1. Классификация

1.1. Задача классификации

Имеется множество объектов, которые некоторым образом разделены на классы. Задано конечное множество объектов и множество классов. Для всех объектов из данного множества известно, к какому классу они относятся. Кроме того, существует множество объектов, для которых неизвестна классовая принадлежность. Требуется разработать алгоритм, позволяющий классифицировать неопределённые объекты и отнести их к каким-либо из заданных классов.

Формально задачу можно представить так: пусть X – множество описаний объектов, каждый объект представлен вектором, значения которого характеризуют объект по различным признакам. Y – конечное множество названий классов. Существует некоторая функция зависимости, отображающая вектор признаков каждого объекта на множество классов $a: X \to Y$. Т.е. функция, определяющая класс объекта по его описанию. Задача классификации разработать алгоритм, позволяющий найти функцию зависимости $a: X \to Y$.

Например, есть множество описаний марок автомобилей (габариты, объем двигателя, год производства, и т.д.). Для каждой марки известен класс автомобиля (например, классификация США – от мини до полноразмерного автомобиля). Имеется множество марок автомобилей с известными параметрами, но их класс неизвестен. Требуется разработать алгоритм, который найдёт зависимость между описанием автомобилей и классами по первому множеству автомобилей и использует эту зависимость для классификации второго множества автомобилей.

1.2. Классификация логистической регрессией

Логистическая регрессия – метод построения линейного классификатора. В простейшем случае позволяет классифицировать объекты по двум классам.

Пусть $Y = \{-1; +1\}$ – множество классов. В логистической регрессии строится линейный классификатор $a: X \to Y$ вида $a(x, w) = sign(\sum_{j=1}^{n}(w_j f_j(x))) = sign < x, w >$, где w – вектор весов, $f_j(x)$ – j-ый признак объекта x, $< x, w >$ - скалярное произведение векторов x и w,

$$sign(x) = \begin{cases} -1, x < 0 \\ +1, x \geq 0 \end{cases}$$

Т.е. для классификации объекта нужно взять вектор его признаков (каждый признак должен быть закодирован каким-то числом), умножить его на некоторый вектор весов и взять знак результата. Знак и будет являться меткой класса. Для того чтобы обучить классификатор нужно найти вектор весов w – единственный неизвестный параметр.

Для нахождения вектора весов w решается задача минимизации эмпирического риска:

$$\min_w Q(w) = \min_w \sum_{i=1}^{m} \ln(1 + e^{-y_i \cdot < x_i, w >}),$$

или максимизации функции правдоподобия:

$$\sum_{i=1}^{m} y_i \ln(f(< x_i, w >)) + (1 - y_i)\ln(1 - f(< x_i, w >)),$$

где x_i – вектор признаков i-го объектов обучающей выборки, y_i – класс i-го объекта, $f(z)$ – логистическая функция, $f(z) = \frac{1}{1+e^{-z}}$.

2. Алгоритмы оптимизации

Неформально задачу оптимизации можно назвать задачей поиска экстремума (минимума или максимума) целевой функции. Все представленные ниже алгоритмы оптимизации являются стохастическими и не гарантируют сходимость за конечное число итераций. Но при правильной настройке быстро достигают решения заданной точности.

2.1. Метод роя частиц

Оптимизация методом роя частиц заключается в том, что в пространстве размещаются частицы в произвольных координатах, для координат каждой частицы вычисляется значение целевой функции и запоминается наилучшее значение среди всех. Все последующие итерации заключаются в смещении, расставленных на первом шаге частиц. Скорость и направление каждой частицы зависит от координат лучшего решения задачи оптимизации в данный момент и координат лучшего решения задачи оптимизации, найденного каждой частицей. Т.е. каждая частица смещается в сторону наилучшего решения в данный момент, но направление её движения корректируется с учётом её наилучшего результата. После смещения каждой частицы обновляется наилучшее решение, если какая-то частица улучшила результат.

Пусть $x_{i,d}$ – d-ая координата i-ой частицы. $p_{i,d}$ – d-ая координата наилучшего решения, найденного i-ой частицей. g_d – d-ая координата наилучшего решения, найденного всеми частицами. r_p, r_g – случайные числа в диапазоне $[0;1]$. φ_p, φ_g – параметры, подбираемые эмпирически. ω – коэффициент инерции, число в диапазоне $[0;1]$. Тогда скорость частицы по d-ой оси вычисляется по формуле:

$$v_{i,d} = \omega v_{i,d} + \varphi_p \cdot r_p \cdot (p_{i,d} - x_{i,d}) + \varphi_g \cdot r_g \cdot (g_d - x_{i,d}),$$

Координаты частиц обновляются по следующей формуле:

$$x_{i,d} = x_{i,d} + v_{i,d}$$

Так как выбор коэффициентов φ_p, φ_g сильно влияет на сходимость метода, один из самых распространённых вариантов алгоритма предлагает ввести нормировку этих коэффициентов:

$$\varphi = \varphi_p + \varphi_g, \varphi > 4,$$

$$\chi = \frac{2k}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|},$$

коэффициент k лежит в интервале $[0;1]$. Тогда скорость рассчитывается по следующей формуле:

$$v_{i,d} = \chi[\omega v_{i,d} + \varphi_p \cdot r_p \cdot (p_{i,d} - x_{i,d}) + \varphi_g \cdot r_g \cdot (g_d - x_{i,d})]$$

Псевдокод алгоритма:

Для каждой частицы i = 1, ..., S выполнить

Инициализировать координат частицы вектором имеющим равномерное распределение: xi ~ U(blo, bup)

Инициализировать лучшее известное положение частицы начальными координатами: $p_i \leftarrow x_i$

Если $f(p_i) < f(g)$ то

Обновить лучшее известное положение: $g \leftarrow p_i$

Инициализировать вектор скорости частицы: v_i ~ U(-|bup-blo|, |bup-blo|)

Пока не достигнуто условие останова:

Для каждой частицы i = 1, ..., S выполнить

Для каждого измерения d = 1, ..., n выполнить

Выбрать случайные числа: $r_p, r_g \sim U(0,1)$

Обновить скорость:
$v_{i,d} = \chi[\omega v_{i,d} + \varphi_p \cdot r_p \cdot (p_{i,d} - x_{i,d}) + \varphi_g \cdot r_g \cdot (g_d - x_{i,d})]$

Обновить позицию: $x_{i,d} = x_{i,d} + v_{i,d}$

Если $f(x_i) < f(p_i)$ то

Обновить лучшую позицию для текущей частицы: $p_i \leftarrow x_i$

Если $f(p_i) < f(g)$ то

Обновить лучшую позицию для всех частиц: $g \leftarrow p_i$

Здесь bup и blo – границы поиска экстремума функции. f(x) – целевая функция.

Важно отметить, что во время выполнения алгоритма частицы могут выходить за границы поиска. В таком случае их нужно перемещать каким-то образом в нужный диапазон (например, менять координаты, по которым частица вышла за границы).

2.2. Метод искусственной пчелиной колонии

Алгоритм оптимизации методом пчелиной колонии заключается в повторении вычислений значений функции в нескольких случайных точках и вычислении значений в окрестностях лучших результатов. После некоторого количества итераций наилучшее решение берётся как экстремум целевой функции.

Параметры алгоритма:

S – количество точек в которых будет рассчитано значение функции на первом шаге каждой итерации.

n – количество лучших участков. Лучшие участки – первые n лучших значений функции, вычисленных ранее. Например, если ставится задача поиска минимума, то берётся n точек в которых функция принимает наименьшее значение относительно других вычисленных значений функции.

m – количество выбранных участков. Участки с лучшими значениями функции после значений на лучших участках.

N – количество значений, вычисляемых в окрестностях лучших участков.

M – количество значений, вычисляемых в окрестностях выбранных участков.

d – размер окрестности

Алгоритм метода:

3. `Инициализировать S случайных решений`
4. `Найти n лучших и m выбранных решений из всех известных решений`
5. `Инициализировать N решений для каждого лучшего участка`
6. `Инициализировать M решений для каждого выбранного участка`
7. `Вернуться к шагу 1`

Например, пусть $f(x)$ – функция для которой нужно найти минимум на отрезке $[-500;500]$. $S = 10, n = 3, m = 2, N = 5, M = 5, d = 10$. Берётся 10 случайных точек в пределах диапазона решения и в них вычисляются значения функции $[f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}]$. Из этих значений выбираются 3 лучших участка (т.к. поиск минимума – три точки в которых функция принимает наименьшие значения) и 2 выбранных участка (точки в которых функция принимает наименьшее значение без учёта точек помеченных как лучшие участки). В пяти случайных точках вокруг каждого лучшего участка вычисляется значение функции. В двух случайных точках вокруг каждого выбранного участка вычисляется значение функции. После этого алгоритм возвращается к первому шагу, но теперь на каждом шаге лучшие и выбранные участки выбираются не только из 10 случайных точек, а с учётом значений, найденных вокруг лучших и выбранных участков из предыдущего шага.

2.3. Дифференциальная эволюция

В этом алгоритме оптимизации генерируется точки n-мерного пространства. Для каждой точки x_i выбираются три случайных точки $x_j, x_k, x_z, j \neq i, k \neq i, z \neq i$. Генерируется новая точка по формуле:

$$v = x_j + F \cdot (x_k - x_z),$$

где F – некоторый действительный параметр в диапазоне $[0,2]$. Некоторые случайные координаты точки x_i заменяются координатами точки v. Затем вычисляется значение целевой функции в новой точке, если это значение оказывается лучше чем в точке x_i, новая точка заменяет точку x_i.