Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и информатики
Кафедра вычислительной математики и программирования

Лабораторная работа № 1,2

по курсу «Машинное обучение»

Преподаватель: Ахмед Самир Халид
Студент: Чурсина Н.А.
Группа: 80-308Б
Оценка:

Москва, 2021

Задание

1 ЛР:

Найти себе набор данных (датасет), для следующей лабораторной работы, и проанализировать его. Выявить проблемы набора данных, устранить их. Визуализировать зависимости, показать распределения некоторых признаков. Реализовать алгоритмы К ближайших соседа с использованием весов и Наивный Байесовский классификатор и сравнить с реализацией библиотеки sklearn.

2 ЛР:

Необходимо реализовать алгоритмы машинного обучения. Применить данные алгоритмы на наборы данных, подготовленных в первой лабораторной работе. Провести анализ полученных моделей, вычислить метрики классификатора. Произвести тюнинг параметров в случае необходимости. Сравнить полученные результаты с моделями реализованными в scikit-learn. Аналогично построить метрики классификации. Показать, что полученные модели не переобучились. Также необходимо сделать выводы о применимости данных моделей к вашей задаче.

Вариант: Логистическая регрессия

Описание алгоритмов

1. Алгоритм n-ближайших соседей (KNN)

Пусть имеется набор данных, состоящий из k наблюдений $Xi(i=1,...,k)$, для каждого из которых задан класс $Cj(j=1,...,m)$. Тогда на его основе может быть сформировано обучающее множество из пар Xi,Cj.

Алгоритм KNN можно разделить на два этапа:

- Обучение: алгоритм запоминает векторы признаков наблюдений и их метки классов. Также задаётся параметр алгоритма n, который задаёт число «соседей», которые будут использоваться при классификации.

- Классификация: предъявляется новый объект, для которого метка класса не задана. Для него определяются n ближайших (в смысле некоторой метрики) предварительно классифицированных наблюдений. Затем выбирается класс, которому принадлежит большинство из n ближайших примеров-соседей, и к этому же классу относится классифицируемый объект.

В моей реализации используется стандартный алгоритм с использованием Евклидовой нормы.

2. Гауссовский Наивный Байесовский классификатор

Основная идея — построить классификатор в предположении того, что функция *p(Xi, Cj)* известна для каждого класса и равна плотности многомерного нормального (гауссовского) распределения:

$$p(x_i \mid c_j) = \frac{1}{\sqrt{2\pi\sigma_{i,j}^2}} e^{-\frac{1}{2}\left(\frac{x_i - \mu_{i,j}}{\sigma_{i,j}}\right)^2} \text{ for } i = 1, 2 \text{ and } j = 1, 2, 3$$

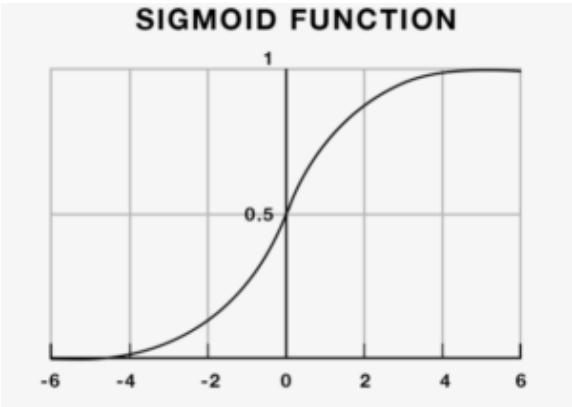
где $\mu_{i,j}$ - среднее значение, а $\sigma_{i,j}$ - стандартное отклонение, которое мы должны оценить по данным. Это означает, что мы получаем одно среднее значение для каждого признака i в паре с классом c_j.

3. Логистическая регрессия

Целью линейной регрессии является оценка значений для коэффициентов модели c, w1, w2, w3 ... .wn и подгонки обучающих данных с минимальной квадратичной ошибкой и прогнозирования вывода y.

Модель логистической регрессии вычисляет взвешенную сумму входных переменных, аналогичную линейной регрессии, но она вычисляет результат с помощью специальной нелинейной функции, логистической или сигмоидной функции для получения значения y.

Сигмоидальная/логистическая функция задается уравнением: y = 1 / 1+ e^{-x}

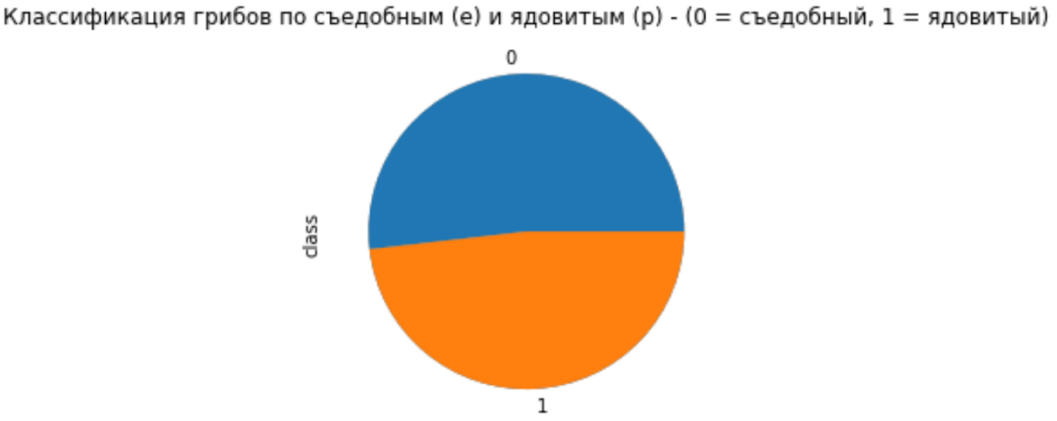


Таким образом, если результат больше 0,5, мы можем классифицировать результат как 1 (или положительный), а если он меньше 0,5, мы можем классифицировать его как 0 (или отрицательный).

Используемые метрики

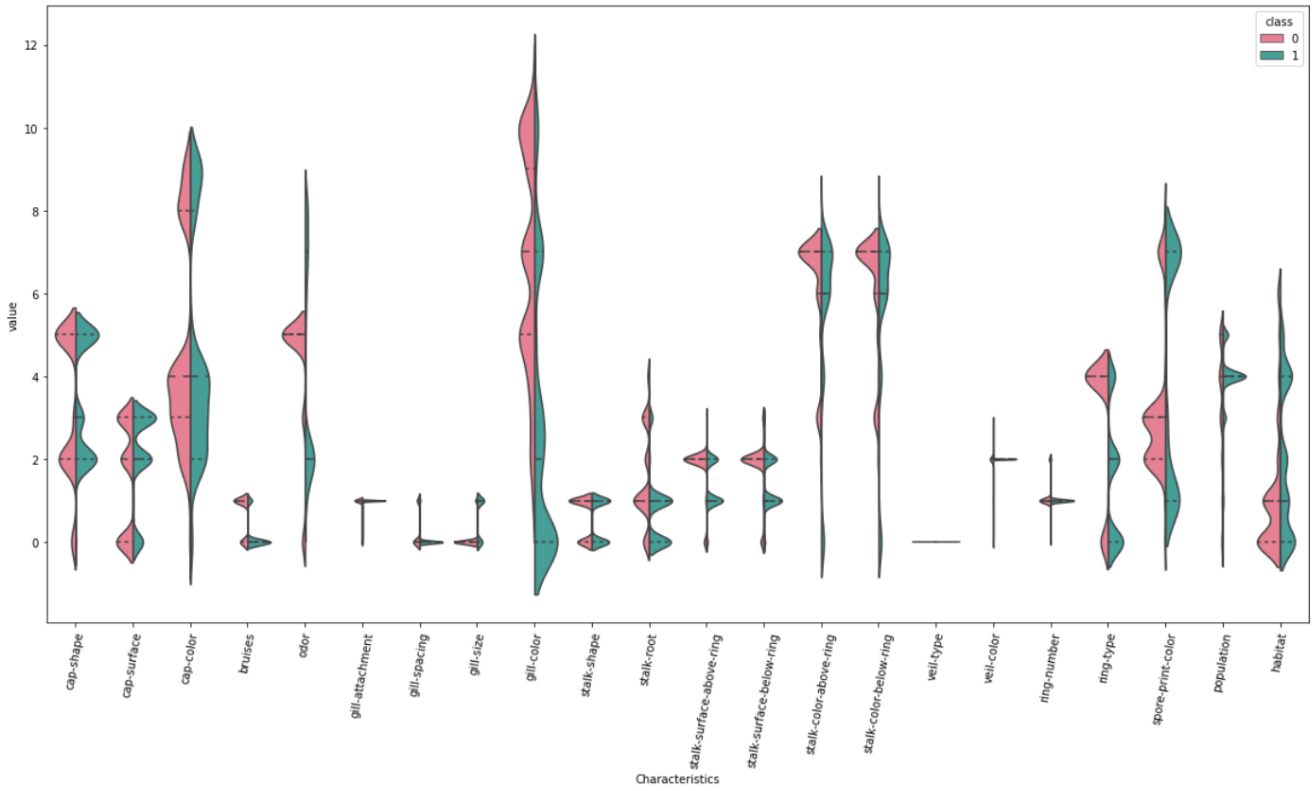
Для оценки качества классификации использованы метрика accuracy, так как в обеих задачах отсутсвует дисбаланс. Валидирование проводилось посредством кросс-валидации.

Мною был выбран mushroom датасет. Данные были проверены на баланс по классу:



Из построенной зависимости видно, что класс не нуждается в балансировке.

Распределение по признакам:



Результаты

Алгоритмы KNN, Байесовского классификатора и Логистической регрессии оказались достаточно медленнее в собственной реализации по сравнению с алгоритмами библиотеки scikit-learn, поскольку в моих реализациях нет оптимизации.

```
%%time
model = LogisticRegression()
scores = cv(model, X, y)
print("Accuracy: ", scores.mean())
```

```
Accuracy:  0.9497783251231526
CPU times: user 1.04 s, sys: 97.2 ms, total: 1.13 s
Wall time: 618 ms
```

```
%%time
model = algo.LR()
scores = cv(model, X, y)
print("Accuracy: ", scores.mean())
```

```
Accuracy:  0.9164206138688897
CPU times: user 8.81 s, sys: 707 ms, total: 9.51 s
Wall time: 5.08 s
```

```
%%time
model = KNeighborsClassifier(n_neighbors=5)
scores = cv(model, X, y)
print("Accuracy: ", scores.mean())
```

```
Accuracy:  0.998645850701023
CPU times: user 2.06 s, sys: 281 ms, total: 2.35 s
Wall time: 1.47 s
```

```
%%time
model = algo.KNN(nn=5)
scores = cv(model, X, y)
print("Accuracy: ", scores.mean())
```

```
Accuracy:  0.9986459264873059
CPU times: user 4.89 s, sys: 178 ms, total: 5.07 s
Wall time: 5.1 s
```

```
%%time
model = GaussianNB()
scores = cv(model, X, y)
print("Accuracy: ", scores.mean())
```

```
Accuracy:  0.9217145888594166
CPU times: user 45.7 ms, sys: 2.59 ms, total: 48.3 ms
Wall time: 46.9 ms
```

```
%%time
model = algo.GaussianNaiveBayesClassifier()
scores = cv(model, X, y)
print("Accuracy: ", scores.mean())
```

```
Accuracy:  0.5179697612732096
CPU times: user 466 ms, sys: 11.3 ms, total: 478 ms
Wall time: 469 ms
```

При выбранных параметрах алгоритмов видно, что модель не переобучалась, поскольку разница в точности классификации на обучающей и тестовой выборках в основном достаточно мала.

Выводы

При “ручной” реализации алгоритмов машинного обучения нужно использовать максимально эффективные алгоритмы, чтобы достичь таких же высоких результатов, какие предоставляют нам библиотечные реализации.