به نام خدا

تمرین اول یادگیری عمیق

غزل زمانی‌نژاد

۹۷۵۲۲۱۶۶

.1

- **AI winter: با وجود اینکه یادگیری عمیق دست‌آوردهای بسیار خوبی دارد، اما مقدار پیشرفت آن در زمینه‌های مختلف از مقداری که انتظار می‌رود کمتر است. در گذشته، هوش‌مصنوعی دو بار در سیکل خوش‌بینی شدید به همراه ناامیدی و تردید قرار گرفت و با کمبود بودجه مواجه شد.**

این اتفاق نخست در سال ۱۹۶۰ با symbolic AI شروع شد. یکی از پیشگامان این زمینه، Marvin Minsky در سال ۱۹۶۷ ادعا کرد که طی یک نسل مسئله ساختن هوش‌مصنوعی به طور قابل ملاحظه‌ای حل می‌شود. سپس در سال ۱۹۷۰ پیش‌بینی کرد که در طی ۳ تا ۸ سال به ماشینی دست خواهیم یافت که هوش آن با میانگین هوش انسان‌ها برابری می‌کند. اما تا کنون انسان نتوانسته به این میزان هوش دست یابد و زمان دسترسی به آن قابل پیش‌بینی نیست. در سال‌های بعد از ادعا، با بالا رفتن سطح انتظارات از هوش‌مصنوعی و برآورده نشدن آنها، محققان از کار در این فیلد دست کشیدند. به این دوره AI winter می‌گوییم. (نام آن بر اساس nuclear winter انتخاب شده‌است، چون دقیقا بعد از اوج جنگ سرد رخ داده‌است.)

دومین بار در سال ۱۹۸۰، expert systems با شرکت‌های بزرگ همکاری کرد. چند موفقیت منجر به موج سرمایه‌گذاری در این زمینه شد. در سال ۱۹۸۵ شرکت‌ها سالانه نزدیک به ۱ میلیارد دلار بر روی این تکنولوژی سرمایه‌گذاری کردند. اما در دهه ۹۰ با مشکلاتی از قبیل هزینه‌های زیاد نگهداری، تغییر مقیاس دشوار و اسکوپ محدود رو به رو شدند و در نتیجه علاقه‌مندی به این زمینه کاهش پیدا کرد.

هم‌اکنون ممکن است در سومین سیکل AI winter و در فاز خوش‌بینی آن قرار گرفته باشیم. پس بهتر است انتظارات متعادلی از این فیلد داشته‌باشیم.

- **Backpropagation: در فرآیند ساختن و بهبود شبکه‌های عصبی، یکی از مشکلات تا مدت‌های طولانی یافتن راه بهینه برای آموزش شبکه‌های عصبی بزرگ بود. محققان به صورت جداگانه در حال تحقیق بودند که منجر به کشف دوباره backpropagation شد. پس‌انتشار روشی برای آموزش زنجیره عملیات‌های پارامتری با استفاده از گرادیان کاهشی است. نخستین استفاده موفقیت‌آمیز آن در سال ۱۹۶۸ بود. Yann LeCun دو ایده شبکه‌های عصبی پیچشی و پس‌انتشار را بر روی دیتاست رقم‌های دست‌نویس پیاده‌سازی کرد. این کار منجر به ساختن شبکه LeNet شد.**
- **Objective Function: برای کنترل کردن یک مقدار، ابتدا نیاز به مشاهده آن داریم. برای کنترل خروجی شبکه عصبی باید بتوانیم فاصله خروجی و مقدار مورد انتظارمان را اندازه‌گیری کنیم. این کار با استفاده از loss function یا همان objective function انجام می‌شود. این تابع پیش‌بینی ما و لیبل صحیح را دریافت می‌کند و distance score را محاسبه می‌کند. این امتیاز نشان‌دهنده میزان خوب بودن عملکرد شبکه بر روی داده است.**
- **Kernel Methods: در سال ۱۹۹۰ یک رویکرد جدید بدست آمد و شبکه‌های عصبی به فراموشی سپره شدند. رویکرد kernel method شامل الگوریتم‌های طبقه‌بندی است که معروف‌ترین آن support vector machine است. SVM بدنبال یافتن یک مرز تصمیم‌گیری برای جدا کردن دو مجموعه از نقاط است. این الگوریتم شامل دو مرحله است:**

داده به بازنمایی با ابعاد بالا مپ می‌شود که در آن مرز تصمیم می‌تواند به صورت یک ابرصفحه نمایش داده شود.

در مرحله maximizing the margin تلاش می‌شود تا فاصله بین ابرصفحه و نزدیک‌ترین نقاط هر کلاس بیشینه شود. این کار به مرز تصمیم کمک می‌کند تا به خوبی به نمونه‌های جدید خارج از داده آموزشی تعمیم داده شود.

تکنیک مپ کردن داده‌ها به فضای دیگر از نظر محاسباتی قابل تامل است. Kernel trick برای رفع این مشکل به کار می‌آید. برای محاسبه بازنمایی جدید نیازی به محاسبه مختصات جدید هر یک از نقاط نیست. بلکه می‌توانیم فاصله میان زوج‌های نقاط را محاسبه کنیم. این کار به شکل بهینه توسط kernel function انجام می‌شود.

SVM تا مدت‌های طولانی از فیلدهای محبوب و مشهور بود. اما بر روی دیتاست‌های بزرگ و همچنین مسائل ادراکی از جمله دسته‌بندی تصاویر عملکرد خوبی نداشت.

- **4D tensors vs 4-dimensional vector: بردار 4d تنها یک محور دارد و آن محور ۴ بعد دارد مثال:**

Np.array([1, 2, 3, 4])

اما یک تنسور 4d، ۴ محور دارد و محورهای آن می‌تواند هر مقدار بعد داشته‌باشد. در حقیقت با کنار هم گذاشتن تنسورهای 3d می‌توانیم تنسور 4d بسازیم.

- **Element-wise product vs Tensor product: در element-wise product عملیات به صورت جداگانه بر روی تک تک entryهای تنسور انجام می‌شود. این عملیات‌ها قابلیت پیاده‌سازی موازی را دارند (به آن vectorized implementation می‌گوییم.) در numpy و چند فریم‌ورک دیگر با عملگر * انجام می‌شود. اما در tensor product برخلاف حالت قبل entryهای تنسورها با یکدیگر ترکیب می‌شوند. این نوع ضرب مشابه ضرب ماتریس‌ها در ریاضیات است. برای انجام این عملیات باید بعد اول تنسور اول با بعد صفرم تنسور دوم یکی باشد. نتیجه آن یک اسکالر و یا تنسور با شکل (x.shape[0],y.shape[1]) می‌باشد.**

. .2

2) داده های ورودی این سوال متغیر دودویی دارند پس بهتر است از bernoulli naive bayes استفاده کنیم. (چون فاصله یک feature برای not spam صفر است نمی توانیم از guassian naive bayes استفاده کنیم) محاسبات:

$P(spam) = 6/10 \quad P(not\ spam) = 4/10$

P(fic)	spam	not spam
f_1	$1/6$	1
f_2	$5/6$	$1/4$
f_3	$4/6$	$1/4$

$P(f_1 = 1) = 5/10 = 1/2$

$P(f_2 = 1) = 6/10 = 3/5$

$P(f_3 = 1) = \frac{5}{10} = 1/2$

$x_1 = [1 \ 1 \ 0] \quad P(spam \mid f_1, f_2, \bar{f}_3) = P(f_1, f_2, \bar{f}_3 \mid Spam) \times P(spam) =$

$= P(f_1 \mid spam)\, P(f_2 \mid spam)\, (1 - P(f_3 \mid spam))\, P(spam)$

$= 1/6 \times 5/6 \times \underbrace{(1 - 4/6)}_{2/6} \times 6/10 = 1/36$

$P(not\ spam \mid f_1, f_2, \bar{f}_3) = P(f_1, f_2, \bar{f}_3 \mid not\ spam) \times P(not\ spam) =$

$= P(f_1 \mid not\ spam) \times P(f_2 \mid not\ spam)\, (1 - P(f_3 \mid not\ spam))\, P(not\ spam) =$

$= 1 \times 1/4 \times \underbrace{(1 - 1/4)}_{3/4} \times 4/10 = 3/40$

$\Rightarrow 1/36 < 3/40$ این داده مربوط به کلاس not spam است

$x_2 = [1, 1, 1] \quad P(spam \mid f_1, f_2, f_3) = P(f_1, f_2, f_3 \mid spam) \times P(spam) = P(f_1 \mid spam)\, P(f_2 \mid spam)$

$P(f_3 \mid spam)\, P(spam) = 1/6 \times 5/6 \times 4/6 \times 6/10 = 1/18$

$P(not\ spam \mid f_1, f_2, f_3) = P(f_1, f_2, f_3 \mid not\ spam)\, P(not\ spam) = P(f_1 \mid not\ spam) \times$

$P(f_2 \mid not\ spam)\, P(f_3 \mid not\ spam)\, P(not\ spam) = 1 \times 1/4 \times 1/4 \times 4/10 = 1/40$

$\Rightarrow 1/18 > 1/40$ این داده مربوط به کلاس spam است.

توجه ۱ : می‌توانیم احتمال‌ها را بر evidence تقسیم کنیم:

① $P(x_1) = P(f_1=1)\ P(f_2=1)\ P(f_3=0) = \frac{1}{2} \times \frac{3}{5} \times \frac{1}{2} = \frac{3}{20} = \frac{15}{100}$

$P(spam \mid x_1) = \frac{1/36}{3/20} = \frac{5}{27}$ $\quad$ $P(not\ spam \mid x_1) = \frac{3/40}{3/20} = \frac{1}{2}$

② $P(x_2) = P(f_1=1)\ P(f_2=1)\ P(f_3=1) = \frac{1}{2} \times \frac{3}{5} \times \frac{1}{2} = \frac{3}{20}$

$P(spam \mid x_2) = \frac{1/18}{3/20} = \frac{10}{27}$ $\quad$ $P(not\ spam \mid x_2) = \frac{1/40}{3/20} = \frac{1}{6}$

توجه 2 : بهتر است از laplace smoothing استفاده کنیم تا هیچ یک از احتمال‌های شرطی صفر نشوند.

3. **ابتدا به کمک قطعه کد زیر دیتاست cifar10 را دانلود می‌کنیم.**

```
1 from keras.datasets import cifar10
2 (x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 0us/step
```

سپس موارد خواسته شده را چاپ می‌کنیم.

```
[2]  1 print(x_train.dtype)
     2 print(x_train.ndim)
     3 print(x_train.shape)
```

```
uint8
4
(50000, 32, 32, 3)
```

Cell2: المان‌های ورودی داده آموزشی از نوع int هستند. تعداد محورهای این تنسور ۴ است (تنسور 4d).

دارای 50k تصویر 3 * 32 * 32 است (۳ تعداد کانال‌های هر تصویر است.)

```
[3]   1 print(y_train.dtype)
      2 print(y_train.ndim)
      3 print(y_train.shape)

    uint8
    2
    (50000, 1)
```

Cell3: **المان‌های خروجی داده آموزشی از نوع** int **هستند. تعداد محورهای این تنسور ۲ است (تنسور** 2d**).**

دارای 50k **برچسب برای هریک از تصاویر ورودی است.**

```
[4]   1 print(x_test.dtype)
      2 print(x_test.ndim)
      3 print(x_test.shape)

    uint8
    4
    (10000, 32, 32, 3)
```

Cell4: **المان‌های ورودی داده آزمون از نوع** int **هستند. تعداد محورهای این تنسور ۴ است (تنسور** 4d**).**

دارای 10k **تصویر** 3 * 32 * 32 **است (۳ تعداد کانال‌های هر تصویر است.)**

```
[5]   1 print(y_test.dtype)
      2 print(y_test.ndim)
      3 print(y_test.shape)

    uint8
    2
    (10000, 1)
```

Cell5: **المان‌های خروجی داده آزمون از نوع** int **هستند. تعداد محورهای این تنسور ۲ است (تنسور** 2d**).**

دارای 10k **برچسب برای هریک از تصاویر ورودی است.**

4. **در این سوال می‌خواهیم الگوریتم** Gaussian Naïve Bayes **را بر روی دیتاست** Iris **پیاده‌سازی کنیم.**

دیتاست Iris **برای طبقه‌بندی ۳ نوع گل** setosa**،** versicolor **و** virginica **طراحی شده‌است. شامل ۱۵۰ داده است که برای هر داده ۴ ویژگی استخراج شده‌است.**

ابتدا در محیط google colab این دیتاست را دانلود می‌کنیم.

```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
```

```
--2021-10-02 07:31:54--  https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4551 (4.4K) [application/x-httpd-php]
Saving to: ‘iris.data.1’

iris.data.1         100%[===================>]   4.44K  --.-KB/s    in 0s

2021-10-02 07:31:55 (109 MB/s) - ‘iris.data.1’ saved [4551/4551]
```

سپس با استفاده از دستور with open، فایل دانلود شده را می‌خوانیم. آن را در یک دیکشنری که کلیدها گل‌های مختلف هستند ذخیره می‌کنیم. همچنین در یک لیست با نام test ذخیره می‌کنیم تا در پایان بتوانیم مدل آموزش‌دیده را تست کنیم.

```
import numpy as np
import random

data = {}
data['Iris-setosa'] = []
data['Iris-versicolor'] = []
data['Iris-virginica'] = []

test = []
# process the data

with open('iris.data') as infile:
  for line in infile:
    num = line.split(',')
    # last line shouldn't be read
    if(len(num) == 1):
      break

    label = num[4].strip()
    features = np.array([float(x) for x in num[:4]])

    data[label].append(features)
    test.append((features, label))
```

سپس valueهای دیکشنری را به صورت numpy array در می‌آوریم. و شکل داده‌ها را چاپ می‌کنیم.

```
data['Iris-setosa'] = np.array(data['Iris-setosa'])
data['Iris-versicolor'] = np.array(data['Iris-versicolor'])
data['Iris-virginica'] = np.array(data['Iris-virginica'])

print("class 1 train shape", data['Iris-setosa'].shape)
print("class 2 train shape", data['Iris-versicolor'].shape)
print("class 3 train shape", data['Iris-virginica'].shape)
```

```
class 1 train shape (50, 4)
class 2 train shape (50, 4)
class 3 train shape (50, 4)
```

در سلول بعد میانگین و واریانس هر یک از ویژگی‌های کلاس‌های مختلف را محاسبه می‌کنیم. و در دو دیکشنری ذخیره می‌کنیم.

```
# compute mean and variance for each feature of each class
def compute_mean(v):
  return np.mean(v, axis=0)

def compute_var(v):
  return np.var(v, axis=0)

mean = {}
var = {}
for k in data.keys():
  mean[k] = compute_mean(data[k])
  var[k] = compute_var(data[k])
  print("mean in class", k, mean[k])
  print("var in class", k, var[k])
```

بعد احتمال پیشین هریک از کلاس‌ها را محاسبه می‌کنیم. و این اطلاعات را چاپ می‌کنیم.

```
# compute probability of each label
p = {}
total = len(data['Iris-setosa']) + len(data['Iris-versicolor']) + len(data['Iris-virginica'])
for k in data.keys():
  p[k] = len(data[k]) / total

print(p)
```

```
mean in class Iris-setosa [5.006 3.418 1.464 0.244]
var in class Iris-setosa [0.121764 0.142276 0.029504 0.011264]
mean in class Iris-versicolor [5.936 2.77  4.26  1.326]
var in class Iris-versicolor [0.261104 0.0965   0.2164   0.038324]
mean in class Iris-virginica [6.588 2.974 5.552 2.026]
var in class Iris-virginica [0.396256 0.101924 0.298496 0.073924]
{'Iris-setosa': 0.3333333333333333, 'Iris-versicolor': 0.3333333333333333, 'Iris-virginica': 0.3333333333333333}
```

در سلول آخر مدل را بر روی داده‌ها ارزیابی می‌کنیم. برای محاسبه احتمال پیشین هر یک از داده‌ها از فرمول زیر استفاده می‌کنیم:

$$\text{Posterior} = \frac{\text{Likelihood* Prior}}{\text{Evidence}}$$

$$P(C_j \mid A_1, A_2, ..., A_n) = \frac{\left(\prod_{i=1}^{n} P(A_i \mid C_j)\right) P(C_j)}{P(A_1, A_2, ..., A_n)}$$

و برای محاسبه هر یک از احتمال‌های شرطی از فرمول زیر استفاده می‌کنیم:

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

برای محاسبه این فرمول‌ها از دو تابع زیر استفاده شده‌است. (برای محاسبه احتمال پسین تقسیم صورت بر evidence **اعمال نشده چون در نتیجه نهایی تاثیر ندارد.)**

```
1 def compute_conditional(v, mean, var):
2   return (1 / np.sqrt(2 * np.pi * var)) * np.exp((-1 * (v - mean)**2) / (2 * var))
3
4 def predict(l, v):
5   conditional = compute_conditional(v, mean[l], var[l])
6   return p[l] * np.prod(conditional)
7
```

سپس لیست test **را شافل می‌کنیم تا ترتیب داده‌ها بهم بریزد. بعد در یک لوپ تمامی داده‌ها را به مدل می‌دهیم (با استفاده از تابع** predict**،** P(t | class) **ها را محاسبه می‌کنیم). پیش‌بینی مدل از آن داده متناظر با بالاترین احتمال پسین است.**

در صورتی که پیش‌بینی مدل با برچسب داده یکسان باشد یک عدد به پاسخ‌های درست اضافه می‌کنیم.

در پایان دقت مدل را پرینت می‌کنیم.

```
 8 # shuffle test data
 9 random.shuffle(test)
10
11 print("prediction***************true label")
12 accuracy = 0
13 for t in test:
14   max_p = 0
15   for k in data.keys():
16     pr = predict(k, t[0])
17     if pr > max_p:
18       max_p = pr
19       prediction = k
20   if prediction == t[1]:
21     accuracy += 1
22
23   print(prediction, "**********", t[1])
24
25 print("total accuracy is", accuracy/total)
```

```
total accuracy is 0.96
```