به نام خدا

داده‌کاوی

تمرین چهارم

خانم دکتر شاکری

محمد امین عرب خراسانی

۸۱۰۱۰۲۲۰۵

بهار ۱۴۰۳

بخش تشریحی

۱) برای محاسبه‌ی انتروپی امکان برگزاری مسابقه از رابطه‌ی ۱ استفاده می‌شود.

$$Info(D) = -\sum_{i=1}^{m} p_i \, log_2(p_i)$$ رابطه‌ی ۱

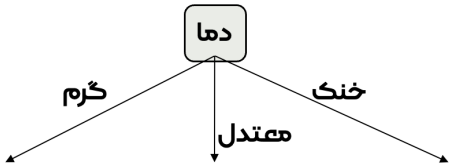
$p_1 = \frac{9}{14} = 0.643$ نسبت تعداد داده‌ها با لیبل برگزاری به تعداد کل داده‌ها

$p_2 = \frac{5}{14} = 0.357$ نسبت تعداد داده‌ها با لیبل عدم برگزاری به تعداد کل داده‌ها

$$Info(D) = -(\mathrm{p}_1 \log_2(p_1) + \mathrm{p}_2 \log_2(p_2))$$
$$\Longrightarrow Info(D) = -(0.643 \log_2(0.643) + 0.357 \log_2(0.357))$$
$$\Longrightarrow Info(D) = -((0.643)(-0.637) + (0.357)(-1.486))$$
$$\Longrightarrow Info(D) = 0.94$$

---

۲) برای به دست آوردن مقدار information gain ویژگی دما، مقادیر یونیک آن مشخص می‌شوند و در ادامه با استفاده از رابطه‌ی ۱ انتروپی هر کدام محاسبه می‌شود.



$$info_{temp}(D) = \frac{4}{14} I(2,2) + \frac{6}{14} I(4,2) + \frac{4}{14} I(3,1)$$

$I(2,2) = 1$

$I(4,2) = 0.918$

$I(3,1) = 0.811$

$$\Longrightarrow Info_{temp}(D) = (\frac{4}{14})(1) + (\frac{6}{14})(0.918) + (\frac{4}{14})(0.811)$$
$$\Longrightarrow Info_{temp}(D) = (\frac{4}{14})(1) + (\frac{6}{14})(0.918) + (\frac{4}{14})(0.811)$$
$$\Longrightarrow Info_{temp}(D) = 0.911$$

در ادامه با استفاده از رابطه‌ی ۲ مقدار information gain برای ویژگی دما محاسبه می‌شود.

$$Gain(temp) = Info(D) - Info_{temp}(D)$$ رابطه‌ی ۲

$$\Longrightarrow Gain(temp) = 0.94 - 0.911 = 0.029$$

---

۳) به منظور از ID3 به عنوان الگوریتم رسم درخت، می‌بایست برای همه‌ی ویژگی‌های موجود information gain محاسبه شده و با یکدیگر مقایسه شوند. در هر مرحله دیتاست بر اساس ویژگی‌ای که بیشترین information gain را داشته باشد تقسیم می‌شود.

$$Gain(temp) = 0.029$$

$$info_{humidity}(D) = \frac{7}{14} I(3,4) + \frac{7}{14} I(6,1) = \frac{7}{14}(0.985) + \frac{7}{14}(0.592) = 0.789$$

$$Gain(humidity) = 0.1515$$

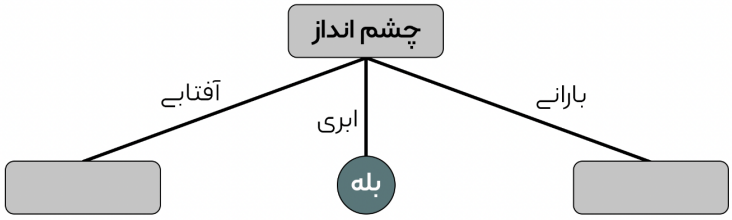
$$info_{wind}(D) = \frac{8}{14} I(6,2) + \frac{6}{14} I(3,3) = \frac{8}{14}(0.811) + \frac{6}{14}(1) = 0.892$$

$$Gain(wind) = 0.048$$

$$info_{outlook}(D) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = \frac{5}{14}(0.971) + \frac{4}{14} I(4,0) + \frac{5}{14}(0.971)$$

$$Gain(outlook) = 0.2464$$

با توجه به information gain های حاصل، ویژگی‌ای که بیشترین information gain را دارد به عنوان ریشه‌ی اصلی انتخاب می‌شود. سه حالت برای چشم انداز به عنوان ریشه‌ی اصلی وجود دارد. شکل زیر این حالت‌ها را نشان می‌دهد.



---

۴) با توجه به انتخاب ریشه‌ی اصلی برای ۳ ویژگی دیگر، یک به یک برای شرط آفتابی بودن هوا information gain محاسبه می‌شود. سپس ویژگی‌ای که بیشترین information gain را دارد انتخاب می‌شود.

۵ نمونه داریم که چشم انداز آن‌ها "آفتابی" است. ۳ تا از آن‌ها منجر به عدم برگزاری مسابقه و ۲ تا از آن‌ها منجر به برگزاری مسابقه می‌شود. بنابراین داریم:

$p_1 = \frac{2}{5} = 0.4$ نسبت تعداد داده‌های آفتابی با لیبل برگزاری به تعداد کل داده‌ها

$p_2 = \frac{3}{5} = 0.6$ نسبت تعداد داده‌های آفتابی با لیبل عدم برگزاری به تعداد کل داده‌ها

$$Info(D) = -(\mathrm{p}_1 \log_2(p_1) + \mathrm{p}_2 \log_2(p_2))$$
$$Gain(A) = Info(D) - Info_A(D)$$

Entropy:

$$\Longrightarrow Info(D) = -(0.4 \log_2(0.4) + 0.6 \log_2(0.6))$$
$$\Longrightarrow Info(D) = -((0.643)(-0.637) + (0.357)(-1.486))$$
$$\Longrightarrow Info(D) = 0.971$$

Temp:

$$info_{temp}(D) = \frac{2}{5} I(2,0) + \frac{2}{5} I(1,1) + \frac{1}{5} I(1,0) = 0 + 0.4 + 0 = 0.4$$
$$Gain(temp) = 0.571$$

Humidity:

$$info_{humidity}(D) = \frac{3}{5} I(0,3) + \frac{2}{5} I(2,0) = 0$$
$$Gain(humidity) = 0.971$$

wind:

$$info_{wind}(D) = \frac{3}{5} I(1,2) + \frac{2}{5} I(1,1) = 0.551 + 0.4$$
$$Gain(wind) = 0.02$$

با توجه به information gain های به دست آمده، برای چشم انداز آفتابی، ویژگی رطوبت در نظر گرفته می‌شود.

برای تمام نمونه‌هایی از چشم انداز که مقادیر آن ابری است، امکان برگزاری بازی تنیس وجود دارد. بنابراین به برگ می‌رسیم.

برای نمونه‌های چشم انداز که مقادیر آن بارانی هستند، برای ۲ ویژگی باقی‌مانده information gain محاسبه می‌شود.

$p_1 = \frac{3}{5} = 0.6$ نسبت تعداد داده‌های بارانی با لیبل برگزاری به تعداد کل داده‌ها

$p_2 = \frac{2}{5} = 0.4$ نسبت تعداد داده‌های بارانی با لیبل عدم برگزاری به تعداد کل داده‌ها

$$Info(D) = -(\mathrm{p}_1 \log_2(p_1) + \mathrm{p}_2 \log_2(p_2))$$
$$Gain(A) = Info(D) - Info_A(D)$$

Entropy:

$$\Longrightarrow Info(D) = -(0.6 \log_2(0.6) + 0.4 \log_2(0.4))$$
$$\Longrightarrow Info(D) = -((0.357)(-1.486) + (0.357)(-1.486))$$
$$\Longrightarrow Info(D) = 0.971$$

Temp:

$$info_{temp}(D) = \frac{3}{5} I(2,1) + \frac{2}{5} I(1,1) = 0.551 + 0.4 = 0.951$$

$$Gain(temp) = 0.02$$

Humidity:

$$info_{humidity}(D) = \frac{3}{5} I(2,1) + \frac{2}{5} I(1,1) = 0.551 + 0.4 = 0.951$$

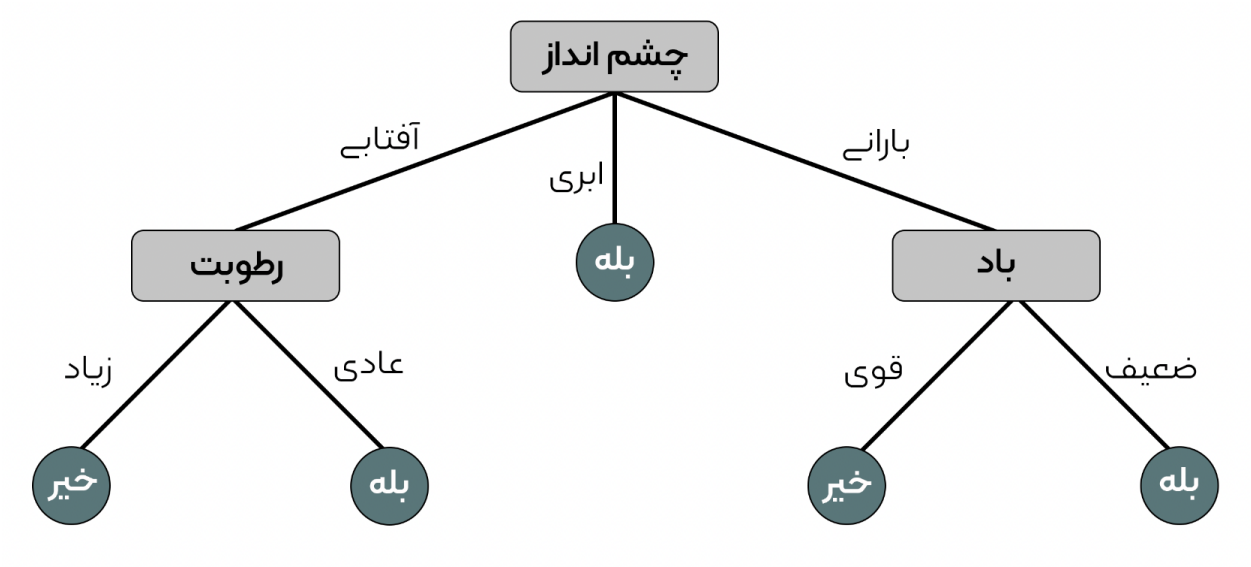
$$Gain(humidity) = 0.02$$

wind:

$$info_{wind}(D) = \frac{3}{5} I(3,0) + \frac{2}{5} I(2,0) = 0$$

$Gain(wind) = 0.971$

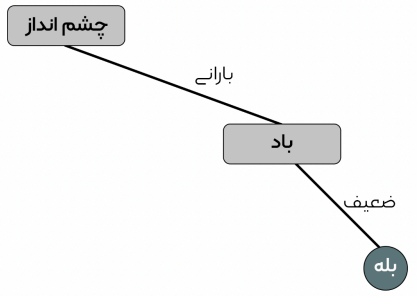
با توجه به information gain های به دست آمده از ویژگی باد برای سمت راست استفاده می‌شود.



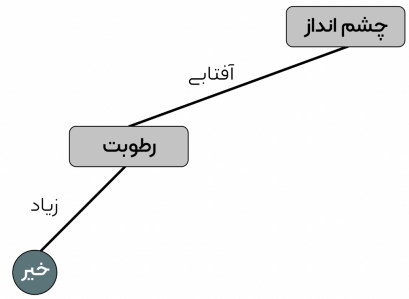
---

۵) برای هر دو نمونه باید از ویژگی‌ای که به عنوان ریشه‌ی اصلی در نظر گرفته شده است شروع شود.

برای نمونه‌ی اول از مسیر زیر به "بله" می‌رسیم



برای نمونه‌ی دوم از مسیر زیر به "خیر" می‌رسیم



---

۶) برای محاسبه‌ی صحت، تعداد نمونه‌هایی که درست پیش‌بینی می‌شوند بر تعداد نمونه‌هایی که پیش‌بینی می‌شوند تقسیم می‌شود بنابراین داریم:

$$Accuracy = \frac{1}{2} \times 100\% = 50\ \%$$

---

۶) با توجه به درخت تصمیم‌گیر ۵ قانون IF-THEN به دست می‌آید که در زیر آورده شده‌اند.

1. اگر چشم‌انداز آفتابی و رطوبت زیاد باشد، آنگاه بازی تنیس برگزار نمی‌شود.
2. اگر چشم‌انداز آفتابی و رطوبت عادی باشد، آنگاه بازی تنیس برگزار می‌شود.
3. اگر چشم‌انداز ابری باشد، آنگاه بازی تنیس برگزار می‌شود.
4. اگر چشم‌انداز بارانی و باد ضعیف باشد، آنگاه بازی تنیس برگزار می‌شود.
5. اگر چشم‌انداز بارانی و باد قوی باشد، آنگاه بازی تنیس برگزار نمی‌شود.

بخش عملی

قسمت اول)

ابتدا فایل فشرده شده‌ی dataset در google colab بارگذاری می‌شود. سپس این فایل از حالت فشرده خارج می‌شود و برای قسمت اول از question_1.csv یک دیتافریم ایجاد می‌شود. در ادامه برای چک کردن این مورد که تمامی داده‌های موجود حاوی لیبل هستند یا خیر از دستور dataframe1['class'].isnull().sum() استفاده می‌شود. نتیجه‌ی اجرای این خط کد نشان می‌دهد که تمامی نمونه‌ها لیبل خورده‌اند.

الف) برای به دست آوردن فرمت هر ویژگی از دستور dataframes1.dtypes استفاده می‌شود، نتیجه‌ی اجرای این دستور در زیر آورده شده است.

```
Unnamed: 0               int64
cap-diameter           float64
cap-shape               object
cap-surface             object
cap-color               object
does-bruise-or-bleed    object
gill-attachment         object
gill-spacing            object
gill-color              object
stem-height            float64
stem-width             float64
stem-root               object
stem-surface            object
stem-color              object
veil-type               object
veil-color              object
has-ring                object
ring-type               object
spore-print-color       object
habitat                 object
season                  object
class                   object
dtype: object
```

همچنین برای به دست آوردن مقادیر از دست رفته برای هر ویژگی از دستور dataframe1.isnull().sum() اجرا می‌شود. نتیجه‌ی اجرای این کد در زیر آورده شده است.

```
Unnamed: 0                  0
cap-diameter                0
cap-shape                   0
cap-surface             14120
cap-color                   0
does-bruise-or-bleed        0
gill-attachment          9884
gill-spacing            25063
gill-color                  0
stem-height                 0
stem-width                  0
stem-root               51538
stem-surface            38124
stem-color                  0
veil-type               57892
veil-color              53656
has-ring                    0
ring-type                2471
spore-print-color       54715
habitat                     0
season                      0
class                       0
dtype: int64
```

ب) ابتدا به کمک دستور dataframe1.shape[0] تعداد ردیف‌های موجود در دیتافریم مشخص می‌شود. بعد از اجرای این دستور مشخص می‌شود که تعداد داده‌ها در دیتافریم برابر ۶۱۰۶۹ می‌باشد. با توجه به مقادیر null در هر کدام از ویژگی‌ها، راهکار برای هر کدام توضیح داده می‌شود.

با توجه به مقادیر null موجود در هر ستون، از cap-surface شروع می‌کنیم. در این ستون ۱۴۱۲۰ ردیف null می‌باشد که حدود ۲۳ درصد کل دیتاست را شامل می‌شود. بنابراین حذف ردیف‌هایی که null هستند منجر به حذف اطلاعات مفیدی از دیتا می‌شوند. بنابراین مقادیر null موجود در ردیف‌ها باید با یک روش مشخص پر شود. برای انتخاب بهترین رویکرد از دستور dataframe1['cap-surface'].value_counts() استفاده می‌شود. نتیجه در زیر آورده شده است.

```
cap-surface
t    8196
s    7608
y    6341
h    4974
g    4724
d    4432
e    2584
k    2303
i    2225
w    2150
l    1412
Name: count, dtype: int64
```

برای پر کردن مقادیر خالی یک رویکرد پر کردن آن‌ها با مد است. با پر کردن مقادیر خالی با 't'، ۱۴۱۲۰ تا به ۸۱۹۶ اضافه می‌شود. از آن‌جایی که 't'، با مقادیر دیگر اختلاف فاحشی ندارد از این رویکرد استفاده نمی‌شود و مقادیر null به صورت رندوم از تمامی کاراکترهای موجود بر اساس فرکانس تکرار انتخاب می‌شوند. در نهایت پس از پر کردن مقادیر null به صورت نرمال، فرکانس مقادیر ستون cap-surface به شکل زیر خواهد بود.

```
cap-surface
t    10605
s     9983
y     8273
h     6451
g     6185
d     5714
e     3315
k     2994
i     2928
w     2786
l     1835
Name: count, dtype: int64
```

حال به سراغ پر کردن مقادیر ستون gill-attachment می‌رویم. در این ستون ۹۸۸۴ ردیف داریم که null می‌باشند که حدود ۱۶ درصد از دیتا می‌باشند. بنابراین حذف این ردیف‌ها منجر به از دست رفتن اطلاعات زیادی از دیتاست می‌شود. ابتدا به کمک دستور dataframe1['gill-attachment'].value_counts() فرکانس تکرار مقادیر در این ستون به دست می‌آید.

```
gill-attachment
a    12698
d    10247
x     7413
p     6001
e     5648
s     5648
f     3530
Name: count, dtype: int64
```

با توجه به نتیجه‌ی به دست آمده مانند ویژگی قبلی با این ویژگی برخورد می‌شود و از توزیع خود مقادیر برای پر کردن مقادیر null استفاده می‌شود. نتیجه به شکل زیر خواهد بود.

```
gill-attachment
a    15167
d    12265
x     8860
p     7129
e     6765
s     6692
f     4191
Name: count, dtype: int64
```

برای این دو ویژگی لازم است توضیحاتی ارائه شود. از آن جایی که مقادیر از دست رفته برای این دو ویژگی به اندازه‌ای بودند که پر کردن آن‌ها قابل توجیه بود، این عمل انجام شد. اما برای زمانی که مقادیر null در یک ویژگی به اندازه‌ای

باشد که پر کردن آن‌ها بر اساس توزیع مقادیر موجود هیچ توجیهی نداشته باشد و این مقادیر به قدری کم هستند که برای پر کردن مقادیر null قابل اتکا نباشند کل ویژگی حذف می‌شود.

ستون بعدی gill-spacing می‌باشد. مقادیر از دست رفته در این ویژگی برابر ۲۵۰۶۳ می‌باشد که حدود ۴۱ درصد دیتا می‌باشد. در نگاه اول این ویژگی کاندیدای حذف می‌باشد. اما چک می‌شود که با حذف این ویژگی مشکلی پیش نیاید. فرکانس مقادیر در این ویژگی به شکل زیر خواهد بود.

```
gill-spacing
c    24710
d     7766
f     3530
Name: count, dtype: int64
```

در این قسمت با توجه به فرکانس بالا مقدار c، چک می‌شود که هر کدام از این سه مقدار، منجر به دسته‌بندی قارچ‌ها در کدام بخش شده‌اند. در واقع با این کار می‌توانیم متوجه شویم که آیا مقادیر این ستون نقش تعیین کننده‌ای در مشخص کردن سمی بودن یا نبودن قارچ دارند یا خیر. در واقع هم‌بستگی آن با مشخص‌سازی کلاس چقدر است. نتیجه‌ی اجرای dataframe1.groupby(['gill-spacing', 'class']).size().unstack(fill_value=0) منجر به نتیجه‌ی زیر می‌شود.

class	e	p
gill-spacing		
c	10237	14473
d	4589	3177
f	1412	2118

با توجه به این نتیجه می‌توان متوجه شد که حذف ویژگی gill-spacing تاثیر چندانی بر روی دیتا ندارد، حتی گزینه‌ی بهتری می‌باشد. بنابراین این ویژگی حذف، یا اصطلاحا drop می‌شود.

ویژگی‌های stem-root، stem-surface، veil-type، veil-color و spore-print-color به ترتیب به اندازه‌ی ۸۴، ۶۲، ۸۹، ۸۷ و ۸۹ درصد دیتاست مقادیر از دست رفته دارند. بنابراین بهترین رویکرد برای این ویژگی‌ها حذف کامل آن‌ها است.

برای ring-type با توجه به آن‌که مقادیر از دست رفته تقریبا شامل ۴ درصد دیتا می‌باشد و این مقادیر از نوع categorical می‌باشد، استفاده از مد بهترین رویکرد برای پر کردن مقادیر از دست رفته می‌باشد. فرکانس مقادیر این ویژگی در زیر آورده شده است که نشان می‌دهد مد این ویژگی f می‌باشد.

```
ring-type
f    48361
e     2435
z     2118
l     1427
r     1399
p     1265
g     1240
m      353
Name: count, dtype: int64
```

پس از پر کردن مقادیر از دست رفته با مد، مقادیر این ویژگی به شکل زیر خواهد بود.

```
ring-type
f    50832
e     2435
z     2118
l     1427
r     1399
p     1265
g     1240
m      353
Name: count, dtype: int64
```

حال مقادیر از دست رفته برای هر ستون برابر با صفر می‌باشد.

```
Unnamed: 0              0
cap-diameter            0
cap-shape               0
cap-surface             0
cap-color               0
does-bruise-or-bleed    0
gill-attachment         0
gill-color              0
stem-height             0
stem-width              0
stem-color              0
has-ring                0
ring-type               0
habitat                 0
season                  0
class                   0
dtype: int64
```

ج) برای آن که بتوان ویژگی‌های غیر عددی را به ویژگی‌های عددی، با روش مناسب تبدیل کرد، نیاز است که برای هر ویژگی، مقادیر آن دیده شود تا بهترین روش انتخاب شود. در واقع بررسی ماهیت دیتا می‌تواند کمک زیادی کند. بنابراین هر کدام از ویژگی‌هایی که نوع مقادیر در آن‌ها categorical می‌باشد، به صورت جداگانه بررسی می‌شود.
همچنین باید در نظر داشت که این دیتا در نهایت با چه مدلی آموزش می‌بیند. از آن جایی که در ادامه‌ی سوال خواسته شده که به کمک KNN و Decision tree مدل آموزش ببینند، باید در نظر داشت که متدهای مختلف تبدیل مقادیر categorical به مقادیر عددی به عنوان ورودی مدل، چه تاثیری روی خروجی مدل خواهند داشت.
روش‌هایی که برای تبدیل دیتای categorical به دیتای عددی وجود دارد، Label encoding، One-hat encoding، Ordinal encoding و Binary encoding می‌باشد که بسته به ماهیت دیتا بهترین روش انتخاب می‌شود.

برای cap-shape

ابتدا مقادیر یونیک در این ویژگی استخراج می‌شود.

```
cap-shape
x    26934
f    13404
s     7164
b     5694
o     3460
p     2598
c     1815
Name: count, dtype: int64
```

با توجه به آن که شکل کلاه قارچ‌ها مقادیر اسمی هستند که ترتیبی نمی‌باشند و تعداد زیاد مقادیر یونیک در این ویژگی، استفاده از Binary encoder می‌تواند گزینه‌ی مناسبی باشد. اما در این قسمت از One-hot encoder نیز استفاده می‌شود و نتیجه‌ی آن با Binary encoder مقایسه می‌شود تا مشخص شود کدام encoder نتیجه‌ی بهتری خواهد داشت. دیتافریم‌های جداگانه‌ی مرتبط با این ویژگی جداگانه بررسی می‌شوند تا فضایی که در نهایت اشغال می‌کنند مقایسه شود.

در Binary encoder تعداد ستون‌هایی که به دیتافریم اضافه می‌شود برابر با $\log_2 N$ می‌باشد که N بیانگر تعداد مقادیر یونیک در ویژگی است. از آن‌جایی که این تعداد برای این ستون برابر با ۷ می‌باشد، تعداد ستون‌های ایجاد شده برابر با ۳ خواهد بود. همچنین ستون مربوط به ویژگی اصلی از دیتافریم حذف می‌شود. با اجرای دستور df_binary_encoded.info() نتیجه‌ی زیر حاصل می‌شود.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61069 entries, 0 to 61068
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            61069 non-null  int64
 1   cap-diameter          61069 non-null  float64
 2   cap-shape_0           61069 non-null  int64
 3   cap-shape_1           61069 non-null  int64
 4   cap-shape_2           61069 non-null  int64
 5   cap-surface           61069 non-null  object
 6   cap-color             61069 non-null  object
 7   does-bruise-or-bleed  61069 non-null  object
 8   gill-attachment       61069 non-null  object
 9   gill-color            61069 non-null  object
 10  stem-height           61069 non-null  float64
 11  stem-width            61069 non-null  float64
 12  stem-color            61069 non-null  object
 13  has-ring              61069 non-null  object
 14  ring-type             61069 non-null  object
 15  habitat               61069 non-null  object
 16  season                61069 non-null  object
 17  class                 61069 non-null  object
dtypes: float64(3), int64(4), object(11)
memory usage: 8.4+ MB
```

در One-hot encoder به ازای هر مقدار یونیک یک ستون اضافه می‌شود و مقدار ۰ یا ۱ بسته به این که در ویژگی اصلی کدام کاراکتر استفاده شده است به آن نسبت داده می‌شود. بنابراین ۷ ستون به آن اضافه می‌شود و ستون اصلی مربوط به ویژگی حذف می‌شود. بعد از اجرای دستور df_one_hot_encoded.info() نتیجه‌ی زیر به دست می‌آید.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61069 entries, 0 to 61068
Data columns (total 22 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   cap-shape_b           61069 non-null  object
 1   cap-shape_c           61069 non-null  object
 2   cap-shape_f           61069 non-null  object
 3   cap-shape_o           61069 non-null  object
 4   cap-shape_p           61069 non-null  object
 5   cap-shape_s           61069 non-null  object
 6   cap-shape_x           61069 non-null  object
 7   Unnamed: 0            61069 non-null  object
 8   cap-diameter          61069 non-null  object
 9   cap-surface           61069 non-null  object
 10  cap-color             61069 non-null  object
 11  does-bruise-or-bleed  61069 non-null  object
 12  gill-attachment       61069 non-null  object
 13  gill-color            61069 non-null  object
 14  stem-height           61069 non-null  object
 15  stem-width            61069 non-null  object
 16  stem-color            61069 non-null  object
 17  has-ring              61069 non-null  object
 18  ring-type             61069 non-null  object
 19  habitat               61069 non-null  object
 20  season                61069 non-null  object
 21  class                 61069 non-null  object
dtypes: object(22)
memory usage: 10.3+ MB
```

همانطور که از قبل مشخص بود، استفاده از Binary encoder فضای کمتری اشغال می‌کند. از طرفی باید در نظر داشت که این روش منجر به آن می‌شود که دیتاها حالت ترتیبی به خود بگیرند که تاثیر زیادی روی مدل‌های Decision tree و KNN ندارد.

نکته‌ی دیگری که حائز اهمیت است آن است که دو مدل Decision tree و KNN به طور کلی با Binary encoder نتیجه‌ی بهتری ارائه می‌دهند. از آن جایی که KNN بر پایه‌ی محاسبه‌ی فاصله است، افزایش Dimensionality باعث پراکندگی دیتا می‌شود که در نهایت منجر به آن می‌شود که مدل به سختی آموزش ببیند. همین موضوع برای Decision tree نیز برقرار است با این تفاوت که حساسیت این الگوریتم به انتخاب روش Encoding کم است.

با در نظر گرفتن همه‌ی این موارد، برای cap-shape از Binary encoder استفاده می‌شود.

در ادامه بر اساس مطالبی که در این قسمت ارائه شد Encoder مناسب انتخاب می‌شود.

برای بقیه‌ی Featureها

با توجه به مطالبی که اشاره شد، برای ویژگی‌هایی که مقادیر یونیک زیادی دارند از Binary encoder استفاده می‌شود. برای ویژگی‌هایی که مقادیر یونیک آن‌ها ۲ تا می‌باشد دیتا به صورت ۰ یا ۱ در دیتافریم ذخیره می‌شود تا فضای کمتری اشغال شود. اطلاعات دیتافریم نهایی به شکل زیر خواهد بود.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61069 entries, 0 to 61068
Data columns (total 40 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            61069 non-null  int64
 1   cap-diameter          61069 non-null  float64
 2   cap-shape_0           61069 non-null  int64
 3   cap-shape_1           61069 non-null  int64
 4   cap-shape_2           61069 non-null  int64
 5   cap-surface_0         61069 non-null  int64
 6   cap-surface_1         61069 non-null  int64
 7   cap-surface_2         61069 non-null  int64
 8   cap-surface_3         61069 non-null  int64
 9   cap-color_0           61069 non-null  int64
 10  cap-color_1           61069 non-null  int64
 11  cap-color_2           61069 non-null  int64
 12  cap-color_3           61069 non-null  int64
 13  does-bruise-or-bleed  61069 non-null  int64
 14  gill-attachment_0     61069 non-null  int64
 15  gill-attachment_1     61069 non-null  int64
 16  gill-attachment_2     61069 non-null  int64
 17  gill-color_0          61069 non-null  int64
 18  gill-color_1          61069 non-null  int64
 19  gill-color_2          61069 non-null  int64
 20  gill-color_3          61069 non-null  int64
 21  stem-height           61069 non-null  float64
 22  stem-width            61069 non-null  float64
 23  stem-color_0          61069 non-null  int64
 24  stem-color_1          61069 non-null  int64
 25  stem-color_2          61069 non-null  int64
 26  stem-color_3          61069 non-null  int64
 27  has-ring              61069 non-null  int64
 28  ring-type_0           61069 non-null  int64
 29  ring-type_1           61069 non-null  int64
 30  ring-type_2           61069 non-null  int64
 31  ring-type_3           61069 non-null  int64
 32  habitat_0             61069 non-null  int64
 33  habitat_1             61069 non-null  int64
 34  habitat_2             61069 non-null  int64
 35  habitat_3             61069 non-null  int64
 36  season_0              61069 non-null  int64
 37  season_1              61069 non-null  int64
 38  season_2              61069 non-null  int64
 39  class                 61069 non-null  int64
dtypes: float64(3), int64(37)
memory usage: 18.6 MB
```

د) از آن جایی که در قسمت‌های قبل از Binary encoder استفاده شد، برای نرمال کردن دیتا فقط باید ویژگی‌هایی که نوع داده در آن‌ها float است نرمال شوند. برای این که از چه روشی برای نرمال کردن دیتاها استفاده شود نیاز است که به ماهیت مدل‌ها و دیتا توجه شود. برای Decision tree تفاوت چندانی ندارد که از کدام روش استفاده کنیم. اما برای KNN، با توجه به ماهیت مدل که فاصله معیار اصلی آن است بهتر است از z-score normalization استفاده شود. به همین دلیل در این قسمت از z-score normalization برای داده‌های float استفاده می‌شود. نتیجه‌ی نرمال‌سازی ۵ دیتای اول دیتافریم در زیر آورده شده است. (توضیحات مربوط به z-score normalization در تمرین ۱ آورده شده است)

	cap-diameter	stem-height	stem-width
0	1.619462	3.076705	0.492293
1	1.873982	3.385311	0.601900
2	1.393432	3.328931	0.557061
3	1.412426	2.726555	0.381690
4	1.501699	2.952075	0.503254

۵) برای آن که محاسبه شود که دیتا Imbalance هست یا خیر، نسبت مقادیری که تعداد بیشتری دارد به مقادیری که تعداد کمتری دارد، محاسبه می‌شود. اگر این مقدار بیشتر از نسبت ۲ به ۱ باشد، دیتا Imbalance در نظر گرفته می‌شود. با پیاده‌سازی کد مربوط به این قسمت این نسبت برابر ۱.۲۴۷ به دست می‌آید که نشان می‌دهد به صورت کلی دیتاست Imbalance نمی‌باشد. اما برای این که بتوانیم دیتاست بالانس‌تری داشته باشیم، از الگوریتم SMOTE استفاده می‌شود. این الگوریتم لیبلی که تعداد کمتری دارد را در نظر می‌گیرد و تا زمانی که تعداد لیبل انتخابی به لیبل بیشتر نرسد sampling را ادامه می‌دهد. برای فرآیند sampling از منطق الگوریتم KNN استفاده می‌کند. با همین روند نسبت لیبل‌ها تبدیل به یک می‌شود.

الف) برای جداسازی دیتاست با نسبت ۸۰\۲۰، از کتابخانه‌ی scikit-learn و دستور train_test_split استفاده می‌شود. با مشخص سازی نسبت ذکر شده، تعداد داده‌های آموزش برابر با ۵۴۲۲۰ می‌باشد و بقیه‌ی دیتا به عنوان دیتای تست در نظر گرفته می‌شود.

ب) برای تعریف یک مدل درخت تصمیم‌گیر، از کتابخانه‌ی scikit-learn و DecisionTreeClassifier استفاده می‌شود.
جزئیات بیشتر در فایل کد وجود دارد. همچنین با استفاده از کتابخانه‌ی time زمان آموزش دیدن مدل محاسبه می‌شود. زمان آموزش مدل برابر با ۰.۳۵ ثانیه و زمان پیش‌بینی برابر با ۰.۰۰۶۹ ثانیه می‌باشد.
در نهایت با استفاده از accuracy_score از کتابخانه‌ی scikit-learn دقت پیش‌بینی محاسبه می‌شود که برابر با ۹۹.۸۴ درصد می‌باشد.

ج) برای تعریف یک مدل KNN، از کتابخانه‌ی scikit-learn و KNeighborsClassifier استفاده می‌شود.
جزئیات بیشتر در فایل کد وجود دارد. همچنین با استفاده از کتابخانه‌ی time زمان آموزش دیدن مدل محاسبه می‌شود. زمان آموزش مدل برابر با ۰.۰۲۱ ثانیه و زمان پیش‌بینی برابر با ۳.۶۷ ثانیه می‌باشد.

در نهایت با استفاده از accuracy_score از کتابخانه‌ی scikit-learn دقت پیش‌بینی محاسبه می‌شود که برابر با ۹۹.۹۱ درصد می‌باشد.

د) ابتدا به صورت کلی زمان پیش‌بینی و آموزش برای هر دو الگوریتم بررسی می‌شود. سپس بر اساس ویژگی هر کدام از الگوریتم‌ها زمان اجرای آموزش و پیش‌بینی به دست‌آمده بررسی می‌شود.

الگوریتم درخت تصمیم‌گیری:

زمان اجرای آموزش درخت تصمیم‌گیری با توجه به این که در هر مرحله می‌بایست معیارهای جداسازی برای هر ویژگی محاسبه شود نسبتا زیاد است. اما از طرفی با توجه به این که درخت تصمیم‌گیر در مرحله‌ی آموزش ساخته می‌شود، در مرحله‌ی پیش‌بینی نمونه‌های تست فقط در ریشه‌ی اصلی درخت قرار می‌گیرند و مسیر اصلی خود را پیدا می‌کنند. از طرفی این زمان پیش‌بینی می‌تواند بر اساس عمق درخت کمتر شود. هر چه عمق درخت کمتر باشد زمان اجرا نیز کمتر خواهد بود اما این تغییر می‌تواند روی صحت مدل تاثیر منفی بگذارد. بنابراین زمان اجرای آموزش بالا با زمان اجرای پیش‌بینی کم جبران می‌شود. کاربرد این الگوریتم در مواردی که نیاز به پیش‌بینی‌های زیاد و مکرر و real-time دارند می‌باشد.

الگوریتم KNN:

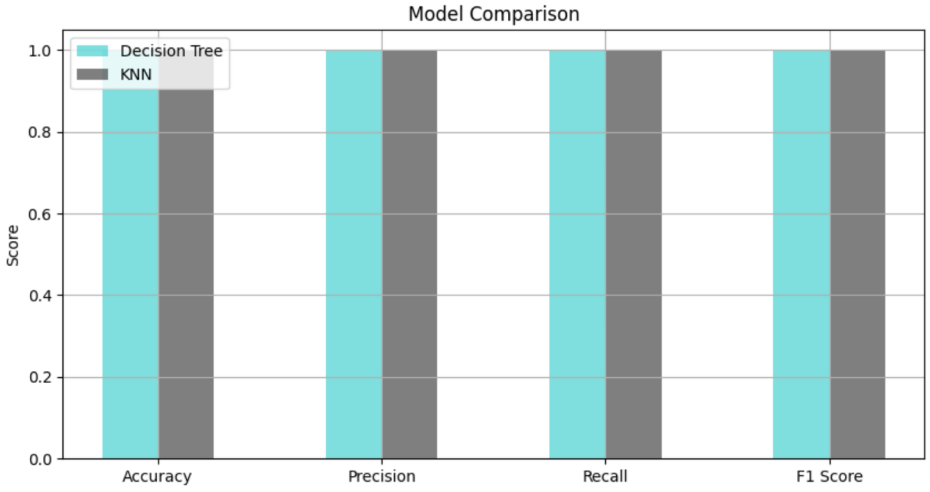
زمان اجرای آموزش KNN نسبتا کم می‌باشد. زیرا اساسا آموزشی در این مرحله انجام نمی‌شود و فقط زمان اجرای مربوط به ذخیره‌ی دیتاست می‌باشد. بنابراین زمان اجرای آموزش بسیار کم می‌باشد. اما قسمت اصلی این الگوریتم مربوط به زمان اجرای پیش‌بینی می‌باشد. این زمان زیاد است زیرا در این الگوریتم فاصله‌ی هر نمونه‌ی تست با هر نمونه در دسته‌ی آموزش محاسبه می‌شود و این ویژگی باعث افزایش زمان پیش‌بینی این الگوریتم می‌شود. این الگوریتم برای کاربردهایی مناسب است که دیتاست کوچک می‌باشد یا زمان پیش‌بینی اهمیت زیادی ندارد.

زمان‌های اجرایی که برای هر الگوریتم محاسبه شد در جدول زیر آورده شده است.

	Decision tree	KNN
زمان اجرای آموزش (ثانیه)	۰.۳۵	۰.۰۰۶۹
زمان اجرای پیش‌بینی (ثانیه)	۰.۰۲۱	۳.۶۷

ویژگی‌های هر دو الگوریتم منجر به تفاوت زمان اجرای آموزش و پیش‌بینی در یک دیتاست یکسان شده است.

ه) برای هر کدام از الگوریتم‌ها به کمک کتابخانه‌ی scikit-learn متریک‌های عنوان شده محاسبه می‌شود. سپس با استفاده از کتابخانه‌ی matplotlib متریک‌های هر الگوریتم در کنار متریک‌های یکسان الگوریتم در یک نمودار میله‌ای رسم می‌شود. شکل زیر نتیجه‌ی این رسم می‌باشد.

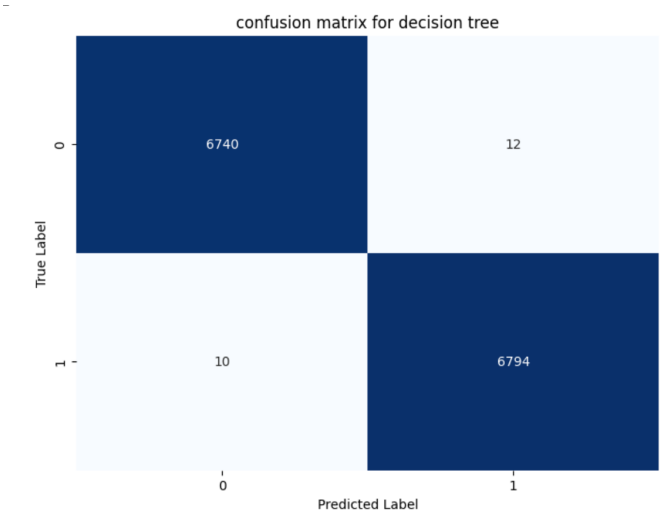


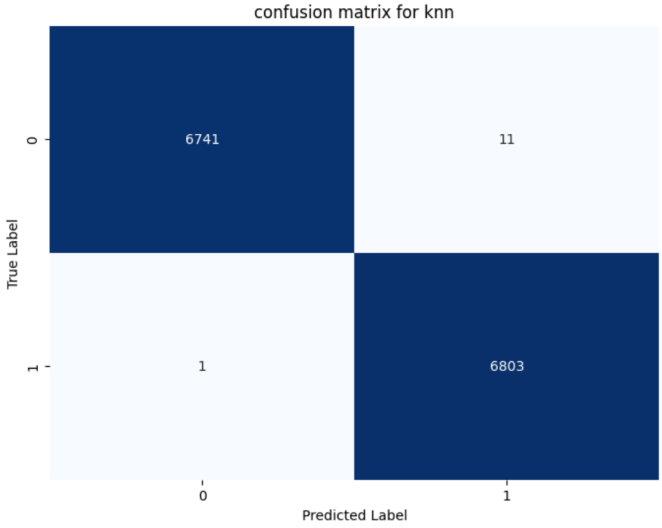
از روی نمودار نمی‌توان با دقت بالایی الگوریتم بهتر را انتخاب کرد. به همین منظور این متریک‌ها برای هر دو الگوریتم در یک دیتافریم ثبت می شوند تا از نظر عددی مقایسه شوند. دیتافریم اشاره شده به شکل زیر خواهد بود.

	model	accuracy	precision	recall	f1 score
0	Decision tree	0.998377	0.998378	0.998377	0.998377
1	KNN	0.999115	0.999119	0.999112	0.999115

با توجه به مقادیر فوق الگوریتم KNN انتخاب مناسب‌تری می‌باشد زیرا متریک‌های بالاتری دارد. (بدون در نظر گرفتن زمان اجرا و کاربرد هر الگوریتم در موارد متفاوت)

و) به کمک کتابخانه‌های scikit-learn، seaborn و matplotlib ماتریس آشفتگی برای هر الگوریتم رسم می‌شود. دو شکل زیر به ترتیب ماتریس آشفتگی برای Decision tree و KNN را نشان می‌دهند.

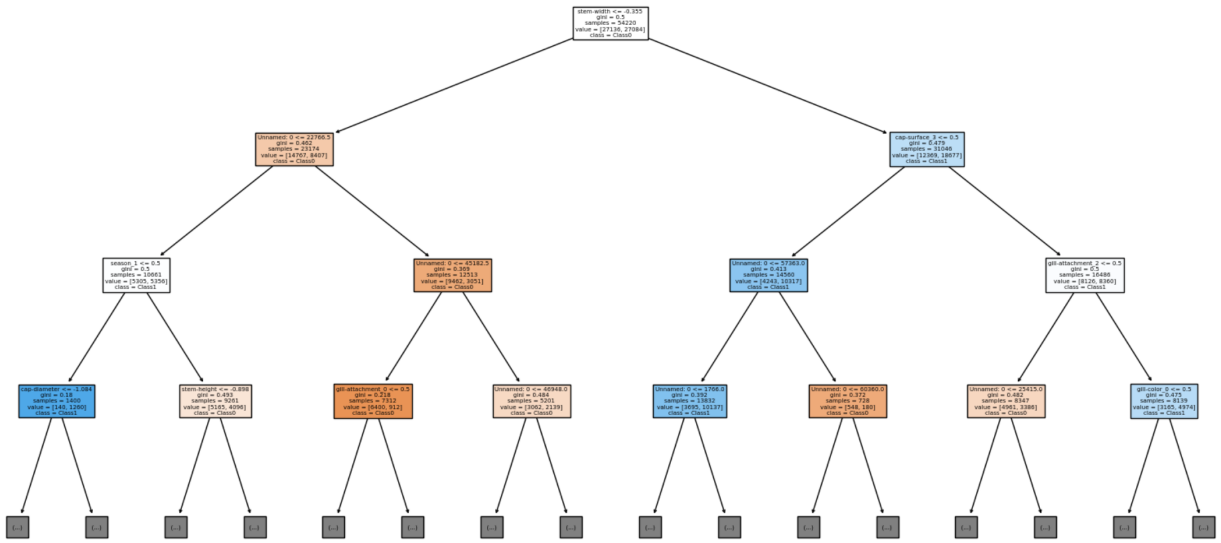




با توجه به ماتریس‌های آشفتگی از دو الگوریتم می‌توان متوجه شد که مقادیر زیاد در قطر اصلی هر دو ماتریس نشان دهنده‌ی عملکرد خوب این دو مدل است. این دو مقدار به ترتیب true positives و true negatives می‌باشند. از طرفی قطر غیر اصلی نشان‌دهنده‌ی false positives و false negatives می‌باشند. هر چه مقادیر این قطر کمتر باشد، نشان از عملکرد مناسب مدل می‌باشد.

این ماتریس شهودی از مدل ارور که مدل می‌تواند داشته باشد ایجاد می‌کند. کاربردهای متنوع مشخص می‌کند که کدام قسمت این ماتریس از اهمیت بالایی برخوردار است. در این دیتاست با توجه به این که سمی بودن یا نبودن قارچ هدف مدل است، تعداد false negatives بیشتر در مدل درخت تصمیم‌گیر باعث می‌شود KNN انتخاب بهتری باشد. در واقع الگوریتم درخت تصمیم‌گیر به اشتباه قارچ‌ها را سالم پیش‌بینی می‌کند که نکته‌ی منفی‌ای است.

ز) برای رسم درخت تصمیم‌گیری در این قسمت از دو روش استفاده شده است. روش اول استفاده از کتابخانه‌ی scikit-learn است که درخت تصمیم‌گیری را کامل و با جزئیات دقیق نشان نمی‌دهد. نتیجه‌ی این نمودار در زیر آورده شده است.



برای داشتن درخت تصمیم‌گیر به صورت کامل از کتابخانه‌ی graphviz استفاده می‌شود. که نتیجه‌ی آن به صورت فایل pdf ضمیمه‌ی فایل ارسالی شده است.

از آنجایی که در هر مرحله از درخت تصمیم‌گیر پارامترهای متفاوتی محاسبه می‌شوند تا زیردرخت‌های دیگر رسم شود تا در نهایت به تارگت برسیم، تفسیر این درخت امکان‌پذیر است. فرآیند تصمیم برای هر نمونه از ویژگی‌ای که با بیشترین information gain در ریشه‌ی اصلی قرار دارد شروع می‌شود و به یک زیردرخت دیگر منتقل می‌شود تا در نهایت به تارگت برسیم.

برای تفسیرپذیری بیشتر این درخت، مشخص کردن max-depth می‌تواند موثر باشد. این روش منجر به آن می‌شود که درخت پیچیدگی کمتری داشته باشد و شهود بهتری ارائه دهد. هزینه‌ای که برای این روش وجود دارد کاهش عملکرد مدل است.

روش دیگر آن است که حذف کردن یا pruning است. در واقع در این روش از ویژگی‌هایی استفاده می‌شود که شامل اطلاعات مفید و تاثیرگذار بیشتری هستند. هزینه‌ی احتمالی برای این روش ایجاد بایاس است که ممکن است برای دیتاست‌های دیگر عملکرد خوبی نداشته باشد.

قسمت دوم)

الف) به منظور ساختن دیتاستی با توجه به خواسته‌ی این بخش از مسئله که دیتا به توالی زمانی تبدیل شود، در کد یک تابع با نام create_time_sequenced_data پیاده‌سازی می‌شود که دیتاست و سایز ویژگی‌ها را به عنوان ورودی می‌گیرد. این تابع روی تمامی نمونه‌های دیتاست پنجره‌های زمانی با طول ۱۰ ایجاد می‌کند که ویژگی‌های دیتافریم را می‌سازند و نمونه‌ی ۱۱م را به عنوان تارگت در نظر می‌گیرد. در نهایت خروجی این تابع ویژگی‌ها و تارگت‌ها می‌باشند که در ادامه در قالب دیتافریم از آن‌ها استفاده می‌شود تا برای آنالیز راحت‌تر مورد استفاده قرار گیرند. اندازه‌ی ویژگی‌های دیتاست جدید یک دیتافریم ۳۶۴۰ می‌باشد که هر نمونه شامل ۱۰ ویژگی می‌باشد. مقادیر تارگت نیز برابر با ۳۶۴۰ می‌باشند که دمای تارگت می‌باشند. جزئیات بیشتر کد در فایل ارسالی ضمیمه شده است.

ب) برای تقسیم دیتاست به دو بخش آموزش و تست با نسبت ۸۰\۲۰ همانند قسمت اول از scikit-learn استفاده می‌شود. در نتیجه‌ی این تقسیم، ۲۹۱۲ دیتا در دیتاست آموزش وجود دارد.

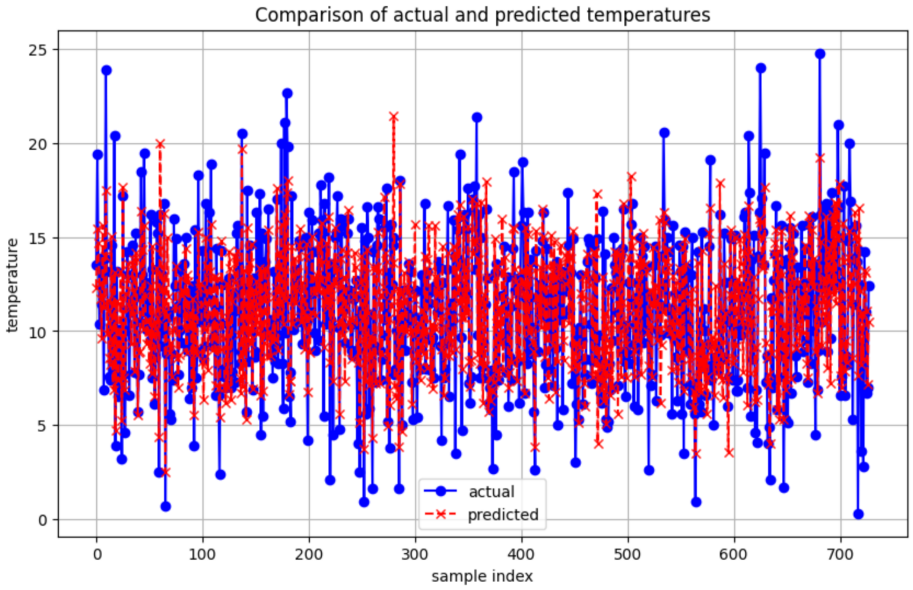
ج) با کمک کتابخانه‌ی scikit-learn یک مدل رگرسیون روی دیتاست آموزش فیت می‌شود تا بر اساس این دیتاست ترین شود. نتایج زیر وزن‌ها و بایاس مدل ترین شده را نشان می‌دهد.

```
Coefficients: [ 0.04730306  0.03824898  0.0468321   0.04071717  0.06068967  0.05703717
  0.03463836  0.08232561 -0.0966962   0.60654376]
Intercept: 0.9275698762994473
```

د) با توجه به مدلی که در بخش قبل به دست آمد، دیتاست مربوط به تست پیش‌بینی می‌شود. سپس به کمک کتابخانه‌ی scikit-learn، مقادیر RMSE و MAE محاسبه می‌شود. نتیجه‌ی این محاسبه در زیر آورده شده است.

```
Root Mean Squared Error: 2.4113535227841543
Mean Absolute Error: 1.9076905202696846
```

ه) به کمک کتابخانه‌ی matplotlib نمودار زیر برای مقایسه‌ی دیتای واقعی با دیتای پیش‌بینی شده رسم می‌شود.



همانطور که از نمودار بالا برمی‌آید، هم دمای واقعی و هم دمای پیش‌بینی‌شده تا حد خوبی به یکدیگر نزدیک هستند و یک روند را طی می‌کنند. این نشان می‌دهد که مدل رگرسیون خطی که در دو بخش قبل آموزش دیده، به طور کلی الگو یا روند تغییرات دما را در طول زمان درست ثبت می‌کند. اما از طرفی در حالی که دماهای پیش‌بینی شده تمایل دارند از دمای واقعی پیروی کنند، انحرافات قابل مشاهده‌ای وجود دارد که در آن مقادیر پیش‌بینی شده با مقادیر واقعی مطابقت ندارند. این اختلافات می‌تواند نشان‌دهنده خطاهای پیش‌بینی باشد.

همچنین از این نمودار می‌توان متوجه شد که مقادیر در برخی از مناطق نمودار، به ویژه در اطراف مقادیر مرکزی محدوده دما، مطابقت خوبی دارند. از این رو می‌توان متوجه شد که این مدل در این محدوده بر خلاف دیتاها با دمای زیاد، دقت خوبی دارد. به طور کلی این نمودار نشان می‌دهد که مدل می‌تواند در دماهایی که از میانگین دما فاصله دارند، بهتر عمل کند.

و) برای انجام این بخش، آخرین دما از ۱۰ روز که به عنوان ویژگی در نظر گرفته شده‌اند انتخاب می‌شود و با دمای روز بعد خود مقایسه می‌شود. بعد از این مقایسه اگر از آن بزرگتر بود لیبل جدید ۰ و اگر کوچکتر بود لیبل جدید ۱ می‌باشد. شکل زیر ۵ دیتای اول آن را نشان می‌دهد.

	Label (Day 11 Temp)	New Label
0	16.2	0
1	13.3	0
2	16.7	1
3	21.5	1
4	25.0	1

ز) با استفاده از کتابخانه‌ی scikit-learn یک مدل SVM با linear kernel طراحی می‌شود که با اجرای دستور svm_model.fit() روی دیتاست آموزش، اجرا می‌شود. جزئیات کد در فایل ضمیمه شده قرار دارد.

ح) با توجه به مدلی که در بخش قبل ارائه شد، متریک‌های ارزیابی مدل که در صورت سوال به آن‌ها اشاره شده است بعد از پیش‌بینی مدل از دیتاست تست، محاسبه می‌شوند. نتیجه‌ی این محاسبات در زیر آورده شده است.

```
Accuracy: 0.6428571428571429
Precision: 0.6675062972292192
Recall: 0.6743002544529262
F1 Score: 0.6708860759493671
```

ط) از آنجایی که مدل رگرسیون خطی مقدار عددی دما در روز ۱۱م را پیش‌بینی می‌کند و SVM افزایش یا کاهش دما در روز ۱۱م را نسبت به روز آخر پیش‌بینی می‌کند (مقادیر باینری)، بنابراین یک روش آن است که برای مقایسه‌ی این دو مدل باید پیش‌بینی مدل رگرسیون خطی را باینری کرد تا بتوان مقایسه را انجام داد.
به همین منظور مقدار دمای پیش‌بینی شده در روز ۱۱م، با دمای روز آخر مقایسه می‌شود. اگر تفاضل این دو از مقدار ۰ (همان حد آستانه‌ای) بیشتر بود به معنای آن است که دما افزایش داشته است یا به عبارتی طبق SVM این لیبل آن یک است. سپس مقایسه می‌شود که چه تعداد از این پیش‌بینی‌ها با پیش‌بینی‌های SVM یکسان است. نحوه‌ی پیاده سازی کد این قسمت در فایل ارسالی ضمیمه شده است. نتیجه‌ی اجرای این کد تناظر دو مدل را با عدد ۰.۹۴ نشان می‌دهد.