Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа № 2 по курсу "Искусственный интеллект"

Студент: А. В. Скворцов
Группа: М8О-308Б

Москва, 2019

Условие

Требуется реализовать класс на выбранном языке программирования, который реализует один из алгоритмов машинного обучения. Обязательным является наличия в классе двух методов fit, predict. Необходимо проверить работу вашего алгоритма на ваших данных (на таблице и на текстовых данных), произведя необходимую подготовку данных. Также необходимо реализовать алгоритм полиномиальной регрессии, для предсказания значений в таблице. Сравнить результаты с стандартной реализацией sklearn, определить в чем сходство и различие ваших алгоритмов. Замерить время работы алгоритмов.

Вариант: 14 % 6 + 1 = 3 (SVM)

Теоретические сведения

Метод опорных векторов — алгоритм машинного обучения, который применяется для решения задач классификации и регрессии. Основная идея метода заключается в построении гиперплоскости, разделяющей объекты выборки наиболее оптимальным способом. Можно показать, что такую плоскость можно найти, решив следующую задачу оптимизации:

$$\begin{cases} \|\vec{\omega}\| \to min \\ M_i(\vec{\omega}, b) \geq 1, \quad i = 1, \ldots, l \end{cases}$$

где ω - нормаль разделящей плоскости, $M_i(\vec{\omega}, b) = y_i((\vec{\omega}, \vec{x_i}) + b)$ - отступ. Причем вместо этой задачи можно решить двойственную ей:

$$\begin{cases} \Phi(\lambda) = \sum_{i=1}^{N} -\frac{1}{2} \sum_{i=1}^{N} \sum_{i=1}^{N} \lambda_i \lambda_j y_i y_j K(x_i, x_j) \to max \\ \sum_{i=1}^{N} \lambda_i y_i = 0 \\ 0 \leq \lambda \leq C \end{cases}$$

где $K(x_i, x_j)$ - ядро, скалярное произведение векторов в некотором пространстве более высокой размерности. Тогда итоговое решение выглядит следующим образом:

$$f(x) = sign(\sum_{i=1}^{N} \lambda_i y_i K(x_i, x) + y_s - \sum_{i=1}^{N} \lambda_i y_i K(x_i, x_s))$$

где x_s - любой опорный вектор, т.е. такой, для которого $\lambda_s \neq 0$

Метод решения

Метод опорных векторов

Как указано выше, для того чтобы найти разделяющую гиперплоскость, нужно решить задачу оптимизации квадратичного программирования. Эту задачу можно решить разными способами, специально для SVM был разработан алгоритм SMO, учитывающий его специфические особенности. В данной работе я реализовал упрощенный SMO (Simplified SMO): `http://cs229.stanford.edu/materials/smo.pdf`. В нем есть некоторые допущения, которые не гарантируют сходимость в определенных случаях, но реализовать его гораздо проще.

Проведем тестирование алгоритма и сравним его со стандартной реализацией sklearn.

Сначала проведем тестирование на датасете проданных домов и попытаемся определить, является ли дом элитным или нет. Как и в первой л.р. будем считать дом элитным, если его стоимость превышает 700000$.

```
In [63]: runfile('D:/myprog/ai/lab2/test_algos.py', wdir='D:/myprog/ai/lab2')
Reloaded modules: MLalgos
sklearn SVM:
              precision    recall  f1-score   support

          -1       0.93      0.97      0.95      5143
           1       0.87      0.73      0.80      1341

   micro avg       0.92      0.92      0.92      6484
   macro avg       0.90      0.85      0.87      6484
weighted avg       0.92      0.92      0.92      6484

[[5002  141]
 [ 359  982]]
Time:  3.281806534

custom SVM:
              precision    recall  f1-score   support

          -1       0.89      0.98      0.93      5143
           1       0.89      0.51      0.65      1341

   micro avg       0.89      0.89      0.89      6484
   macro avg       0.89      0.75      0.79      6484
weighted avg       0.89      0.89      0.87      6484

[[5056   87]
 [ 651  690]]
Time:  193.386777598
```

Как видно, точность моей реализации похуже, почти половину элитных домов он не распознает, а скорость работы в 60 раз хуже. Тем не менее результат неплохой, алгоритм неправильно определил класс только у 12% объектов.

Для проведения тестирования на корпусе документов, я выбрал другой датасет для определения спама: `https://www.kaggle.com/ozlerhakan/spam-or-not-spam-dataset`. Такое решение связано, во-первых, с медленной работой программы (а этот датасет весит гораздо меньше), а во-вторых с тем, что точность классификации на старом датасете в предыдущей работе была немногим выше 50%.

Обработку данных я провел следующим образом. Выбрал 100 случайных писем и с помощью CountVectorizer из sklearn определил их токены. Будем считать эти токены репрезентативными и проведем обучение модели на них. Результаты:

```
In [64]: runfile('D:/myprog/ai/lab2/test_algos.py', wdir='D:/myprog/ai/lab2')
Reloaded modules: MLalgos
sklearn SVM:
              precision    recall  f1-score   support

          -1       0.93      1.00      0.96       629
           1       0.97      0.64      0.77       121

   micro avg       0.94      0.94      0.94       750
   macro avg       0.95      0.82      0.87       750
weighted avg       0.94      0.94      0.93       750

[[627   2]
 [ 44  77]]
Time:  7.237673483

custom SVM:
              precision    recall  f1-score   support

          -1       0.99      0.89      0.94       629
           1       0.63      0.97      0.76       121

   micro avg       0.90      0.90      0.90       750
   macro avg       0.81      0.93      0.85       750
weighted avg       0.93      0.90      0.91       750

[[560  69]
 [  4 117]]
Time:  41.401005138
```

На удивление алгоритм сошелся достаточно быстро, а точность оказалась немногим хуже. Можно заметить, что стандартная реализация склонна определять не спам письма, как спам, а моя наоборот.

Полиномиальная регрессия

В отличие от линейной регрессии, в полиномиальной могут встречаться комбинации нескольких независимых переменных, степень которых может быть выше первой. Все такие комбинации я нашел с помощью PolynomialFeatures из sklearn, а коэффициенты нашел методом наименьших квадратов.

Ниже представлены результаты тестирования. Здесь я пытаюсь предсказать цену на дом в зависимости от его характеристик. Похоже, что связь между переменными нелинейная, при полиномиальной регрессии второй степени как абсолютная, так и среднеквадратическая ошибка наименьшие.

```
In [67]: runfile('D:/myprog/ai/lab2/test_algos.py', wdir='D:/myprog/ai/lab2')
Polynomial Regression of 1 order
Mean Absolute Error:  125211.89358357605
Root Mean Squared Error:  197935.63939154643
Time:  0.011403817

Polynomial Regression of 2 order
Mean Absolute Error:  104699.98478533959
Root Mean Squared Error:  163763.75556981342
Time:  0.149013732

Polynomial Regression of 3 order
Mean Absolute Error:  103503.09247021786
Root Mean Squared Error:  169258.5891334728
Time:  2.864534445
```

Выводы

В данной лабораторной работе я узнал, как алгоритмы машинного обучения работают изнутри, в частности, я узнал много нового об SVM. Так в отличие от логистической регрессии этот алгоритм может работать даже в случае линейно неразделимой выборки, однако чувствителен к шумам и выбросам. Для его корректной работы нужно проводить нормализацию данных.

Кроме того, я также научился работать с библиотекой sklearn.