Московский авиационный институт

(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Лабораторная работа №1 по искусственному интеллекту

6 семестр

Студент: Кареткин Д.В.

Группа: М8О-301б

Дата:

Москва, 2022

Оглавление

Постановка задачи

1) реализовать следующие алгоритмы машинного обучения: Linear/ Logistic Regression, SVM, KNN, Naive Bayes в отдельных классах
2) Данные классы должны наследоваться от BaseEstimator и ClassifierMixin, иметь методы fit и predict (подробнее: https://scikit-learn.org/stable/developers/develop.html)
3) Вы должны организовать весь процесс предобработки, обучения и тестирования с помощью Pipeline (подробнее: https://scikit-learn.org/stable/modules/compose.html)
4) Вы должны настроить гиперпараметры моделей с помощью кросс валидации (GridSearchCV,RandomSearchCV, подробнее здесь: https://scikit-learn.org/stable/modules/grid_search.html), вывести и сохранить эти гиперпараметры в файл, вместе с обученными моделями
5) Проделать аналогично с коробочными решениями
6) Для каждой модели получить оценки метрик:Confusion Matrix, Accuracy, Recall, Precision, ROC_AUC curve (подробнее: Hands on machine learning with python and scikit learn chapter 3, mlcourse.ai, https://ml-handbook.ru/chapters/model_evaluation/intro)
7) Проанализировать полученные результаты и сделать выводы о применимости моделей
8) Загрузить полученные гиперпараметры модели и обученные модели в формате pickle на гит вместе с jupyter notebook ваших экспериментов

Логистическая регрессия

Алгоритм

В отличие от обычной регрессии, в методе логистической

регрессии не производится предсказание значения числовой переменной исходя из выборки исходных значений. Вместо этого, значением функции является вероятность того, что данное

исходное значение принадлежит к определенному классу. Для простоты, давайте предположим, что у нас есть только два класса и вероятность, которую мы будем определять, P_+ вероятности того, что некоторое значение принадлежит классу "+". И конечно $P_- = 1 - P_+$. Таким образом, результат логистической регрессии всегда находится в интервале [0, 1].

Основная идея логистической регрессии заключается в том, что пространство исходных значений может быть разделено линейной границей (т.е. прямой) на две соответствующих классам области. Итак, что же имеется ввиду под линейной границей? В случае двух измерений — это просто прямая линия без изгибов. В случае

трех — плоскость, и так далее. Эта граница задается в зависимости от имеющихся исходных данных и обучающего алгоритма. Чтобы все работало, точки исходных данных должны разделяться линейной границей на две вышеупомянутых области. Если точки исходных данных удовлетворяют этому требованию, то их можно назвать линейно разделяемыми.

Обучение и метрики

```
metrics for custom_logreg for train data


confusion_matrix =
 [[852 331]
 [ 40 577]]


accuracy_score =  0.7938888888888889
recall_score =  0.9351701782820098
precision_score= 0.6354625550660793


metrics for custom_logreg for test data


confusion_matrix =
 [[104  32]
 [  4  60]]


accuracy_score =  0.82
recall_score =  0.9375
precision_score= 0.6521739130434783
```

```
metrics for sklearn_logreg for train data


confusion_matrix =
 [[817  55]
 [ 75 853]]


accuracy_score =  0.9277777777777778
recall_score =  0.9191810344827587
precision_score= 0.9394273127753304


metrics for sklearn_logreg for test data


confusion_matrix =
 [[100   8]
 [  8  84]]


accuracy_score =  0.92
recall_score =  0.9130434782608695
precision_score= 0.9130434782608695
```

Выводы о моделях по метрикам

- Переобучение не наблюдается у моей модели и модели из sklearn
- Разница метрик между моей моделью и моделью из sklearn присутствует, но некритична.

Метод опорных векторов

Алгоритм

Алгоритм Главная цель SVM как классификатора — найти уравнение разделяющей гиперплоскости в пространстве, которая бы разделила два класса неким оптимальным образом. После настройки весов алгоритма (обучения), все объекты, попадающие по одну сторону от построенной гиперплоскости, будут предсказываться как первый класс, а объекты, попадающие по другую сторону — второй класс.

Обучение и метрики

```
metrics for custom_svm for train data


confusion_matrix =
 [[892 908]
 [  0   0]]


accuracy_score =  0.4955555555555556
recall_score =  0.0
precision_score= 0.0


metrics for custom_svm for test data


confusion_matrix =
 [[108  92]
 [  0   0]]


accuracy_score =  0.54
recall_score =  0.0
precision_score= 0.0
```

```
metrics for sklearn_svm for train data


confusion_matrix =
 [[504   0]
 [388 908]]


accuracy_score =  0.7844444444444445
recall_score =  0.7006172839506173
precision_score= 1.0


metrics for sklearn_svm for test data


confusion_matrix =
 [[  1   0]
 [107  92]]


accuracy_score =  0.465
recall_score =  0.4623115577889447
precision_score= 1.0
```

Выводы

- Моя модель не переобучилась, т к разница на метриках между трейном и тестом минимальна.
- Моделт из sklearn переобучилась сильнее.
- Моя модель показывает себя хуже по метрикам на трейне чем модель из sklearn на трейне, но при этом моя модель показывает себя лучше чем модель из sklearn на тесте.

Алгоритма k ближайших соседей с весами

Алгоритм

Для классификации каждого из объектов тестовой выборки необходимо последовательно выполнить следующие операции:

1) Вычислить расстояние до каждого из объектов обучающей выборки и посчитать веса для каждого объекта обучающей выборки
2) Отобрать k объектов обучающей выборки, расстояние до которых минимально
3) Самый частый класс является результатом работы алгоритма

Обучение и метрики

```
metrics for custom_KNN for train data


confusion_matrix =
 [[887  32]
 [  5 876]]


accuracy_score =  0.9794444444444445
recall_score =  0.9943246311010215
precision_score= 0.9647577092511013

metrics for custom_KNN for test data


confusion_matrix =
 [[108   1]
 [  0  91]]


accuracy_score =  0.995
recall_score =  1.0
precision_score= 0.9891304347826086
```

```
metrics for sklearn_KNN for train data


confusion_matrix =
 [[887  32]
 [  5 876]]


accuracy_score =  0.9794444444444445
recall_score =  0.9943246311010215
precision_score= 0.9647577092511013

metrics for sklearn_KNN for test data


confusion_matrix =
 [[108   1]
 [  0  91]]


accuracy_score =  0.995
recall_score =  1.0
precision_score= 0.9891304347826086
```

Выводы по метрикам

- Метрики на трейне у моей модели и модели из sklearn примерно одинаковые
- Модели почти не переобучились
- Модель показала себя лучше всех остальных

Наивный байесовский классификатор

Алгоритм

1. Преобразуем набор данных в частотную таблицу (frequency table).
2. Создадим таблицу правдоподобия (likelihood table), рассчитав соответствующие вероятности.
3. С помощью теоремы Байеса рассчитаем апостериорную вероятность для каждого класса
4. Класс с наибольшей апостериорной вероятностью будет результатом прогноза.

Обучение и метрики

```
metrics for custom_NaivBaisClassificator for train data


confusion_matrix =
 [[830  58]
 [ 62 850]]


accuracy_score =  0.9333333333333333
recall_score =  0.9320175438596491
precision_score= 0.9361233480176211


metrics for custom_NaivBaisClassificator for test data


confusion_matrix =
 [[103   1]
 [  5  91]]


accuracy_score =  0.97
recall_score =  0.9479166666666666
precision_score= 0.9891304347826086
```

```
metrics for sklearn_NaivBaisClassificator for train data


confusion_matrix =
 [[831  58]
 [ 61 850]]


accuracy_score =  0.9338888888888889
recall_score =  0.9330406147091108
precision_score= 0.9361233480176211


metrics for sklearn_NaivBaisClassificator for test data


confusion_matrix =
 [[103   1]
 [  5  91]]


accuracy_score =  0.97
recall_score =  0.9479166666666666
precision_score= 0.9891304347826086
```

Выводы по метрикам

- Метрики на трейне и тесте моделей отличаются несильно.