**Федеральное агентство по образованию**Государственное образовательное учреждение высшего профессионального   
образования **«Тихоокеанский Государственный университет»**

Факультет компьютерных и фундаментальных наук

Кафедра ПОВТАС

**Лабораторная работа №4**

по дисциплине: «Методы машинного обучения»

на тему: «Реализация наивного байесовского классификатора»  
Вариант №4

Выполнил: студент группы ПИИ(м)-21

Латынцев А.В.

Проверил: преподаватель кафедры ПОВТАС

Тормозов В.С.

# Постановка задачи

**Цель работы**: научиться строить наивный байесовский классификатор и с его помощью выполнять бинарную классификацию образов.

**Задания на лабораторную работу** (8 вариант)

1. Необходимо построить (реализовать на языке Python) наивный байесовский классификатор на основе, следующих данных обучающей выборки (для своего варианта):

[http://tk.ulstu.ru/files/iris data.py](http://tk.ulstu.ru/files/iris%20data.py)

Полагать, что признаки независимы и распределены по гауссовскому закону (нормальной плотности распределения вероятностей).

1. Для данной обучающей выборки подсчитать число и процент неверных классификаций.
2. Отобразить обучающую выборку в виде графика точек на плоскости (объекты разных классов должны быть иметь разные маркеры и цвет).

# Краткая теория

Плотность распределения случайной величины в соответствии с теоремой Байеса определяется выражением:

,

* – априорное распределение (доля соответствующего класса в обучающей выборке);
* – распределение образов для класса ;
* – распределение образов всех классов (без принадлежности к тому или иному классу);

Предположим, мы знаем величины и распределения Тогда, правило выбора класса (модель классификации) можно записать, следующим образом:

Главная проблема в реализации байесовского классификатора – это оценить условные плотности распределения вероятности . И, в общем виде, эта задача гораздо сложнее, чем построение классификатора с позиции выбора параметрической функции и последующего поиска вектора параметров :

При условии, что признаки независимы, функции правдоподобия классов представимы в виде:

Или, эквивалентный вывод часто делают по логарифму от произведения величин (уходят от произведений):

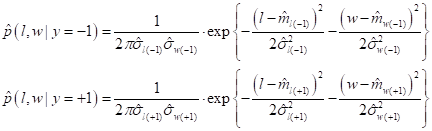
Часто в сторонних библиотеках алгоритм наивного байесовского классификатора реализуют с использованием гауссовских плотностей распределения вероятности. Конечно, это достаточно распространенный случай, но не всегда признаки подчиняются нормальному закону распределения. Это необходимо учитывать.

Если два зависимых признака и независимы и подчиняются нормальному распределению, справедливо:

* – математические ожидания и признаков;
* , – дисперсии и признаков.

Дисперсия определяется по формуле:

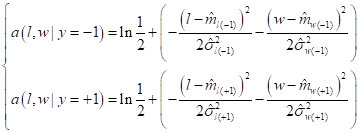
Перепишем, учитывая, что :



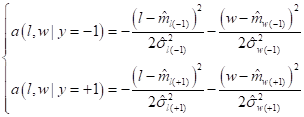
Априорные вероятности равны 0,5. Выполнив уход от произведений, построим эквивалентный классификатор по правилу:



Тогда получим:



Т.к. слагаемое одинаково у обоих классов, его можно не учитывать:



# Результаты работы

Работа была выполнена на языке программирования Python 3 с использованием Jupyter Notebook.

Первым делом (Листинг 1) импортировал необходимые библиотеки, определил данные в соответствии с вариантом:

|  |
| --- |
| **import** **numpy** **as** **np** # 1.23.4  **import** **ipywidgets** **as** **widgets** # 8.0.2  **import** **matplotlib.pyplot** **as** **plt** # 3.6.1  **from** **copy** **import** deepcopy  # использовать системное приложение для взаимодействия с графиками  # в моем случае - TkAgg  %matplotlib  data\_x = [(**2.9**, **6.0**), (**3.8**, **5.1**), (**3.0**, **4.9**), (**3.5**, **5.0**), (**2.6**, **5.5**), (**3.4**, **4.6**), (**3.8**, **5.1**), (**3.5**, **5.5**), (**2.3**, **5.0**), (**3.6**, **4.9**), (**3.5**, **5.1**), (**2.8**, **5.7**), (**3.0**, **5.4**), (**2.9**, **6.4**), (**3.0**, **4.3**), (**3.0**, **4.8**), (**3.5**, **5.1**), (**3.2**, **4.7**), (**2.8**, **5.7**), (**4.2**, **5.5**), (**2.5**, **6.3**), (**2.4**, **4.9**), (**3.1**, **4.8**), (**3.7**, **5.4**), (**3.0**, **5.6**), (**2.7**, **5.6**), (**3.1**, **6.9**), (**2.7**, **6.0**), (**3.4**, **4.8**), (**2.4**, **5.5**), (**3.3**, **5.1**), (**2.5**, **5.6**), (**2.9**, **6.2**), (**3.0**, **5.9**), (**2.8**, **6.1**), (**3.0**, **4.4**), (**2.7**, **5.2**), (**2.9**, **5.7**), (**3.3**, **5.0**), (**3.2**, **6.4**), (**3.4**, **5.2**), (**3.4**, **5.0**), (**3.1**, **4.9**), (**4.4**, **5.7**), (**2.8**, **6.1**), (**3.4**, **5.0**), (**3.1**, **6.7**), (**3.7**, **5.1**), (**3.1**, **4.9**), (**4.0**, **5.8**), (**2.3**, **4.5**), (**3.1**, **6.7**), (**3.2**, **5.0**), (**2.4**, **5.5**), (**3.6**, **5.0**), (**3.9**, **5.4**), (**3.5**, **5.0**), (**2.6**, **5.7**), (**2.8**, **6.8**), (**3.9**, **5.4**), (**2.2**, **6.0**), (**3.2**, **4.4**), (**3.8**, **5.7**), (**3.2**, **4.7**), (**2.9**, **6.6**), (**3.0**, **4.8**), (**2.6**, **5.8**), (**3.0**, **5.0**), (**3.4**, **5.1**), (**3.8**, **5.1**), (**2.3**, **6.3**), (**3.6**, **4.6**), (**2.7**, **5.8**), (**2.9**, **4.4**), (**3.2**, **4.6**), (**3.5**, **5.2**), (**3.1**, **4.6**), (**2.5**, **5.5**), (**2.2**, **6.2**), (**3.2**, **7.0**), (**3.3**, **6.3**), (**3.0**, **6.1**), (**3.4**, **4.8**), (**3.4**, **5.4**), (**2.3**, **5.5**), (**2.5**, **5.1**), (**3.4**, **6.0**), (**2.0**, **5.0**), (**2.9**, **5.6**), (**2.7**, **5.8**), (**2.8**, **6.5**), (**3.4**, **5.4**), (**3.7**, **5.3**), (**4.1**, **5.2**), (**3.0**, **5.6**), (**3.0**, **6.6**), (**2.9**, **6.1**), (**3.0**, **6.7**), (**3.0**, **5.7**), (**3.2**, **5.9**)]  data\_y = [**1**, -**1**, -**1**, -**1**, **1**, -**1**, -**1**, -**1**, **1**, -**1**, -**1**, **1**, **1**, **1**, -**1**, -**1**, -**1**, -**1**, **1**, -**1**, **1**, **1**, -**1**, -**1**, **1**, **1**, **1**, **1**, -**1**, **1**, -**1**, **1**, **1**, **1**, **1**, -**1**, **1**, **1**, -**1**, **1**, -**1**, -**1**, -**1**, -**1**, **1**, -**1**, **1**, -**1**, -**1**, -**1**, -**1**, **1**, -**1**, **1**, -**1**, -**1**, -**1**, **1**, **1**, -**1**, **1**, -**1**, -**1**, -**1**, **1**, -**1**, **1**, -**1**, -**1**, -**1**, **1**, -**1**, **1**, -**1**, -**1**, -**1**, -**1**, **1**, **1**, **1**, **1**, **1**, -**1**, -**1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, -**1**, -**1**, -**1**, **1**, **1**, **1**, **1**, **1**, **1**]  clean\_data = set(zip(data\_x, data\_y)) # чистим данные от дублей  classes = {  'green': {  'data': np.array([data[**0**] **for** data **in** clean\_data **if** data[**1**] == -**1**]),  'label': 'Образы 1 класса',  'alpha': **1**,  },  'blue': {  'data': np.array([data[**0**] **for** data **in** clean\_data **if** data[**1**] == **1**]),  'label': 'Образы 2 класса',  'alpha': **1**,  }  } |

Листинг 1. Первоначальная инициализация

Следующий шаг – вычисление математического ожидания, дисперсии и определение классификатора:

|  |
| --- |
| classes['green']['math\_exp'] = {  'x': np.mean(classes['green']['data'], axis=**0**)[**0**],  'y': np.mean(classes['green']['data'], axis=**0**)[**1**],  }  classes['blue']['math\_exp'] = {  'x': np.mean(classes['blue']['data'], axis=**0**)[**0**],  'y': np.mean(classes['blue']['data'], axis=**0**)[**1**],  }  classes['green']['dispersion'] = {  'x': np.var(classes['green']['data'], axis=**0**)[**0**],  'y': np.var(classes['green']['data'], axis=**0**)[**1**],  }  classes['blue']['dispersion'] = {  'x': np.var(classes['blue']['data'], axis=**0**)[**0**],  'y': np.var(classes['blue']['data'], axis=**0**)[**1**],  }  classes['green']['classifier'] = \  **lambda** x: -(x[**0**] - classes['green']['math\_exp']['x']) \*\* **2** / (**2** \* classes['green']['dispersion']['x']) - (x[**1**] - classes['green']['math\_exp']['y']) \*\* **2** / (**2** \* classes['green']['dispersion']['y'])  classes['blue']['classifier'] = \  **lambda** x: -(x[**0**] - classes['blue']['math\_exp']['x']) \*\* **2** / (**2** \* classes['blue']['dispersion']['x']) - (x[**1**] - classes['blue']['math\_exp']['y']) \*\* **2** / (**2** \* classes['blue']['dispersion']['y'])  **def** **classifier**(xy\_value):  green\_probably = classes['green']['classifier'](xy\_value)  blue\_probably = classes['blue']['classifier'](xy\_value)  **if** green\_probably > blue\_probably:  **return** 'green', {'green': green\_probably, 'blue': blue\_probably}  **return** 'blue', {'green': green\_probably, 'blue': blue\_probably} |

Листинг 2. Определение классификатора

В Листинг 3 я прогоняю обучающую выборку через классификатор для оценки точности классификации:

|  |
| --- |
| classes['green']['classifier\_data'] = {  'good\_defined': [],  'bad\_defined': [],  'bad\_color': 'red',  }  classes['blue']['classifier\_data'] = {  'good\_defined': [],  'bad\_defined': [],  'bad\_color': 'orange',  }  count\_of\_bad\_defined = **0**  **for** class\_i **in** classes:  **for** point **in** classes[class\_i]['data']:  **if** classifier(point)[**0**] == class\_i:  classes[class\_i]['classifier\_data']['good\_defined'].append(point)  **else**:  classes[class\_i]['classifier\_data']['bad\_defined'].append(point)  count\_of\_bad\_defined += **1** |

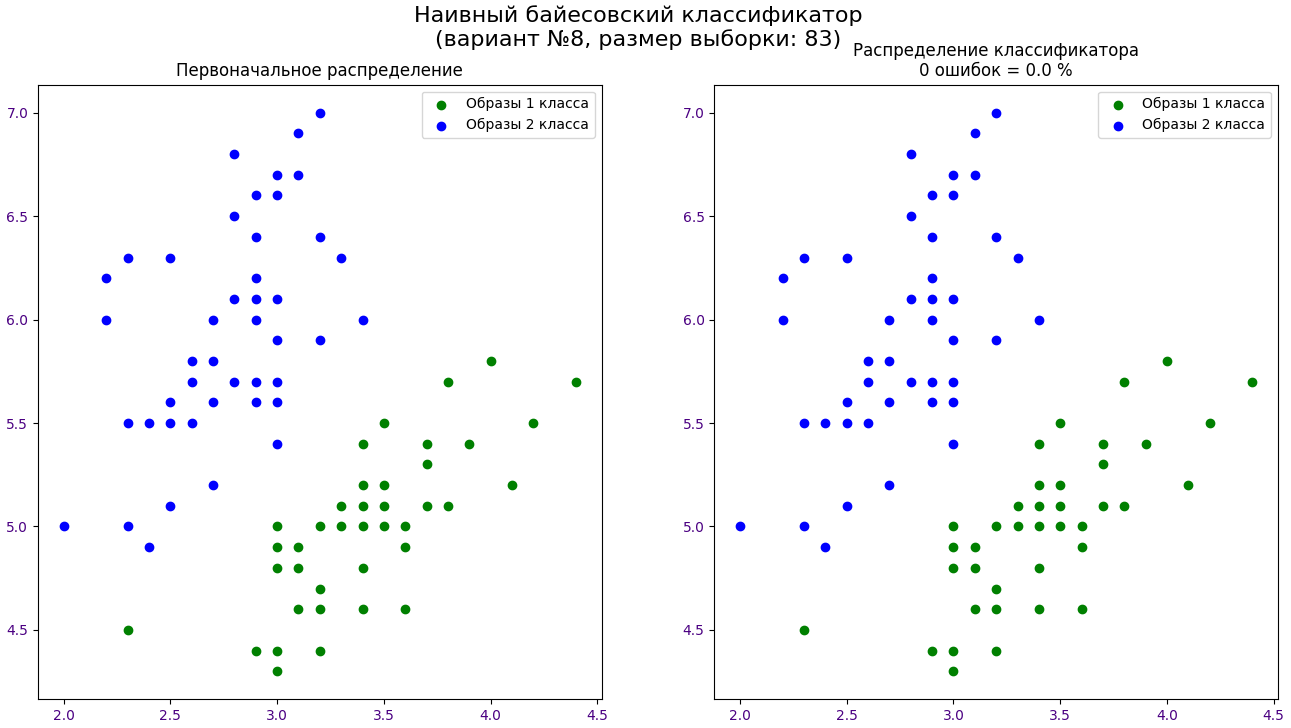
Листинг . Оценка точности классификации

В Листинг 4 описаны директивы для построения графиков:

|  |
| --- |
| fig, ax = plt.subplots(nrows=**1**, ncols=**2**, figsize=(**16**, **8**))  fig.suptitle(f'Наивный байесовский классификатор**\n**(вариант №8, размер выборки: {len(clean\_data)})', fontsize=**16**)  ax[**0**].set\_title('Первоначальное распределение', color='black')  **for** class\_i **in** classes:  ax[**0**].scatter(  classes[class\_i]['data'][:, :**1**],  classes[class\_i]['data'][:, **1**:**2**],  color=class\_i, label=classes[class\_i]['label'],  alpha=classes[class\_i]['alpha'],  )  ax[**0**].tick\_params(labelcolor='indigo')  ax[**0**].legend()  ax[**1**].set\_title("Распределение классификатора**\n**" +  f"{count\_of\_bad\_defined} ошибок = " +  f"{100 \* count\_of\_bad\_defined / len(clean\_data)} %",  color='black'  )  **if** classes[class\_i]['classifier\_data']['good\_defined']:  ax[**1**].scatter(  np.array(classes[class\_i]['classifier\_data']['good\_defined']).transpose()[**0**],  np.array(classes[class\_i]['classifier\_data']['good\_defined']).transpose()[**1**],  color=class\_i, label=classes[class\_i]['label'],  alpha=classes[class\_i]['alpha']  )  **if** classes[class\_i]['classifier\_data']['bad\_defined']:  ax[**1**].scatter(  np.array(classes[class\_i]['classifier\_data']['bad\_defined']).transpose()[**0**],  np.array(classes[class\_i]['classifier\_data']['bad\_defined']).transpose()[**1**],  color=classes[class\_i]['classifier\_data']['bad\_color'],  label="Ошибки " + classes[class\_i]['label'],  alpha=**0.5**  )  ax[**1**].tick\_params(labelcolor='indigo')  ax[**1**].legend()  plt.show() |

Листинг . Построение графиков

В результате работы программы были получены следующие графики:



# Вывод

В ходе лабораторной работы был реализован наивный байесовский классификатор, определено число и процент ошибок, построены соответствующие графики.