Лабораторная работа №7

Наивные байесовские классификаторы

Наивные байесовские классификаторы представляют собой семейство классификаторов, которые очень схожи с линейными моделями, рассмотренными в предыдущем разделе. Однако они имеют тенденцию обучаться быстрее. Цена, которую приходится платить за такую эффективность – немного более низкая обобщающая способность моделей Байеса по сравнению с линейными классификаторами типа **LogisticRegression** и **LinearSVC**.

Причина, по которой наивные байесовские модели столь эффективны, заключается в том, что они оценивают параметры, рассматривая каждый признак отдельно и по каждому признаку собирают простые статистики классов. В scikit-learn реализованы три вида наивных байесовских классификаторов: **GaussianNB**, **BernoulliNB** и **MultinomialNB**. **GaussianNB** можно применить к любым непрерывным данным, в то время как **BernoulliNB** принимает бинарные данные, **MultinomialNB** принимает счетные или дискретные данные (то есть каждый признак представляет собой подсчет целочисленных значений какой-то характеристики, например, речь может идти о частоте встречаемости слова в предложении). **BernoulliNB** и **MultinomialNB** в основном используются для классификации текстовых данных.

Классификатор **BernoulliNB** подсчитывает ненулевые частоты признаков по каждому классу. Это легче всего понять на примере:

```
X = np.array([[0, 1, 0, 1], [1, 0, 1, 1], [0, 0, 0, 1], [1, 0, 1, 0]])
y = np.array([0, 1, 0, 1])
```

Здесь у нас есть четыре точки данных с четырьмя бинарными признаками. Есть два класса **0** и **1**. Для класса 0 (первая и третья точки данных) первый признак равен нулю два раза и отличен от нуля ноль раз, второй признак равен нулю один раз и отличен от нуля один раз и так далее. Те же самые частоты затем подсчитываются для точек данных во втором классе. Подсчет ненулевых элементов в каждом классе по сути выглядит следующим образом:

```
counts = {}
for label in np.unique(y):
    counts[label] = X[y == label].sum(axis=0)
print("Частоты признаков:\n{}".format(counts))
```

```
Частоты признаков:
{0: array([0, 1, 0, 2]), 1: array([2, 0, 2, 1])}
```

Преимущества, недостатки и параметры

Две другие наивные байесовские модели **MultinomialNB** и **GaussianNB**, немного отличаются с точки зрения вычисляемых статистик. **MultinomialNB** принимает в расчет среднее значение каждого признака для каждого класса, в то время как **GaussianNB** записывает среднее значение, а также стандартное отклонение каждого признака для каждого класса.

Для получения прогноза точка данных сравнивается со статистиками для каждого класса и прогнозируется наиболее подходящий класс. Интересно отметить, что для **MultinomialNB** и **BernoulliNB** это приводит прогнозной формуле, которая имеет точно такой же вид, что и формула для линейных моделей (см. «Линейные модели классификации»). К сожалению, **coef_** для наивных байесовских моделей имеет несколько иной смысл, чем **coef_** для линейных моделей, здесь **coef_** не тождественен **w**.

MultinomialNB и **BernoulliNB** имеют один параметр **alpha**, который контролирует сложность модели. Параметр **alpha** работает следующим образом: алгоритм добавляет к данным зависящее от **alpha** определенное количество искусственных наблюдений с положительными значениями для всех признаков. Это приводит к «сглаживанию» статистик. Большее значение **alpha** означает более высокую степень сглаживания, что приводит к построению менее сложных моделей. Алгоритм относительно устойчив к разным значениям **alpha**. Это означает, что значение **alpha** не оказывает значительного влияния на хорошую работу модели. Вместе с тем тонкая настройка этого параметра обычно немного увеличивает правильность.

GaussianNB в основном используется для данных с очень высокой размерностью, тогда как остальные наивные байесовские модели широко используются для разреженных дискретных данных, например, для текста. **MultinomialNB** обычно работает лучше, чем **BernoulliNB**, особенно на наборах данных с относительно большим количеством признаков, имеющих ненулевые частоты (т.е. на больших документах).

Наивные байесовские модели разделяют многие преимущества и недостатки линейных моделей. Они очень быстро обучаются и прогнозируют, а процесс обучения легко интерпретировать. Модели очень хорошо работают с высокоразмерными разреженными данными и относительно устойчивы к изменениям параметров. Наивные байесовские модели являются замечательными базовыми моделями и часто используются на очень больших наборах данных, где обучение даже линейной модели может занять слишком много времени.

Код к лабораторной работе:

```
import mglearn
import sklearn
import matplotlib.pyplot as plt
import numpy as np
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB

# Классификатор BernoulliNB

X = np.array([[0, 1, 0, 1], [1, 0, 1, 1], [0, 0, 0, 1], [1, 0, 1, 0]])
y = np.array([0, 1, 0, 1])

counts = {}
for label in np.unique(y):
    counts[label] = X[y == label].sum(axis=0)
print("Частоты признаков:\n{}".format(counts))

clf = BernoulliNB()
clf.fit(X, Y)
print("clf.predict:\n{}".clf.predict(X[2:3]))


# Классификатор MultinomialNB

rng = np.random.RandomState(1)
X = rng.randint(5, size=(6, 100))
y = np.array([1, 2, 3, 4, 5, 6])

clf = MultinomialNB()
clf.fit(X, y)

print(clf.predict(X[2:3]))

# Классификатор GaussianNB

X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
Y = np.array([1, 1, 1, 2, 2, 2])
clf = GaussianNB()
clf.fit(X, Y)

print(clf.predict([[-0.8, -1]]))

clf_pf = GaussianNB()
clf_pf.partial_fit(X, Y, np.unique(Y))

print(clf_pf.predict([[-0.8, -1]]))
```