

به نام خدا



دانشگاه تهران

پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



درس تحلیل ها و سیستم های داده های حجیم

پروژه شماره یک
فاز اول

نام و نام خانوادگی :

سحر رجبی

شماره دانشجویی :

۸۱۰۱۹۹۱۶۵

خرداد ۱۴۰۱

فهرست گزارش سوالات (لطفاً پس از تکمیل گزارش، این فهرست را بهروز کنید.)

3.....سوال ۱-

11.....سوال ۲-

21.....سوال ۳-

سوال ۱ -

برای این بخش، از کتاب pride and prejudice از سایت Gutenberg استفاده شده است. این کتاب بیش از ۵۰ هزار کلمه دارد. با استفاده از `wc -c` می‌توانیم تعداد کلمات را تست کنیم که در این مورد تعداد برابر با ۷۹۶۸۰۲ کلمه بود.

برای اجرای کدهای این بخش، کدهای پایتون را در سرور با دستور زیر اجرا کرده‌ایم:

```
Spark-submit --executor-memory 4GB --total-executor-cores <total num of cores> --executor-cores <num of cores in single machine> <file-name>
```

که `total-executor-cores` مجموع کورهای قابل استفاده در تمام کلاستر را مشخص می‌کند و `executor-cores` کورهای قابل استفاده در یک ماشین را مشخص می‌کند.

(a) بخش

برای پیش‌پردازش متن‌ها، کارهای زیر صورت گرفت:

۱- کلماتی مثل `Mr.`, `Ms.` و `Mrs.` با `Mr.`, `Ms.` و `Mrs.` جایجا شدند تا در هنگام شکستن به جملات، این نقاط مشکلی ایجاد نکنند.

۲- حرف "s" مالکیت در کلمات باید حذف می‌شد تا بعد از حذف `punctuation`‌ها، s به کلمات ما نچسبد.

۳- تمامی `punctuation`‌ها به جز علائمی که برای پایان جمله استفاده می‌شود، یعنی «.»، «!» و «?» حذف شدند.

۴- چندین `space` متوالی را تبدیل به یک `space` کردیم (پترن `regex` آن در تصویر زیر مشخص است).

```
data = data.withColumn('value', regexp_replace(col('value'), ' +', ' '))
```

۵- کarakتر `space` پایان یک جمله، بین علامت نگارشی آخر جمله و کلمه‌ی بعد می‌آید را حذف کردیم. (پترن `regex` آن در تصویر زیر مشخص است)

```
data = data.withColumn('value', regexp_replace(col('value'), '(\. ! | \?) ', '$1'))
```

۶- تمامی فضاهای خالی ابتدای جملات را حذف کردیم. (پترن `regex` آن در تصویر زیر مشخص است).

```
data = data.withColumn('value', regexp_replace(col('value'), '^\\s+', ''))
```

۷- در نهایت تمامی کarakترهایی که جز حروف، فاصله و علائم انتهای جمله بودند را هم حذف کردیم. (پترن `regex` آن در تصویر زیر مشخص است).

```
data = data.withColumn('value', regexp_replace(col('value'), '^[A-Za-z\\.\\?! ]', ''))3
```

- ۸- تمامی کاراکترها را lower-case کردیم.
- ۹- عنوان فصل‌ها که به صورت chapter <digit> بود را حذف کردیم. (پتن regex آن در تصویر زیر قابل مشاهده است).

```
data = data.withColumn('value', regexp_replace(col('value'), 'chapter \d+', ''))
```

برای اعمال این پردازش‌ها، از تابع withColumn برای اعمال تغییرات بر روی یک Column استفاده کردیم و با تابع regexp_replace یک ستون از داده را (با استفاده از تابع col) به آن دادیم و مشخص کردیم چه پتن regex را با چه چیزی جابجا کند.

بعد از اعمال این پیش‌پردازش‌ها، می‌توانیم به سراغ خواسته‌های سوال برویم.

۱۰ خط اول کتاب بعد از اعمال پیش‌پردازش‌ها به صورت زیر است:

value
it is a truth uni...
however little kn...
my dear mr bennet...
mr bennet replied...
but it is returne...
mr bennet made no...
do not you want t...
you want to tell ...
this was invitati...

برای شکستن کتاب به جملات، در تابع withColumn، ستونی به اسم sentence ایجاد می‌کنیم که با اعمال تابع explode بر روی split با علائم نگارشی بر روی متن کتاب ایجاد می‌شود.

با این روش یک Dataframe داریم که در هر سطر آن، یک جمله از کتاب قرار دارد. سپس یک ستون تحت عنوان طول جملات، به این دیتا فریم اضافه کردیم. برای این کار، با تابع withColumn یک ستون با عنوان length اضافه کردیم و با استفاده از تابع size که بر روی split ستون sentence با استفاده از space صدا شده، این ستون را پر می‌کنیم. خروجی به صورت زیر است.

sentence	length
it is a truth uni...	23
however little kn...	48
my dear mr bennet...	21
mr bennet replied...	7
but it is returne...	19
mr bennet made no...	5
do not you want t...	10
cried his wife im...	4
you want to tell ...	13
this was invitati...	4

only showing top 10 rows

این ستون مانند یک برچسب از جنس int است.

سپس با استفاده از Tokenizer، ستون sentence را به عنوان ستون ورودی و words را به عنوان ستون خروجی مشخص می‌کنیم. با این روش، بعد از اعمال transform، یک ستون با نام words ایجاد می‌شود که کلمات را جدا کرده و به صورت توکن در یک لیست ذخیره کرده. خروجی به صورت زیر است:

sentence	length	words
it is a truth uni...	23	[it, is, a, truth...]
however little kn...	48	[however, little,...]
my dear mr bennet...	21	[my, dear, mr, be...]
mr bennet replied...	7	[mr, bennet, repl...]
but it is returne...	19	[but, it, is, ret...]
mr bennet made no...	5	[mr, bennet, made...]
do not you want t...	10	[do, not, you, wa...]
cried his wife im...	4	[cried, his, wife...]
you want to tell ...	13	[you, want, to, t...]
this was invitati...	4	[this, was, invit...]

only showing top 10 rows

از آنجایی که words و ستون خروجی آن no-stop-words است. نتیجه به صورت زیر است.

sentence	length	words	no-stop-words
it is a truth uni...	23	[it, is, a, truth...]	[truth, universal...]
however little kn...	48	[however, little,...]	[however, little,...]
my dear mr bennet...	21	[my, dear, mr, be...]	[dear, mr, bennet...]
mr bennet replied...	7	[mr, bennet, repl...]	[mr, bennet, repl...]
but it is returne...	19	[but, it, is, ret...]	[returned, mrs, l...]
mr bennet made no...	5	[mr, bennet, made...]	[mr, bennet, made...]
do not you want t...	10	[do, not, you, wa...]	[want, know, taken]
cried his wife im...	4	[cried, his, wife...]	[cried, wife, imp...]
you want to tell ...	13	[you, want, to, t...]	[want, tell, obje...]
this was invitati...	4	[this, was, invit...]	[invitation, enough]

only showing top 10 rows

سپس با استفاده از HashingTF و مشخص کردن ستون no-stop-words به عنوان ورودی و ستون features به عنوان خروجی، با استفاده از hash هر کلمه را به یک مقدار تبدیل کردیم که آن در hash-table به عنوان feature در جمله قابل استفاده است. خروجی به صورت یک لیست است که المان اول آن اندازه‌ی hash-table است، المان دوم، یک لیست شامل hash-index کلماتی که در هر جمله استفاده شده‌اند است و المان سوم، شامل فرکانس تکرار هر یک hash-index است. خروجی به صورت زیر است:

sentence	length	words	no-stop-words	features
it is a truth uni...	23	[it, is, a, truth...]	[truth, universal...]	{(262144, [5968, 788...],
however little kn...	48	[however, little,...]	[however, little,...]	{(262144, [21823, 33...],
my dear mr bennet...	21	[my, dear, mr, be...]	[dear, mr, bennet...]	{(262144, [5381, 218...],
mr bennet replied...	7	[mr, bennet, repl...]	[mr, bennet, repl...]	{(262144, [21653, 97...],
but it is returne...	19	[but, it, is, ret...]	[returned, mrs, l...]	{(262144, [52914, 13...],
mr bennet made no...	5	[mr, bennet, made...]	[mr, bennet, made...]	{(262144, [82035, 97...],
do not you want t...	10	[do, not, you, wa...]	[want, know, taken]	{(262144, [38004, 14...],
cried his wife im...	4	[cried, his, wife...]	[cried, wife, imp...]	{(262144, [12716, 82...],
you want to tell ...	13	[you, want, to, t...]	[want, tell, obje...]	{(262144, [85530, 16...],
this was invitati...	4	[this, was, invit...]	[invitation, enough]	{(262144, [113004, 1...],

only showing top 10 rows

نهایتاً با استفاده از `IDF` که ستونی `features` را به عنوان ورودی دریافت می‌کند، در ستون `idf` یک عبارت مشابه ستون `features` داریم که شامل تعداد فیچرها و آیدی فیچرهای استفاده شده است و المان سوم، به جای فرکانس تکرار، امتیاز آن در مدل `IDF` فیت شده را دارد. نتیجه به صورت زیر است:

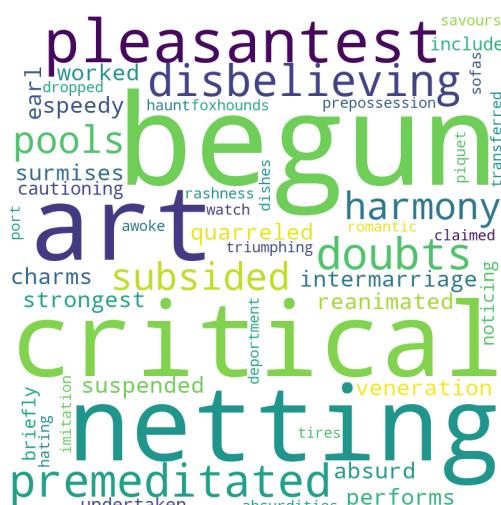
sentence	length	words	no-stop-words	features	idf
it is a truth uni...	23	[it, is, a, truth...]	[truth, universal...]	(262144, [5968,788...]	(262144, [5968,788...]
however little kn...	48	[however, little,...]	[however, little,...]	(262144, [21823,33...]	(262144, [21823,33...]
my dear mr bennet...	21	[my, dear, mr, be...]	[dear, mr, bennet...]	(262144, [5381,218...]	(262144, [5381,218...]
mr bennet replied...	71	[mr, bennet, repl...]	[mr, bennet, repl...]	(262144, [21653,97...]	(262144, [21653,97...]
but it is returne...	19	[but, it, is, ret...]	[returned, mrs, l...]	(262144, [52914,13...]	(262144, [52914,13...]
mr bennet made no...	5	[mr, bennet, made...]	[mr, bennet, made...]	(262144, [82035,97...]	(262144, [82035,97...]
do not you want t...	10	[do, not, you, wan...]	[want, know, taken]	(262144, [38004,14...]	(262144, [38004,14...]
cried his wife im...	4	[cried, his, wif...]	[cried, wife, imp...]	(262144, [12716,82...]	(262144, [12716,82...]
you want to tell ...	13	[you, want, to, t...]	[want, tell, obje...]	(262144, [85530,16...]	(262144, [85530,16...]
this was invitati...	4	[this, was, invit...]	[invitation, enough]	(262144, [113004,1...]	(262144, [113004,1...]

برای به دست آوردن ۵۰ کلمه‌ی با امتیاز بیشتر برای رسم word-cloud، کارهای زیر را انجام دادیم:

ابتدا یک select dataframe با no-stop-words کردن ستون از دیتافریم قبلی و اعمال تابع explode بر روی آن ایجاد می‌کنیم که نام ستون آن word خواهد بود. سپس با استفاده از distinct() از هر کلمه تنها یک رکورد نگه میداریم و برای ورودی دادن آن به hashingTF یک ستون به عنوان word-as-array ایجاد می‌کنیم که همان کلمه را در یک آرایه ذخیره کرده است.

HashingTF را بر روی این دیتافریم اعمال می‌کنیم و سپس ستون features حاصل از آن را برابر استفاده از idf که در گذشته بر روی جملات fit شده است استفاده می‌کنیم. (به این دلیل که مدل-`tf-idf` به جایگیری کلمات حساس نیست و تنها فرکانس را در نظر می‌گیرد؛ مشکلی نخواهیم داشت). برای اینکه در یک ستون، امتیاز هر کلمه را داشته باشیم، یک user-defined-function تعریف می‌کنیم که با دریافت ستون idf، امتیاز اولی که در المان سوم آن وجود دارد را به ما برگرداند (از آنجایی که در هر خط تنها یک کلمه داریم، اولین همان امتیاز کلمه‌ی مورد نظر است) و آن را در ستون جدید به نام value می‌ریزیم (با استفاده از `withColumn` و نهایتاً این دیتافریم را بر روی این ستون به صورت نزولی مرتب می‌کنیم و ۵۰ کلمه‌ی اول را برای رسم word-cloud برمی‌داریم.

تصویر رسم شده در نهایت یه صورت زیر است:



(b) بخش

برای این بخش هم، همان پردازش‌های قسمت قبل و تغییراتی که منتج به رسیدن به ستون words که شامل کلمات هر جمله بود را انجام دادیم و از ستون words که بعد از اعمال پیش‌پردازش‌ها و tokenize کردن جملات (بدون حذف stop-wordها) ایجاد شده بود استفاده خواهیم کرد.

برای شمارش تعداد تکرار کلمات، بعد از انتخاب ستون words از دیتا فریم قبلی، ابتدا با استفاده از explode بر روی ستون words، به ازای هر کلمه یک خط ایجاد کردیم و بعد بر روی کلمات group کردیم و تعداد هر دسته را به دست آوردیم. تمامی کلمات ابتدای لیست، جز stop-wordها هستند. می‌توانیم یک بار دیگر این کار را بر روی ستون no-stop-words اعمال کنیم تا تعداد تکرار سایر کلمات را مشاهده کنیم. نتیجه‌ی این بخش برو روی ستون words به صورت زیر است:

	value	count
the	2243	
to	2043	
of	1847	
and	1829	
her	1068	
a	1037	
i	1016	
in	956	
was	846	
she	796	
that	776	
not	754	
it	745	
he	694	
you	693	
his	632	
with	572	
be	571	
as	546	
mr	519	
had	510	
for	508	

همچنین اگر آن را بر روی ستون no-stop-words اعمال کنیم، کلمات زیر در صدر جدول قرار می‌گیرند:

	value	count
mr	519	
elizabeth	300	
said	226	
darcy	219	
miss	190	
much	181	
bingley	179	
mrs	175	
benet	171	
one	137	

only showing top 10 rows

برای ایجاد bi-gram و three-gram بر روی دادگان، دو instance از کلاس Ngram ساختیم که در یکی n برابر ۲ و در دیگری برابر ۳ است. سپس آن‌ها را بر روی ستون words اعمال کردیم تا در دو ستون 2-gram و 3-gram نتیجه را ذخیره کنیم.

دیتا فریم بعد از اضافه کردن 2-gram‌ها و 3-gram‌ها:

sentence	words	2-grams	3-grams
it is a truth uni... [it, is, a, truth... [it is, is a, a t... [it is a, is a tr...			
however little kn... [however, little,... [however little, ... [however little k...			
my dear mr bennet... [my, dear, mr, be... [my dear, dear mr... [my dear mr, dear...			
mr bennet replied... [mr, bennet, repl... [mr bennet, benne... [mr bennet replie...			
but it is returne... [but, it, is, ret... [but it, it is, i... [but it is, it is...			
mr bennet made no... [mr, bennet, made... [mr bennet, benne... [mr bennet made, ...			
do not you want t... [do, not, you, wa... [do not, not you, ... [do not you, not ...			
cried his wife im... [cried, his, wife... [cried his, his w... [cried his wife, ...			
you want to tell ... [you, want, to, t... [you want, want t... [you want to, wan...			
this was invitati... [this, was, invit... [this was, was in... [this was invitat...			
why my dear you m... [why, my, dear, y... [why my, my dear,... [why my dear, my ...			
that he came down... [that, he, came, ... [that he, he came... [that he came, he...			
morris immediate... [morris, immediat... [morris immediate... [morris immediate...			
what is his name [what, is, his, n... [what is, is his,... [what is his, is ...			
bingley] [bingley] [] []			
is he married or ... [is, he, married,... [is he, he marrie... [is he married, h...			
oh] [oh] [] []			
single my dear to... [single, my, dear... [single my, my de... [single my dear, ...			
a single man of l... [a, single, man, ... [a single, single... [a single man, si...			
four or five thou... [four, or, five, ... [four or, or five... [four or five, or...			

برای شمارش 3-gram‌ها هم، مشابه قسمت اول که فرکانس تکرار کلمات را به دست آوردیم؛ این بار تابع explode را بر روی ستون 3-grams می‌زنیم و بعد از groupby کردن روی نتیجه، تعداد تکرار هر کدام را به دست می‌آوریم. نهایتاً می‌توانیم نتیجه را بر روی ستون count مرتب کنیم و رایج‌ترین ۳تایی‌ها از کلمات را ببینیم.

value	count
i do not	39
i am sure	33
she could not	29
it would be	20
as soon as	20
on the subject	19
i am not	18
that he had	18
one of the	18
by no means	17
i dare say	17
and i am	16
a great deal	16
mr collins was	16
it is a	15
the honour of	15
and she was	14
in the world	14
in spite of	14
that it was	14

only showing top 20 rows

بخش C

برای ایجاد یک pipeline، نیاز به ساخت sequence‌هایی از Transformer‌ها و Estimator‌ها داریم. Transformer‌هایی مانند Tokenizer یا HashingTF یا ... به صورت پیش‌ساخته در PySpark وجود دارند اما برای برخی از کاربردها، نیاز به تعریف custom Transformer یا Estimator‌هایی به صورت مطابق کاربرد مورد انتظار ایجاد کنیم.

در اولین مرحله، یک کلاس Preprocess با ارث‌بری از Transformer ایجاد کردیم تا پیش‌پردازش‌هایی که قبل انجام شده بود را در متدهای transform این کلاس قرار دهیم و بتوانیم از آن در pipeline خود استفاده کنیم.

همچنین کلاس SplitToSentences هم با دریافت دیتافریم، با علائم نگارشی جمله‌های هر خط را از یکدیگر جدا می‌کند و سپس با استفاده از تابع explode برای هر کدام از آن‌ها یک خط در نظر می‌گیرد. در ادامه از Transformer‌های خود Pyspark شامل Tokenizer و Ngram استفاده می‌کنیم تا ابتدا کلمات را با توکن جدا کند و سپس bi-gram‌ها را تشکیل بدهد.

با تعریف کلاس AddBiGramWordLength که از کلاس Transformer ارثبری کرده است، طول کلمات bigram‌ها را به دست می‌اوریم. به این صورت که تابع explode را بر روی ستون bigrams اعمال می‌کنیم و در هر سطر از دیتافریم جدید، با استفاده از getItem، المان اول و یا دوم bigram را بدست می‌اوریم و طول آن را در ستون‌های len1 یا len2 ذخیره می‌کنیم. همچنین برای ساخت برچسب داده، ستون‌هایی که اختلاف طول کلمه‌ی اول و دوم آن‌ها بیشتر از ۲ باشد (و یا کوچک‌تر از ۲) برچسب ۱ و سایر داده‌ها برچسب ۰ می‌خورند.

همچنین برای بررسی n-gram‌های بالاتر، ما برای 3-gram‌ها هم بررسی کردیم. به این صورت که ستون len3 را هم مشابه دو ستون قبلی ایجاد کردیم، سپس ستون برچسب را به این صورت لیبل زدیم که اگر اختلاف طول کلمات اول و دوم، و دوم و سوم هر کدام کوچک‌تر از ۲ یا بزرگ‌تر از ۲ بود، لیبل ۱ و در غیر این صورت ۰ تعیین می‌کنیم.

برای استفاده از ویژگی‌های استخراج شده برای مدل، باید با استفاده از vectorAssembler ستون ویژگی‌های خود را بسازیم تا ویژگی‌ها برای مدل قابل استفاده باشد. ستون ویژگی ما در این سوال len1 و len2 است که می‌خواهیم بر اساس آن more_than_2 را پیش‌بینی کنیم. (در حالت 3-gram، len1 و len2 ویژگی‌های ما هستند).

در نهایت از LogisticRegression از Pyspark Estimator خود استفاده می‌کنیم تا دو مدلی برای پیش‌بینی طول کلمه‌ی دوم از روی طول کلمه‌ی اول بسازیم و نهایتاً با بررسی دقت این مدل ببینیم که آیا ارتباطی بین این دو وجود دارد یا خیر.

با مشخص کردن stage‌های این pipeline می‌توانیم آن را اجرا کنیم.

بعد از اجرای pipeline، برای اینکه بررسی کنیم آیا بین طول کلمات، ارتباطی وجود دارد یا خیر، با استفاده از تابع فیلتر، آن‌هایی که prediction آن‌ها برابر با برچسب اصلی بوده را جدا می‌کنیم و تعداد آن‌ها را بر تعداد کل داده‌ها تقسیم می‌کنیم. در نتیجه‌ی این کار، دقت ۷۵۵.۰ گزارش شد که در طی ۱۰ ایتریشن به دست آمد. در نتیجه، نتیجه می‌گیریم که بین آن‌ها ارتباط وجود دارد. جدول زیر، زمان اجرا برای هر یک از حالات خواسته‌شده را گزارش کرده است.

3-grams	2-grams	
1m 19s	1m 17s	۱ ماشین و ۱ هسته
1m 19s	1m 18s	۱ ماشین و ۲ هسته
1m 20s	1m 17s	۲ ماشین، هر کدام ۱ هسته
1m 21s	1m 20s	۲ ماشین، هر کدام ۲ هسته

برای اجرای این کد با هسته‌ها و ماشین‌های متفاوت، از همان دستوری که در ابتدای سوال یک توضیح داده شد استفاده کردیم. مقادیر executor-cores و total-executor-cores برای هر یک از کافیگ‌ها به همین ترتیب گزارش شده‌اند:

برای ۱ ماشین و ۱ هسته: ۱ و ۱

برای ۱ ماشین و ۲ هسته: ۲ و ۲

برای ۲ ماشین و ۱ هسته: ۲ و ۱

برای ۲ ماشین و ۲ هسته: ۴ و ۲

در مورد تحلیل زمان‌ها گزارش شده، انتظار می‌رود که با توزیع کردن تسک بر روی کورهای مختلف، زمان اجرای تسک کاهش پیدا کند. اما از آنجایی که اینجا حجم دیتای ما خیلی بالا نیست، این تغییر اصلاً محسوس نیست و تقریباً همه‌ی تسک‌ها در یک زمان انجام شده‌اند که زمان صرف شده ناشی از aggregate کردن پاسخ‌های هر بخش و سربار ارتباط با دیسک و ... است، و بخش محاسبات زمان زیادی از ما نگرفته است.

اما در صورتی که کار محاسباتی سنگین، که بر روی پارتیشن‌های مجزا قابل انجام باشد، داشته باشیم، این تفاوت محسوس‌تر خواهد بود.

سوال ۲-

با استفاده از **RDD**

برای خواندن داده‌ها، ابتدا یک spark-context ایجاد کردیم، سپس با تابع `textFile` در `spark-` فایل را از `hdfs` خواندیم و بر روی آن تابع `mapPartitions` را صدا کردیم تا هر خط را به صورت یک خط `csv` با `delimiter` با `tab` تفسیر کند. کد این بخش در تصویر زیر قابل مشاهده است:

```
data = sc.textFile('hdfs://raspberrypi-dml0:9000/rajabi/web_Graph.txt')\
    .mapPartitions(lambda line: csv.reader(line, delimiter='\t'))
```

بخش (a)

برای این بخش، ابتدا با استفاده از تابع `map` و یک `lambda function`، با دریافت هر خط، آیدی نود اول، که نود شروع یال است را به عدد ۱ مپ می‌کنیم. به این ترتیب برای هر آیدی، به تعداد درجه‌ی خروجی خط داریم که در هر کدام عدد ۱ نوشته شده است، سپس با استفاده از تابع `reduceByKey` که یک `lambda function` دریافت می‌کند، مقدار `value` برای `key` یکسان را با هم جمع می‌کنیم و از آنجایی که به ازای هر یال خروجی یک عدد ۱ داشتیم، برای هر `key` یا هر نود، مقدار درجه‌ی خروجی به دست می‌آید. با مرتب کردن نتایج بر حسب تعداد یال خروجی، نودهای زیر نودهای قرار گرفته در ابتدای این لیست خواهند بود:

```
[(456, '506742'),
 (372, '305229'),
 (372, '203748'),
 (330, '768091'),
 (277, '808643'),
 (268, '412410'),
 (265, '600479'),
 (258, '376428'),
 (257, '156950'),
 (256, '885728'),
 (253, '667584'),
 (248, '685695'),
 (247, '282140'),
 (245, '598188'),
 (244, '579314'),
 (231, '411593'),
 (229, '321091'),
 (225, '838278'),
 (216, '302733'),
```

البته برای مرتب کردن نتایج بر حسب ستون درجات، باید یک تابع `map` اعمال کنیم تا کلید را برابر با درجه و `value` را برابر با آیدی نود قرار دهد و نهایتاً تابع `sortByKey` را به صورت نزولی بر روی آن صدا کنیم.

برای به دست آوردن تعداد یال ورودی هم مشابه بالا عمل کردیم اما این بار در تابع `map`، ابتدا نود دوم در هر یال را به یک ۱ مپ کردیم. به این صورت هر نود به تعداد یال‌های ورودی، رکورد در دیتا خواهد داشت که آیدی آن را به عدد یک مپ کرده است و مشابه توضیحات داده شده می‌توان درجه‌ی ورودی را با `reduceByKey` به دست آورد. همچنین برای مرتب کردن نتایج هم مشابه قسمت قبل، باید تابع `map` ای که کلید را برابر ستونی که می‌خواهیم بر روی آن مرتب کنیم قرار می‌دهد را اعمال کنم. نودهای زیر، نودهای با بیشترین یال ورودی هستند:

```
[ (6326, '537039'),
  (5354, '597621'),
  (5271, '504140'),
  (5182, '751384'),
  (5097, '32163'),
  (4847, '885605'),
  (4731, '163075'),
  (4620, '819223'),
  (4550, '605856'),
  (4484, '828963'),
  (4220, '551829'),
  (4219, '41909'),
  (4206, '558791'),
  (4187, '459074'),
  (4180, '407610'),
  (4084, '213432'),
  (4015, '765334'),
  (4010, '384666'),
  (3988, '173976'),
```

بخش (b)

سه حالت برای وجود یک hub در گراف می‌توانیم متصور شویم. اول اینکه hub را با توجه به درجهٔ ورودی مشخص کنیم، دوم با توجه به درجهٔ خروجی، و سوم با توجه به مجموع درجات ورودی و خروجی.

برای حالت اول و دوم که از آنجایی که قبلاً داده را بر اساس درجهٔ ورودی و خروجی مرتب کرده‌بودیم، hub‌ها را داریم. همچنین با توجه به تعریف hub که درجهٔ بسیار بیشتر از میانگین است، میانگین درجهٔ خروجی برای نودهای این گراف برابر با ۶.۹ است و میانگین درجات خروجی هم برابر با ۷.۱۴ که با اعمال یک تابع map و مپ‌کردن یک کلید یکسان، به درجات مذکور و سپس reduceByKey و جمع همهٔ درجات به منظور میانگین‌گیری به‌دست آمده است. با توجه به فاصلهٔ این اعداد با اعدادی که در سوال قبل به عنوان نودهای با بیشترین درجه گزارش شده‌اند، همهٔ نودهای اول این لیست به اضافهٔ نودهای دیگری که تفاوت قابل توجه با میانگین دارند، هاب به حساب می‌آیند.

برای حالت سوم، rdd مربوط به درجهٔ ورودی و rdd مربوط به درجات خروجی را با یکدیگر join می‌کنیم. به این ترتیب، یک rdd سوم خواهیم داشت که به ازای هر نود، درجهٔ ورودی و خروجی آن را دارد. سپس با اعمال یک تابع map بر روی این rdd سوم، هر خط آن را می‌گیریم و به جای اینکه (node, (in-degree, out-degree)) داشته‌باشیم، آن را مپ می‌کنیم تا (node, (in-degree, out-degree)) داشته‌باشیم. سپس برای مرتب کردن این rdd بر اساس مجموع درجات، باید یک تابع map دیگر اعمال کنیم تا مجموع درجات را به عنوان key قرار بدهد و سایر مقادیر را در value قرار بده و نهایتاً تابع sortByKey به صورت نزولی را بر روی آن اعمال کنیم. نودهایی که بیشترین مجموع درجات را دارند در زیر قابل مشاهده‌اند:

```
[ (6353, ('537039', 6326, 27)),
  (5376, ('597621', 5354, 22)),
  (5290, ('504140', 5271, 19)),
  (5250, ('751384', 5182, 68)),
  (5129, ('32163', 5097, 32)),
  (4767, ('163075', 4731, 36)),
  (4637, ('819223', 4620, 17)),
  (4572, ('605856', 4550, 22)),
  (4491, ('828963', 4484, 7)),
  (4233, ('551829', 4220, 13)),
  (4229, ('41909', 4219, 10)),
  (4228, ('558791', 4206, 22)),
  (4210, ('459074', 4187, 23)),
  (4206, ('407610', 4180, 26)),
  (4102, ('213432', 4084, 18)),
  (4030, ('384666', 4010, 20)),
  (4019, ('765334', 4015, 4)),
  (4014, ('173976', 3988, 26)),
  (3970, ('687325', 3956, 14)),
```

در تمام نودهای بالای این لیست، یال ورودی وزن بسیار بیشتری دارد که با توجه به ماهیت web هم قابل توجیه است. چرا که یک صفحه نمی‌تواند به تعداد بسیار بالایی صفحه‌ی دیگر لینک داشته باشد.

همچنین میانگین مجموع درجات که مشابه میانگین‌گیری قبل انجام شده است، برابر با ۱۵.۷۴۸ است که باز هم نودهای اول لیست قبلی، همگی به عنوان هاب قابل تگ زدن هستند.

بخش (d)

برای چک کردن همبندی گراف، از الگوریتم‌های پیمایش گراف استفاده کردیم تا در صورتی که در یک پیمایش همه‌ی نودها را دیدیم، بتوانیم ادعا کنیم که گراف همبند است.

ابتدا یک dictionary از آیدی نود به دیده شدن یا نشدن (که در ابتدا همگی False هستند) به اسم visited ایجاد می‌کنیم که با دستور زیر از اجتماع نودهای اول و دوم یال‌ها ایجاد شده است.

```
visited = {k: False for k in data.keys().collect()}
visited.update({k: False for k in data.values().collect()})
```

سپس روی داده‌ها، یک تابع map صدا می‌کنیم که نود اول را به «لیست» نود دوم map کند و با صدا کردن تابع reduceByKey بر روی آن‌ها، این لیست‌ها را بهم اضافه می‌کنیم تا نهایتاً هر نود، به لیست نودهایی که به آن‌ها یال دارد وصل شود. سپس برای کاهش زمان دسترسی به این داده، روی آن collectAsMap را صدا می‌کنیم تا خروجی یک dictionary باشد. این کار به این دلیل است که در صورتی که در پیمایش، بخواهیم برای پیدا کردن همسایه‌ها filter کنیم، زمان زیادی از ما صرف خواهد شد.

سپس یک تابع DFS غیربازگشتی می‌نویسیم که از یک stack برای پیاده‌سازی استفاده می‌کند و هر نودی که به واسطه‌ی پیش‌روی در لیست همسایگان نودها مشاهده می‌کنیم را در visited با قرار دادن True به عنوان value، به عنوان دیده شده تگ می‌زنیم. بعد از پایان DFS از یک نود، در صورتی که حتی یک نود با مقدار False در visited وجود داشته باشد، گراف ما همبند نیست که این مقدار وجود داشت و حال باید اندازه‌ی اولین و دومین مولفه‌ی همبندی را پیدا کنیم.

برای این کار از کتابخانه networkx استفاده کردیم. به این صورت که با iterate کردن روی لیست یال‌ها، یال‌ها را با تابع edge.add به گراف اضافه کردیم و در نهایت با استفاده از تابع connected_components کامپوننت‌های همبند گراف را به دست آوردیم. نهایتاً با توجه به طول کامپوننت این مولفه‌ها را مرتب کردیم و نتیجه این بود که بزرگ‌ترین کامپوننت این گراف اندازه‌ی ۸۵۵۸۰۲ دارد و اندازه‌ی دومین مولفه هم ۴۰۴ است.

بخش (e)

برای به دست آوردن قطر گراف هم مجدداً از کتابخانه networkx استفاده کردیم. با توجه به اینکه این گراف همبند نیست، قطر آن بی‌نهایت است. اما می‌توانیم قطر را برای بزرگ‌ترین مولفه به دست بیاوریم که با تابع diameter بر روی subgraph بزرگ‌ترین مولفه، محاسبه می‌شود.

بخش (f)

برای بررسی وجود یا عدم وجود دور اویلری، در صورتی که گراف را بدون جهت در نظر بگیریم، باید تمامی نودها درجه‌ی زوج داشته باشند. به این ترتیب با استفاده از `rdd` ای که شامل درجه‌ی ورودی، درجه‌ی خروجی و مجموع درجات بود (در بخش b توضیح داده شد) با اعمال تابع `filter` با شرطی که مجموع درجات آن فرد باشد، به این نتیجه می‌رسید که این لیست خالی نیست و تعداد زیادی نود با درجه‌ی فرد داریم. در نتیجه این گراف دور اویلری ندارد.

همچنان اگر گراف را جهت‌دار در نظر بگیریم، باید تعداد یال خروجی و ورودی برای هر نод مساوی باشد. در نتیجه، با اعمال فیلتر بر روی همان `dd` بخش قبل با شرط اینکه درجه‌ی ورودی و خروجی یکسان نباشند، باز هم به این نتیجه می‌رسیم که نتیجه‌ی این فیلتر شامل تعداد زیادی نود است در نتیجه از این طریق هم مجدد عدم وجود دور اویلری استنباط می‌شود.

بخش (g)

برای به دست آوردن مثلث‌های موجود در گراف، ابتدا یک `rdd` که هر نod را به تمام نودهایی که به آن‌های یال دارد وصل می‌کند، ایجاد می‌کنیم. به این صورت که ابتدا بر روی داده‌ی اصلی یک تابع `map` که نود اول را به لیست نود دوم وصل می‌کند اعمال می‌کنیم و سپس با تابع `reduceByKey` که برای هر نود، لیست مجاورت آن را داریم (نودهایی که به آن‌ها یال دارد) سپس `map` را بر روی آن اعمال می‌کنیم تا نتیجه یک `dictionary` باشد که هر نود را به لیست مجاورت آن مپ می‌کند. از آنجایی که دسترسی به دیکشنری از O(1) است، در بخش بعدی، الگوریتم زمان زیادی را سیو می‌کند.

سودوکد کد پیاده‌سازی شده به این صورت است:

```
For u, adjacency_list(u):
    for v in adjacency_list(u):
        for w in adjacency_list(v):
            if w->u is in adjacency_list(w):
                add (u, v, w) to triangles
```

و به این ترتیب با دیکشنری ساخته شده لیست مثلث‌ها را به دست می‌آوریم که تعداد از آن‌ها را در تصویر زیر می‌بینید.

```
{('532327', '753875', '797584'),
 ('523452', '78244', '659654'),
 ('395500', '267647', '682431'),
 ('434868', '355595', '692663'),
 ('97827', '138941', '897545'),
 ('471639', '437199', '119096'),
 ('503727', '335016', '793635'),
 ('569548', '163412', '876631'),
 ('137774', '680301', '367408'),
```

با استفاده از DataFrame

برای خواندن داده در این بخش، یک spark-session ایجاد کردیم و سپس از تابع `read.csv` در `spark-sesession` برای خواندن داده از `hdfs` استفاده کردیم. نتیجه به صورت یک `DataFrame` است.

(a) بخش

برای این بخش، برای بهدست آوردن درجهی ورودی، ابتدا داده‌ها را بر روی `node1` گروه کردیم و تعداد هر گروه را بهدست اوردیم و سپس بر روی ستون `out_degree` نتایج را `sort` کردیم (با استفاده از `orderBy` که نام یک ستون را می‌گیرد، و به صورت نزولی). برای بهدست آوردن درجهی ورودی هم داده‌ها را بر روی `node2` گروه کردیم و باز هم تعداد هر گروه را بهدست آوردیم و مشابه قبل، نتایج را مرتب کردیم.

۱۰ نод ابتدایی مرتب‌سازی بر حسب بیشترین درجهی خروجی به شرح زیر است:

node	out_degree
506742	456
305229	372
203748	372
768091	330
808643	277
412410	268
600479	265
376428	258
156950	257
885728	256

همچنین ۱۰ نود ابتدایی بیشترین درجهی ورودی هم به صورت زیر است:

node	in_degree
537039	6326
597621	5354
504140	5271
751384	5182
32163	5097
885605	4847
163075	4731
819223	4620
605856	4550
828963	4484

(b) بخش

برای بهدست اوردن هاب‌ها، مشابه قبل سه حالت داریم که یکی بر حسب درجهی ورودی، دیگری درجهی خروجی و سومی مجموع درجات است. دو مورد اول که در بخش a محاسبه شده‌اند و با توجه به میانگین هر کدام که به ترتیب ۷.۱۴ و ۶.۹ است، در تعریف hub می‌گنجند.

برای بهدست آوردن هابها با تعریف بیشترین مجموع درجات، یک دیتافریم از `join` دو دیتافریم قبلی بر روی ستون `node` بهدست می‌آوریم و با استفاده از تابع `withColumn` یک ستون با نام `degree` به آن اضافه می‌کنیم که از جمع `col(out-degree)` و `col(in_degree)` بهدست می‌آید و سپس آن را با استفاده از `orderBy('degree', ascending=False)` مرتب می‌کنیم. ۱۰ نود برتر، که ۱۰ بزرگ‌ترین هاب ما هم هستند به شرح زیر است:

node	in_degree	out_degree	degree
537039	6326	27	6353
597621	5354	22	5376
504140	5271	19	5290
751384	5182	68	5250
32163	5097	32	5129
163075	4731	36	4767
819223	4620	17	4637
605856	4550	22	4572
828963	4484	7	4491
551829	4220	13	4233

(d) بخش

مشابه توضیحات قسمت RDD می‌خواهیم یک دیکشنری از نودها داشته باشیم که نشان دهد هر نود در DFS دیده شده است یا نه. به این منظور با اجتماع روی نودهای اول و دوم یک دیکشنری ایجاد می‌کنیم که در ابتداء مقدار تمام آن‌ها `False` است.

سپس برای ساخت لیست مجاورت برای هر نود (لیست نودهایی که به سمت آن‌ها یال دارد) ابتدا داده را بر روی ستون `node1` گروه می‌کنیم، سپس در تابع `agg`، تابع `collect_list` را استفاده می‌کنیم. سپس برای تبدیل این دیتافریم به دیکشنری، ابتدا آن را به `pandas-dataframe` تبدیل می‌کنیم و سپس با ایندکس کردن بر روی `node1` آن را به دیکشنری تبدیل می‌کنیم. پیاده‌سازی تابع DFS در این بخش، درست مانند تابع RDD است و باز هم بعد از اجرای DFS، همچنان نودی بدون دیده شدن داریم و در نتیجه گراف ما همبند نیست.

برای بهدست آوردن کامپوننت‌ها هم مجدداً از کتابخانه `networkx` استفاده کردیم و این بار برای ساخت گراف، می‌توانیم در زمان لوپ زدن بر روی گراف، می‌توانیم مقدار `node1` و `node2` را بگیریم و با ایجاد یک یال، کم کم گراف را ایجاد کنیم. البته یک راه دیگر برای این کار، این است که ابتداء دیتافریم را تبدیل به یک `pandas-dataframe` کنیم و با استفاده از آن، به یکباره گراف را ایجاد کنیم.

در نهایت مشابه قبل، با استفاده از تابع `connected_components` می‌توانیم مولفه‌های همبندی را بهدست بیاوریم و سپس با مرتب کردن به صورت نزولی و بر روی اندازه مولفه‌ها، اندازه بزرگ‌ترین مولفه و دومین بزرگ‌ترین مولفه به ترتیب برابر با ۴۰۴ و ۸۵۵۸۰۲ است.

(e) بخش

برای این بخش هم از گراف ساخته شده در بخش d استفاده می‌کنیم و با استفاده از `connected_components`، قطر بزرگ‌ترین مولفه را با استفاده از تابع `diameter` بهدست می‌آوریم.

(f) بخش

مشابه توضیحات قسمت RDD، فرض می‌کنیم که یک بار گراف جهتدار است، در این صورت باید تعداد یال‌های ورودی و خروجی یکسان باشد. برای چک‌کردن این شرط، روی دیتافریمی که در بخش b ساختیم و شامل درجهٔ خروجی، ورودی و مجموع درجات بود؛ یک تابع filter صدا می‌کنیم که در آن چک می‌کنیم که آیا مقدار out_degree و in_degree برای نودی نامساوی هست یا خیر. که هست و گراف دور اویلری ندارد.

اگر گراف را بدون جهت فرض کنیم، باید مجموع درجات تمامی نودها زوج باشد. برای چک‌کردن این شرط هم با استفاده از تابع filter بر روی دیتافریم مذکور، چک می‌کنیم که آیا نودی وجود دارد که مجموع درجات آن فرد باشد یا خیر؛ که وجود دارد. پس از این راه هم نتیجهٔ می‌گیریم که گراف دور اویلری ندارد.

(g) بخش

در این بخش هم درست مشابه بخش RDD، سعی کردیم که یک دیکشنری از لیست مجاورت برای هر نод (شامل نودهایی که به سمت آن‌ها یال دارد) ایجاد کنیم. این کار را در بخش چک‌کردن همبندی انجام دادیم، به این صورت که بر روی node1 گروه کردیم و سپس در تابع agg تابع collect_list را استفاده کردیم تا لیستی از نودهایی که به آن یال دارد ایجاد کند و سپس با تبدیل این دیتافریم به و ایندکس‌کردن بر روی node1 آن را به دیکشنری تبدیل کردیم.

در ادامه، همان سودوکدی که در بخش RDD ارائه دادیم را بر روی این دیکشنری پیاده‌سازی کردیم.
بخشی از مثلث‌های موجود در این گراف را در زیر می‌بینید:

```
('10000', '245186', '9111'),  
('10000', '245186', '145200'),  
('10000', '245186', '419669'),  
('10000', '245186', '662845'),  
('10000', '245186', '721534'),  
('10000', '245186', '722570'),  
('10000', '245186', '746615'),  
('10000', '245186', '789565'),  
('10000', '896395', '56905'),  
('10000', '896395', '846625'),  
('100008', '226332', '472008'),  
('100008', '472008', '15409'),  
('100008', '472008', '225239'),  
('100008', '472008', '226332'),
```

با استفاده از SQL

برای استفاده از داده‌هایی که به وسیلهٔ sparksession و استفاده از تابع read.csv، مشابه قسمت DataFrame خوانده شده است؛ باید تابع createOrReplaceTempView را اعمال کنیم تا یک view با نامی که دریافت می‌کند برای ما بسازد و سپس کوئری‌های sql را اجرا کنیم.

(a) بخش

برای پیدا کردن درجهٔ خروجی، ابتدا بر روی `node1` گروه می‌کنیم و تعداد هر گروه را به دست می‌آوریم؛ سپس باید بر روی تعداد آن مرتب کنیم. برای این کار تابع `sql` را بر روی `sparkSession` صدا می‌کنیم و کوئری `sql` متناظر این کار را به آن می‌دهیم. خروجی چند نod اول این لیست به صورت زیر است:

node	out_degree
506742	456
305229	372
203748	372
768091	330
808643	277
412410	268
600479	265
376428	258
156950	257
885728	256
667584	253
685695	248
282140	247
598188	245
579314	244
411593	231
321091	229
838278	225
302733	216
915273	213

همین کار را باید برای به دست آوردن درجهٔ ورودی هر nod انجام دهیم. این بار داده‌ها را بر روی `node2` گروه می‌کنیم و همان کارها را تکرار می‌کنیم. خروجی ۲۰ نod برتر این ترکیب، در زیر آمده است.

node	in_degree
537039	6326
597621	5354
584140	5271
751384	5182
32163	5097
885605	4847
163075	4731
819223	4620
695856	4550
828963	4484
551829	4220
41909	4219
558791	4206
459074	4187
407610	4180
213432	4084
765334	4015
384666	4010
173976	3988
687325	3956

(b) بخش

مشابه تمامی قسمت‌های قبلی، در صورتی که منظور، هاب در بین درجه‌های خروجی یا ورودی باشد، همان نودهای گزارش شده در قسمت a پاسخ این سوال هم هستند. اما با فرض اینکه می‌خواهیم هاب‌ها را با مجموع درجات ورودی و خروجی گزارش کنیم، باید ابتدا پاسخ دو بخش a رو با اسم‌های `in_degrees` و `out_degrees` با تابع `createOrReplaceTempView` ثبت کنیم و سپس این دو جدول را بر روی ستون `node` جوین کنیم و آیدی نod و مجموع درجات خروجی و ورودی را برگردانیم و بر روی ستون مجموع مرتب کنیم. نودهای بالای این لیست به شرح زیر است:

node	degree
537039	6353
597621	5376
504140	5290
751384	5250
32163	5129
163075	4767
819223	4637
605856	4572
828963	4491
551829	4233
41909	4229
558791	4228
459074	4210
407610	4206
213432	4102
384666	4030

با توجه به میانگین درجات در هر یک از این سه حالت، نودهای گزارش شده در هر سه پاسخ مناسبی هستند.

(d) بخش

برای اینکه با استفاده از sql بتوانیم همبندی را چک کنیم، باید بتوانیم از یک نود DFS بزنیم و بینیم آیا تمامی نودها مشاهده می‌شوند یا خیر. این کار با استفاده از recursive sql امکان‌پذیر است اما pyspark از آن پشتیبانی نمی‌کند. به همین دلیل راه دومی وجود دارد که هزینه‌ی محاسباتی بسیار بالایی هم دارد و به همین دلیل در سیستم شخصی موفق به دریافت جواب از آن نشدم. این راه این است که به صورت متوالی، یک یال از v به w را بگیریم، و به ازای هر یال w ، رکورد w به w را هم اگر در گراف نبود، اضافه کنیم. زمانی که دیگر رکوردي اضافه نمی‌شد، چک می‌کنیم که آیا مجموعه رکوردهایی که در سمت اول آن‌ها v قرار دارد، در سمت دوم، شامل تمام نودهای گراف می‌شود یا خیر.

(e) بخش

برای به دست آوردن قطر گراف، با استفاده از pymysql ، ابتدا با دستور select تمامی یال‌ها را دریافت می‌کنیم، سپس به ازای هر رکورد این پاسخ، یک یال به networkx در کتابخانه اضافه می‌کنیم. بعد از آن، از آنجایی که گراف همبند نیست، باید ابتدا بزرگ‌ترین مولفه‌ی همبندی را با استفاده از این کتابخانه، درست مشابه قسمت‌های قبل پیدا کنیم و سپس قطر این مولفه را به دست بیاوریم.

(f) بخش

دیتاست ایجاد شده در بخش b را هم با createOrReplaceTempView ثبت می‌کنیم. سپس مشابه قسمت‌های RDD و Dataframe از دو طریق وجود یا عدم وجود دور اویلری را چک می‌کنیم. ابتدا با فرض جهت‌دار بودن گراف، باید تعداد یال‌های ورودی و تعداد یال‌های خروجی گراف برای همه‌ی نودها برابر باشند؛ اما با نوشتن کوئری‌ای که تعداد نودهایی با درجه‌ی ورودی و خروجی متفاوت را گزارش می‌کنند (با استفاده از جوین دو جدول out_degrees و in_degrees بر روی ستون node جدول مناسب ایجاد می‌شود)، متوجه می‌شویم که تعداد این نودها صفر نیست پس گراف دور اویلری ندارد.

سپس با فرض بدون جهت بودن گراف، باید چک کنیم که درجه‌ی تمام رئوس زوج باشد. پس با کوئری sql‌ای که تعداد نودهای با درجه‌ی فرد را در جدول degrees پیدا می‌کند، متوجه می‌شویم که این گراف دور اویلری ندارد چرا که نودهای با درجه‌ی فرد وجود دارند.

(g) بخش

برای این بخش، ابتدا با استفاده از `with clause` یک جدول موقت با نام `third_pair` ایجاد کردیم که سه ستون با نام‌های `node1`, `node2` و `node3` دارد که از جوین جدول `data` با خودش به وجود می‌آید. هر عضو از این جدول، به صورتی است که هم از `node1` به `node2` یال موجود باشد، و هم به `node2`. سپس این جدول را بار دیگر با `data` جوین می‌کنیم و سه تایی‌هایی از این جدول را انتخاب می‌کنیم که `node1` به `node3` یالی در `data` داشته باشد.

سوال - ۳

(a) بخش

در این بخش، برای انتخاب ویژگی‌ها، تمامی ویژگی‌ها به جز نام رسپی و آیدی آن انتخاب کردیم. شاید نام هم می‌توانست تاثیری در پیش‌بینی بهتر داشته باشد، اما نیاز به امبدینگ کلمات و ... بود و این کار را انجام ندادیم.

همچنین برای ادامه‌ی این سوال، با دو روش کار را دنبال کردیم، در یک روش داده‌ها را بر اساس مقدار RecipeCategory فیلتر کردیم و تنها آن‌هایی که زمان پخت را ارائه داده‌بودند نگه داشتیم، و در روش دیگر، مقدار این ستون را برای سایر رسپی‌ها برابر با NotKnown قرار دادیم. در روش اول، بخش زیادی از داده را دور ریخته‌ایم. اما در روش دوم، به علت حجم بسیار بالای این دسته از داده‌ها، احتمال بایاس شدن و افرایش دقت مدل همانطور که در بخش آخر خواهیم دید وجود دارد.

همچنین ستون خروجی، یا همون RecipeCategory را با استفاده از StringIndexer ایندکس کردیم تا در نهایت بتوانیم به عنوان label در آموزش مدل از آن استفاده کنیم و نام آن هم CategoryIndex است.

(b) بخش

زمانی که داده‌ها از فایل csv خوانده می‌شود، تایپ تمامی داده‌ها string است. به این منظور، تمامی ویژگی‌ها را با استفاده از تابع withColumn و استفاده از cast در داخل آن، به FloatType کست کردیم. همچنین ستون خروجی یا CategoryIndex را هم به IntegerType کست کردیم.

بعد از cast کردن ویژگی‌ها، با استفاده از VectorAssembler ستون‌های ویژگی را انتخاب کردیم و آن‌ها را به عنوان یک وکتور، در ستون features ذخیره کردیم تا بتوانیم در PCA از آن‌ها استفاده کنیم. سپس با استفاده از PCA که ستون ورودی آن features و ستون خروجی آن pcaFeatures است، بهترین ۴ ویژگی را برای آموزش مدل انتخاب کردیم.

(c) بخش

با استفاده از randomSplit که بر روی DataFrame می‌توانیم صدا کنیم، ۲۰ درصد داده را به عنوان تست و ۸۰ درصد آن را به عنوان دادگان آموزش استفاده کردیم و برای مدل از یک MultilayerPerceptronClassifier استفاده می‌کنیم که در مجموع ۷ لایه دارد که یک لایه‌ی آن ورودی، یک لایه‌ی خروجی (در صورت حذف دادگان بدون لیبل سایز خروجی ۴ است و در غیر این صورت ۵) و ۵ لایه‌ی مخفی دارد و این مدل را بر روی داده‌های آموزش fit می‌کنیم.

بعد از آموزش مدل، داده‌های تست را به مدل می‌دهیم تا خروجی آن‌ها را در ستون predict قرار بدهد. در نهایت ستون label (که از روی ستون CategoryIndex ایجاد کردہ‌ایم) و ستون prediction را

به یک instance از کلاس MulticlassClassificationEvaluator می‌دهیم تا مدل را گزارش کند.

مدل ما در ۱۰۰ ایتریشن، بر روی بلاک‌سایزهای ۱۲۸ تایی و با اندازه‌ی لایه‌های میانی ۱۰، ۱۲، ۱۵، ۱۰ آموزش دیده است.

دقت داده‌های تست در حالتی که به داده‌های بدون زمان، برچسب "NotKnown" داده‌باشیم، برابر با ۹۳ درصد و در صورت حذف این داده‌ها برابر با ۳۵ درصد است.