УДК 681.3.07

Н. Г. Полетаева

КЛАССИФИКАЦИЯ СИСТЕМ МАШИННОГО ОБУЧЕНИЯ

Приведена классификация методов машинного обучения в зависимости от способа обучения, рассмотрены основные алгоритмы машинного обучения и область применения. Особое внимание уделено обучению с подкреплением на основе Марковских процессов принятия решений.

The classification of machine learning methods is given depending on the technique of learning, the main algorithms of machine learning and the scope of application are considered. Particular attention is paid to reinforcement learning based on Markov decision processes.

Ключевые слова: обучение с учителем, обучение без учителя, частичное обучение, обучение с подкреплением.

Keywords: supervised learning, unsupervised learning, semisupervised learning, reinforcement learning.

Машинное обучение — это методика анализа данных, которые позволяют компьютерам самостоятельно обучаться посредством решения массива сходных задач. Классификация и область применения систем машинного обучения приведена на рисунке 1 (с. 6). В [1] выделяют еще категорию «частичное обучение», большинство алгоритмов частичного обучения являются комбинациями алгоритмов обучения с учителем и без.

Обучение с учителем

Обучающие данные включают желательные решения, называемые метками (label). Можно выделить самые важные алгоритмы обучения с учителем:

— линейная регрессия;
— логистическая регрессия;
— метод *k*-ближайших соседей;
— метод опорных векторов;
— деревья принятия решений и случайные леса;
— нейронные сети.

Обучение с учителем (или контролируемое обучение) может быть использовано для решения задач регрессии, чтобы прогнозировать целевое числовое значение переменной, располагая набором характеристик или признаков. Регрессионные проблемы, для которых входы имеют временную разницу, иногда называют анализом временных ря-

дов. Обнаружение аномалий — это слой поверх регрессии: он относится к проблеме определения, когда наблюдаемое значение существенно отличается от прогнозируемого значения.

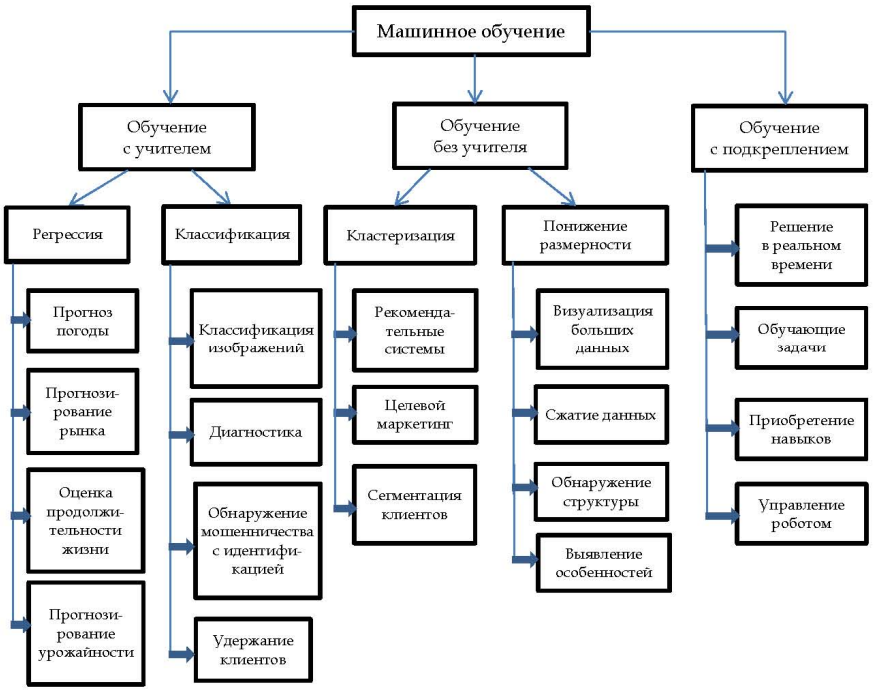

Рис. 1. Классификация и область применения систем машинного обучения

Некоторые алгоритмы регрессии могут применяться также для классификации. Например, логистическая регрессия, так как она способна получать вероятность принадлежности к заданному классу.

Так, метод ***k-ближайших соседей*** (k-nearest neighbors — KNN) — один из простых и часто используемых непараметрических методов классификации. Он состоит из следующих шагов:

— выбрать число *k* и метрику расстояния;

— найти *k*-ближайших соседей образца, который необходимо классифицировать;

— присвоить метку класса мажоритарным голосованием.

Метрики расстояния для определения того, как «ближние» точки расположены друг к другу в пространстве признаков, обычно представляют собой евклидово расстояние для непрерывных переменных и расстояние Хэмминга — для дискретных переменных. Основной недостаток алгоритма KNN заключается в создании моделей больших размеров, которые должны хранить все обучающие данные, содержащие векторы признаков и метки.

Идея ближайшего соседа расширяется и на другие задачи. Например, в задаче восстановления плотности вероятности с применением обобщенных оценок типа *k*-ближайших узлов [2]:

$$\hat{f}_N(x,k)=\frac{1}{N\delta_N(x,k)}\sum_{i=1}^{N}h\left(\frac{x-X^{(i)}}{\delta_N(x,k)}\right), \quad (1)$$

где параметр локальности $\delta_N(x,k)$ есть радиус наименьшего шара с центром в точке x такого, что в нем находится не менее k узлов $X^{(i)}$, $X^{(i)}\in R^n$.

Настроечным параметром оценки (1) является значение k, то есть количество ближайших узлов. Для его определения применяется принцип максимума эмпирического перекрестного правдоподобия в форме

$$\hat{k}_N=\arg\max_k \hat{L}(k), \quad \hat{L}(k)=\prod_{j=1}^{N}\hat{f}_N^{(j)}(X^{(j)},k), \quad (2)$$

где оценка плотности вероятности $\hat{f}_N^{(j)}(x,k)$ вычислена по всем узлам $\{X^{(i)}\}_1^N$, кроме j-го:

$$\hat{f}_N^{(j)}(x,k)=\frac{1}{N\delta_N(x,k)}\sum_{\substack{i=1\\ i\neq j}}^{N}h\left(\frac{x-X^{(i)}}{\delta_N(x,k)}\right). \quad (3)$$

Задача разбиения экспериментального материала на обучающую и контрольную группы в данном случае имеет тот же смысл и порождает те же проблемы, что и в задачах классификации и построения регрессионных зависимостей. Соотношения (2), (3) определяют один из вариантов экономного использования экспериментального материала, при котором одни и те же данные различным образом многократно разбиваются на обучающее и контрольное множества. Таким способом формирования функции правдоподобия приходится пользоваться при сравнительно небольших объемах выборки N. Подчеркнем содержательно наиболее важный момент: то или иное разбиение наблюдений на две группы необходимо для проверки качества оценок — возможности их использования для интерполяции восстанавливаемой зависимости.

Метод опорных векторов (support vector machine — SVM) является (в простейшем виде) линейным классификатором подобно логистической регрессии, так как создает гиперплоскость в векторном пространстве, которая пытается разделить два класса в наборе данных. Метод SVM отличается от логистической регрессии функцией издержек. Логистическая регрессия использует функцию логарифмического правдоподобия, которая штрафует все точки пропорционально ошибке в оценке вероятности — даже те, которые находятся на правильной стороне гиперплоскости. Классификатор SVM пытается найти гиперплоскость с максимальным зазором, разделяющую два класса, где «зазор» указывает расстояние от разделяющей плоскости (границы решения) до ближайших обучающих образцов с каждой стороны, так называемых опорных векторов. Метод SVM использует *петлевую функцию*, ко-

торая штрафует только точки, расположенные на неправильной стороне гиперплоскости или очень близко к ней на правильной стороне (нарушение зазора образцом равно 0, если он расположен вне полосы на корректной стороне).

Еще одна причина, по которой модели SVM столь популярны, в том, что их можно легко модифицировать с использованием ядра для решения нелинейных задач классификации. Ключевая идея в основе ядерных методов для решения задач с линейно неразделимыми данными состоит в том, чтобы создать нелинейные комбинации исходных признаков и функцией отображения $\phi(\cdot)$ спроецировать их на пространство более высокой размерности, где они становятся линейно разделимыми. Затем ту же самую функцию отображения $\phi(\cdot)$ можно использовать для трансформации новых, ранее не наблюдавшихся данных, чтобы их классифицировать при помощи линейной модели SVM [3]. Именно в этом подходе можно применить так называемый ядерный трюк (или подмену скалярного произведения функцией ядра). На математическом уровне ядро преобразует одно векторное пространство V_1 в другое пространство V_2 и представляет собой функцию $K(x,y)$ на $V_1 \times V_2$. Каждый $x \in V_1$ отображается с помощью $K(x,\cdot)$, а V_2 — пространство, охватываемое всеми такими функциями. На практике нужно всего лишь на шаге прямого вычисления скалярного произведения между двумя точками подставить ядерную функцию:

$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)}).$$

Одним из наиболее часто используемых ядер является гауссово ядро радиальной базисной функции (radial basis function — RBF):

$$K(x^{(i)}, x^{(j)}) = \exp\left(-\gamma \left\| x^{(i)} - x^{(j)} \right\|^2\right),$$

где $\gamma = \frac{1}{2\sigma^2}$ — параметр, который нужно оптимизировать; $\left\| x^{(i)} - x^{(j)} \right\|$ — евклидово расстояние. Термин ядро можно интерпретировать как функцию подобия между парой образцов. Теперь необходимо обучить ядро SVM, чтобы оно могло провести нелинейную границу решения, хорошо разделяющую данные.

Чтобы продемонстрировать работу метода, сначала создадим набор данных для нелинейной задачи классификации, который имеет вид логического элемента XOR, используя для этого функцию logical_xor из библиотеки NumPy языка программирования Python, где первым 100 образцам назначается метка класса 1, другим 100 образцам — метка класса –1:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import random
```

```
np.random.seed(0)
X_xor = np.random.randn(200 , 2)
y_xor = np.logical_xor(X_xor[ :, 0] >0 , X_xor[: , 1] > 0)
y_xor = np.where (y_xor , 1, - 1)
plt.scatter(X_xor[y_xor==1 , 0] , X_xor[y_xor==1 , 1] ,
   c= 'tab:orange', marker= '^', label= '1')
plt.scatter(X_xor[y_xor==-1 , 0] , X_xor[y_xor==-1 , 1],
   c= 'tab:blue', marker= 's', label= '-1')
plt.ylim (-3.0)
plt.legend ()
plt.show ()
```


Набор данных XOR со случайным шумом, сгенерированный приведенным выше кодом, показан на рисунке 2. Совершенно очевидно, что не получится хорошо разделить образцы из положительного и отрицательного классов, применив в качестве границы решения линейную гиперплоскость, построенную методом линейной логистической регрессии либо линейной модели SVM.

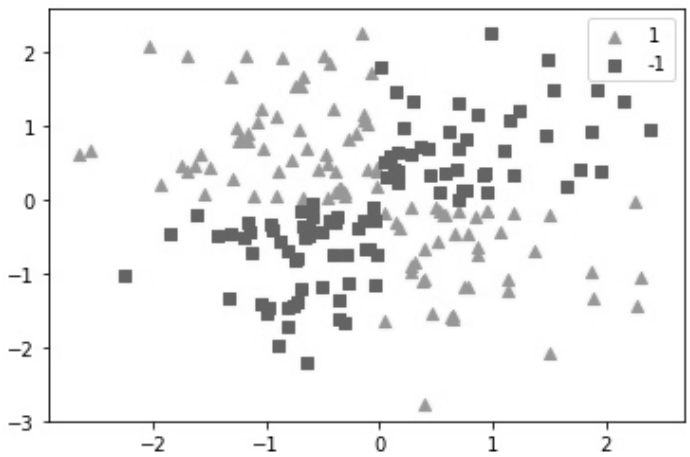

Рис. 2. Набор данных для нелинейной задачи классификации

Для решения поставленной задачи будем использовать Python-пакет Scikit-Learn (универсальная библиотека с открытым исходным кодом для анализа данных), в котором реализованы все основные алгоритмы машинного обучения. Применим класс SVC из библиотеки Scikit-Learn со следующими параметрами:

```
from sklearn.svm import SVC
svm = SVC(kernel= 'rbf', random_state=0 , gamma=0.10, C=10.0)
```

Здесь kernel — ядро RBF; gamma — параметр ядра; C — параметр регуляризации SVC, управляющий зазором и общий для всех ядер метода SVM.

После обучения SVM с ядром радиальной базисной функции на демонстрационном наборе данных получим относительно хорошее разделение данных XOR (рис. 3):

```
from mlxtend.plotting import plot_decision_regions
svm.fit(X_xor, y_xor)
plot_decision_regions (X_xor, y_xor, clf=svm)
plt.legend(loc='upper right')
plt.show()
```


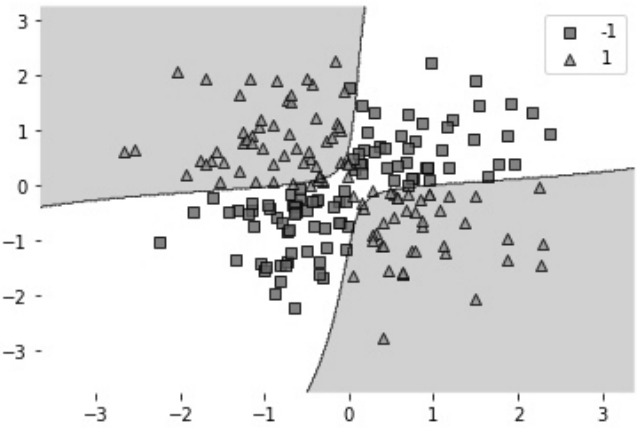

Рис. 3. Использование ядерного трюка
для нахождения разделяющих гиперплоскостей

Параметр γ (gamma) является параметром отсечения для гауссовой сферы. С ростом величины γ увеличивается охват обучающих образцов, что ведет к более мягкой границе решения.

Наивные байесовские классификаторы очень популярны в приложениях фильтрации спама в электронных сообщениях, так как имеют тенденцию обучаться быстрее других классификаторов. Классификатор называют «наивным», потому что он делает сильное статистическое предположение: признаки выбираются независимо от некоторого (неизвестного) распределения, по каждому признаку собираются простые статистики классов.

Деревья принятия решений — универсальные контролируемые модели обучения, которые имеют важное свойство — быть легко интерпретируемым. Этот алгоритм можно использовать как для классификации, так и для прогнозирования. Дерево решений представляет собой древовидную структуру в виде набора связанных узлов. Во многих задачах применяется бинарное дерево, каждый узел такого дерева разбивает данные на два множества с помощью порогового значения одного из информативных признаков. Дочерние подмножества далее рекурсивно разделяются на меньшие подмножества на основе других условий. Условия разделения автоматически выбираются на каждом шаге для наилучшего разбиения набора элементов.

Для того чтобы расщепить узлы в самых информативных признаках, определим в качестве целевой функции, которую необходимо оптимизировать алгоритмом обучения на основе дерева, *прирост информации* (information gain — IG) при каждом расщеплении:

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j), \quad (4)$$

где f — это признак, по которому выполняется расщепление; D_p и D_j — набор данных родительского и j-го дочернего узла; I — мера неоднородности; N_p — общее число образцов в родительском узле и N_j — число образцов в j-ом дочернем узле.

Прирост информации является мерой «чистоты» подмножеств, полученных в результате разделения. Из соотношения (4) видно, что IG — это разница между неоднородностью родительского узла и суммой неоднородностей дочерних узлов (чем меньше неоднородность дочерних узлов, тем больше прирост информации). Для уменьшения комбинаторного пространства поиска в большинстве библиотек (включая Scikit-Learn) реализованы бинарные деревья решений, то есть каждый родительский узел расщепляется на два дочерних узла.

В бинарных деревьях решений обычно используются три *меры неоднородности*, или критерия расщепления: мера неоднородности Джини (I_G), энтропия (I_H) и ошибка классификации (I_E).

Энтропия для всех непустых классов $p(i \mid t) \neq 0$ определяется следующим образом:

$$I_H(t) = -\sum_{i=1}^{c} p(i \mid t) \log_2 p(i \mid t),$$

где $p(i \mid t)$ — это доля образцов, которая принадлежит классу i для отдельно взятого узла t. Энтропия равна 0, если все образцы в узле принадлежат одному и тому же классу, и энтропия максимальна, если распределение классов равномерное.

Мера неоднородности Джини определяется как критерий, который минимизирует вероятность ошибочной классификации:

$$I_G(t) = \sum_{i=1}^{c} p(i \mid t)(1 - p(i \mid t)) = 1 - \sum_{i=1}^{c} p^2(i \mid t).$$

Подобно энтропии, мера неоднородности Джини максимальна, если классы полностью перемешаны — например, в конфигурации с бинарными классами (c = 2): $I_G(t) = 0{,}5$. Однако на практике мера неоднородности Джини и энтропия, как правило, дают очень похожие результаты.

Еще одной мерой неоднородности является ошибка классификации:

$$I_E(t) = 1 - \max_i \{p(i \mid t)\}.$$

Этот критерий менее чувствителен к изменениям вероятностей классов в узлах.

Теперь, используя библиотеку Scikit-Learn, можно обучить дерево решений с максимальной глубиной 2 для набора данных «ирисы Фишера», применив в качестве критерия неоднородности энтропию (рис. 4). Соответствующий исходный код выглядит следующим образом:


```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
iris = load_iris()
X = iris.data[:, 2:]      # длина и ширина лепестка
y = iris.target
tree_clf = DecisionTreeClassifier(criterion='entropy', max_depth=2)
tree_clf.fit(X, y)
plt.figure(figsize=((10,8)))
plot_tree(tree_clf, filled=True, feature_names=iris.feature_names[2:],
  class_names=iris.target_names, rounded=True)
plt.show()
```

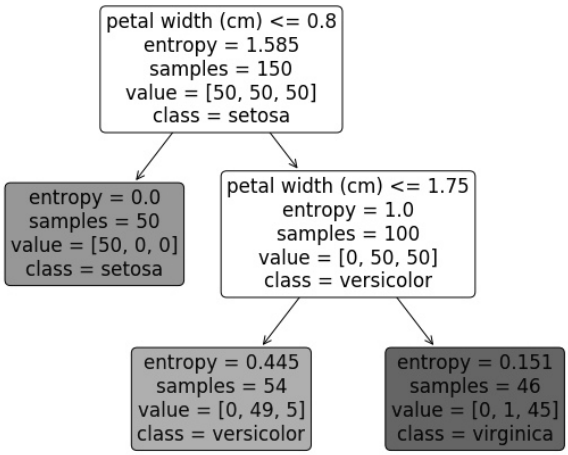

Рис. 4. Дерево принятия решений для набора данных «ирисы Фишера»

Важным качеством деревьев принятия решений является относительная простота объяснения результатов классификации или регрессии, поскольку каждое предсказание может быть выражено в серии логических условий, которые прослеживают путь от корня дерева до конечного узла.

Случайный лес — модель, состоящая из множества деревьев решений. Вместо того чтобы просто усреднять прогнозы разных деревьев (такая концепция называется просто «лес»), эта модель обучает группу классификаторов на основе деревьев принятия решений, задействовав для каждого отличающийся случайный поднабор обучающего набора.

В процессе обучения каждое дерево случайного леса учится на случайном образце из набора данных. Выборка образцов происходит с заменой (возвращением) — в статистике этот метод называется *бутстреппингом* (bootstrapping). Это дает возможность повторно использовать образцы одним и тем же деревом. При тестировании результат выводится путем усреднения прогнозов, полученных от каждого дерева. Подход, при котором каждый обучающийся элемент получает собственный набор обучающих данных (с помощью бутстреппинга), после чего результат усредняется, называется *бэггинг* (bagging, сокр. от bootstrap aggregating). Когда выборка выполняется без замены, такой метод называется вставкой или вклеиванием (pasting). После обучения каждого отдельного дерева решений общие предсказания случайных лесов делаются путем выбора статистического режима прогнозов отдельных деревьев для деревьев классификации (то есть каждое дерево «голосует») и среднего статистического значения прогнозов отдельных деревьев для деревьев регрессии. Однако повышенная сложность случайных лесов затрудняет анализ предсказаний по сравнению с одиночными деревьями решений.

Класс RandomForestClassifier из библиотеки Scikit-Learn имеет определенные параметры, которые либо уникальны для случайных лесов, либо особенно важны: параметр n_estimators задает число деревьев решений для включения в лес; параметр max_features определяет максимальное число отбираемых признаков; параметр bootstrap позволяет установить тип выборки — с заменой (настройка по умолчанию) или без замены.

Одиночные деревья принятия решений имеют тенденцию перестраиваться под свои обучающие наборы, и случайные леса смягчают этот эффект, принимая среднее значение нескольких деревьев принятия решений, что обычно улучшает производительность модели. Кроме того, поскольку каждое дерево в случайном лесу может быть обучено независимо от всех других деревьев, распараллелить алгоритм обучения очень просто. Поэтому случайные леса крайне эффективны для обучения.

Еще одним замечательным качеством случайных лесов является то, что они позволяют легко измерить относительную важность каждого признака. Scikit-Learn измеряет важность признака путем выяснения, насколько узлы дерева, использующие этот признак, уменьшают загрязненность в среднем (по всем деревьям в лесу). Точнее, это взвешенное среднее, где вес каждого узла равен количеству связанных с ним обучающих образцов.

Scikit-Learn вычисляет эту оценку автоматически для каждого признака после обучения, а затем масштабирует результаты так, чтобы сумма всех значимостей была равна 1. Можно получить доступ к итоговому признаку с помощью переменной feature_importances_. Например, следующий код обучает RandomForestClassifier на наборе данных «ирисы Фишера» и выводит значение каждого признака:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1)
rnd_clf.fit(iris["data"], iris["target"])
for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
print(name, score)
sepal length (cm) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```


Очевидно, что наиболее важными признаками являются длина лепестка (44 %) и ширина (42 %), в то время как значимость длины и ширины чашелистика по сравнению с ними намного ниже (11 % и 2 % соответственно).

Нейронные сети. Другим способом прогнозирования данных временных рядов является использование искусственных нейронных сетей, в частности рекуррентных нейронных сетей (recurrent neural network — RNN), которые уникально спроектированы для изучения тенденций и моделей при вводе временных рядов в целях классификации или прогнозирования.

Многослойный персептрон (multilayer perceptron — MLP) относится к классу простых нейронных сетей прямого распространения, которые могут создавать нелинейные границы принятия решений. Для обучения MLP применяют *алгоритм обратного распространения ошибки*. Его можно описать как градиентный спуск, использующий автоматическое дифференцирование в обратном режиме. Для каждого обучающего образца алгоритм сначала вырабатывает прогноз (прямой проход), измеряет ошибку, затем проходит через каждый слой в обратном направлении, чтобы измерить вклад в ошибку каждой связи (обратный проход), и, наконец, немного подстраивает веса связей с целью уменьшения ошибки (шаг градиентного спуска). Чтобы этот алгоритм работал правильно, разработчики внесли ключевое изменение в архитектуру MLP: заменили ступенчатую функцию активации логистической функцией $f(z) = 1/(1+\exp(-z))$.

Нейронные сети применяются для построения очень сложных моделей, особенно в случае больших наборов данных. Однако, во-первых, они чувствительны к масштабированию данных, а во-вторых, требуется много времени для обучения таких моделей. Гибкость нейронных сетей также является одним из их главных недостатков: существует много гиперпараметров для подстройки. Можно не только применять любую воображаемую топологию сети (способ связывания нейронов друг с другом), но даже в простом MLP допускается изменять число слоев, количество нейронов на слой, тип функции активации, используемой в каждом слое, логику инициализации весов и др. Для нахождения наи-

лучшей комбинации значений гиперпараметров применяют два метода: решетчатый поиск и рандомизированный. Чтобы выполнить решетчатый поиск, необходимо задействовать класс GridSearchCV из библиотеки Scikit-Learn, передав ему набор параметров и значения, которые нужно опробовать. GridSearchCV оценит всевозможные комбинации значений гиперпараметров, применяя перекрестную проверку. Подход с решетчатым поиском хорош для небольшого числа комбинаций, но, когда пространство поиска гиперпараметров является большим, предпочтительнее воспользоваться рандомизированным поиском, который реализован классом RandomizedSearchCV. Этот класс можно применять почти так же, как и класс GridSearchCV, но, вместо того чтобы опробовать всевозможные комбинации, он оценивает заданное количество случайных комбинаций, выбирая случайное значение для каждого гиперпараметра на любой итерации.

Для *контролируемого обучения* необходимо разделить набор данных на обучающие и тестовые наборы. По сути, алгоритм машинного обучения берет в себя набор учебных материалов и выводит модель. Модель представляет собой алгоритм, который принимает новые точки данных в том же виде, что и данные обучения, и выводит предсказание. Все алгоритмы машинного обучения определяются тремя взаимозависимыми компонентами:

1) семейство моделей, в котором описывается множество моделей, из которых можно выбирать;

2) функция издержек, которая позволяет количественно сравнивать разные модели;

3) процедура оптимизации, выбирающая лучшую модель из множества.

Обучение без учителя

Обучающие данные не помечены, система пытается обучаться без учителя. Если в контролируемом обучении система на маркированных тренировочных данных пытается извлечь модель, которая позволяет делать прогнозы о ранее не встречавшихся или будущих данных, то в неконтролируемом обучении система старается самостоятельно найти шаблоны непосредственно из набора данных. Основные алгоритмы обучения без учителя:

а) кластеризация:
- метод *k*-средних;
- иерархический кластерный анализ;
- максимизация ожиданий;
- нейронная сеть Кохонена;

б) визуализация и понижение размерности:
- анализ главных компонентов;
- ядерный анализ главных компонентов;
- локальное линейное вложение;
- стохастическое вложение соседей с t-распределением;

в) обучение ассоциативным правилам.

Метод k-средних применяется к вещественным векторам, когда точно известно количество кластеров. Цель алгоритма — назначить каждую точку набора данных кластеру таким образом, чтобы сумма расстояний от каждой точки до центра кластера была минимальной. Здесь понятие «расстояние» является обычным евклидовым расстоянием в векторном пространстве:

$$d(x,y)=\sqrt{\sum_i (x_i-y_i)^2}=\|X-Y\|.$$

В математических терминах алгоритм k-средних вычисляет кластерное назначение W: $X\to\{1,...,k\}$, которое минимизирует внутрикластерную сумму квадратичных ошибок (*SSE*):

$$SSE=\sum_{i=1}^{N}\sum_{j=1}^{k}\omega^{(i,j)}\left\|X^{(i)}-\mu^{(j)}\right\|^2,$$

где k — число кластеров; $\mu^{(j)}$ — центры масс векторов $X^{(i)}\in R^n$, $i=1,2,\ldots,N$, $\omega^{(i,j)}=1$ в случае, если образец $X^{(i)}$ находится в группе j, и $\omega^{(i,j)}=0$ в противном случае. Значение *SSE* иногда также называется «инерцией» кластера.

Иерархический кластерный анализ используется для группировки непомеченных точек данных, имеющих сходные характеристики. Существует два основных подхода к иерархической кластеризации: дивизивный (разделяющий) и агломеративный (объединяющий). Дивизивный алгоритм начинается с единственного кластера, который охватывает все образцы и итеративно расщепляет кластер на более мелкие, пока каждый кластер не будет содержать всего один образец. В агломеративном алгоритме принят противоположный подход: каждый образец рассматривается как отдельный кластер, далее алгоритм объединяет ближайшие пары кластеров.

Агломеративная иерархическая кластеризация представлена двумя стандартными алгоритмами: методом одиночной связи (single linkage) и методом полной связи (complete linkage). Используя метод одиночной связи для каждой пары кластеров вычисляются расстояния между самыми похожими членами и объединяются два кластера, для которых расстояние между самыми похожими членами наименьшее. Подход на основе полной связи подобен методу одиночной связи, но в каждой паре кластеров сравниваются наиболее различающиеся члены для объединения.

Результаты работы иерархических алгоритмов обычно представляются в виде дендрограммы, построенной по сжатой матрице расстояний [3].

Анализ главных компонент (principal component analysis — PCA) — это метод линейного преобразования, который широко используется в самых разных областях, чаще всего для снижения размерности. Алгоритм PCA определяет ось, на долю которой приходится самая крупная

величина дисперсии в обучающем наборе. Он также находит вторую ось, ортогональную к первой, на долю которой приходится самая крупная величина оставшейся дисперсии, и т.д. При проекции на такие оси (снижении размерности) сохраняется наибольшее количество информации. Ортогональные оси (главные компоненты) нового подпространства можно интерпретировать как направления максимальной дисперсии при условии, что оси новых признаков ортогональны друг другу. Класс PCA из библиотеки Scikit-Learn реализует метод PCA с использованием сингулярного разложения SVD. Следующий код применяет PCA для уменьшения размерности набора данных до двух измерений (алгоритм автоматически выполняет центрирование данных):


```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

После подгонки преобразователя PCA к набору данных можно обращаться к главным компонентам с помощью переменной components_, которая хранит их в виде горизонтальных векторов, поэтому, например, первым главным компонентом будет pca.components_.T[:, 0].

Более того, ядерный трюк может быть применен к PCA, что позволяет выполнять сложные нелинейные проекции для понижения размерности. Этот алгоритм называется ядерным PCA (kPCA). Например, следующий код использует класс KernelPCA из библиотеки Scikit-Learn для выполнения kPCA с ядром RBF:

```
from sklearn.decomposition import KernelPCA
rbf_pca = KernelPCA(n_components = 2, kernel='rbf', gamma=0.04)
X_reduced = rbf_pca.fit_transform(X)
```

Обучение на базе многообразий (manifold learning) — это класс алгоритмов без учителя, нацеленных на описание наборов данных как низкоразмерных многообразий, вложенных в пространство большей размерности.

1. *Метод t-SNE* (t-distributed stochastic neighbor embedding — распределенное стохастическое соседнее вложение) используется для визуализации отображения пространства высокой размерности в пространство меньшей размерности. Алгоритм t-SNE начинается с преобразования многомерного евклидового расстояния между точками в условные вероятности, отражающие сходство точек:

$$p_{j|i} = \frac{\exp\left(-\left\|x_i - x_j\right\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\left\|x_i - x_k\right\|^2 / 2\sigma_i^2\right)},$$

где среднеквадратичное отклонение σ_i — параметр, который нужно оптимизировать.

Теперь определим матрицу сходства для исходного набора данных как симметричный вариант условной вероятности:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}. \quad (5)$$

Также определим матрицу сходства для точек отображения:

$$q_{ij} = \frac{f(|x_i - x_j|)}{\sum_{k \neq i} f(|x_i - x_k|)} \quad c \quad f(z) = \frac{1}{1+z^2}. \quad (6)$$

Здесь применяется такой же подход, как и для точек данных, но используется другое распределение (распределение Стьюдента с одной степенью свободы или распределение Коши вместо гауссова распределения). Алгоритм t-SNE стремится к тому, чтобы две эти матрицы были максимально близкими, это соответствует минимизации расстояния Кульбака — Лейблера между распределениями (5) и (6):

$$KL(P||Q) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

2. *Многомерное шкалирование* (MDS) понижает размерность, одновременно пытаясь сохранить расстояния между образцами.

3. *Изометрическое отображение* (Isomap) создает граф, соединяя каждый образец с его ближайшими соседями, а затем понижает размерность, пытаясь сохранить геодезические расстояния между образцами (геодезическое расстояние между двумя узлами в графе представляет собой количество узлов на кратчайшем пути между этими узлами).

Локально-линейные методы снижения размерности базируются в основном на сохранении свойств в малых окрестностях точек.

Например, вот как работает метод *локально линейного вложения* (LLE): сначала для каждого обучающего образца $x^{(i)} \in R^n$ алгоритм идентифицирует его *k*-ближайших соседей, а затем пытается восстановить $x^{(i)}$ как линейную функцию этих соседей. Более конкретно, он находит такие веса $w_{i,j}$, которые минимизируют сумму квадратичных ошибок:

$$\hat{W} = \arg\min_{w} \sum_{i=1}^{N} \left(x^{(i)} - \sum_{j=1}^{N} w_{i,j} x^{(j)} \right)^2$$

при условии

$$\begin{cases} w_{i,j} = 0, x^{(j)} \notin S_k^{(i)}, \\ \sum_{j=1}^{N} w_{i,j} = 1, i = 1,2,...,N' \end{cases}$$

где W — матрица весов, содержащая все веса $w_{i,j}$; $S_k^{(i)}$ — множество k-ближайших соседей $x^{(i)}$. Второе ограничение нормализует веса для каждого обучающего образца $x^{(i)}$.

После этого шага матрица весов $\hat{W}(\hat{w}_{i,j})$ представляет локальные линейные связи между обучающими образцами. Теперь второй шаг — отобразить обучающие образцы на d-мерное пространство (где $d < n$) с одновременным сохранением как можно большего числа имеющихся локальных связей. Если $z^{(i)}$ является образом $x^{(i)}$ в этом d-мерном пространстве, то необходимо минимизировать квадрат расстояния между $z^{(i)}$ и $\sum_{j=1}^{N} \hat{w}_{i,j} z^{(j)}$. Таким образом, получаем задачу безусловной оптимизации:

$$\hat{Z} = \arg\min_{z} \sum_{i=1}^{N} \left(z^{(i)} - \sum_{j=1}^{N} \hat{w}_{i,j} z^{(j)} \right)^2 ,$$

где Z — это матрица, содержащая все $z^{(i)}$. Это очень похоже на первый шаг, но, вместо того чтобы фиксировать образцы и находить оптимальные веса, делаем обратное: фиксируем веса и находим оптимальное положение отражений образцов в пространстве с более низким числом измерений.

Обучение ассоциативным правилам. Выявление закономерностей между связанными событиями может быть использовано для понимания природы анализируемых данных. Основные меры поиска ассоциативных правил: поддержка (support) — частота появления правила, достоверность (confidence) — показатель, характеризующий уверенность в том, что правило на самом деле верное (оценка условной вероятности).

Обучение с подкреплением

Обучающая система или программный агент может наблюдать, выбирать и выполнять действия внутри среды (среда — это замкнутый физический или виртуальный объект), получая в ответ награды или штрафы в форме отрицательных наград. Затем агент должен определить наилучшую стратегию, называемую политикой, для получения максимальной награды [1]. Например, политикой может быть нейронная сеть, которая на входе принимает наблюдения и выдает действие, подлежащее выполнению, по оценочным вероятностям для каждого действия (рис. 5). Другие способы исследования пространства политик связаны с использованием генетических алгоритмов и градиентных методов для определения оптимальной политики.

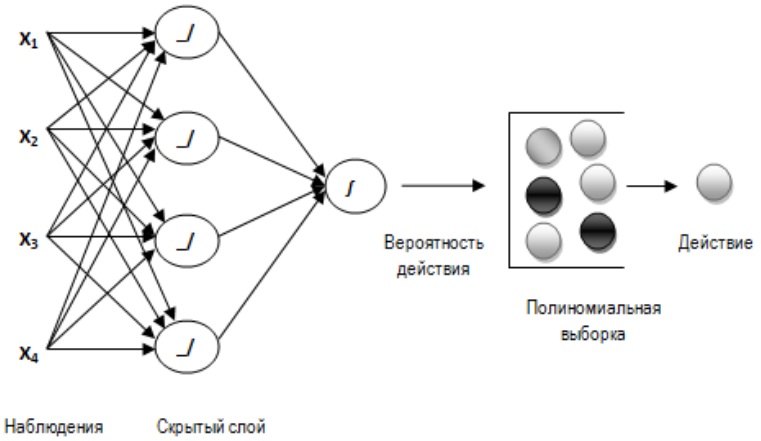

Рис. 5. Политика в форме нейронной сети

Марковские системы принятия решений. Марковский процесс принятия решений состоит из множества состояний S, множества действий A, функции перехода $T(s,a,s')$ из состояния s в состояние s' при условии, что агент выбрал действие $a \in A$ и функции вознаграждения $R(s,a,s')$ для этого перехода. Для марковского процесса справедливо утверждение: вероятности переходов между состояниями не зависят от истории предыдущих переходов.

Оценка оптимальной ценности $V^*(s)$ состояния, предложенная Р. Беллманом, представляет собой ожидаемую суммарную прибыль с учетом дисконтной ставки γ:

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left\{R(s,a,s') + \gamma \cdot V^*(s')\right\}.$$

Обозначим через $\pi(a,s)$ вероятность выбора действия $a \in A$ в состоянии s, $Q^*(s,a)$ — оптимальная Q-ценность пары «состояние — действие» (s,a). Тогда, если агент будет следовать оптимальной стратегии $\pi(a,s)$, то для всех пар (s,a) справедливо:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left\{R(s,a,s') + \gamma \cdot \max_{a'} Q_k(s',a')\right\}. \qquad (7)$$

Выбор оптимальной политики определяется из подсчитанных значений $Q^*(s,a)$:

$$\pi^*(s) = \arg\max_a Q^*(s,a). \qquad (8)$$

Данный подход (7), (8) был адаптирован для поиска оптимального маршрута передвижения мобильного агента в условиях дорожного движения и сохранения конфиденциальности его местоположения [4].

Коммуникационные возможности мобильных устройств позволяют отслеживать местоположение мобильных агентов по их уникальному идентификатору путем прослушивания сети.

Предположим, что злоумышленник имеет ограниченное число прослушивающих станций для расположения в сети с целью обнаружения мобильных агентов на перекрестках. Ограничение на количество прослушивающих станций связано с затратами на установку каждой станции.

Для обеспечения своей конфиденциальности мобильные агенты могут использовать несколько уникальных идентификаторов. Смена идентификаторов происходит в зонах смешивания, которые требуют определенных затрат — стоимости смены псевдонимов и пребывания в состоянии молчания для мобильного агента [5].

В моделируемой конфликтной ситуации, основанной на теоретико-игровом взаимодействии мобильных агентов и злоумышленника, при расчете ценности состояний вводятся некоторые определенные правила в качестве исходных данных для выбора оптимальной стратегии [4].

Предположим, что a — действие, выполняемое в состоянии s (выбор пути). Оно определяется следующим образом (рис. 6):

$$a = \begin{cases} 0{,}8 = const & \text{для выбранного направления движения}, \\ 0{,}1 = const & \text{для всех остальных возможных вариантов}. \end{cases}$$

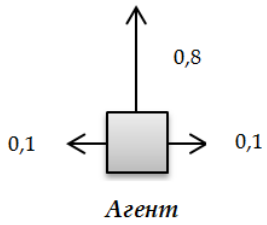

Рис. 6. Действие a, выполняемое агентом в состоянии s

Применим метод Беллмана для связанного графа, представленного на рисунке 7.

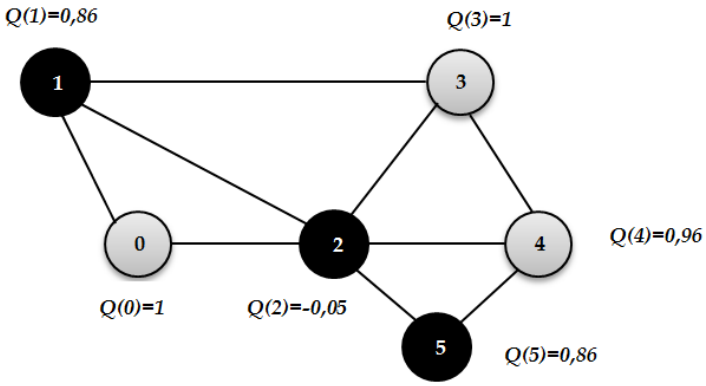

Рис. 7. Ценности состояний в связанном графе, $\gamma = 1$ и $R(s) = -0{,}04$

Для оценивания предлагаемого метода Беллмана были рассмотрены следующие алгоритмы: Дейкстры, Флойда — Уоршелла. Среднее время выполнения алгоритмов оптимизации приведено в таблице.

Сравнение разработанных алгоритмов по скорости

Процессор AMD Athlon II X4 630 2,8 ГГц	Метод Беллмана	Алгоритм Дейкстры	Алгоритм Флойда — Уоршелла
Среднее время поиска	0,821 с	0,648 с	0,810 с

Классические алгоритмы поиска выдали путь 0-2-3 (рис. 7), так как расстояние между этими узлами минимальное. Это ведет к раскрытию агента, поскольку он проходит через перекресток, на котором с большой долей вероятности находится прослушивающая станция $Q(2) = -0{,}05$. Алгоритм Беллмана предложил маршрут 0-2-4-3, так как ценность данного маршрута максимальна: агент меняет идентификатор в зоне смешивания $Q(4) = 0{,}96$, чтобы безопасно достичь цели в вершине 3. При этом реализация метода Беллмана не сильно уступает реализации алгоритма Дейкстры по скорости, находясь на том же уровне, что и алгоритм Флойда — Уоршелла [4].

Список литературы

1. *Жерон О.* Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем. СПб., 2018.

2. *Катковник В.Я., Полетаева Н.Г.* Принцип максимума эмпирического правдоподобия для выбора параметра сглаживания в непараметрических оценках плотности // Вопросы кибернетики. Актуальные задачи адаптивного управления. М., 1982. Вып. 89. С. 90—102.

3. *Рашка С.* Python и машинное обучение. М., 2017.

4. *Васильев С.П., Полетаева Н.Г.* Применение методов машинного обучения в задачах оптимизации // Информационные системы и технологии: теория и практика : сб. науч. тр. СПб., 2019. Вып. 11. С. 28—40.

5. *Buttyan L., Holczer T., Vajda I.* On the effectiveness of changing pseudonyms to provide location privacy in VANETs // Proceedings of 4th European Workshop on Security and Privacy in Ad-hoc and Sensor Networks. Cambridge, 2007. P. 129—141.

Об авторе

Наталья Григорьевна Полетаева — канд. техн. наук, ст. преп., Балтийский федеральный университет им. И. Канта, Россия.

E-mail: spur49@mail.ru

The author

Dr Natalya G. Poletaeva, Assistant Professor, Immanuel Kant Baltic Federal University, Russia.

E-mail: spur49@mail.ru