Московский авиационный институт
(Национальный исследовательский университет)
Факультет "Информационные технологии и прикладная математика"

Лабораторная работа №1
по курсу “Машинное обучение”

Студент: Живалев Е.А.

Группа: М8О-306Б

Преподаватель: Ахмед Самир Халид

Вариант: -

Оценка: _____

Дата: _____

Москва
2021

1 Задание

Задание : 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 соседей с использованием весов и Наивный Байесовский классификатор и сравнить с реализацией библиотеки sklearn.

2 Описание

2.1 Датасет

Для лабораторной работы мной был выбран датасет Somerville Happiness Survey Data Set, содержащий результаты опроса жителей некоторого города об их условиях жизни.

Подробное описание данных, содержащихся в датасете:
D = decision attribute (D) with values 0 (unhappy) and 1 (happy)
X1 = the availability of information about the city services
X2 = the cost of housing
X3 = the overall quality of public schools
X4 = your trust in the local police
X5 = the maintenance of streets and sidewalks
X6 = the availability of social community events
Attributes X1 to X6 have values 1 to 5.

Соответственно, ставим себе задачу определить является человек счастливым или нет, зная ответы на вопросы X1-X6.

Для загрузки датасета и первичной обработки используется библиотека pandas. Посмотрим на часть данных, вызывая функцию head:

```
[12] data = pd.read_csv("SomervilleHappinessSurvey2015.csv", encoding='utf-16')
     data.head()
```

	D	X1	X2	X3	X4	X5	X6
0	0	3	3	3	4	2	4
1	0	3	2	3	5	4	3
2	1	5	3	3	3	3	5
3	0	5	4	3	3	3	5
4	0	5	4	3	3	3	5

Чтобы получить более подробную информацию вызовем функцию describe:

```
data.describe()
```

	D	X1	X2	X3	X4	X5	X6
count	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000
mean	0.538462	4.314685	2.538462	3.265734	3.699301	3.615385	4.216783
std	0.500271	0.799820	1.118155	0.992586	0.888383	1.131639	0.848693
min	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	0.000000	4.000000	2.000000	3.000000	3.000000	3.000000	4.000000
50%	1.000000	5.000000	3.000000	3.000000	4.000000	4.000000	4.000000
75%	1.000000	5.000000	3.000000	4.000000	4.000000	4.000000	5.000000
max	1.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000

Как можно видеть, пропусков в данных нет, выбросов тоже, все значения признаков соответствуют ожидаемым, то есть лежат в интервале от 1 до 5.

Посмотрим есть ли корреляция между признаками, для этого вызовем функцию corr:

```
[290] data.corr()
```

	D	X1	X2	X3	X4	X5	X6
D	1.000000	0.312740	0.019368	0.163639	0.113356	0.206685	0.220729
X1	0.312740	1.000000	0.092676	0.301971	0.104378	0.399203	0.417521
X2	0.019368	0.092676	1.000000	0.181081	0.107432	-0.002141	0.024546
X3	0.163639	0.301971	0.181081	1.000000	0.298898	0.329874	0.207006
X4	0.113356	0.104378	0.107432	0.298898	1.000000	0.269420	0.199151
X5	0.206685	0.399203	-0.002141	0.329874	0.269420	1.000000	0.307402
X6	0.220729	0.417521	0.024546	0.207006	0.199151	0.307402	1.000000

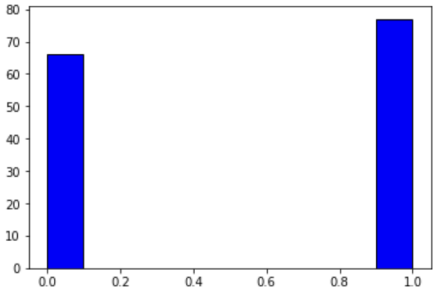
Сильной корреляции между признаками нет.

Построим гистограммы для некоторых признаков:

1. Гистограмма для целевого значения D

```
plt.hist(data['D'], color = 'blue', edgecolor = 'black')
```

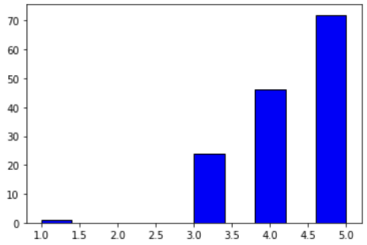
```
(array([66.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 77.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```



2. Гистограмма для признака X1

```
plt.hist(data['X1'], color = 'blue', edgecolor = 'black')
```

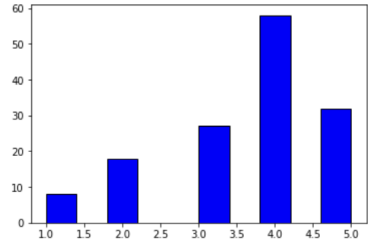
```
(array([ 1.,  0.,  0.,  0.,  0., 24.,  0., 46.,  0., 72.]),
 array([1. , 1.4, 1.8, 2.2, 2.6, 3. , 3.4, 3.8, 4.2, 4.6, 5. ]),
 <a list of 10 Patch objects>)
```



3. Гистограмма для признака X3

```
plt.hist(data['X3'], color = 'blue', edgecolor = 'black')
```

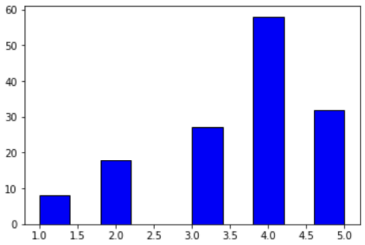
```
(array([ 8.,  0., 18.,  0.,  0., 27.,  0., 58.,  0., 32.]),
 array([1. , 1.4, 1.8, 2.2, 2.6, 3. , 3.4, 3.8, 4.2, 4.6, 5. ]),
 <a list of 10 Patch objects>)
```



4. Гистограмма для признака X5

```
plt.hist(data['X5'], color = 'blue', edgecolor = 'black')
```

```
(array([ 8.,  0., 18.,  0.,  0., 27.,  0., 58.,  0., 32.]),
 array([1. , 1.4, 1.8, 2.2, 2.6, 3. , 3.4, 3.8, 4.2, 4.6, 5. ]),
 <a list of 10 Patch objects>)
```



Разобьем датасет на обучающую и тестовую выборки в процентном соотношении 80 к 20 с помощью метода train_test_split библиотеки sklearn:

```
from sklearn.model_selection import train_test_split
x = data.iloc[:, 1:]
d = data.iloc[:, 0]
x_train, x_test, d_train, d_test = train_test_split(x, d, test_size=0.20)
```

2.2 Алгоритм K ближайших соседей

Идея данного алгоритма состоит в следующем - пусть у нас есть некоторая обучающая выборка $X = ((x_1, y_1), \ldots, (x_n, y_n))$, где x_i - набор признаков для конкретного элемента, y_i - то, к какому классу данный элемент относиться. Зададим некоторую функцию $\rho(x_i, x_k)$, которая будет как-то адекватно показывать насколько элементы x_i и x_k похожи друг на друга(или близки друг к другу). Затем для каждого элемента, который необходимо классифицировать, посчитаем расстояния до всех элементов обучающей выборки и выберем из них k ближайших соседей. Для того чтобы избежать неопределенности при классификации каждому соседу сопоставим некоторый вес, который будет определяться весовой функцией, зависящей от расстояния между классифицируемым элементом и соседом. Я использовал функцию равную обратному квадрату расстояния. Затем для каждого класса суммируем получившиеся веса и относим классифицируемый элемент к тому классу, сумма весов которого получилась наибольшей. Стоит отметить, что для данного алгоритма иногда стоит проводить предобработку датасета, поскольку признаки могут быть распределены в разных промежутках, и, соответственно, тот, что имеет большее значение будет вносить больший импакт в функцию расстояния. В моем случае это не было необходимо, поскольку значения всех признаков находятся в промежутке от 1 до 5.

Для реализации данного мной был написан класс SimpleKNNClassifier с интерфейсом, похожим на интерфейс объектов из библиотеки sklearn, а именной функцией fit для обучения, которой на вход передаются данные обучающей выборки и функцией predict, которой на вход передаются признаки классифицируемого элемента. В функции fit ничего интересного не происходит - полученные данные просто кладутся в поля класса, а в функции predict реализован описанный выше алгоритм.

Исходный код:

```
class SimpleKNNClassifier:
  def __init__(self, neighbours = 5):
    self.neighbours = neighbours

  def fit(self, data, labels):
    self.data = data
    self.labels = labels
    self.number_of_labels = len(np.unique(labels))

  def predict(self, item):
    distances = np.sum((item[np.newaxis, :] - self.data[:]) ** 2,
     axis=1)
    nearest = np.argsort(distances)
    scores = np.zeros(self.number_of_labels)
    for i in range(self.neighbours):
      if distances[nearest[i]] == 0:
        return self.labels[nearest[i]]
      else:
        weight = 1 / distances[nearest[i]]
      scores[self.labels[nearest[i]]] += weight
    return scores.argmax()
```

Было интересно посмотреть как меняется точность классификации в зависимости от выбора количества числа соседей и сравнить результат с реализацией данного алгоритма из библиотеки sklearn - KNeighborsClassifier. Для этого были получены точности предсказаний для количества соседей от 1 до 39. Получившийся результат для тестовой выборки:

График для моей реализации:

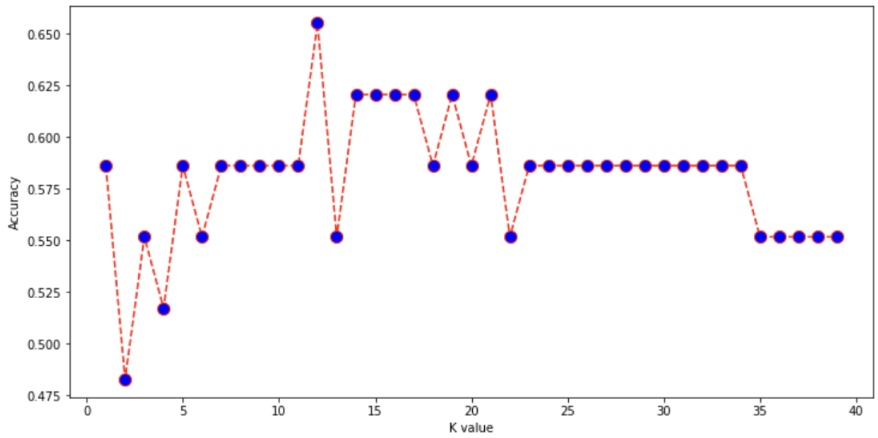
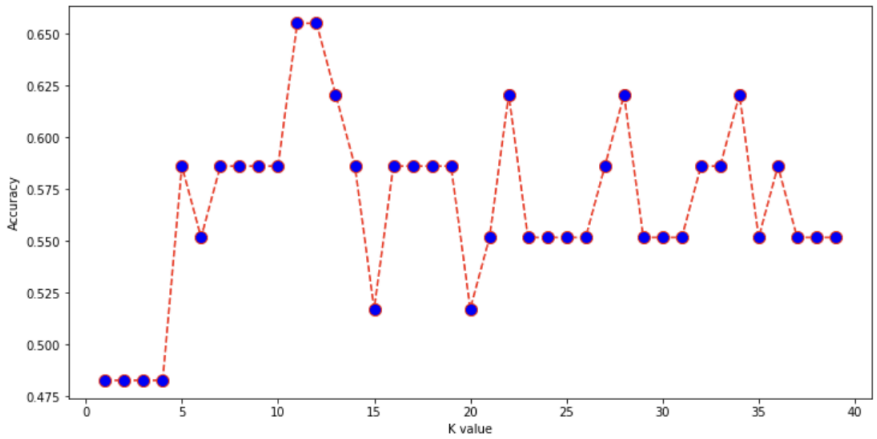


График для реализации библиотеки sklearn:



Как можно видеть, максимальная точность совпадает, но достигается при различных значениях k. Большим минусом моего датасета является его размер - он достаточно небольшой. Из $5^5 = 3125$ возможных комбинаций ответов там лишь 143 и при этом некоторые повторяются. Из-за этого точность очень сильно может разниться при разных разбиениях на обучающую и тестовую выборки. В данном случае моя реализация в среднем предсказывает точнее, однако общая тенденция за несколько запусков такова, что реализация из библиотеки sklearn в среднем точнее предсказывает, но максимальные значения для точности совпадают(как правило, при разных значениях k). Тонким моментом, ощутимо влияющим на точность является способ обработки ситуации, когда расстояния до одного из соседов равно 0. В этом случае функция весов неопределенна. Я в таком случае возвращаю класс этого соседа, а библиотека sklearn, насколько мне известно, в этом случае берет вес равным 1. Я пробовал делать также, но результаты получались хуже. К сожалению, определить почему так значительно отличаются результаты моей реализации от реализации библиотеки sklearn мне установить не удалось.

2.3 Наивный Байесовский классификатор

Данный метод основан на использовании форумлы Байеса:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

и предположении о независимости признанков друг от друга. Имея обучающую выборки мы можем посчитать значения $P(x_i|y_i)$ для всех пар x_i, y_i, где x_i - значение некоторого признака, y_i - некоторый класс. Тогда, пользуясь формулой Байеса:

$$P(y_i|x_1, \ldots, x_n) = \frac{P(y_i) * \prod_{i=1}^{n} P(x_i|y_i)}{P(x_1, \ldots, x_n)}$$

Поскольку для всех y_i значение в знаменателе равно, его можно опустить. Таким образом, оценка для некоторого y по заданным $x_1, \ldots, x_n$:

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i|y_i)$$

Поскольку в произведении может получиться очень малое число имеет смысл заменить его на сумму логарифмов, например, натуральных:

$$\hat{y} = \arg\max_y ln(P(y)) + \sum_{i=1}^{n} ln(P(x_i|y_i))$$

Исходный код:

```
from math import exp, pi, sqrt, log

class NaiveBayesClassifier:
  def fit(self, data, labels):
      self.labels_prob = dict()
      for label in labels:
        self.labels_prob.setdefault(label, 0)
        self.labels_prob[label] += 1
      self.features_prob = [dict() for i in range(len(data[0]))]
      for feat in range(len(data[0])):
        for i in range(len(data)):
          self.features_prob[feat].setdefault(data[i][feat], [0
    for _ in range(len(self.labels_prob))])
          self.features_prob[feat][data[i][feat]][labels[i]] += 1
        for key in self.features_prob[feat]:
          for i in range(len(self.features_prob[feat][key])):
            self.features_prob[feat][key][i] /= self.labels_prob[i
    ]
      for key in self.labels_prob:
        self.labels_prob[key] /= len(labels)

  def predict(self, item):
    max = 0
    max_i = -1
    for i in range(len(self.labels_prob)):
      current = 0
      for j in range(len(item)):
```

```
26          self.features_prob[j].setdefault(item[j], [0 for _ in
    range(len(self.labels_prob))])
27          if self.features_prob[j][item[j]][i] != 0:
28            current += log(self.features_prob[j][item[j]][i])
29        current += log(self.labels_prob[i])
30        if current > max or max_i == -1:
31          max = current
32          max_i = i
33      return max_i
```

2.4 Наивный Гауссовский Байесовский классификатор

Поскольку в библиотеке sklearn нет реализации описанного выше наивного Байесвского классификатора описанного выше было решено реализовать одну из его вариаций - наивный Гауссовский Байесовский классификатор, который также реализован в sklearn. Главным отличием от обычного наивного классификатора является предположение о том, что все значения распределены по нормальному закону распределения. Поэтому искомые вероятности определяются с помощью функции плотности вероятности для нормального распределения:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp(\frac{-(x_i - \mu_y)^2}{2\sigma_y^2})$$

Исходный код:

```
1  def calc_prob(val, mean, std):
2    return exp(-(val - mean) ** 2 / (2 * std ** 2)) / sqrt(2 * pi *
      std ** 2)
3
4  class NaiveGaussianBayesClassifier:
5    def fit(self, data, labels):
6      self.labels_prob = dict()
7      self.feat_props = []
8      for label in labels:
9        self.labels_prob.setdefault(label, 0)
10       self.labels_prob[label] += 1
11     for key in self.labels_prob:
12       self.labels_prob[key] /= len(labels)
13     for feat in range(len(data[0])):
14       counts = [[] for _ in range(len(self.labels_prob))]
15       props = []
16       for i in range(len(data)):
17         counts[labels[i]].append(data[i][feat])
18       for i in range(len(self.labels_prob)):
19         arr = np.array(counts[i])
20         props.append((arr.mean(), arr.std()))
21       self.feat_props.append(props)
22
23   def predict(self, item):
24     max = -1 * 10 ** 10
25     max_i = -1
26     for i in range(len(self.labels_prob)):
27       current = 0
28       for j in range(len(item)):
29         current += log(calc_prob(item[j], self.feat_props[j][i
     ][0], self.feat_props[j][i][1]))
30       current += log(self.labels_prob[i])
```

```
      if current > max or max_i == -1:
        max = current
        max_i = i
    return max_i
```

Сравнение результатов:

1. Наивный Байесовский классификатор

```
f = NaiveBayesClassifier()
f.fit(x_train.to_numpy(), d_train.to_numpy())
results = []
for j in range(len(d_test)):
  results.append(f.predict(x_test.iloc[j]))
counter = 0
print(accuracy_score(d_test, results))
```

```
0.5862068965517241
```

2. Наивный Гауссовский Байесовский классификатор

```
xd = NaiveGaussianBayesClassifier()
xd.fit(x_train.to_numpy(), d_train.to_numpy())
results = []
for j in range(len(d_test)):
  results.append(xd.predict(x_test.iloc[j]))
print(accuracy_score(d_test, results))
```

```
0.5517241379310345
```

3. Наивный Гауссовский Байесовский классификатор из библиотеки sklearn

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(x_train, d_train)
model.score(x_test, d_test)
```

```
0.5517241379310345
```

Во всех проведенных мной запусках результаты для моей реализации Наивного Гауссовского Байесовского классификатора и реализации из sklearn совпали, что не может не радовать. Наивный Байесовский классификатор иногда предсказывал точнее, а иногда нет.

3 Выводы

Пожалуй, данная лабораторная работа была наиболее интересной за довольно долгий период времени. При её выполнении пришлось многое вспомнить, например, базовые вещи из курса матстата, познакомиться со многим новым - библиотеками pandas, numpy, sklearn, неизвестными до этого вещами из Python. Несмотря на то, что данные алгоритмы являются одними из самых простых из области машинного обучения, их тем не менее было интересно реализовывать. Очень обидно, что мне не удалось установить почему так сильно отличаются предсказания моей реализации алгоритма knn и реализации из библиотеки sklearn. Точность предсказания у обоих алгоритмов получилась достаточно невысокая - скорее всего, это связано с малым размером датасета и, как следствие, с малым количеством данных для обучения.

4 Список литературы

1. http://archive.ics.uci.edu/ml/index.php - сайт, с которого был взят датасет
2. www.kaggle.com/learn - много полезных вещей было изучено здесь
3. http://www.machinelearning.ru - описания алгоритмов
4. Ноутбуки преподавателя
5. https://scikit-learn.org/ - документация библиотеки sklearn