МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Лабораторная работа №10
Наивный Байесовский классификатор

Дисциплина «Теория вероятностей и математическая статистика»

Студент группы Б9123-01.03.02ии
Моттуева Уруйдана Михайловна

г. Владивосток
2025

Целью данной лабораторной работы является построение модели наивного Байесовского классификатора в виде реализации класса NaiveBias.

Постановка задачи классификации

Рассматривается генеральная совокупность объектов с признаками $X = (X^1, X^2, \dots, X_\mathrm{m})$. Каждому объекту присвоен один из нескольких классов Y.

Требуется на основе выборки определить, к какому классу относится объект, отсутствующий в выборке.

Наивный Байесовский классификатор

Идея

Если новый объект имеет значения признаков $x = (x_1, x_2, \dots, x_m)$, то можно вычислить условную вероятность принадлежности к классу y_k:

$$p_{Y|X}(y_k|x) = \frac{p_Y(y_k) \cdot p_{X|Y}(x|y_k)}{p_X(x)},$$

Где:

1. $p_{Y|X}\,(\,y_k \mid x\,)$ – это условная вероятность того, что событие $Y = y_k$ произойдет при условии, что событие $X = x$ уже произошло.
2. $p_Y(y_k)$ – это априорная вероятность события $Y = y_k$. Показывает, какова вероятность события $Y = y_k$ без учета информации о X.
3. $p_{X|Y}(\,x \mid y_k\,)$ – это условная вероятность события $X = x$ при условии, что событие $Y = y_k$ произошло. Показывает, насколько вероятно наблюдение $X = x$, если известно, что $Y = y_k$.
4. $p_X(x)$ – вероятность события $X = x$. Представляет собой общую вероятность события $X = x$, независимо от значения Y.

Проблема

Неизвестны ни частные, ни совместные, ни условные функции вероятности.

Оценка функций вероятности

Для функции вероятности $p_{X_i}(x)$ можно использовать несмещённую состоятельную оценку $\hat{p}_{X_i}(x)$ — относительная частота значения x у признака X_i.

Аналогично для $p_Y(y)$ оценка $\hat{p}_Y(y)$— относительная частота класса y.

Для условной функции вероятности $p_{X_i|Y}(x|y)$ оценка $\hat{p}_{X_i|Y}(x|y)$ — относительная частота значения x у признака X_i среди объектов класса y.

Наивное предположение

Для вычисления совместной функции вероятности предположим, что признаки независимы в совокупности. Тогда

$$p_{X|Y}(\boldsymbol{x}|y) = \prod_{i=1}^{m} p_{X_i|Y}(x_i|y).$$

Принятие решения о классе

Одно из возможных правил выбора класса — выбрать класс с наибольшей вероятностью.

Тогда для объекта со значениями признаков x будет выбран класс

$$y = \underset{y_k}{argmax} \frac{\hat{p}_Y(y_k) \cdot \prod_{i=1}^{m} p_{X_i|Y}(x_i|y_k)}{p_{\boldsymbol{X}}(\mathrm{x})}.$$

Можно заметить, что $p_{\boldsymbol{X}}(x)$ не зависит от y_k, а вместо вероятностей можно брать их логарифм. Следовательно, итоговый вид модели:

$$y = \underset{y_k}{argmax} \left(\log \hat{p}_Y(y_k) + \sum_{i=1}^{m} \log \hat{p}_{X_i|Y}(x_i|y_k) \right).$$

Замечания о модели

1. Для непрерывных признаков следует использовать плотность нормального распределения:

$$\hat{f}_{X_i|Y}(x|y_k) = \frac{1}{\sqrt{2\pi}\sigma_{ik}} e^{-\frac{(x-\mu_{ik})^2}{2\sigma_{ik}^2}},$$

где:

- μ_{ik} — среднее арифметическое,

- σ_{ik} — выборочное среднеквадратическое отклонение значений признака X_i у объектов класса y_k.

2. Чтобы относительная частота значения не равнялась 0, следует при её вычислении прибавить 1 к числителю и знаменателю.

Реализация

1. Конструктор принимает флаги, отвечающие за то, какие признаки являются дискретными, а какие непрерывными;

```
class NaiveBias:
    def __init__(self, discrete_flags):
        self.discrete_flags = discrete_flags
        self.classes_ = None
        self.class_probs_ = None
        self.feature_params_ = None
```

2. Метод fit принимает массив объектов X и вектор классов y из тренировочной выборки. Вычисляет и сохраняет необходимые значения функций вероятности и плотностей;

```
def fit(self, X, y):
    self.classes_ = np.unique(y)
    n_classes = len(self.classes_)
    n_features = X.shape[1]

    self.class_probs_ = np.zeros(n_classes)
    for i, c in enumerate(self.classes_):
        self.class_probs_[i] = np.sum(y == c) / len(y)

    self.feature_params_ = []

    for feature_idx in range(n_features):
        if self.discrete_flags[feature_idx]:
            # Для дискретных признаков
            unique_vals = np.unique(X[:, feature_idx])
            prob_table = np.zeros((n_classes, len(unique_vals)))

            for i, c in enumerate(self.classes_):
                class_mask = (y == c)
                feature_vals = X[class_mask, feature_idx]

                for j, val in enumerate(unique_vals):
                    prob_table[i, j] = np.sum(feature_vals == val) / len(feature_vals)

            self.feature_params_.append({
                'type': 'discrete',
                'unique_vals': unique_vals,
                'prob_table': prob_table
            })
```

```
        else:
            # Для непрерывных признаков
            params = np.zeros((n_classes, 2))  # (mean, std)

            for i, c in enumerate(self.classes_):
                class_mask = (y == c)
                feature_vals = X[class_mask, feature_idx]
                params[i, 0] = np.mean(feature_vals)  # mean
                params[i, 1] = np.std(feature_vals)   # std

            self.feature_params_.append({
                'type': 'continuous',
                'params': params
            })
```

3. Метод predict принимает массив объектов X и возвращает вектор предсказанных классов.

```
def predict(self, X):
    n_samples = X.shape[0]
    n_classes = len(self.classes_)
    log_probs = np.zeros((n_samples, n_classes))

    for i in range(n_classes):
        log_probs[:, i] = np.log(self.class_probs_[i])

        for feature_idx in range(len(self.discrete_flags)):
            if self.discrete_flags[feature_idx]:
                # Дискретный признак
                param = self.feature_params_[feature_idx]
                prob_table = param['prob_table']
                unique_vals = param['unique_vals']

                val_indices = np.array([np.where(unique_vals == x)[0][0]
                                        if x in unique_vals else 0
                                        for x in X[:, feature_idx]])

                log_probs[:, i] += np.log(prob_table[i, val_indices] + 1e-10)  # +1e-10 чтобы избежать log(0)

            else:
                # Непрерывный признак
                params = self.feature_params_[feature_idx]['params']
                mean, std = params[i, 0], params[i, 1]

                log_probs[:, i] += sps.norm.logpdf(X[:, feature_idx], loc=mean, scale=std)

    return self.classes_[np.argmax(log_probs, axis=1)]
```

Разбейте выборку на тренировочную и тестовую, указав долю тестовой выборки, например 1/3, и выбрав случайное число random_state для перемешки выборки.

```
iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=42)
```

```python
iris_df = pd.DataFrame(X, columns=iris.feature_names)

print(iris_df.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
```

```python
# Все признаки непрерывные (False)
our_nb = NaiveBias(discrete_flags=[False, False, False, False])
our_nb.fit(X_train, y_train)
our_pred = our_nb.predict(X_test)
our_accuracy = accuracy_score(y_test, our_pred)

# Все признаки дискретные (True)
our_nb_ = NaiveBias(discrete_flags=[True, True, True, True])
our_nb_.fit(X_train, y_train)
our_pred_ = our_nb_.predict(X_test)
our_accuracy_ = accuracy_score(y_test, our_pred_)

# не все признаки дискретные
_nb = NaiveBias(discrete_flags=[False, False, True, True]) #accure = 0.94
_nb.fit(X_train, y_train)
_pred_ = _nb.predict(X_test)
_accuracy_ = accuracy_score(y_test, _pred_)

# GaussianNB для непрерывных признаков
gnb = GaussianNB()
gnb.fit(X_train, y_train)
gnb_pred = gnb.predict(X_test)
gnb_accuracy = accuracy_score(y_test, gnb_pred)

# MultinomialNB для дискретных признаков
mnb = MultinomialNB()
mnb.fit(X_train, y_train)
mnb_pred = mnb.predict(X_test)
mnb_accuracy = accuracy_score(y_test, mnb_pred)
```

```
Результаты классификации:
мой непрерывный NaiveBias: accuracy = 0.96
мой дискретный NaiveBias: accuracy = 0.92
мой смешанный NaiveBias: accuracy = 0.94
GaussianNB: accuracy = 0.96
MultinomialNB: accuracy = 0.96
```