.5; i + 0.5]$ соответствуют классу i.

На этапе предварительной обработки данных происходит нормализация элементов обучающей выборки. С помощью линейного преобразования

$$x(t)_i^* = \frac{x(t)_i - \min\limits_{t=\overline{1,N}} x(t_i)}{\max\limits_{t=\overline{1,N}} x(t_i) - \min\limits_{t=\overline{1,N}} x(t_i)}, i = \overline{1,4}$$

значения координат входных векторов переводятся в единичный гиперкуб $[0; 1]^4$.

Модель TSK0 включает в себя правила нечёткого вывода вида

$$\text{if}(x_1 \text{ is } A_1^k) \wedge (x_2 \text{ is } A_2^k) \wedge (x_3 \text{ is } A_3^k) \wedge (x_4 \text{ is } A_4^k) \Rightarrow y(x_1, x_2, x_3, x_4) = b_k, k = \overline{1, K}$$

Функции принадлежностей термов в антецедентах правил вывода являются гауссовыми:

$$\mu_{i,k}(x_i) = \exp\left(\frac{(x_i - c_{i,k})^2}{2a_{i,k}^2}\right)$$

Чтобы произвести структурную идентификацию модели, надо определить количество правил K и найти все параметры $c_{i,k}, a_{i,k}, b_k$. Для этого производится кластеризация входных данных с помощью конкурентного обучения нейронной сети Кохонена [2]. Алгоритм кластеризации описан в разделе 2.1. Пусть в результате обучения сети были получены K центров кластеров в гиперкубе $[0;1]^4$. Тогда каждому правилу нечёткой модели ставится в соответствие кластер, параметры $c_{i,k}$ для этого правила совпадают с координатами кластера. Параметры $a_{i,k}$ выбираются по правилу ближайшего соседа:

$$a_{i,k} = \frac{\|s_k - s_h\|}{1.5}, h = \arg\min_j \|s_k - s_j\|$$

где s_j — центр j-го кластера, $\| * \|$ — квадрат евклидовой нормы.

Параметры b_k вычисляются как нормированная взвешенная сумма значений выходной переменной y на обучающей выборке. В качестве весов здесь выступают значения функции истинности левой части правила k на элементах выборки:

$$b_k = \frac{\sum_{t=1}^{N}[\alpha_k(x(t))y(t)]}{\sum_{t=1}^{N}\alpha_k(x(t))}, \alpha_k(x) = \prod_{j=1}^{4}\mu_{j,k}(x_j)$$

Выход модели TSK0 вычисляется по формуле:

$$b_k = \frac{\sum_{k=1}^{K}[\alpha_k(x)b_k]}{\sum_{k=1}^{K}\alpha_k(x)}$$

и является нормированной взвешенной суммой коэффициентов b_k, где в качестве весов выступают степени уверенности в антецеденте каждого правила модели.

После структурной идентификации для более точного определения параметров производится оптимизация по какому-либо критерию качества. В данном случае ипользуется функция эмпирического риска $Q(w) = \frac{1}{N}\sum_{t=1}^{N} L(y(t) - f(x(t), w))$, где w — вектор параметров модели, $L(x, y) = (x - y)^2$. Оптимизация производится методом роя частиц [3]. Размерность пространства параметров: $K(2 * 4 + 1)$. Сам метод описан в разделе 2.2.

Для финальной оценки качества конкретного классификатора в данной работе применяется метод одного выстрела. Выбирается число $l : 1 \leqslant l \leqslant N$, затем множество D разбивается на $D_{train} = \{x(t) : x(t) \in R^4, 1 \leqslant t \leqslant l-1\}$ и $D_{test} = \{x(t) : x(t) \in R^4, l \leqslant t \leqslant N\}$. Выборка D_{train} используеся для структурной идентификации модели и при параметрической оптимизации. По выборке D_{test} вычисляется значение функционала $Q(w)$.

Чтобы оценить процесс построения модели в целом, а не качество конкретного классификатора, используется метод перекрёстного контроля, состоящий в разбиении исходной выборки D на $m > 1$ непересекающихся частей D_j и последовательном использовании каждой из частей в качестве тестовой выборки, а объединения всех остальных в роли обучающей. Таким образом, метод предполагает построение и оценку m моделей, а затем усреднение этой оценки: $Q(w) = \frac{1}{m}\sum_{i=1}^{m} Q_i(w)$.

Конкурентное обучение сети Кохонена

Сеть является двухслойной полносвязной. Первый слой входной, на него поступают компоненты векторов пространства R^N. Каждый нейрон первого слоя связан с H нейронами второго слоя, представляющими собой центры кластеров в R^N. Выход сети вычисляется по правилу «победитель получает всё»: на нейроне второго слоя с номером $j = \arg\max_{i=\overline{1,H}}(\|x - c_i\|)$ выход равен единице, а на остальных нулю. Таким образом, сеть позволяет кластеризовать данные, поступающие на вход.

Сеть Кохонена может обучаться без учителя. Для этого используется алгоритм конкурентного обуения. Пусть сгенерированы начальные координаты центров кластеров c_i, заданы параметры $\alpha_r < \alpha_w \in [0; 1]$, обучающая выборка $D = \{x(t) : x(t) \in R^N, 1 \leqslant t \leqslant N\}$. Для каждой частицы определён счётчик побед n_k, задано максимальное количество итераций $maxIterations$ и точность ε.

Пока не выполнен критерий остановки:

1. $epochNumber+ = 1$
2. $\forall x(t) \in D$
   - 2.1. $w = \arg\min_{k=\overline{1,H}} d(x(t), c_k)$, $d(x(t), c_k) = \frac{n_k}{\sum_{i=1}^{H} n_i}\|x(t) - c_k\|^2$.
   - 2.2. $r = \arg\min_{k \neq w} d(x(t), c_k)$
   - 2.3. $n_w+ = 1$
   - 2.4. $c_w = c_w + \alpha_w(x(t) - c_w)$
   - 2.5. $c_r = c_r - \alpha_r(x(t) - c_r)$
3. $\alpha_w = \alpha_w - \alpha_w \frac{epochNumber}{maxIterations}$
4. $\alpha_r = \alpha_r - \alpha_r \frac{epochNumber}{maxIterations}$
5. Проверить критерий остановки: $\frac{1}{H}\sum_{k=1}^{H} \|c_k^{epochNumber} - c_k^{epochNumber-1}\|^2 < \varepsilon$

Из вида формул для обновления α_w и α_r следует, что остановка алгоритма гарантированно произойдёт не позднее, чем через $maxIterations$ итреаций.

Алгоритм роя частиц

Алгоритм роя частиц является прямым стохастическим алгоритмом оптимизации. Он производит случайный направленный поиск в многомерном пространстве, моделируя поведение некоторой социальной группы.

Пусть есть S частиц с координатами $x_i(0)$ и скоростями $v_i(0), i = \overline{1, s}$. В начальный момент $t = 0$ все частицы лежат в допустимой области G, их скорости нулевые. Далее пока не

выполнен критерий остановки, координаты частиц и их скорости обновляются по следующим формулам:

$$v_i(t+1) = w * v_i(t) + c_1 * rand() * (p_i(t) - x_i(t)) + c_2 * rand() * (p_g(t) - x_i(t))$$
$$x_i(t+1) = x_i(t) + v_i(t)$$

Здесь w, c_1, $c_2 \in [0; 1]$ — параметры алгоритма, функция $rand()$ возвращает число из $[0; 1]$, $p_i(t)$ — лучшее положение i-ой частицы на момент t, $p_g(t)$ — лучшее положение среди всех частиц роя на момент t. Если частица $x_i(t)$ выйдет за пределы G при прибавлении к ней вектора скорости $v_i(t)$, необходимо последовательно уменьшать $v_i(t)$, пока $x_i(t+1)$ не будет лежать в G.

В качестве критерия остановки можно использовать достижение максимального числа итераций, приемлемого значения целевой функции или малость изменения значения целевой функции для лучшей частицы роя. За оценку оптимума после окончания работы принимается точка $p_g(t)$.

Реализация

Программная реализация алгоритмов построения и верификации модели выполнена на языке Python 2.7 с использованием библиотеки Numpy. Исходный код можно найти по ссылке [4]. Программа позволяет задавать через командную строку следующие параметры:

- начальный параметр для генератора псевдослучайных чисел;
- количество кластеров в сети Кохонена;
- количество частиц в методе роя частиц;
- размер тестовой выборки в методе одного выстрела;
- количество разбиений в методе скользящего контроля;
- режим работы («oneshot» или «crossv»).

В режиме «oneshot» качество модели оценивается до и после параметрической оптимизации по методу одного вытрела, а в процессе оптимизации при нахождении лучшего значения целевой функции оно выводится на экран.

В режиме «crossv» запускается метод скользящего контроля. Построение моделей при этом осуществляется в разных процессах и может быть выполнено параллельно, если есть соответствующие аппаратные ресурсы. На экран выводится финальная оценка качества и оценки качества всех построенных в процессе скользящего контроля моделей.

Результаты экспериментов

Для того, чтобы убедиться, что программная реализация обучения модели работает корректно, были проведены её запуски в различных режимах. В режиме «oneshot» были построены модели с различным количеством кластеров (NC) и частиц в рое (NP). Чем больше кластеров, тем больше правил в модели, т.е. она становится сложнее, а значит и более склонна к переобучению. С увеличением числа частиц растёт интенсивность подгонки параметров модели под обучающую выборку, что также может привести к переобучению. Резуьтаты экспериментов приведены в таблице 4.1. Во всех экспериментах объём тестовой выборки равен трети от объёма всех данных. В каждой ячейке таблицы указана доля верных ответов классификатора в процентах до и после этапа параметрической оптимизации.

Выборка	NP=20 NC=10	NP=50 NC=10	NP=25 NC=20	NP=100 NC=20
Обучающая	90,4 / 95,2	90,4 / 98	93,3 / 99	93,3 / 99
Тестовая	95,5 / 91,1	95,5 / 93,3	95,5 / 91,1	95,5 / 93,3

Таблица 4.1: Результаты работы модели в режиме «oneshot»

Как видно из таблицы 4.1, оптимизация в большинстве случаев улучшает результат на обучающей выборке и ухудшает на тестовой, происходит переобучение. В рассматриваемой задаче входные данные не настолько сложные, чтобы дополнительно подстраивать модель после структурной идентификации. Это имеет смысл делать, только при получении плохих результатов после идентификации.

Аналогичные эксперименты были проведены и в режиме «crossv» при числе разбиений, равном 5. Их результаты приведены в таблице 4.2. Дисперсия результатов оценки качества моделей, построенных при перекрёстном контроле, довольно большая, что объясняется недостаточным для получения устойчивых статистических характеристик объёмом входных данных. При рассматриваемых параметрах только в одном случае среднее качество классификации оказалось ниже, чем 92%. Чтобы подобрать более оптимальные параметры алгоритмов струкрурной идентификации и роя частиц, можно провести гиперпараметрическую оптимизацию, используя в роли критерия значение качества, получаемое скользящим контролем, но подобные эксперименты выходят за рамки данной лабораторной работы.

	NP=20 NC=10	NP=50 NC=10	NP=25 NC=20	NP=100 NC=20
Среднее качество	92,6	93,3	90,6	93,3
Дисперсия	7,1	6,9	5,7	2,1

Таблица 4.2: Результаты работы модели в режиме «crossv»

Список литературы

[1] *Iris data set*. `http://archive.ics.uci.edu/ml/datasets/Iris`.

[2] Осовский С.. *Нейронные сети для обработки информации*. М.: Финансы и статистика, 2002, с. 344.

[3] R.C. Eberhart J. Kennedy. «Particle Swarm Optimization». В: *Proceedings of IEEE International Conference on Neural Networks*. Т. IV. Perth, Australia, 1995, с. 1942—1948.

[4] *Исходный код алгоритмов, реализованных в ходе лабораторной работы*. `https://github.com/sovrasov/fuzzy-ml`.