Московский авиационный институт
(национальный исследовательский университет)
Кафедра 806
“Вычислительная математика и программирование”

Лабораторная работа №1-2
по Машинному Обучению
Вариант №2

Группа: М8О-308Б-18

Студент: Синявский А.В.

Преподаватель: Ахмед Самир Халид

Москва, 2021

Задание

Лаба 1:

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 K ближайших соседа с использованием весов и Наивный Байесовский классификатор и сравнить с реализацией библиотеки sklearn.

Лаба 2:

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 2

1) Логистическая регрессия
2) Дерево решений
3) Random Forest

Алгоритмы

1. KNN

Один из простейших алгоритмов классификации. Суть алгоритма заключается в определении класса объекта по классам K уже классифицированных объектов, ближайших к новому в смысле какой-либо метрики. Выбранная мною метрика – Евклидова норма

2. Наивный Байесовский классификатор

В основе лежит формула условной вероятности Байеса. Наивным алгоритм делает предположение о независимости признаков. Данное предположение позволяет рассчитать функции правдоподобия классов

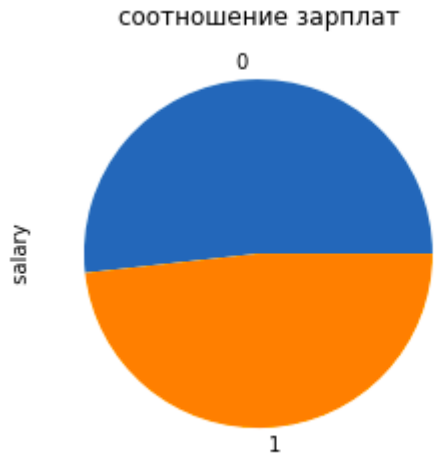
3. Логистическая регрессия

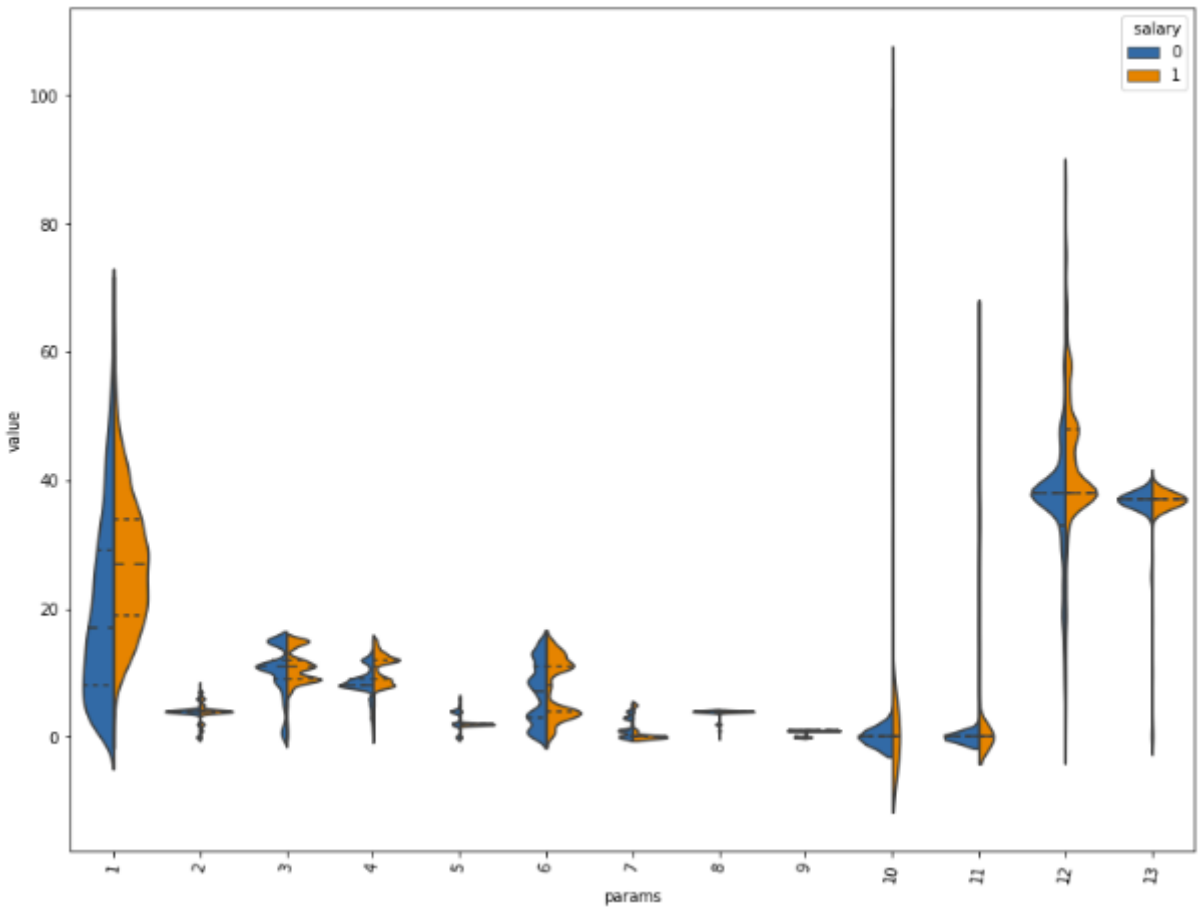
Данная модель, аналогично линейной регрессии, рассчитывает взвешенную сумму, однако определяет принадлежность объекта классу по значению сигмоидной функции:

$$y = 1 / 1 + e^{-x}$$

Данные

Используемый мною датасет – http://archive.ics.uci.edu/ml/datasets/Adult. В нём содержится информация об объёме заработной платы различых людей. Будем разделять людей на 2 группы: ЗП <=50K$, >50K$





Для оценки качества классификации рассчитывается точность предсказаний

Результаты

In [25]:
```
%%time
model = KNeighborsClassifier(n_neighbors=9)
metrics = cv(model, X, y)
print("Accuracy: ", metrics[0].mean())
print("Precision: ", metrics[1].mean())
print("Recall: ", metrics[2].mean())
```

```
Accuracy:  0.6565318456531846
Precision:  0.6462574020377003
Recall:  0.6508883194658993
CPU times: user 789 ms, sys: 7.3 ms, total: 796 ms
Wall time: 797 ms
```

In [26]:
```
%%time
model = bmlf.KNN(9)
metrics = cv(model, X, y)
print("Accuracy: ", metrics[0].mean())
print("Precision: ", metrics[1].mean())
print("Recall: ", metrics[2].mean())
```

```
Accuracy:  0.6565318456531846
Precision:  0.6461896177855908
Recall:  0.6510828673922646
CPU times: user 13.8 s, sys: 4.79 s, total: 18.6 s
Wall time: 18.6 s
```

In [27]:
```
%%time
model = GaussianNB()
metrics = cv(model, X, y)
print("Accuracy: ", metrics[0].mean())
print("Precision: ", metrics[1].mean())
print("Recall: ", metrics[2].mean())
```

```
Accuracy:  0.7531380753138075
Precision:  0.8136593592651288
Recall:  0.6394830242498057
CPU times: user 178 ms, sys: 0 ns, total: 178 ms
Wall time: 187 ms
```

In [28]:
```
%%time
model = bmlf.NBC()
metrics = cv(model, X, y)
print("Accuracy: ", metrics[0].mean())
print("Precision: ", metrics[1].mean())
print("Recall: ", metrics[2].mean())
```

```
Accuracy:  0.7548117154811715
Precision:  0.8130970751709785
Recall:  0.6446383254489493
CPU times: user 712 ms, sys: 9.18 ms, total: 722 ms
Wall time: 721 ms
```

In [21]:
```
%%time
model = LogisticRegression()
metrics = cv(model, X, y)
print("Accuracy: ", metrics[0].mean())
print("Precision: ", metrics[1].mean())
print("Recall: ", metrics[2].mean())
```

```
Accuracy:  0.7247791724779172
Precision:  0.728693554296994
Recall:  0.6918742306685761
CPU times: user 4.71 s, sys: 3.29 s, total: 8 s
Wall time: 2.6 s
```

In [7]:
```
%%time
model = bmlf.LR()
metrics = cv(model, X, y)
print("Accuracy: ", metrics[0].mean())
print("Precision: ", metrics[1].mean())
print("Recall: ", metrics[2].mean())
```

```
Accuracy:  0.5105532310553231
Precision:  0.586915673225394
Recall:  0.42377352372620686
CPU times: user 1min 13s, sys: 48.8 s, total: 2min 2s
Wall time: 33.9 s
```

In [29]:
```
%%time
model = DecisionTreeClassifier()
metrics = cv(model, X, y)
print("Accuracy: ", metrics[0].mean())
print("Precision: ", metrics[1].mean())
print("Recall: ", metrics[2].mean())
```

```
Accuracy:  0.7568572756857276
Precision:  0.7527622709086285
Recall:  0.7449750027700051
CPU times: user 350 ms, sys: 6.75 ms, total: 357 ms
Wall time: 367 ms
```

In [32]:
```
%%time
model = bmlf.Node()
metrics = cv(model, X, y)
print("Accuracy: ", metrics[0].mean())
print("Precision: ", metrics[1].mean())
print("Recall: ", metrics[2].mean())
```

```
Accuracy:  0.7525801952580194
Precision:  0.7522067842546241
Recall:  0.7373880536085041
CPU times: user 8min 43s, sys: 611 ms, total: 8min 44s
Wall time: 8min 44s
```

In [30]:
```
%%time
model = RandomForestClassifier()
metrics = cv(model, X, y)
print("Accuracy: ", metrics[0].mean())
print("Precision: ", metrics[1].mean())
print("Recall: ", metrics[2].mean())
```

```
Accuracy:  0.8128312412831242
Precision:  0.7998326088733402
Recall:  0.8211597815524195
CPU times: user 6.03 s, sys: 7.31 ms, total: 6.04 s
Wall time: 6.05 s
```

In [31]:
```
%%time
model = bmlf.RF(num_trees=100)
metrics = cv(model, X, y)
print("Accuracy: ", metrics[0].mean())
print("Precision: ", metrics[1].mean())
print("Recall: ", metrics[2].mean())
```

```
Accuracy:  0.802138540213854
Precision:  0.7589840395828503
Recall:  0.8692796566673093
CPU times: user 10.9 s, sys: 0 ns, total: 10.9 s
Wall time: 10.9 s
```

Выводы

По результатам работы видно, что при корректной реализации точность классификации не сильно отличается от моделей из sklearn. Основной проблемой является оптимизация. Разница во времени выполнения у большинства моделей отличается в 10 и более раз. Алгоритм KNN показал себя наименее эффективным из рассмотренных на выбранном мною датасете. Я полагаю, это связано с тем, что в алгоритме учитывается значимость различных атрибутов.