دانشگاه صنعتی امیرکبیر
(پلیتکنیک تهران)
دانشکده مهندسی کامپیوتر

درس بیوانفورماتیک
پروژه پایانی

امیرمهدی زرین‌نژاد
۹۷۳۱۰۸۷

نیم‌سال دوم ۱۴۰۰

گزارش کامل همراه با کد در نوت بوک نیز نوشته‌شده و ضمیمه شده‌است.

- **ابتدا بخش پیش پردازش را داریم:**

در قدم اول توابع مربوط به kmer را بررسی می‌کنیم که جلوتر از آن‌ها استفاده می‌شود:

```
[58] def kmer_for_one_sequence(seq, k):
0s       number_of_windows = len(seq) - k + 1
         kmers = {}

         alphabet = ['A', 'C', 'G', 'T']
         products = [''.join(p) for p in itertools.product(alphabet, repeat=k)]
         for product in products:
             record = ''.join(product)
             kmers[record] = 0
         # print(kmers)

         for i in range(number_of_windows):
             current_window = seq[i:i + k]
             # print('window:', current_window)
             kmers[current_window] += 1

         if number_of_windows:
             for record in kmers:
                 kmers[record] = kmers[record] / number_of_windows
         return kmers
```

این تابع برای اعمال به یک رشته پیاده‌سازی شده است.

حلقه for اول:

به این صورت کار می‌کند که ابتدا k یا همان سایز پنجره را همراه با رشته مورد نظر دریافت می‌کند؛ با توجه با الفبایی که برای رشته‌ها داریم یک دیکشنری از تمام حالت‌هایk تایی الفبا ایجاد می‌کند و این دیکشنری را با صفر مقداردهی اولیه می‌کند.

حلقه for دوم:

سپس در رشته پیمایش می‌کند و در هر مرحله پنجرهk تایی موجود و پیش‌رو را درنظر می‌گیرد. و برای این پنجره در دیکشنری،۱ واحد اضافه می‌کند(در واقع با این کار تعداد هرکدام از پنجره‌های ممکن را در رشته‌ی داده‌شده می‌شمارد(

حلقه for سوم:

نهایتا در حلقه سوم مقادیر دیکشنری، تقسیم بر تعداد کل پنجره‌های ممکن می‌شوند و به این صورت نرمال‌سازی می‌شوند و اثر طول خنثی می‌شود.

```
[59] def kmer(k, data):
        for index, row in data.iterrows():
            if len(row) > 1:
              updated_data = kmer_for_one_sequence(row[1], k)
              # row[1] = np.array(list(updated_data.values()))
              row[1] = list(updated_data.values())
            else:
              updated_data = kmer_for_one_sequence(row[0], k)
              row[0] = list(updated_data.values())
```

این تابع kmer را بر روی مجموعه‌ای از رشته‌ها اعمال می‌کند. (درواقع از تابع قبلی استفاده می‌کند و آن را به مجموعه‌ای از رشته‌ها اعمال می‌کند)

همانطور که در کد مشخص است بر روی مجموعه داده ورودی پیمایش میکند و index و row را برای هر سر استخراج می‌کند و رشته را از داخل row بدست می‌آورد.

و تابع kmer_for_one_sequence را برروی هر رشته اعمال می‌کند و آن رشته‌را با برداری که از تابع قبلی بدست می‌آید جایگزین می‌کند

(چک می‌کند که:

اگر طول row بیش‌تر از ۱ بود: داده‌ی آموزشی است که هم Type دارد و همSequence

درغیر این صورت: داده‌ی تست که فقط Sequence دارد.

به این صورت رشته را پیدا می‌کند)

نهایتا رشته‌های ما با یکسری بردار عددی نرمال‌شده، جایگزین می‌شوند.

حال که این توابع را تعریف کردیم، از آن‌ها همراه با یکسری تکنیک‌های دیگر استفاده می‌کنیم و بر مجموعه‌داده‌هایمان اعمال می‌کنیم تا به فرمت مناسب برای ارائه به مدل و پردازش در بیایند:

داده‌ها را می‌خوانیم:

```
import pandas as pd
# import seaborn as sns
# import matplotlib.pyplot as plt

train_set = pd.read_csv("training_set.csv")
train_set
```

برچسب‌ها را با مقادیر عددیشان جایگزین می‌کنیم تا بتوان در مدل از آن‌ها استفاده کرد:

```
train_set.iloc[:, 0:1] = train_set.iloc[:, 0:1].replace('Class1', 1)
train_set.iloc[:, 0:1] = train_set.iloc[:, 0:1].replace('Class2', 2)
train_set.iloc[:, 0:1] = train_set.iloc[:, 0:1].replace('Class3', 3)
train_set.iloc[:, 0:1] = train_set.iloc[:, 0:1].replace('Class4', 4)
train_set.iloc[:, 0:1] = train_set.iloc[:, 0:1].replace('Class5', 5)
train_set.iloc[:, 0:1] = train_set.iloc[:, 0:1].replace('Class6', 6)

train_set
```

	Type	Sequence
0	5	TACCACCTACGCTGACAATGGATGTTATTGTACCCGATTCGAATTA...

و kmer را اعمال می‌کنیم:

kmer را برروی رشته‌های مجموعه‌داده‌مان اعمال می‌کنیم و به این صورت بردارهای kmer متناظرشان را جایگزین می‌کنیم.

در بخش‌های مختلف کد از k=۲ برای پیاده‌سازی kmer استفاده شده. این مقدار از تست مقادیر مختلف بدست آمده و نتیجه شده. درواقع مدل شبکه عصبی که ایجاد کردیم با چند مقدار مختلف k نتیجه مناسب را می‌دهد اما k=۲ برای مدل mlp ما از بقیه بهتر بود. درواقع این برتری زمانی مشخص شد که یک شبکه عصبی ساده با تعداد پارامترها و نورون‌های کم داشتیم. در تست‌های مکرر حالت‌های مختلف و مقادیر مختلف k مشاهده شد که در یک شبکه عصبی سبک با افزایش k درصد خطا برروی داده‌های develop افزایش می‌یابد . اما با k=۲ شبکه ی ساده‌ی ما می‌توانست درصد زیادی از داده‌های ارزیابی را درست تشخیص دهد و فقط تعداد کمی را با تفاوت تنها ۱واحد متفاوت تشخیص می‌داد. اما با افزایش k مقدار خطا بیش‌تر و بزرگ‌تر می‌شد. (علت این امر هم این است که پنجره های کوچک‌تر زیربخش‌های بیش‌تری از رشته‌ها را بررسی می‌کنند و به عبارتی جزئی تر رشته را بررسی می‌کنند. این مطلب را می‌توان با درکنار هم قرار دادن یک k خیلی کوچک و یک k خیلی بزرگ به خوبی درک کرد)

```
kmer(k=2, data=train_set)

train_set
```

	Type	Sequence
0	5	[0.07471585615800261, 0.04713992919694429, 0.0...
1	5	[0.0692064083457526, 0.043964232488822655, 0.0...
2	3	[0.03645266594124048, 0.06641050054406965, 0.0...
3	6	[0.0641025641025641, 0.05042735042735043, 0.03...

سپس مجموعه‌داده و برچسب‌های آموزشی را در پارامترهای train_data و train_label میریزیم. و نهایتا داده‌ها و برچسب را به صورت لیست درمی‌آوریم تا در مدل بتوانیم استفاده کنیم.

```
train_data = train_set.iloc[:, 1:2]
train_labels = train_set.iloc[:, 0:1]

train_data
```

	Sequence
0	[0.07471585615800261, 0.04713992919694429, 0.0...
1	[0.0692064083457526, 0.043964232488822655, 0.0...

```
[71] train_data = list(train_data['Sequence'])
     train_labels = list(train_labels['Type'])

     print(train_data)
     print(train_labels)

     [[0.07471585615800261, 0.04713992919694429, 0.04574250046580958, 0.09288242966275387, 0.07881498043599776, 0.050307434
     [5, 5, 3, 6, 2, 3, 3, 1, 4, 3, 1, 1, 2, 4, 1, 2, 3, 1, 5, 2, 2, 1, 3, 5, 4, 5, 4, 3, 4, 5, 5, 4, 5, 4, 5, 5, 1, 1, 6,
```

همین‌کارها و مراحل پیش‌پردازش را برای مجموعه داده‌های develop و test هم انجام می‌دهیم تا قابل استفاده، پردازش و تحلیل بشوند.

بعد از اعمال پیش‌پردازش بر مجموعه‌داده‌ها گام بعدی فرا می‌رسد:

- **ایجاد مدل، پردازش، یادگیری، ارزیابی و تست**

ما از MLP استفاده کردیم و مدل را ایجاد کردیم:

ابتدا یک مدل دسته‌بندی MLP ایجاد می‌کنیم. این مدل ۳لایه پنهان دارد که هرکدام ۶۴ نورون دارند. از تابع فعال‌ساز relu استفاده شده‌است که غشبکه مارا غیرخطی می‌کند.رندم استیت هسته‌ی شروع و مقداردهی رندم اولیه پارامترها را تعیین می‌کند و مکس ایتر هم حداکثر تعداد پیمایش‌ها را تعیین می‌کند.

این مقادیر پارامترها باتوجه به توزیع داده‌ها خوب عمل می‌کنند و با تست و تحلیل محاسبه شده‌اند. برای برخی از پارامترها مقدار اتیمم وجود دارد به این صورت که بیش‌تر یا کم‌تر از یک مقدار خاص، عملکرد مدل کاهشی می‌شود و ما با پیدا کردن آن نقطه خاص می‌توانیم عملکرد مناسب مدل را بدست بیاوریم.

در طراحی مدل باتوجه به کم بودن داده‌های آموزش از earlystopping می‌توانیم استفاده کنیم تا از بیش‌برازش جلوگیری شود.

نهایتا دادهای آموزشی همراه با برچسب‌هایشان در مدل فیت شدند و مدل با آن‌ها آموزش می‌بینند.

```
from keras.backend import dropout
clf = MLPClassifier(hidden_layer_sizes=(64, 64, 64),activation="relu" ,random_state=1, early_stopping=True)

clf.fit(train_data, train_labels)

MLPClassifier(early_stopping=True, hidden_layer_sizes=(64, 64, 64),
              random_state=1)
```

حال داده‌های develop را به مدل می‌دهیم تا پیش‌بینی کند و نتایج را نشان می‌دهیم.

```
[81] development_predicts = clf.predict(development_data)

     print(development_predicts)

     [5 2 2 6 3 6 3 5 1 6 5 6 6 2 6 5 2 5 5 3 4 2 4 6 3 2 3 5 4 5 4 3 5 2 1 1 5
      2 2 5 6 1 6 1 1 3 5 5 5 4 1 6 1 3 1 6 6 1 3 4 2 4 2 2 4 6 1 1 3 1 3 1 3 1
      5 1 5 2 4 4 2 4 1 2 6 6 2 3 6 5 6 4 4 6 4 2 4 5 5 3 1 3 5 2 5 4 3 2 6 3 1
      3 6 6 1 2 3 3 4 2 5 4 2 3 4 2 1 5 4 6 4 4 4 6 1 4 3 2 6 2 1 2 3 2 3 4 3 6
      5 6 4 4 5 1 5 4 3 1 6 4 3 6 5 5 5 5 3 1 2 4 3 3 1 2 6 2 6 1 1 1]
```

سپس تفاوت نتایج پیش‌بینی شده‌ی مدل را با مقادیر واقعی نشان می‌دهیم که برای همه داده‌های develop برابر صفر است و یعنی مدل عملکرد ۱۰۰ درصدی از خود نشان داده‌است.

این امتیاز هم در همین بخش با r۲_score محاسبه و نمایش داده‌شده است.

```
[82] print(development_predicts - development_labels)
     print("\nThe Score with ", (r2_score(development_predicts, development_labels)))

     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

     The Score with  1.0
```

همین‌کار را برای داده‌های تست انجام می‌دهیم:

```
from keras.backend import dropout
clf = MLPClassifier(hidden_layer_sizes=(64, 64, 64),activation="relu" ,random_state=1, early_stopping=True)

clf.fit(train_data, train_labels)

MLPClassifier(early_stopping=True, hidden_layer_sizes=(64, 64, 64),
              random_state=1)
```

```
[110] test_predicts = clf.predict(test_data)

      print("answers =\n", (test_predicts))

      answers =
       [5 5 3 1 6 1 6 3 4 1 4 3 2 4 4 3 6 3 3 3 4 1 5 6 1 5 3 3 3 3 1 3 6 1 6 1 3
       5 5 6 5 4 5 5 3 6 4 5 1 2 4 3 4 3 3 6 1 6 5 3 1 3 3 3 3 6 4 5 6 6 6 3 2 6
       1 1 4 6 6 2 5 4 1 1 3 5 6 4 3 5 3 1 1 6 5 3 5 1 4 1 3 3 2 1 3 4 3 3 3 3 3
       1 4 1 4 4 3 3 3 4 6 1 6 4 5 3 6 2 1 3 3 6 5 2 2 3 2 1 3 3 5 1 6 1 2 2 1 3
       1 1 6 6 2 6 3 5 5 3 6 3 1 2 4 4 3 3 5 4 1 1 2 3 6 1 3 2 6 4 5 6 5 6 5 5 6
       3 3 3 6 4 6 3 3 6 6 6 1 1 5 1 1 3 3 1 1 1 5 3 2 2 1 1 1 1 3 6 3 3 4 5 6 4
       1 4 3 4 6 5 4 6 5 5 6 6 4 5 3 6 6 3 1 1 4 4 4 5 1 2 6 1 6 3 2 2 1 2 3 6 6
       5 1 3 6 1 3 3 1 1 5 4 1 3 1 2 1 1 3 3 3 6 1 6 4 1 3 4 3 3 4 3 1 3 6 6 2 1
       6 2 1 6 6 1 4 1 6 5 6 3 4 2 4 3 1 6 5 5 2 3 5 4 5 1 3 3 4 1 1 1 1 2 1 5 2
       3 3 4 2 5 1 5 4 1 1 3 3 2 5 4 4 3 2 5 1 2 1 2 1 2 2 3 5 6 1 6 3 1 6 4 1 1
       1 2 3 5 6 3 1 1 4 2 1 6 3 6 3 4 2 6 6 3 3 2 2 6 3 1 3 2 5 1]
```

با توجه به نتایجی که از quera بدست آمد می‌بینیم مدل ما عملکرد بسیار بالا و تقریبا ۱۰۰ درصدی دارد که نشان می‌دهد پیش پردازش، نرمال‌سازی، ساخت و استفاده از مدل به خوبی صورت گرفته است.

باتشکر از توجه شما.

امیرمهدی زرین‌نژاد