Лекция 4. МЕТОДЫ И АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧ КЛАССИФИКАЦИИ

Задача классификации — задача, в которой имеется множество объектов (ситуаций), разделённых некоторым образом на классы. Задано конечное множество объектов, для которых известно, к каким классам они относятся. Это множество называется выборкой. Классовая принадлежность остальных объектов неизвестна. Требуется построить алгоритм, способный классифицировать произвольный объект из исходного множества.

Различают следующие виды классификации:

- Двухклассовая классификация. Наиболее простой в техническом отношении случай, который служит основой для решения более сложных задач.
- Многоклассовая классификация. Когда число классов достигает многих тысяч (например, при распознавании иероглифов или слитной речи), задача классификации становится существенно более трудной.
- Непересекающиеся классы.
- Пересекающиеся классы. Объект может относиться одновременно к нескольким классам.
- Нечёткие классы. Требуется определять степень принадлежности объекта каждому из классов, обычно это действительное число от 0 до 1.

В математической статистике задачи классификации называются также задачами дискриминантного анализа. В машинном обучении задача классификации решается, в частности, с помощью методов искусственных нейронных сетей при постановке эксперимента в виде обучения с учителем.

Существуют также другие способы постановки эксперимента — обучение без учителя, но они используются для решения другой задачи — кластеризации или таксономии. В этих задачах разделение объектов обучающей выборки на классы не задаётся, и требуется классифицировать объекты только на основе их сходства друг с другом. В некоторых прикладных областях, и даже в самой математической статистике, из-за близости задач часто не различают задачи кластеризации от задач классификации.

Некоторые алгоритмы для решения задач классификации комбинируют обучение с учителем с обучением без учителя,

например, одна из версий нейронных сетей Кохонена — сети векторного квантования, обучаемые с учителем.

Задачи классификации встречаются очень часто в самых разных областях деятельности человека. Как и задача регрессионного анализа, задача классификации решается в целях последующего прогнозирования переменной отклика (номера класса). Предполагается, что уже имеется какое-то количество n объектов, для каждого из которых известен некоторый набор из m признаков (факторов) и номер класса, к которому этот объект принадлежит, т.е. сырые данные, используемые для решения задачи классификации, имеют вид, представленный в таблице 4.1.

Таблица 4.1. Наборы признаков, распределенные по классам

Номер наблюдения, i	Значения факторов			Значения переменной отклика (номер класса)
1	$x_{1,1}$	…	x_{1m}	y_1
…	…	…	…	…
i	$x_{i,1}$	…	$x_{i,m}$	y_i
…	…	…	…	…
n	$x_{n,1}$	…	$x_{n,m}$	y_n

Здесь значения переменной отклика – номер класса, которому принадлежит объект, т.е.

$$y_i \epsilon \{1,\dots,K\} \text{ для всех } i=1,\dots,n,$$

K – (известное) количество классов.

Необходимо выработать правило, позволяющее отнести новый объект к одному из возможных классов. В тех случаях, когда число классов K = 2, говорят о бинарной классификации.

Постановка задачи классификации. Как и в задаче регрессионного анализа, предположим, что имеется n объектов, каждый из которых описывается m признаками. Будем нумеровать объекты индексом i ($i=1,\dots,n$), а признаки (значения которых могут быть получены непосредственным измерением) – индексом j ($j=1,\dots,m$). Для объекта с номером i обозначим через $x_{i,j}$ – значения признака j ; y_i – значение зависимого признака объекта i .

Приведём пример. Пусть объекты – это клиенты банка, наблюдаемый признак x – уровень их заработной платы,

прогнозируемый признак y – состояние кредитной карты. Цель исследования – спрогнозировать, «уйдёт ли в минус» тот или иной клиент банка (владелец банковской карты).

В нашем примере m $=1$. Данные n обследованных клиентов запишем как пары (x_i, y_i), $i=1,\ldots,n$. Каждой такой паре можем поставить в соответствие точку на координатной плоскости. «Разобьём» всех клиентов на два класса:

- «благонадёжные» (т.е. имеющие неотрицательный баланс на карте)
- «неблагонадёжные» (т.е. имеющие отрицательный баланс на карте).

Для наглядности благонадёжных клиентов будем обозначать точками синего цвета, неблагонадёжных – красного, как показано на приведённом рисунке 4.1. Задача состоит в том, чтобы сформулировать правило, позволяющее по заданному значению фактора (уровня зарплаты) определить наиболее вероятное значение переменной отклика (точнее, наиболее вероятный цвет соответствующей точки).

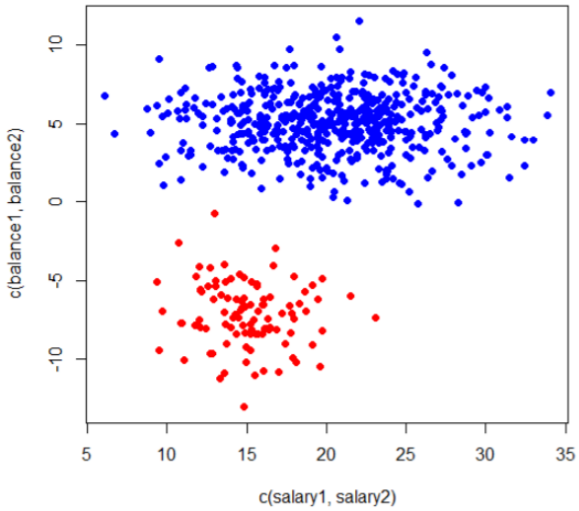

Рис 4.1. Зависимость между заработной платой и балансом карты

Мы видим, что одни и те же значения по оси абсцисс могут соответствовать как синим, так и красным точкам. Однако можно

заметить, что большим значениям признака x (заработной плате) соответствует большее число синих точек, чем красных.

К основным метода решения задачи классификации относятся:

- метод дерева решений
- байесовский классификатор
- метод опорных векторов

Метод дерева решений. Метода деревьев решений (англ.: decision tree) для задачи классификации (см., например, состоит в том, чтобы осуществлять процесс деления исходных данные на группы, пока не будут получены однородные (или почти однородные) их множества. Совокупность правил, которые дают такое разбиение (англ.: partition), позволят затем делать прогноз (т.е. определять наиболее вероятный номер класса) для новых данных.

Метод деревьев решений применим для решения задач классификации, возникающих в самых разных областях, и считается одним из самых эффективных. Среди задач, успешно решаемых с помощью этого метода, можно назвать, например,

• скоринговые модели кредитования (англ.: credit scoring models);

• маркетинговые исследования, направленные на выявление предпочтений клиента или степени его удовлетворённости – обычно эти сведения бывают востребованы маркетинговыми агентствами или рекламными компаниями;

• диагностика (медицинская или техническая), где по набору значений факторов (симптомов, результатов анализов) нужно поставить диагноз или сделать вывод о динамике процесса.

Дерево решений –это модель, представляющая собой совокупность правил для принятия решений. Графически её можно представить в виде древовидной структуры, где моменты принятия решений соответствуют так называемым узлам (англ.: decision nodes). В узлах происходит ветвление процесса (англ.: branching), т.е. деление его на так называемые ветви (англ.: branches) в зависимости от сделанного выбора. Конечные (или, что то же самое, терминальные) узлы называют листьями (англ.: leafs, leaf nodes) – каждый лист – это конечный результат последовательного принятия решений.

Данные, подлежащие классификации, находятся в так называемом «корне» дерева (англ.: root). В зависимости от решения, принимаемого в узлах, процесс в конце концов останавливается в одном из листьев,

где переменной отклика (искомому номеру класса) присваивается то или иное значение.

Метод деревьев решений реализует принцип так называемого «рекурсивного деления» (англ.:recursive partitioning). Эта стратегия также называется «Разделяй и властвуй» (англ.: «Divide and conquer»). В узлах, начиная с корневого, выбирается признак, значение которого используется для разбиения всех данных на 2 класса. Процесс продолжается до тех пор, пока не выполнится критерий остановки. Это возможно в следующих ситуациях:

- Все (или почти все) данные данного узла принадлежат одному и тому же классу;
- Не осталось признаков, по которым можно построить новое разбиение;
- Дерево превысило заранее заданный «лимит роста» (если таковой был заранее установлен).

Проиллюстрируем этот процесс на следующем примере. Представьте, что Вы работаете в кинокомпании и Ваш рабочий стол завален сценариями кинофильмов. Вам нужно разложить их по трём ящикам:

- Популярные (англ.: «mainstream hits»);
- Не популярные у зрителей, но получившие высокую оценку критиков;
- Не имеющие успеха.

Вместо того, чтобы тщательно прочитывать каждый из них, Вы хотите разработать алгоритм, реализующий принцип «разделяй и властвуй» для классификации сценариев.

Для этого Вы идёте в архив киностудии, чтобы собрать данные о киносценариях за последние 10 лет. После просмотра 30 киносценариев Вы начинаете замечать, что фильмы с высоким бюджетом привлекают больше звёзд. Эта закономерность хорошо просматривается на следующей диаграмме (рис. 4.2).

Будем использовать количество снимавшихся в фильме звёзд как первый из признаков, по которому производится разбиение данных (рис. 4.3). Рассмотрим теперь группу сценариев с большим числом задействованных звёзд. Разобьём её на 2 подгруппы по признаку «размер бюджета» как показано на рис. 4.4.

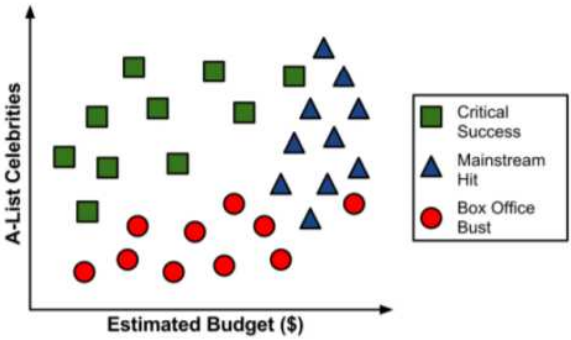

Рис. 4.2 Зависимость количества звезд, снимавшихся в фильме, от его бюджета

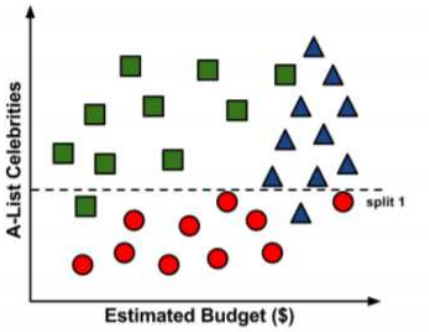

Рис. 4.3 Разбиение множества киносценариев по признаку количества занятых звёзд

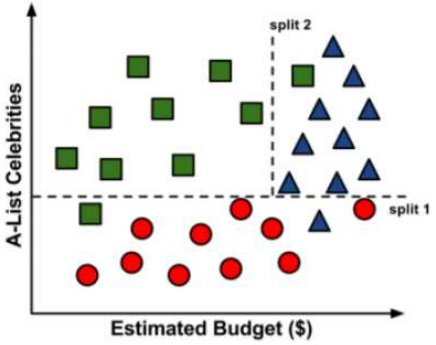

Рис. 4.4 Разбиение группы киносценариев с большим количеством занятых звёзд по признаку размера бюджета

Итак, мы разбили наши данные на 3 группы:

- Левая верхняя группа (зелёные квадраты) содержит исключительно те фильмы, которые были высоко оценены критиками. Для них характерно большое число занятых звёзд, но относительно небольшой бюджет.
- Группа в правом верхнем углу, в основном, состоит из синих треугольников – фильмов с большим бюджетом и большим числом звёзд;
- Третья группа – красные кружки в нижней части диаграммы – состоит преимущественно из фильмов, не имевших успеха – число звёзд, занятых в них, невелико, а их бюджет варьируется от очень малого до очень большого.

В принципе, ничто не мешает нам продолжать процесс разделения данных (по каждому из возможных значений числа звёзд, снявшихся в фильме), пока не получим очень «мелкое» разделение (может даже оказаться, что каждая группа будет содержать лишь по одному элементу). Понятно, что смысла в такой классификации нет. Поэтому мы ограничим ветвление дерева – например, остановим процесс, когда каждая группа хотя бы на 80% будет состоять из элементов одного и того же класса. (В нашем примере в первой группе вс8 элементов принадлежат одному классу – зелёные квадраты; во второй группе из 10 элементов 1 квадрат и 9 треугольников, т.е. 90% элементов принадлежит одному классу; в третьей группе 12 элементов, из них 10 одинаковых – красных кружков, – т.е. показатель однородности = 10/12 = 83,33… %. Это означает, что выполнен критерий оптимальности.

Заметим, что мы говорим лишь о разбиениях данных (точек в Евклидовом пространстве) прямыми (в общем случае – гиперплоскостями), параллельными осям координат. Казалось бы, если бы мы не ограничивали возможность разбиения только такими прямыми (гиперплоскостями), мы могли бы получить лучшее разбиение, например, как показано на рис. 4.5.

Однако Метод деревьев решений не позволяет строить такие разбиения, поскольку для разделения данных используются условия вида: $\{x < a\}$, $\{x > a\}$, где x – значение фактора, a – некоторое фиксированное число. Такие разбиения в англоязычной литературе называются «axisparallel splits», т.е. «разбиения, параллельные осям».

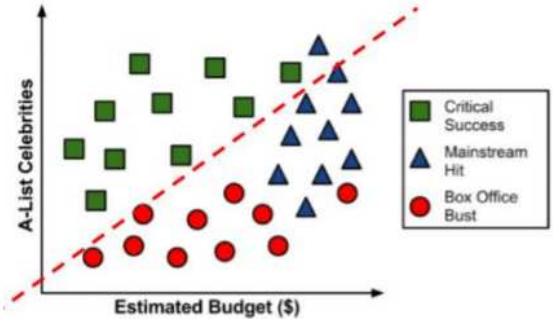

Рис. 4.5 Разбиение, недопустимое в методе деревьев решений группы

В нашем примере мы условно делим число задействованных в фильме звёзд по принципу «много» – «мало», и аналогично различаем малобюджетные и высокобюджетные. Соответствующее дерево решений показано на рис. 4.6.

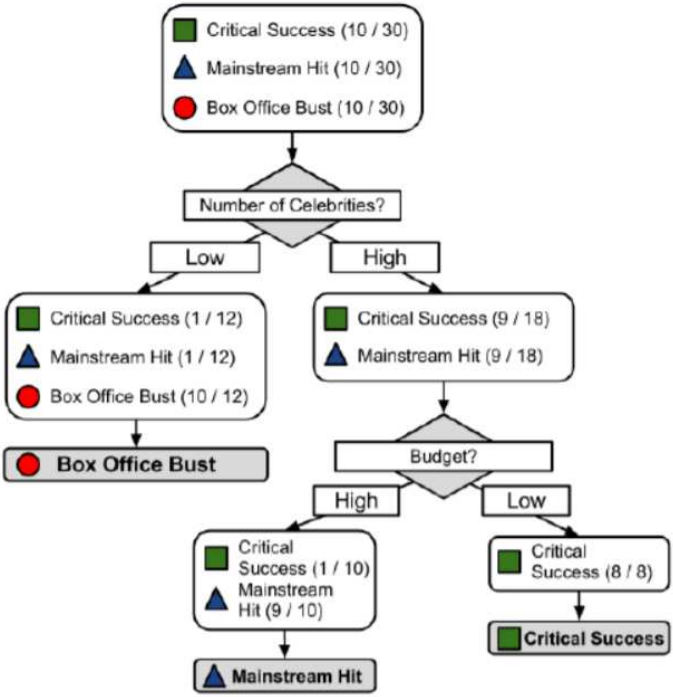

Рис. 4.6 Дерево решений для классификации киносценариев

Поскольку мы классифицировали сценарии лишь по 2-м факторам (количество занятых звёзд и бюджет), дерево получилось небольшим. На практике факторов бывает много больше и дерево содержит больше узлов, ветвей и листьев.

Существуют различные численные алгоритмы построения деревьев решений. Одним из более известных является алгоритм под названием C5.0, разработанный программистом J._Ross Quinlan. Фактически алгоритм C5.0 представляет собой стандарт процедуры построения деревьев решений. Эта программа реализуется на коммерческой основе но версия, встроенная в пакет R (и некоторые другие пакеты) доступны бесплатно. В приведённой таблице 4.2. перечислены сильные и слабые стороны алгоритма C5.0.

Таблица 4.2. Обзор алгоритма C5.0

Сильные стороны алгоритма C5.0	**Слабые стороны алгоритма C5.0**
Алгоритм C5.0 является универсальным, хорошо решает задачи классификации из разных областей.	Алгоритм C5.0 «тяготеет» к разбиениям по признакам с большим количеством уровней/ (Decision tree models are often biased toward splits on features having a large number of levels)
Алгоритм C5.0 может применяться для анализа не только числовых, но и номинальных данных; обеспечивает обработку пропущенных данных/	Модель (т.е. дерево решений) может оказаться как недоопределённой, так и переопредёлённой (It is easy to overfit or underfit the model)
Алгоритм C5.0 для построения дерева решений использует только самые важные признаки объектов (выбирает из множества факторов только те, которые сильно влияют на результат классификации).	Неточности классификации могут возникнуть из-за того, что используются только «параллельные осям» разбиения (axis-parallel splits)
Алгоритм C5.0 требует относительно небольшого объёма обучающей выборки.	Алгоритм C5.0 очень чувствителен к обучающей выборке – даже небольшие изменения в ней могут сильно повлиять на результат.

Интерпретация результатов работы алгоритма C5.0 не требует специальной математической подготовки.	Деревья решений иногда получаются очень большими, что затрудняет их интуитивное понимание.
Алгоритм C5.0 считается более эффективным, чем другие алгоритмы классификации.	

Байесовский классификатор. Байесовский классификатор — широкий класс алгоритмов классификации, основанный на принципе максимума апостериорной вероятности. Для классифицируемого объекта вычисляются функции правдоподобия каждого из классов, по ним вычисляются апостериорные вероятности классов. Объект относится к тому классу, для которого апостериорная вероятность максимальна.

Байесовский подход к классификации основан на теореме, утверждающей, что если плотности распределения каждого из классов известны, то искомый алгоритм можно выписать в явном аналитическом виде. Более того, этот алгоритм оптимален, то есть обладает минимальной вероятностью ошибок.

На практике плотности распределения классов, как правило, не известны. Их приходится оценивать (восстанавливать) по обучающей выборке. В результате байесовский алгоритм перестаёт быть оптимальным, так как восстановить плотность по выборке можно только с некоторой погрешностью. Чем короче выборка, тем выше шансы подогнать распределение под конкретные данные и столкнуться с эффектом переобучения.

Байесовский подход к классификации является одним из старейших, но до сих пор сохраняет прочные позиции в теории распознавания. Он лежит в основе многих достаточно удачных алгоритмов классификации. Теорема Байеса заключается в следующем:

$$P(y=c|x)=\frac{P(x|y=c)P(y=c)}{P(x)}$$

$P(y=c|x)$ - вероятность что объект x принадлежит классу c (апостериорная вероятность класса);

$P(x|y=c)$ - вероятность встретить объект x среди всех объектов класса c;

$P(y=c)$ - безусловная вероятность встретить объект класса с (априорная вероятность класса);

$P(x)$ - безусловная вероятность объекта x.

Цель классификации состоит в том чтобы понять к какому классу принадлежит объект x. Следовательно необходимо найти наиболее вероятный класс объекта x, т.е., необходимо из всех классов выбрать тот, который дает максимум вероятности $P(y=c|x)$:

$$c_{opt} = arg \underbrace{max}_{c \in C} P(y=c|x) = arg \underbrace{max}_{c \in C} \frac{P(x|y=c)P(y=c)}{P(x)}$$

Для каждого класса с вычисляется $P(y=c|x)$ и выбирается класс, имеющий максимальную вероятность. Вероятность $P(x)$ не зависит от с и является константой:

$$c_{opt} = arg \underbrace{max}_{c \in C} P(x|y=c)P(y=c)$$

Наивный байесовский алгоритм – это алгоритм классификации, основанный на теореме Байеса с допущением о независимости признаков. Другими словами, НБА предполагает, что наличие какого-либо признака в классе не связано с наличием какого-либо другого признака. Например, фрукт может считаться яблоком, если он красный, круглый и его диаметр составляет порядка 8 сантиметров. Даже если эти признаки зависят друг от друга или от других признаков, в любом случае они вносят независимый вклад в вероятность того, что этот фрукт является яблоком. В связи с таким допущением алгоритм называется «наивным».

Модели на основе НБА достаточно просты и крайне полезны при работе с очень большими наборами данных. При своей простоте НБА способен превзойти даже некоторые сложные алгоритмы классификации.

Ниже, на рис. 4.7, представлен обучающий набор данных, содержащий один признак «Погодные условия» (weather) и целевую переменную «Игра» (play), которая обозначает возможность проведения матча. На основе погодных условий мы должны определить, состоится ли матч. Чтобы сделать это, необходимо выполнить следующие шаги.

Шаг 1. Преобразуем набор данных в частотную таблицу (frequency table).

Шаг 2. Создадим таблицу правдоподобия (likelihood table), рассчитав соответствующие вероятности. Например, вероятность облачной погоды (overcast) составляет 0,29, а вероятность того, что матч состоится (yes) – 0,64.

Шаг 3. С помощью теоремы Байеса рассчитаем апостериорную вероятность для каждого класса при данных погодных условиях. Класс с наибольшей апостериорной вероятностью будет результатом прогноза.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Рис. 4.7 Пример реализации НБА

Допустим, что дана задача: состоится ли матч при солнечной погоде (sunny)? Такую задачу можно решить с помощью описанного выше подхода.

$$P(Yes \mid Sunny) = P(Sunny \mid Yes) * P(Yes) / P(Sunny)$$

Здесь мы имеем следующие значения:

$$P(Sunny \mid Yes) = 3 / 9 = 0,33$$
$$P(Sunny) = 5 / 14 = 0,36$$
$$P(Yes) = 9 / 14 = 0,64$$

Теперь рассчитаем P(Yes | Sunny):

$$P(Yes \mid Sunny) = 0,33 * 0,64 / 0,36 = 0,60$$

Значит, при солнечной погоде более вероятно, что матч состоится. Аналогичным образом с помощью НБА можно прогнозировать несколько различных классов на основе множества признаков. Этот алгоритм в основном используется в области классификации текстов и при решении задач многоклассовой классификации.

Метод Опорных Векторов. Метод Опорных Векторов или SVM (от англ. Support Vector Machines) — это линейный алгоритм, используемый в задачах классификации и регрессии. Данный алгоритм имеет широкое применение на практике и может решать как линейные так и нелинейные задачи. Основной задачей алгоритма является найти наиболее правильную линию, или гиперплоскость разделяющую данные на два класса. SVM это алгоритм, который получает на входе данные, и возвращает такую разделяющую линию.

Рассмотрим следующий пример, представленный на рис. 4.8. Допустим у нас есть набор данных, и мы хотим классифицировать и разделить красные квадраты от синих кругов (допустим позитивное и отрицательное). Основной целью в данной задаче будет найти “идеальную” линию которая разделит эти два класса.

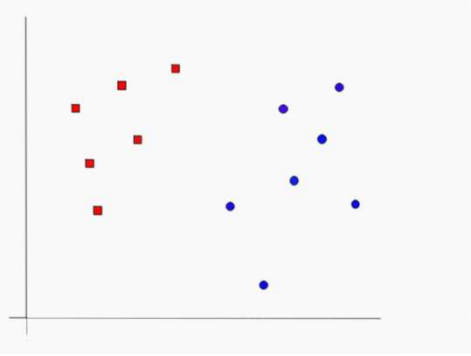

Рис. 4.8 Исходный набор данных

Необходимо найти идеальную линию, или гиперплоскость, которая разделит набор данных на синий и красный классы. Но, как можно заметить, нет одной, уникальной, линии, которая бы решала такую задачу. Мы можем подобрать бесконечное множество таких линий, которые могут разделить эти два класса. Есть два варианта реализации (рис. 4.9).

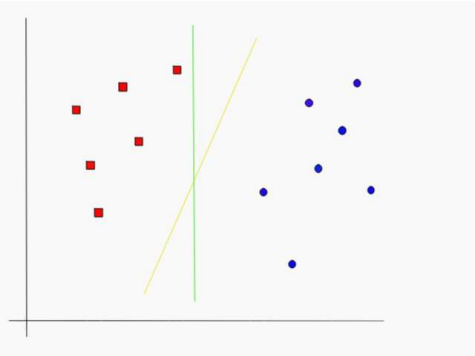

Рис. 4.9 Пример разделение набора на классы

Используя алгоритм SVM решением данной задачи будет выбор желтой линии. В данном примере можно интуитивно понять, что желтая линия разделяет и соответственно классифицирует два класса лучше зеленой.

В случае с зеленой линией — она расположена слишком близко к красному классу. Несмотря на то, что она верно классифицировала все объекты текущего набора данных, такая линия не будет генерализованной — не будет так же хорошо вести себя с незнакомым набором данных. Задача нахождения генерализованной разделяющей двух классов является одной из основных задач в машинном обучении.

Алгоритм SVM устроен таким образом, что он ищет точки на графике, которые расположены непосредственно к линии разделения ближе всего. Эти точки называются опорными векторами. Затем, алгоритм вычисляет расстояние между опорными векторами и разделяющей плоскостью. Это расстояние, которое называется зазором. Основная цель алгоритма — максимизировать расстояние зазора. Лучшей гиперплоскостью считается такая гиперплоскость, для которой этот зазор является максимально большим.

Рассмотрим следующий пример, с более сложным датасетом, который нельзя разделить линейно (рис. 4.10).

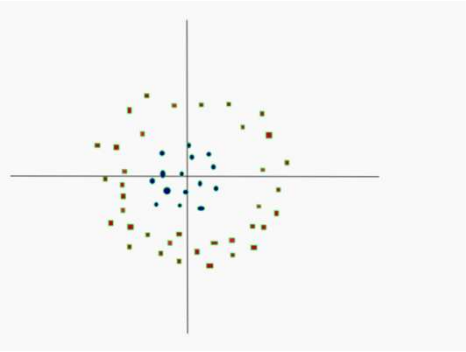

Рис. 4.10 Набор данных, требующий классификации

Очевидно, что этот набор данных нельзя разделить линейно. Невозможно начертить прямую линию, которая бы классифицировала эти данные. Но, этот датасет можно разделить линейно, добавив дополнительное измерение, которое мы назовем осью Z. Представим, что координаты на оси Z регулируются следующим ограничением:

$$z = x^2 + y^2$$

Таким образом, ордината Z представлена из квадрата расстояния точки до начала оси. Визуализация представлена на рис. 4.11.

Теперь данные можно разделить линейно. Допустим пурпурная линия разделяющая данные z=k, где k константа. Если

$$z = x^2 + y^2$$

, то следовательно и

$$k = x^2 + y^2$$

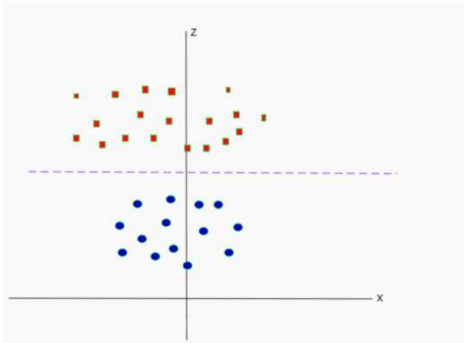

Рис. 4.11 Визуализация набора данных на оси Z.

Таким образом, мы можем спроэцировать наш линейный разделитель, обратно к исходному количеству измерений выборки, используя эту трансформацию (рис. 4.12).

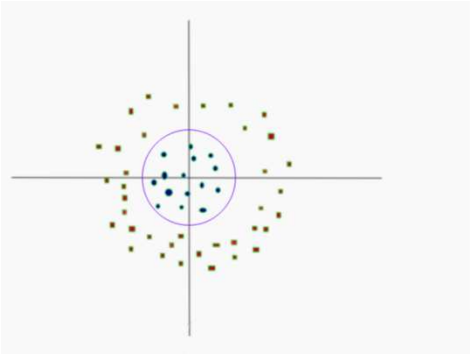

Рис. 4.12 Классификация данных.

В результате, мы можем классифицировать нелинейный набор данных добавив к нему дополнительное измерение, а затем, привести обратно к исходному виду используя математическую трансформацию. Однако, не со всеми наборами данных можно с такой же легкостью провернуть такую трансформацию. К счастью, имплементация этого алгоритма в библиотеке sklearn решает эту задачу за нас.

Для двумерного датасета, представленного выше, разделяющая прямая является гиперплоскостью. Гиперплоскость — это n-1 мерная подплоскость в n-мерном Евклидовом пространстве, которая разделяет пространство на две отдельные части.

Например, представим, что наша линия представлена в виде одномерного Евклидова пространства (т.е. наш набор данных лежит на прямой). Выберите точку на этой линии. Эта точка разделит набор данных, в нашем случае линию, на две части. У линии есть одна мера, а у точки 0 мер. Следовательно, точка — это гиперплоскость линии.

```
  import numpy as np
  X = np.array([[-1, -1], [-2, -1], [1, 1], [2,
1]])
  y = np.array([1, 1, 2, 2])
```

Точки представлены в виде массива X, а классы к которому они принадлежат в виде массива y.

Теперь мы обучим нашу модель этой выборкой. Для данного примера я задал линейный параметр “ядра” классификатора (kernel).

```
from sklearn.svm import SVC c
lf = SVC(kernel='linear')
clf = SVC.fit(X, y)
```

Предсказание класса нового объекта

```
prediction = clf.predict([[0,6]])
```

Параметры — это аргументы которые вы передаете при создании классификатора. Ниже приведены несколько самых важных настраиваемых параметров SVM:

- “C”
- Гамма

Параметр “C” помогает отрегулировать ту тонкую грань между “гладкостью” и точностью классификации объектов обучающей выборки. Чем больше значение “C” тем больше объектов обучающей выборки будут правильно классифицированы (рис. 4.13).

В данном примере есть несколько порогов принятия решений, которые мы можем определить для этой конкретной выборки. Обратите внимание на прямую (представлена на графике в виде зеленой линии) порога решений. Она довольно таки проста, и по этой причине, несколько объектов были классифицированы неверно. Эти точки, которые были классифицированы неправильно называются выбросами в данных.

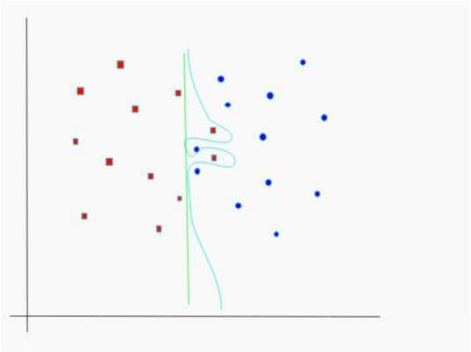

Рис. 4.13Использование параметра “C”

Мы также можем настроить параметры таким образом, что в конечном итоге получим более изогнутую линию (светло-голубой порог решений), которая будет классфицировать асболютно все данные обучающей выборки правильно. Конечно, в таком случае, шансы того, что наша модель сможет генерализовать и показать столь же хорошие результаты на новых данных, катастрофически мала. Следовательно, если вы пытаетесь достигнуть точности при обучении модели, вам стоит нацелиться на что-то более ровное, прямое. Чем выше число “C” тем более запутанная гиперплоскость будет в вашей модели, но и выше число верно-классифицированных объектов обучающей выборки. Поэтому, важно “подкручивать” параметры модели под конкретный набор данных, чтобы избежать переобучения но, в то же время достигнуть высокой точности.

В официальной документации библиотека SciKit Learn говорится, что гамма определяет насколько далеко каждый из элементов в наборе данных имеет влияние при определении “идеальной линии”. Чем ниже гамма, тем больше элементов, даже тех, которые достаточно далеки от разделяющей линии, принимают участие в процессе выбора этой самой линии. Если же, гамма высокая, тогда алгоритм будет “опираться” только на тех элементах, которые наиболее близки к самой линии.

Если задать уровень гаммы слишком высоким, тогда в процессе принятия решения о расположении линии будут учавствовать только самые близкие к линии элементы. Это поможет игнорировать выбросы в данных. Алгоритм SVM устроен таким образом, что точки расположенные наиболее близко относительно друг друга имеют больший вес при принятии решения. Однако при правильной настройке «C» и «gamma» можно добиться оптимального результата,

который построит более линейную гиперплоскость, игнорирующую выбросы, и, следовательно, более генерализуемую.