**Дисциплина «Искусственный интеллект»**

**Рабочая тетрадь № 7**

|  |
| --- |
| **Теоретический материал – Нейронные сети** |
| Обучение персептрона  Персептрон представляет собой элементарную часть нейронной сети. Одиночный персептрон является линейным бинарным классификатором. В этой лекции мы рассмотрим процедуру обучения персептрона для классификации данных. Поскольку персептрон представляет собой бинарный классификатор, то мы будем рассматривать лишь два класса.  Пусть мы рассматриваем некоторое множество (конечное или бесконечное) n-мерных векторов, которые будем обозначать  Будем считать, что это множество разбивается на два класса, которые мы будем обозначать +1 и -1. Поэтому возникает задача построения функции, которая задана на нашем множестве векторов, и принимает значения в множестве . В качестве такой функции может выступать персептрон. С алгебраической точки зрения персептрон состоит из вектора весов .  При этом персептрон работает по формуле  ,  где функция равна , если , и равна , если .  Приведем алгоритм обучения персептрона. Пусть у нас есть набор обучающих данных , где - это различные вектора, а из множества указывает к какому классу относится наш вектор.   1. Положим вектор весов равным нулю. 2. Повторять раз следующие шаги: 3. Для каждого тестового набора : 4. Вычислить . 5. Если , то скорректировать веса .   Описанный алгоритм довольно легко программировать. |

|  |  |
| --- | --- |
| **1.1.1 Пример** | |
| ***Задача:*** | |
|  | Рассмотрим программу обучения персептрона на языке Python. Сначала рассмотрим основной класс персептрона, который умеет учиться по тестовым данным. |
| ***Решение:*** | |
| Начнем с описания объектов класса: | |
|  |  |
| В строке 25 мы осуществляем корректировку весов. Посмотрим, как учится и работает наш персептрон. | |
|  | |
| ***Ответ:*** | |
|  |  |
| Видим, что что наш персептрон отлично научился распознавать образы, относя к классу 1 те вектора, у которых первая компонента больше второй, и к классу -1 в противном случае. Хотя устройство персептронов довольно простое эти конструкции могут решать и практические задачи. Кроме того, из таких персептронов состоят нейронные сети. | |

|  |
| --- |
| **Теоретический материал – Реализация нейронной сети на Python** |
| Нейронная сеть — это функциональная единица машинного или глубокого обучения. Она имитирует поведение человеческого мозга, поскольку основана на концепции биологических нейронных сетей.  Наиболее распространенный тип нейронной сети, называемый многослойным персептроном (MLP), представляет собой функцию, которая отображает входные данные в выходные данные. MLP имеет один входной слой и один выходной слой. Между ними может быть один или несколько скрытых слоев. Входной слой имеет тот же набор нейронов, что и признаки. Скрытые слои также могут иметь более одного нейрона. Каждый нейрон представляет собой линейную функцию, к которой применяется функция активации для решения сложных задач. Выход каждого слоя подается в качестве входных данных для всех нейронов следующих слоев.  Нейронные сети способны решать множество задач. В основном они состоят из таких компонентов:   * входной слой (получение и передача данных); * скрытый слой (вычисление); * выходной слой. Чтобы реализовать нейросеть, необходимо понимать, как ведут себя нейроны. Нейрон одновременно принимает несколько входов, обрабатывает эти данные и выдает один выход. Нейронная сеть представляет собой блоки ввода и вывода, где каждое соединение имеет соответствующие веса (это сила связи нейронов; чем вес больше, тем один нейрон сильнее влияет на другой). Данные всех входов умножаются на веса: * ; * .   Входы после взвешивания суммируются с прибавлением значения порога «c»:  Полученное значение пропускается через функцию активации (сигмоиду), которая преобразует входы в один выход:  Так выглядит сигмоида:    Интервал результатов сигмоиды — от 0 до 1. Отрицательные числа стремятся к нулю, а положительные — к единице.  Например. Пусть нейрон имеет следующие значения:  Входной слой: . |
| **1.1.2 Пример** |
| ***Решение:***  Для написания кода нейрона будем использовать библиотеку Pytnon — NumPy: |
|  |
|  |
| Нейросеть состоит из множества соединенных между собой нейронов. Пример несложной нейронной сети    где:  — входной слой;  — скрытый слой с двумя нейронами;  — выходной слой.  Например. Представим, что нейроны из графика выше имеют веса . Пороговое значение у обоих нейронов равно 0 и они имеют идентичную сигмоиду.  При входных данных получим:  Входные данные по нейронам передаются до тех пор, пока не получатся выходные значения. |
|  |
|  |

|  |
| --- |
| **Теоретический материал – Обучение нейронной сети** |
| Обучение нейросети — это подбор весов, которые соответствуют всем входам для решения поставленных задач.  Класс нейронной сети: |
|  |
| Каждый этап процесса обучения состоит из:   * прямого распространения (прогнозируемый выход); * обратного распространения (обновление весов и смещений).   Например:  Дана двуслойная нейросеть:  В данном случае на выход влияют только две переменные — (веса) и (смещение). Настройку весов и смещений из данных входа или процесс обучения нейросети можно изобразить так:    **Прямое распространение.**  Как видно, формула прямого распространения представляет собой несложное вычисление:  Далее необходимо добавить в код функцию прямого распространения. Предположим, что смещения в этом случае будут равны 0.    Чтобы вычислить ошибку прогноза, необходимо использовать функцию потери. В примере уместно воспользоваться формулой суммы квадратов ошибок — средним значением между прогнозируемым и фактическим результатами: |
| **Обратное распространение**  Обратное распространение позволяет измерить производные в обратном порядке — от конца к началу, и скорректировать веса и смещения. Для этого необходимо узнать производную функции потери — тангенс угла наклона.    Производная функции по отношению к весам и смещениям позволяет узнать градиентный спуск. Производная функции потери не содержит весов и смещений, для ее вычисления необходимо добавить правило цепи:  где .  Благодаря этому правилу можно регулировать веса. Добавляем в код Python функцию обратного распространения:    Нейронные сети базируются на определенных алгоритмах и математических функциях. Сначала может казаться, что разобраться в них довольно сложно. Но существуют готовые библиотеки машинного обучения для построения и тренировки нейросетей, позволяющие не углубляться в их устройство. |
| ***Задание:*** |
| Реализовать классы нейросетей по аналогии с классом OurNeuralNetwork. Данные нейросети:   * три входа ; * три нейрона в скрытых слоях ; * выход .   Нейроны имеют идентичные веса и пороги:       Данные нейросети:   * два входа ; * два нейрона в скрытых слоях ; * два выхода .   Нейроны имеют идентичные веса и пороги:   * ; * . |
| ***Решение:*** |

|  |  |
| --- | --- |
|  | ***Задание:*** |
|  | Реализуйте классы нейронных сетей с использованием других функций активации. |
|  |  |
| ***Решение:*** | |
|  | |

|  |
| --- |
| **1.2. Введение в нейронные сети с помощью Scikit-Learn в Python** |
| Теперь мы знаем, что такое нейронные сети и какие шаги необходимо выполнить, чтобы построить простую нейронную сеть с плотными связями. В этом разделе мы попытаемся построить простую нейронную сеть, которая предсказывает класс, к которому принадлежит данное растение ириса. Мы будем использовать библиотеку Python Scikit-Learn для создания нашей нейронной сети.  Sklearn предоставляет 2 оценщика для задач классификации и регрессии соответственно:   * MLPClassifier; * MLPRegressor   Начнем с импорта необходимых библиотек. |
|  |
| **MLPClassifier** |
| Загрузка данных  Мы будем загружать два набора данных.  Набор данных цифр: мы будем использовать набор данных цифр, который имеет изображения размером 8x8 для цифр 0-9. Ниже мы будем использовать цифровые данные для задач классификации.  Набор данных о жилье в Бостоне: мы будем использовать набор данных о жилье в Бостоне, который содержит информацию о различных свойствах дома, таких как среднее количество комнат, уровень преступности на душу населения в городе и т. д. Мы будем использовать его для задач регрессии.  Sklearn предоставляет оба этих набора данных. Мы можем загрузить их, вызвав методы load\_digits() и load\_boston(). |
|  |
|  |
|  |
|  |
| Классификация  MLPClassifier — это клвсс, доступный как часть модуля neuro\_network sklearn для выполнения задач классификации с использованием многослойного персептрона.  Как обычно разделим набор данных на две части:   * данные обучения, которые будут использоваться для модели обучения; * тестовые данные, по которым будет проверяться точность обученной модели.   Функция train\_test\_split модуля model\_selection sklearn поможет нам разделить данные на два набора: 80% для обучения и 20% для тестирования. Мы также используем seed(random\_state=123) с train\_test\_split, чтобы мы всегда получали одно и то же разделение и могли сравнивать и воспроизволить результаты в будущем. |
|  |
|  |
| Для начала натренируем модель MLPClassifier с параметрами по умолчанию для тренировочных данных. |
|  |
|  |
|  |
|  |
| Cоздадим метод plot\_confusion\_matrix(), который принимает исходные и предсказанные метки данных по модели. Затем он строит матрицу путаницы, используя matplotlib. |
|  |
|  |
| Ниже приведен список важных атрибутов, доступных с MLPClassifier, которые могут предоставить значимую информацию после обучения модели.   * loss\_ — возвращает убыток после завершения процесса обучения. * coefs\_ — возвращает массив длины n\_layers-1, где каждый элемент представляет веса, связанные с уровнем i. * intercepts\_ — возвращает массив длины n\_layers-1, где каждый элемент представляет собой перехват, связанный с персептронами слоя i. * n\_iter\_ — количество итераций, для которых выполнялась оценка. * out\_activation\_ — возвращает имя функции активации выходного слоя. |
|  |
|  |
| **MLPRegressor** |
| MLPRegressor — это класс, доступный как часть библиотеки neuro\_network sklearn для выполнения задач регрессии с использованием многослойного персептрона. Также разделим набор данных на две части:   * данные обучения (80%), которые будут использоваться для модели обучения; * тестовые данные (20%), по которым будет проверяться точность обученной модели. |
|  |
|  |
|  |
|  |
|  |
|  |
| MLPRegressor имеет все атрибуты такие же, как и у MLPClassifier: |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |  |
| --- | --- |
|  |  |
| **Задание** | |
| ***Задача:*** | |
|  | Используйте классы MLPClassified и MLPRegressor для классификации и регрессии произвольных данных из интернета. Проведите анализ атрибуты, полученных моделей.  Для классификации можете взять набор данных Ирисов:  https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21398ea43cba4f7d537619d0e07d5ae3/iris.csv  а для регрессии датасет зависимости заработной платы от опыта работы:  https://raw.githubusercontent.com/AnnaShestova/salary-years-simple-linear-regression/master/Salary\_Data.csv |
| ***Решение:*** | |