基于天猫数据的大数据系统构建

> 李易非：算法讨论，1、3、4、5 题 coding，环境搭建，可视化
>
> 林新迪：算法讨论，2、3、6 coding，环境搭建
>
> 熊苗：算法讨论，8、9、10 coding，可视化

一、Overview

在本次项目中，我们需要基于天猫商品数据集设计一个大数据分析系统，基于十道题目的要求，我们将问题解耦为两个部分，一个是 offline 的处理过程，一个是 online 的推荐过程。总的来说，在 offline 的处理流程中，我们利用 pyspark 对数据集进行分析、处理、整合，生成不同的 table 为大数据系统的不同功能和可视化提供数据。图 1。为大数据系统 offline processing 的计算图，可以看到，不同 table 将在每一步处理中逐渐扩充并生成新的 table，我们将在第二节中详述每个部分中涉及的算法和处理策略。

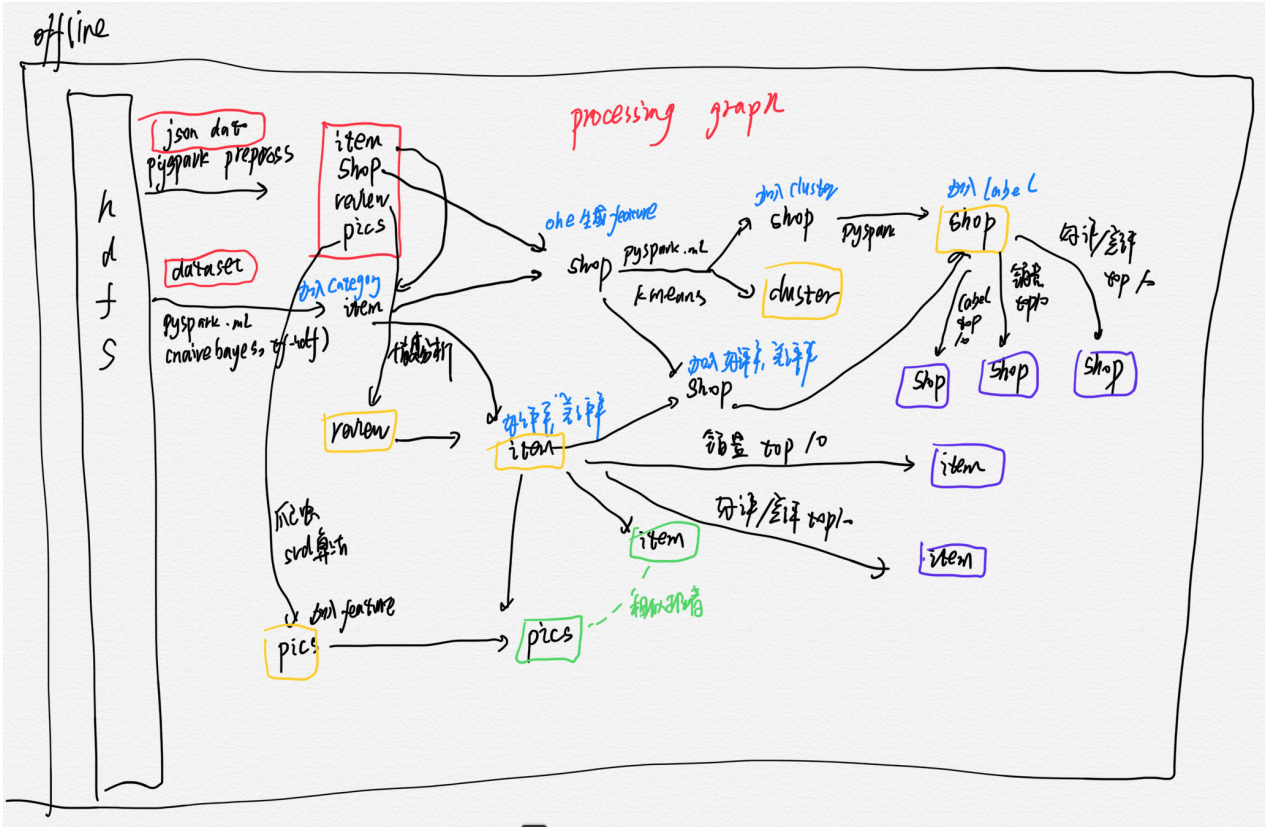


图1 offline processing 计算图

在 table 的设计上，我们将 table 分为三类，第一类是普通的数据表，普通数据表中存储由 json 数据中提取出的 feature 以及不同题目输出的新的 feature （比如商品种类、店铺聚类等）；第二类是用于 top k 预测的预计算表，为了保证大数据系统查询的响应速度，我们将所有要求的 top k 都预计算 top 10 的结果；第三类是用于相似推荐的表格，我们在其中提前计算了不同商品/图片的特征，以便相似推荐的 knn 算法能够更加快速的执行。online 的推荐过程以及可视化过程则是利用 offline 处理生成的表格完成各自的工作。图 2 为最终 offline processing 结束后大数据系统中的表格以及它们之间的联系。其中黄色表格是普通的数据表，绿色表格为用于相似推荐的两张表格，紫色表格为预计算的表格。

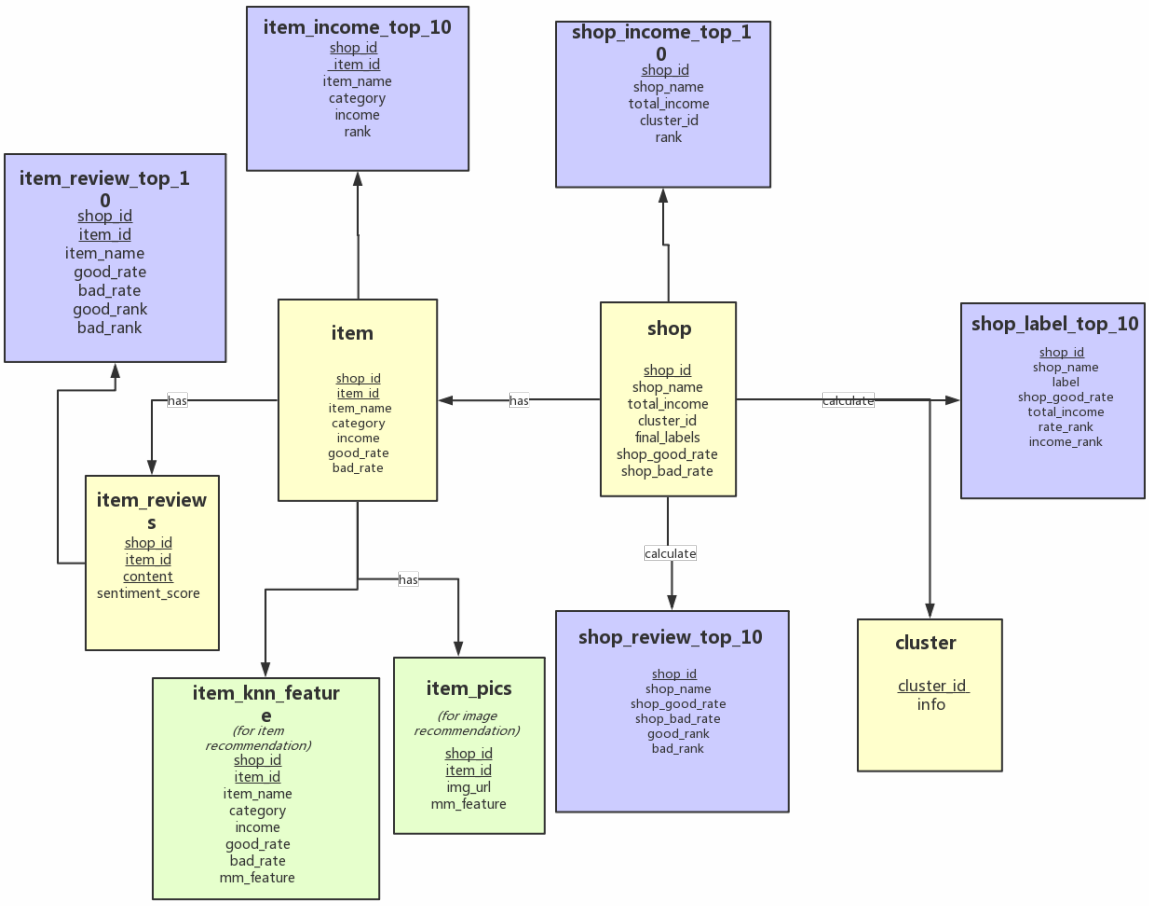


图2 系统数据库表格一览

在可视化上，我们采用了 echarts 作为解决方案，echarts 拥有简单易用的 API 以及绚丽丰富的可视化显示，帮助我们做出了简单而又富有信息量的大数据看版。

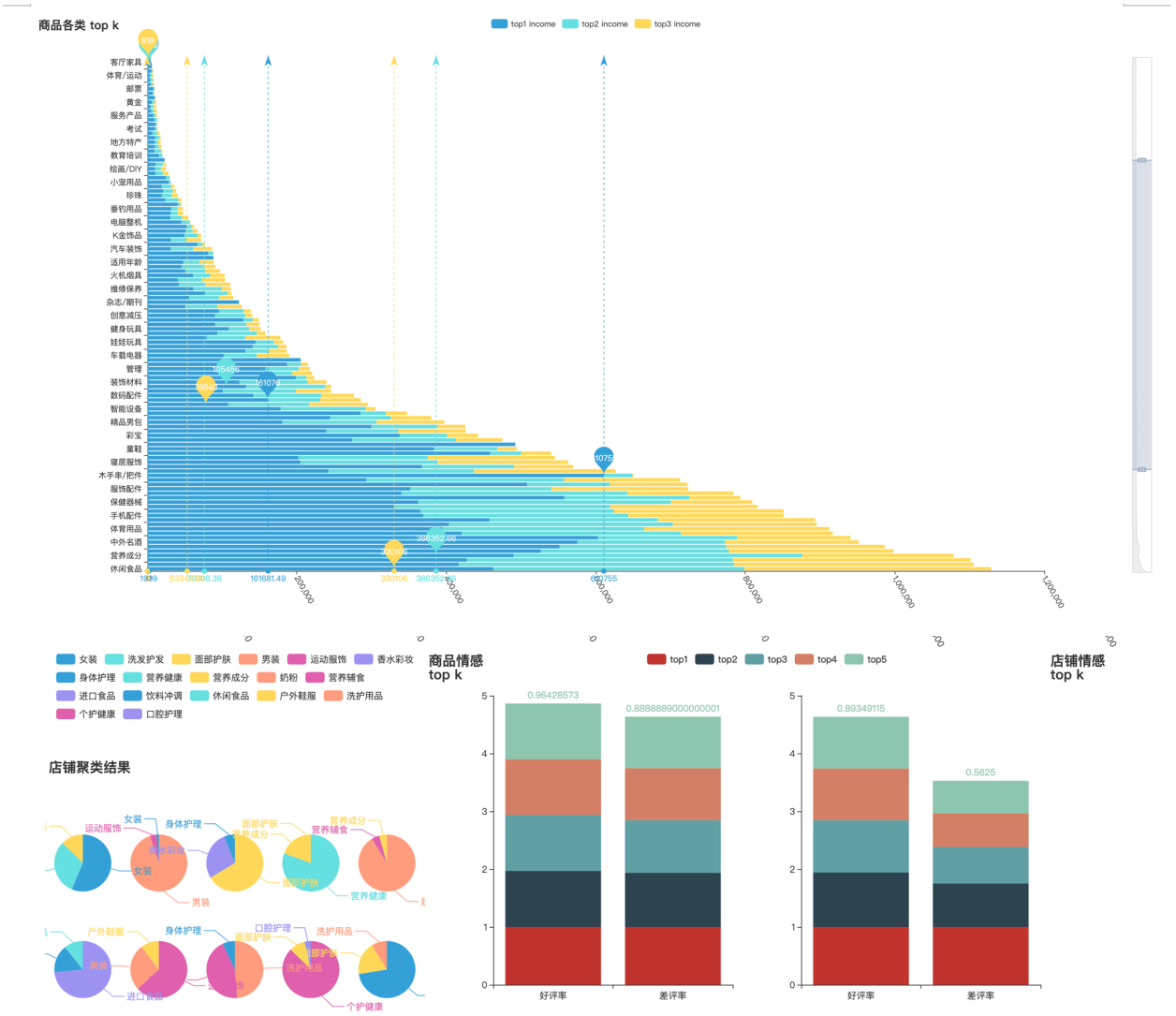


图3 大数据看版部分截图

二、算法详解

1 销售额统计

1.1 要求

销售额统计：计算所有店铺的销售总额，每件商品的销售额定义为（当前价格+邮费+进口税）*月销量，若当前价格或月销量为 null 则忽略。

1.2 实现

计算商品的销售额，首先我们得求出每件商品的当前价格、邮费、进口税和月销量。

其中计算当前价格比较容易，raw data 中有显式的结构表明物品的当前价格，但是邮费和进口税、销量都需要对文本进行分析，通常店家都会用 包邮、免税等字样来说明。

- 首先我们封装一个 shop 类 用来记录店铺的信息和总销售额

- 接下来封装 product 类来计算一件商品的总价格
  - 计算单件商品的当前价格

```
self.price = product_dict.get('price', None)
```

  - 分析商品的邮费，我们使用正则表达式来分析 postage

```
SPECIAL_POSTAGE = {'上门安装', '包邮', '快递包', '快递包邮'}
POSTAGE_PATTERN = re.compile(r'(EMS|快递|运费):\s*(\d+\.\d+)')

def parse_postage(self):
    ...
            # 看是否包邮
        if self.postage in self.SPECIAL_POSTAGE:
            self.postage = 0
        else:
            # 匹配邮费
            match = re.search(self.POSTAGE_PATTERN, self.postage)
            self.postage = float(match.group(2))
```

  - 分析商品的销量，利用正则表达式匹配 sales 对应的文本

```
SALES_PATTERN = re.compile(r'月销量\s*(\d+)\s*件')

def parse_sales(self):
        # 匹配销量
        match = re.search(self.SALES_PATTERN, self.sales)
        self.sales = float(match.group(1))
```

  - 分析商品的进口税

```
TARIFF_PATTERN = re.compile(
        r'进口税[:： ]\s预计(\d+\.\d+|\d+\.\d+\s*-\s*\d+\.\d+)\s*元')
SPECIAL_TARIFF = {'进口税: 买家自行承担',
                        '进口税: 商品售价已包税',
                        '进口税: 商品售价最高全额包税',
                        '进口税: 商家承担',
                        '进口税: 如需征收买家承担'}
```

    进口税的处理方法相较前面稍显复杂

    - 如果识别到 SPECIAL_TARIFF 则税费为零；
    - 如果识别到 TARIFF_PATTERN 字样，则按照识别到的税费处理，
    - 如果存在 a-b 元类型，我们取均值作为最后的税费。
    - 如果没有识别到进口税字样，但识别到 10% 这样的字样，则我们按照当前价格的百分比进行收税。

计算得到相应的 邮费、进口税、当前价格、销量之后，我们只需要针对每件商品将这几者加和即可。最后店铺再将所有的商品的总销售额相加即可。

2 商品归类

2.1 要求

对所有商品进行聚类，类别合理可解释。

2.2 实现

2.2.1 基本思路

- 搜索带类别标签商品信息数据集、数据预处理
- 将商品名称进行分词，映射为tf-feature
- 用Naive Bayes训练模型
- 利用模型对原始数据集商品进行分类

2.2.2 外来数据集试验

有监督学习转化

如果不借助外部的数据集，商品分类只能利用非监督的学习对所有商品进行聚类，并对各个商品类别进行解释。但是无监督方法有三个主要缺点，第一，非监督学习准确率较低；第二、商品类别繁多，难以对聚类结果进行缜密有效的分类；第三，非监督学习无法获得，通用、合理、兼容性好的类别体系，在成熟的电商市场上难以应用。

因此，我们希望找到带类别标签的电商平台商品数据集，以该数据集为训练集，以原始数据集为预测集进行商品归类，将非监督的学习转化为有监督的分类问题。

数据集简介

所找到的外部数据集必须商品涵盖面广，商品体系粒度大小适中。我们在Github，Kaggle，以及一些开放数据平台的电商数据进行试验后，我们选择了北京大学开放数据研究平台的电商商品数据。

该数据集包含703500件商品，商品DataField如下

```
{'class_small': '礼服/演出服',
  'class_strong': '童装',
  'pro_class': '母婴',
  'pro_name': 'HUMG FENG品牌儿童礼服裙女童公主裙花童生日小礼服演出服婚纱主持人蓬蓬裙 藕色 130cm',
  'pro_type': '男装/女装/童装/内衣'},
 {'class_small': '礼服/演出服',
  'class_strong': '童装',
  'pro_class': '母婴',
  'pro_name': '尚品琳儿童舞蹈服装夏季幼儿女童练功服短袖考级服连体服芭蕾舞裙 曜石黑 120cm',
  'pro_type': '男装/女装/童装/内衣'},
```

我们选取pro_type作为一级分类，class_strong作为二级分类，class_small作为三级分类。一级分类的个数为 15，二级分类的个数为 216，三级分类的个数为 2172。

数据预处理

数据预处理主要包括将字符串表示的类别映射为整数类型；去除数据集中存在类别标签噪声，识别出这些错误标签并将其修改为正确标签。

在对数据集进行处理后，我们用sklearn的NB方法对数据进行试验并在训练集上观察正确率。我们分别进行了对一级分类和二级分类的预测试验。

2.2.3 Pyspark 核心算法实现

文本清洗与分词

```
invalid_pattern = re.compile('[a-zA-Z0-9’!"#$%&\'()*+,-./:;<=>?@，。?★、…【】《》？“”‘’！[\\]^_`{|}~\s]+')
def cut_func(row):
    name = re.sub('[a-zA-Z0-9’!"#$%&\'()*+,-./:;<=>?@，。?★、…【】《》？“”‘’！[\\]^_`{|}~\s]+', "", row.name)
    words = jieba.lcut(name, cut_all=False, HMM=False)
    return Row(cate_ID=row.cate_ID, words=words)
```

```
cate_words_rdd = cate_name_df.rdd.map(lambda x: cut_func(x))
cate_words_df = cate_words_rdd.toDF().cache()
```

转换为tf-idf-features

```
hashingTF = HashingTF(inputCol="words", outputCol="tf_features", numFeatures=15000)
tf_result = hashingTF.transform(cate_words_df)
idf = IDF(inputCol="tf_features", outputCol="idf_features")
idf_model = idf.fit(tf_result)
tf_idf_result = idf_model.transform(tf_result)
```

训练Bayes模型

在pyspark的bayes方法中，label必须为double类型，因此我们将int类型的label转为double类型

```
nb = NaiveBayes(featuresCol='idf_features', labelCol='cate_ID', smoothing=1.0, modelType='multinomial')
tf_idf_result = tf_idf_result.select(tf_idf_result['cate_ID'].astype('double'), 'idf_features')
tf_idf_result = tf_idf_result.dropna().cache()
model = nb.fit(tf_idf_result)
```

类别预测

之后，我们读入原始数据中的商品信息，并进行数据清洗，然后将原始数据集的商品名称分词结果的转化为tf-idf-features

```
test_tf_feature = hashingTF.transform(test_df)
test_tfidf_features = idf_model.transform(test_tf_feature)
```

进行预测，并可视化预测结果

```
pred_result = model.transform(test_tfidf_features)
pred_class_df = pred_result.select("words", "prediction")
join_df = pred_class_df.join(dataset_df, pred_class_df.prediction ==
dataset_df.cate_ID, "inner").show()
```

words	prediction	cate_ID	category	name
[骆驼, 春夏, 运动, 休闲, ...	299.0	299	健康运动、户外	户外服饰
[骆驼, 春夏, 运动, 休闲, ...	299.0	299	健康运动、户外	户外服饰
[骆驼, 春夏, 运动, 休闲, ...	299.0	299	健康运动、户外	户外服饰
[骆驼, 春夏, 运动, 休闲, ...	299.0	299	健康运动、户外	户外服饰
[骆驼, 春夏, 运动, 休闲, ...	299.0	299	健康运动、户外	户外服饰

3 商品销量分类统计

3.1 要求

商品销量分类统计：根据2，统计各个类别销量TOP-N的商品。

3.2 实现

当知道了每个商品的类别之后，要计算这个类别销量TOP-N的商品并能够快速展示，我们只需要利用 spark-dataframe API 根据类别对所有商品的销售额进行排序，并输出 rank <= k 的商品即可。

以下是 item table 的基本信息。

shop_id	item_id	item_name	category	income
173275708	0	太平鸟男装 商务夹克男秋装外套刺绣...	男装	-1.0
173275708	11	太平鸟 冬季新款时尚灰色单排扣长款...	女装	39032.0

- 首先创建 spark sql window，利用 'income' （销量）作为排序指标降序排列

以下转换成 SQL 语句 等价于 PARTITION BY category ORDER BY income DESC

```
window = Window.partitionBy("category").orderBy(desc("income"))
```

- 在每一个类别里面进行排序，创建单独的一列 为 rank，并导出成文件，方便以后可视化获取 TOP-N 使用

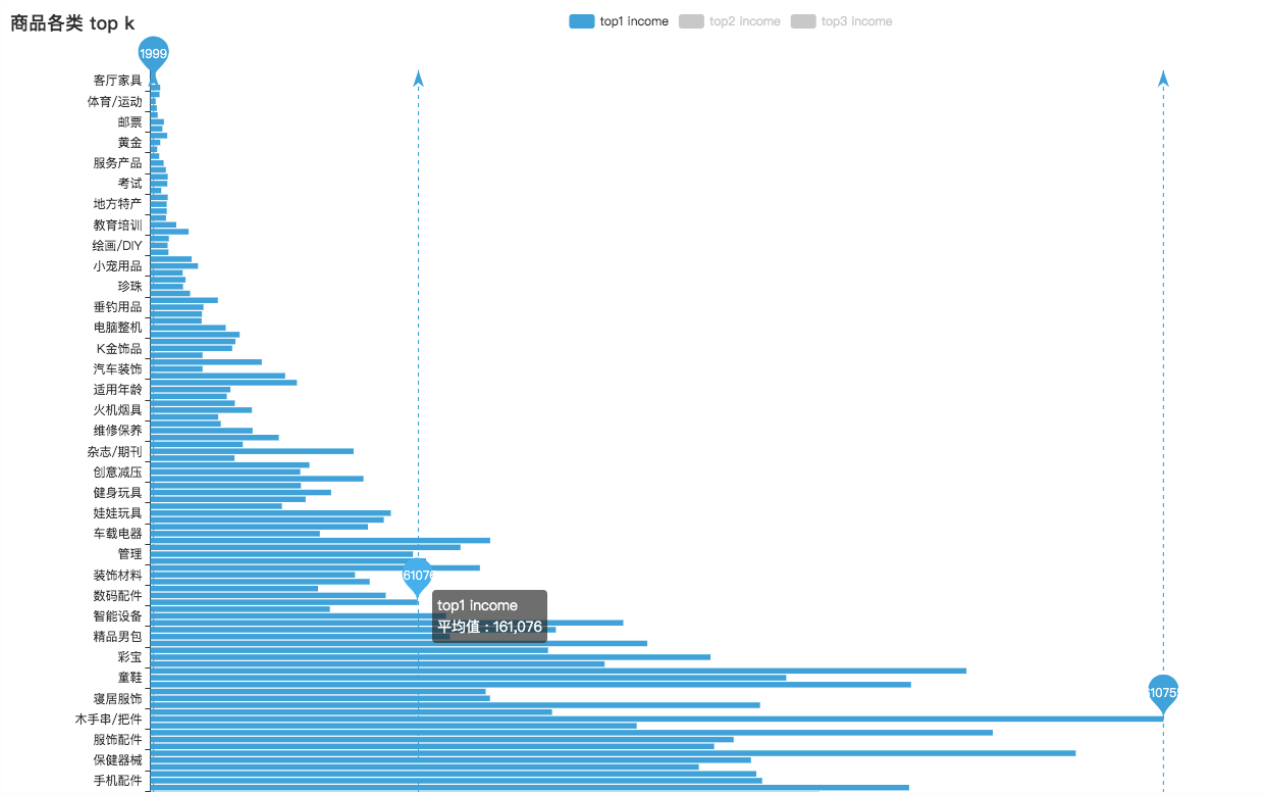
```
top_k_df = item_df.withColumn("rank", dense_rank().over(window))
```

- 获取 TOP-3 的商品，导出到另外一个 dataframe 作为我们最后展示的结果

```
top_3_df = top_k_df.where(top_k_df.rank <= k)
```

这是我们导出的部分结果，可以看到分类的结果和排序的结果都比较理想。

shop_id	item_id	item_name	category	income	rank
1954282799	44	Bally/巴利钱包男短款折叠钱夹...	精品男包	125910.0	1
2194800481	216	Loewe罗意威 Puzzle系列...	精品男包	97995.0	2
2929168542	149	COACH/蔻驰男士单肩包斜挎包商...	精品男包	90909.0	3
196993935	70	可自提 童装/男童 轻型WARM...	童装	6000660.0	1
1771485843	92	Teenie Weenie小熊20...	童装	4804200.0	2
196993935	151	可自提 童装/女童 轻型WARM ...	童装	2207790.0	3



这是其中每个商品类别 top-1 销量商品的直方图，其中最右边的线为销量最高的商品，中间的线是销量均值，上面的图例可以选择展示 TOP-1 TOP-2 还是 TOP-3。

4 店铺归类

4.1 要求

对所有店铺进行聚类，类别合理可解释。

4.2 实现

4.2.1 基本思路

- 每个商品根据类别生成 one-hot vector
- 每个店铺对其所有商品的 one-hot vector 求均值，得到店铺 feature
- 使用 Kmeans (k=10) 算法进行店铺聚类
- 每个类别的 center 在不同位置的权重代表了该 cluster 对不同类别的偏好，我们将前三大的权重类别作为每个 cluster 的类别信息

4.2.2 具体实现

在使用店铺聚类算法的时候，我们使用的仍然是上面题3的 item table。

- 首先我们先创建 类别到数字索引的一一映射，方便后续使用 spark 的算法进行one hot encode。我们使用的是Spark 提供的 StringIndexer 接口。

```
# 建立 string to index map
stringIndexer = StringIndexer(inputCol="category", outputCol="indexed",
                              handleInvalid="error",
stringOrderType="frequencyDesc")
indexer = stringIndexer.fit(item_df)
indexed_item_df = indexer.transform(item_df)
# 建立 index to string map
inverter = IndexToString(inputCol="indexed", outputCol='label',
labels=indexer.labels)
```

得到的结果如下表所示， category 和 indexed 一一对应。

shop_id	item_id	item_name	category	income	indexed
173275708	0	太平鸟男装 商务夹克男秋装外套刺绣...	男装	-1.0	0.0
173275708	14	太平鸟 红色时尚轻型羽绒服修身短款...	女装	-1.0	1.0

- 接下来我们把 index 编译为独热码，仍然使用 Spark 提供的 OneHotEstimator 。

```
ohe = OneHotEncoderEstimator(inputCols=["indexed"], outputCols=
["ohe_category"],
                             dropLast=False)
ohe_model = ohe.fit(indexed_item_df)
ohe_item_df = ohe_model.transform(indexed_item_df)
```

- 计算 每个商店的 feature，利用这个商店所有商品的类别相加取均值，这样比如一家店的 所有商品为（1，5， 0， 2），第一种类别的1件，第二种类别5件，第三、四种分别为0件和2件。取均值可以让每个类别的 数字在 0～1 之间，更利于类别的划分，比如H店 A类别有5件，J店A类别有10件，但本质上他们都只是拥有A类别的商品，我们看重的是他们拥有A类别商品占拥有总商品的比重，所以取了均值。

shop_id	shop_feature
2973180684	[0.0,0.0,0.0,0.0,...
1755496647	[0.97305389221556...
2101639169	[0.99107142857142...

- 利用 shop feature 使用 Spark MLib 的 Kmeans 算法进行聚类。

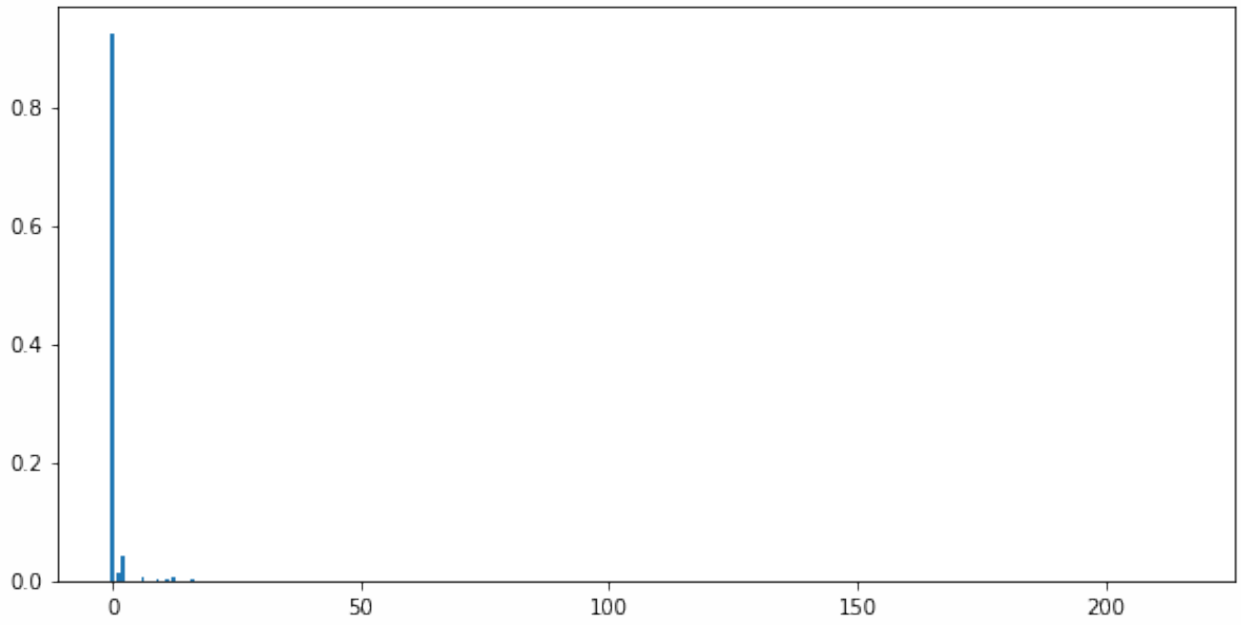
```
kmeans = KMeans(k=10, seed=623, featuresCol="shop_feature",
                predictionCol='cluster_id')
kmeans_model = kmeans.fit(shop_features)
shop_cluster_df = kmeans_model.transform(shop_features)
```

shop_id	shop_feature	cluster_id

2682628788	[0.0,0.0,0.0,0.0,...	0
2818345036	[0.0,0.0,0.0,0.0,...	8
2895810697	[0.0,0.0,0.0,0.0,...	5
2973180684	[0.0,0.0,0.0,0.0,...	5
2984563895	[0.0,0.0,0.0,0.0,...	2
3081409573	[0.0,0.0,0.0,0.0,...	0
1934470798	[0.0,0.0,0.0,0.0,...	0
3386085948	[0.0,0.0,0.0,0.0,...	0
1755496647	[0.97305389221556...	1

- 接下来我们求中心向量，并用直方图展示。

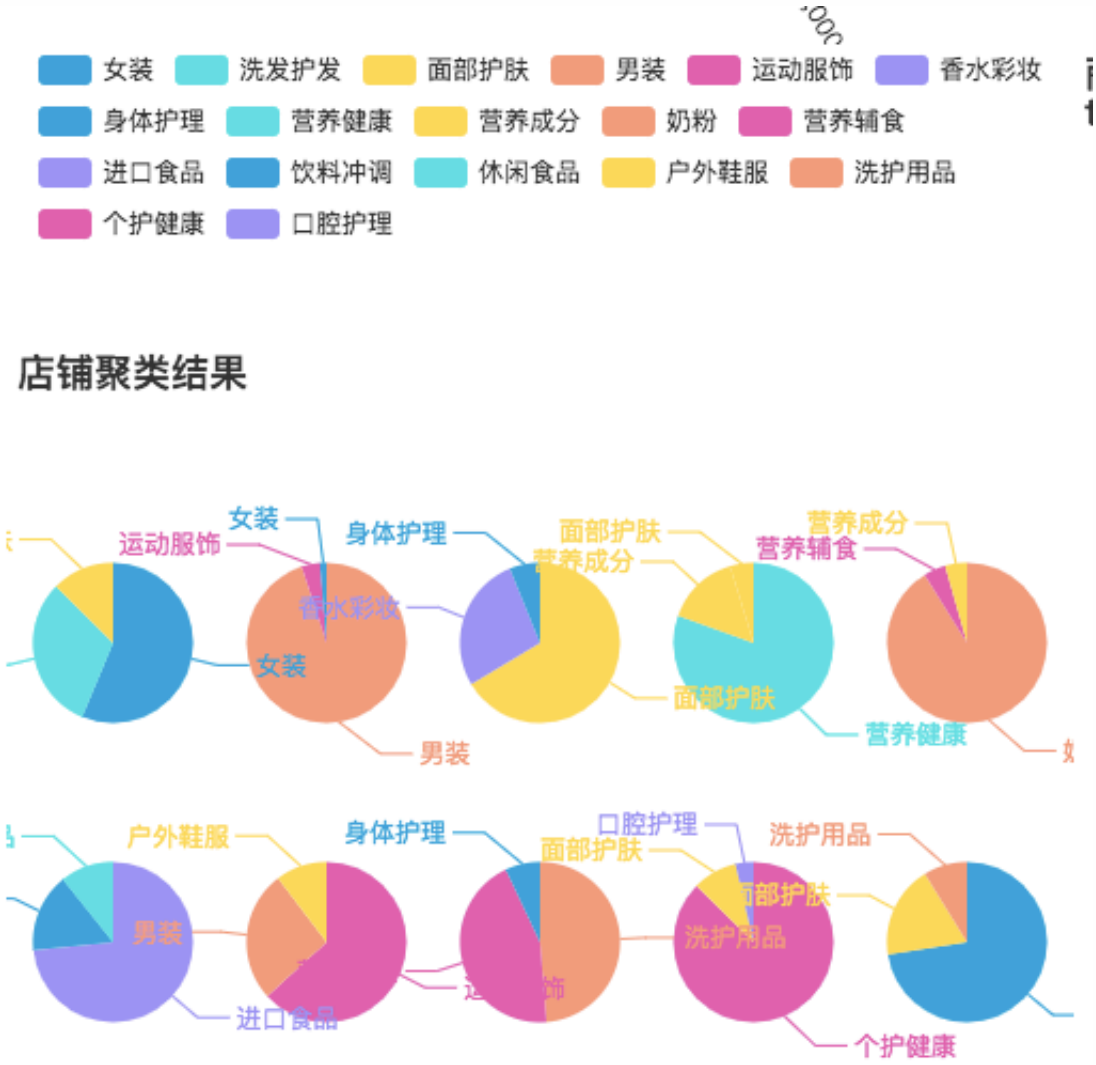
```
def identify_cluster_center(center):
    # 中心向量归一化，利于后面判别该中心每个类别的占比
    center = np.array(center) / np.sum(center)
    # 排序取其中占比前三的类别
    idxs = np.argsort(center)
    valid_category_idx = idxs[-3:][::-1]
    # 匹配 label 和对应的 weight
    valid_category = np.array(indexer.labels)[valid_category_idx]
    valid_category_weight = np.array(center)[valid_category_idx]
    valid_cate_name_weight = list(zip(valid_category,
valid_category_weight))
    return valid_cate_name_weight
```



以上是其中一个 center 的直方图，其中男装的占比最大，超过90%，可见聚类效果较好。

下表是我们聚出来的十个类，我们以每类中心的占比最高商品来对这个类进行类别赋值，可以看到第一类主要的商品为女装，第二个类主要商品为男装，其中男装占比高达92.42%，可见这个类分得比较成功。剩下的洗发护发、营养健康、奶粉、面部护肤、香水彩妆等分类都比较好，其中超过六成以上的商品都是该类别商品。

cluster_id	info
0	女装:0.127285205561...
1	男装:0.924244598583...
2	洗发护发:0.7374914105...
3	营养健康:0.6515122920...
4	奶粉:0.759353671809...
5	洗护用品:0.3893009653...
6	面部护肤:0.7206643061...
7	进口食品:0.5920609811...
8	香水彩妆:0.6854944437...
9	身体护理:0.5375684661...



以上是我们十聚类的结果，每个类别都有一些占绝对优势的类别，可见店铺聚类的效果还是不错。

5 店铺销量分类统计

5.1 要求

根据4，统计各个类别总销量TOP-N的店铺。

5.2 实现

接下来我们统计各个类别销量 TOP-N的店铺，我们首先统计每个店铺的销量总和，接着按销量对每个类别的店铺进行排序，最后输出包含 rank 的表格，以便后来可视化展出。

整个计算流程同第三道题。

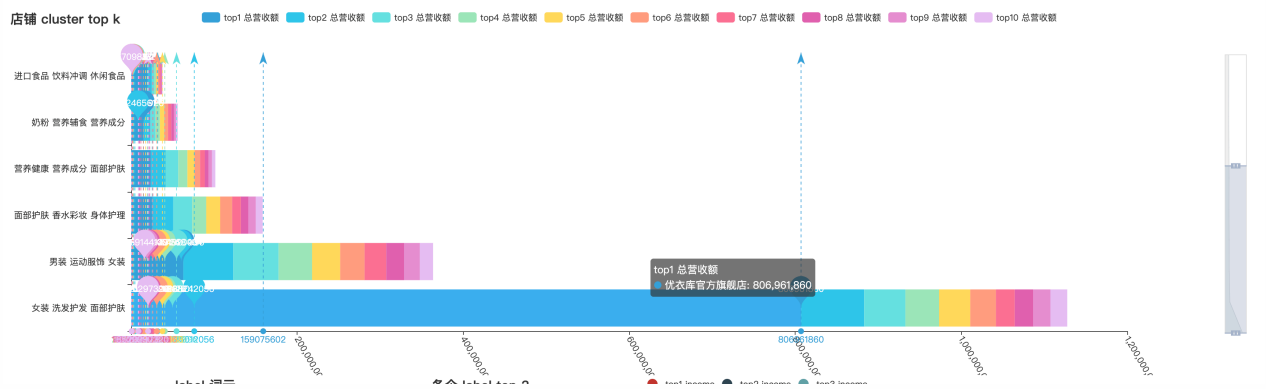
- 首先创建 window 方便使用 Spark SQL API 并就 销量 进行排序

```
window = Window.partitionBy("cluster_id").orderBy(desc("total_income"))
shop_top_k_df = shop_join_cluster_df.withColumn("rank",
dense_rank().over(window))
```

- 接下来计算 top 3 的销量店铺

shop_id	shop_name	total_income	shop_feature	cluster_id	rank
305358018	JackJones官方旗舰店	6.2685036E7	[0.82832948421862...	1	1
289268212	真维斯官方旗舰店	6.0226404E7	[0.55802640722724...	1	2
188124207	唐狮官方旗舰店	5.4652096E7	[0.56094364351245...	1	3
2815462940	AHC官方海外旗舰店	2.5919564E7	[0.0,0.0,0.0,0.0,...	6	1
2963458513	MARTIDERM海外旗舰店	2.4726656E7	[0.0,0.0,0.0,0.0,...	6	2
3014868997	dm海外旗舰店	2.2844176E7	[0.0,0.0,0.0,0.0,...	6	3
2794371653	swisse官方海外旗舰店	2.3755544E7	[0.0,0.0,0.0,0.0,...	3	1
2415882532	GNC健安喜官方海外旗舰店	1.7735794E7	[0.0,0.0,0.0,0.0,...	3	2
2316003304	BLACKMORES海外旗舰店	1.5239114E7	[0.0,0.0,0.0,0.0,...	3	3

从结果来看，整个聚类结果也非常理想，其中男装店铺的前三是 JackJones、真维斯、唐狮，这和我们的日常感受也非常相近，而营养健康类别分别是 swisse、 GNC、BLACKMORES，也是这个领域的一些非常有名的品牌。



这是我们店铺聚类的 TOP-K，其中（女装、洗发护发、面部护肤）店铺类别的 top1 销量最高，TOP-1 店铺 优衣库销量达到 8 亿+。

6 店铺标签设计

6.1 要求

根据店铺信息、店铺中的商品信息、评论信息等为店铺生成若干个关键词标签。

6.2 实现

6.2.1 基本思路

在给店铺打标签时，我们的基本思路为，结合2-5题的信息（商品的一级标签和二级标签，不同类别商品销量Top-5，店铺类别（二级标签kmenas聚类），各雷贝店铺销量Top-5），再利用数据集中的商品价格，商品评论数量，商品评论情感倾向，评论关键词等信息，对信息进行较为简单的分类，自主设计标签，使得算法简单易行同时效果显著。

我们设计了如下的店铺标签。

商品类别：二级标签，选较多的商品类别二级标签

"最多评论"：reviewcount

"今日爆款"：店铺类别top-n的单品数量较多

"便宜好货"/"高端定制"/"中产最爱"：相同行业平均价格（总销售额/销量） revenue/sales volume

"金牌卖家"：总销售额/销量行业领先，类别排名排名数据

“好评如潮”：评论情感倾向较为positive

"品牌老店"/"个性小店"：商品总数较多/商品总数较少

6.2.2 具体实现

店铺主导商品的二级标签

如果某店铺的某类产品超过店铺总数的15%，则将该类别作为店铺标签，最终的结果DF包含两列，一列尾shop_id，一列为主导店铺标签的列表（list）

shop_id	collect_list(category)
2895810697	[喂养用品, 洗护用品]
2973180684	[香水彩妆, 洗护用品]
2984563895	[洗发护发]
2682628788	[传统滋补, 营养健康, 饮料冲调]

最多评论

选取各个类别中的评论数量top-10，添加"最多评论"标签

```
comment_count_df = sentiment_df.groupBy('shop_id').agg(F.count(sentiment_df.item_id))
comment_count_df = comment_count_df.withColumnRenamed('count(item_id)','count_comment')
window = Window.orderBy(desc("count_comment"))
top_comment_df = comment_count_df.withColumn("rank", rank().over(window))
top_comment_df = top_comment_df.withColumn('label',F.when(F.col('rank')<=100,'最多热评').otherwise(''))
```

好评如潮

评论数排名第101开始的店铺评论数为2000多且相差不大。 选择评论数>1000且评价情感分析平均值前100的店铺给予“好评如潮”标签。

```
shop_goodcomment_df = sentiment_df.groupBy('shop_id').agg(F.avg(sentiment_df.sentiment))
window = Window.orderBy(desc("avg(sentiment)"))
shop_goodcomment_df = shop_goodcomment_df.withColumn("rank", rank().over(window))
shop_goodcomment_df = shop_goodcomment_df.withColumn('gclabel',F.when(F.col('rank')<=100,'好评如潮').otherwise(''))
```

品牌老店/个性小店

商品总数较多/商品总数较少；商品总数>10的为品牌老店；商品总数>30且小于等于70的为个性小店。

```
item_count_df = item_df_item_count.withColumn('old_young_label',F.when(F.col('count(item)')>100,'品牌老店').otherwise(''))
item_count_df = item_count_df.withColumn('old_young_label',F.when((F.col('count(item)')<=70)&(F.col('count(item)')>=30),'个性小店').otherwise(F.col('old_young_label')))
```

爆款生产商

即该店铺产出爆款产品数量大于等于2.此处爆款产品为类别销售收入top-5。

```
window = Window.partitionBy("category").orderBy(desc("income"))
hot_item_df =
item_df.drop('item_name').withColumn('rank',rank().over(window))
hot_item_df = hot_item_df.where(hot_item_df.rank<=5)
hot_item_count_df =
hot_item_df.groupBy('shop_id').agg(F.count(hot_item_df.item_id))
hot_item_count_df =
hot_item_count_df.withColumnRenamed('count(item_id)','hot_item_count')
hot_item_count_df =
hot_item_count_df.withColumn('hotitem_label',F.when(F.col('hot_item_count'
)>2,'爆款生产商').otherwise(''))
```

金牌卖家

即在聚类而成的各个类别中的商家top-5

```
gold_seller_df = shop_cluster_df.select('shop_id','rank')
gold_seller_df =
gold_seller_df.withColumn('goldseller_label',F.when(F.col('rank')<=5,'金牌
卖家').otherwise(''))
```

便宜好货/中产最爱/高端定制

总销售额/总销量。前1/3为高端定制，中间1/3为中产最爱，后面1/3为便宜好货。

标签整合与数据处理

数据处理主要是将dominate_cate的列表转化为一个空格间隔的字符串，标签整合是最后将所有的标签转为一个以空格间隔的字符串。

首先将之前所得的DF合并为在一起。

```
shop_label_df = shop_price_level_df.select('shop_id','price_label')\
    .join(hot_item_count_df,on=
['shop_id'],how='full').join(gold_seller_df,on=['shop_id'],how='full')\
    .join(item_count_df,on=
['shop_id'],how='full').join(item_dominate_cate,on=
['shop_id'],how='full')\
    .join(top_comment_df,on=
['shop_id'],how='full').join(shop_goodcomment_df,on=
['shop_id'],how='full')

  shop_label_df =
shop_label_df.drop('rank','avg(sentiment)','hot_item_count','count(item)',
'count_comment')
```

将有多个类别标签列表转为字符串。最后将所有的标签整合在一起。最终各店铺的标签如下：

```
shop_label_final_df.show()
```

shop_id	final_labels
1934470798	高端定制 金牌卖家 个护健康
2682628788	中产最爱 爆款生产商 个性小店 ...
2818345036	便宜好货 香水彩妆
2895810697	便宜好货 喂养用品 洗护用品
2973180684	便宜好货 香水彩妆 洗护用品
2984563895	中产最爱 洗发护发
3081409573	高端定制 传统滋补

8 识图找物

8.1 要求

每个商品有若干张描述图片，对商品图片进行聚类。要求给定一张数据集外的商品图片，返回相关商品。

8.2 实现

8.2.1 基本思路

- 利用爬虫将每个商品的一张图片爬取
- 对每张图片 resize 到 50*50，归一化，进行 svd 分解，取一条 U，V 向量拼接称为图片的 feature，并存入相应 table
- 使用 knn 算法进行相似推荐

8.2.2 具体实现

第八题的题目需求里包含两个部分，第一个是图片聚类，第二个是图片相似度计算。

- 我们将图片 resize 到 50 * 50 并归一化处理，利用 SVD分解得到 U V向量，并 concat 到一起，得到图片的 feature。
- 接下来我们创建 image dataframe 方便后续进行操作。

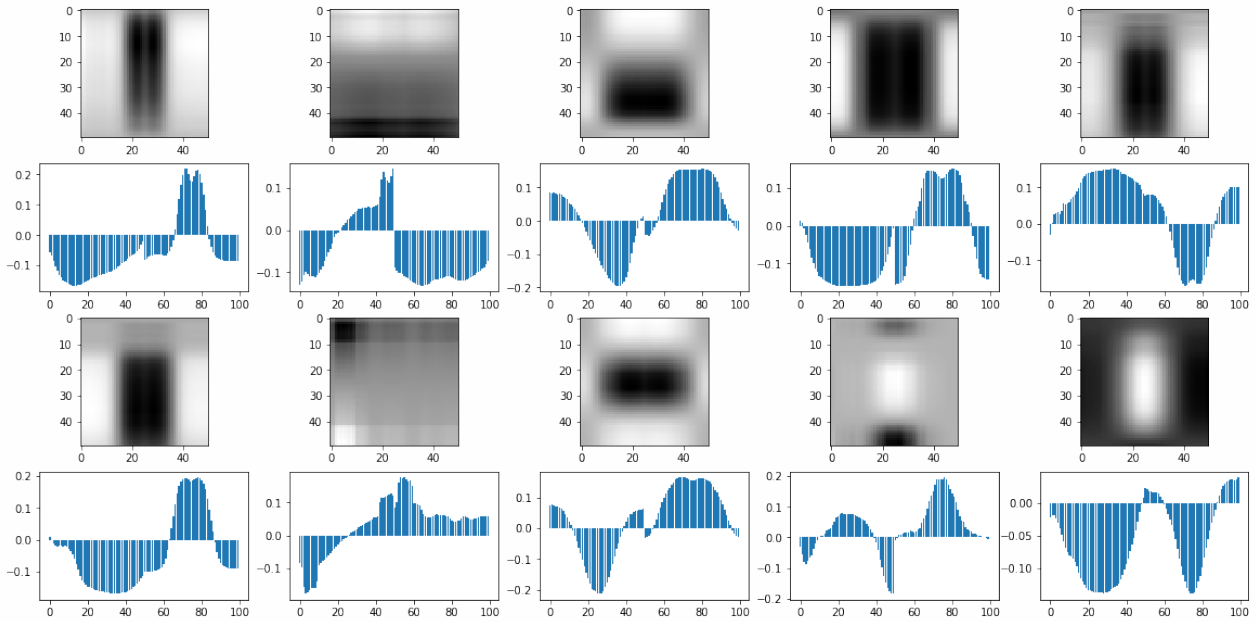
```
item_pic_feature_df = spark.createDataFrame(raw_data).join(item_pic_df,
['shop_id',

'item_id'],how='right')
```

此时我们得到的 dataframe 包含 以下四个 column： shop_id、item_id、feature、img_url信息。

- 接下来我们利用 feature 信息做图片的 Kmeans 聚类。

```
kmeans = KMeans(k=10, seed=623, featuresCol="feature",
predictionCol='cluster_id')
kmeans_model = kmeans.fit(item_pic_feature_df.dropna())
item_pic_cluster_df = kmeans_model.transform(item_pic_feature_df.dropna())
```

同样聚出十个类，聚类的结果如下：



这里的蓝色图是两百维向量及其对应的取值，上面的图片是经过 SVD反变换之后的结果。

- 之后我们使用 Spark MLib 的 KNN 算法进行相似图片计算

以下是相似度评分，相似度评分使用的是 feature 的欧式距离。

```
def compute_dist(vec, query_feature):
    if vec is None:
        return 100000
    vec_arr = vec.values
    result = 0
    for idx in range(len(vec_arr)):
        result += (query_feature[idx] - vec_arr[idx]) ** 2
    return float(result)

def udf_score(query_feature):
    return udf(lambda vec: compute_dist(vec, query_feature), FloatType())
```

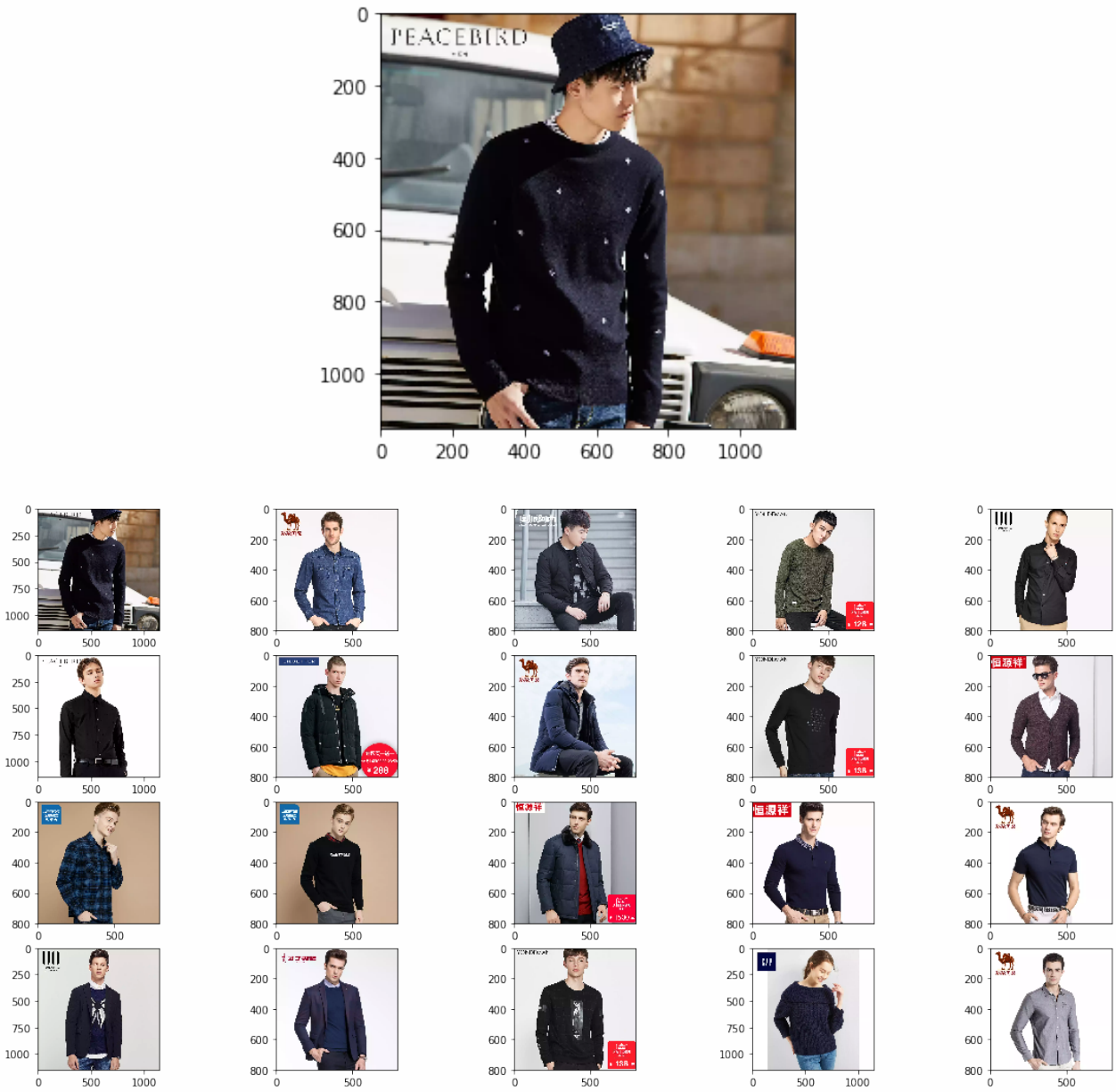
- 计算 传入图片 和所有图片的相似度

```
neighbor_df = item_pic_cluster_df\
                .withColumn("dist", udf_score(query_feature_vec)
                                (item_pic_cluster_df.feature).astype("float"))
```

- 取 距离最小的 k 个

```
top_k_neighbors_df = neighbor_df.orderBy(neighbor_df.dist)
top_k_neighbors = top_k_neighbors_df.head(k)
```

以下是效果展示，第一张是原图，后面是最相似的20张图片。



从图片中看到，都是单人、同样的姿势、黑色衣服，相似度还是比较高的，只是并没有针对商品进行进一步优化，但也算完成了不错的识图找物。

9 相似推荐

9.1 要求

基于前面的归类，要求给定数据集中的一件商品，返回相似的TOP-N商品，需有可解释的推荐理由。

9.2 实现

9.2.1 基础思路

- 在这之前，我们已经计算出了每个商品的类别、销量、好评率、差评率
- 相似推荐首先仅推荐同一类商品
- 再将销量、好评率、差评率组合，通过 min-max-scale 之后成为商品 feature
- 利用 knn 算法进行商品相似推荐

9.2.2 具体实现

- 首先我们创建一个新的表，把商品和对应的评论 join 起来，该表有shop_id 、item_id、good_rate、bad_rate、good_rank、bad_rank 等column。为便于展示，我们下表省略了其中shop_id 、item_id、good_rate、bad_rate等column。

good_rank	bad_rank	item_name	category	income
216	463	太平鸟男装 商务夹克男秋装外套刺绣...	男装	-1.0
808	298	太平鸟男装秋冬新款黑色休闲裤商务青...	男装	-1.0
551	384	太平鸟男装 秋蓝色牛津纺长袖衬衫男...	男装	-1.0
1250	1	太平鸟男装新款男士修身显瘦直筒小脚...	男装	-1.0
1	1082	太平鸟男装 秋冬薄款纯棉白色修身衬...	男装	-1.0

- 结合上面的 table 创建商品的 feature vactor

```
# 利用 good_rate 和 bad_rate 生成feature向量
feature_df = new_item_df.withColumn("feature",
                    array_to_dense_vector(array('good_rate', 'bad_rate',
'income')))
# 使用 minmax scale
mm_scaler = MinMaxScaler(inputCol='feature', outputCol='mm_scale_feature')
mm_scaler_model = mm_scaler.fit(feature_df)
new_feature_df = mm_scaler_model.transform(feature_df)
```

- 再利用 KNN 算法进行相似度计算，其中我们的计算标准有以下几个要求。
  - 类别相同
  - 价格相近
  - 好评率相近
  - 差评率相近

```
neighbor_df = new_feature_df.where(new_feature_df['category'] ==
category)\
                .withColumn("dist", udf_score(query_feature)
                            (new_feature_df.mm_feature))
```

从这里看到，首先我们会判断 category 筛选出相同类别的商品，然后判断 feature 的similarity，返回相似度最高的商品。其中feature 由好评率、差评率、销量组成，所以相近的商品必然具有相近的价格、好评率、差评率。

- 返回结果

```
neighbor_df.orderBy(neighbor_df.dist).select("item_name", "good_rate",
"bad_rate",
                                             "income", "dist").show(k)
```

我们给出的商品为男装类别、拥有 0.5 的好评, 0.3636363744735718 的差评, 销量为0.

item_name	good_rate	bad_rate	income	dist
太平鸟男装 秋蓝色牛津纺长袖衬衫男...	0.5	0.36363637	-1.0	0.0
JOW/乔沃纯棉短袖polo衫男修...	0.5	0.36363637	490.0	2.3697244E-10
2017春秋新款TOMMY HIL...	0.5	0.36363637	2550.0	6.3966974E-9
男士日系哈伦裤男秋季宽松裤子 青少...	0.5	0.36363637	4266.0	1.7896998E-8
Baleno班尼路男装 纯棉修身牛...	0.5	0.36363637	4928.0	2.3881E-8

这是返回的结果，其中太平鸟男装是我们搜索的结果，剩下4个是在我们规定的相似度下，最为接近的结果。

10 情感分析

10.1 要求

分析商品的firstNReviews中的评论和追评，归类为好评、中评、差评。列出差评率最高和好评率最高的TOP-N店铺和TOP-N商品。

10.2 实现

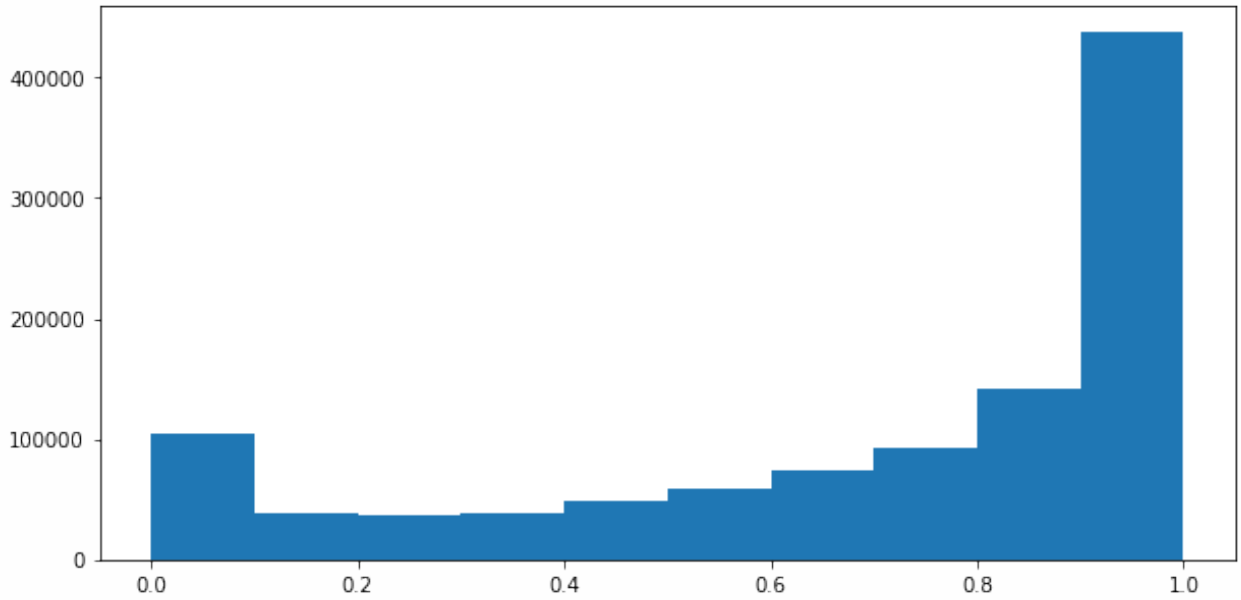
情感分析部分功能我们采用了 SnowNLP 的库。

```
review_df = review_df.withColumn("sentiment",
review_sentiment(review_df.content))
```

shop_id	item_id	content	sentiment
173275708	0	夹克还可以，不薄不厚，适合秋天，刺...	0.9734554797987359
173275708	0	版型超正，立体刺绣，料子厚重，做工...	0.9997615920112265

173275708	0	这件衣服在光线下有点发红，稍有点沾...	0.003099763076949147
173275708	1	其实裤子质量还挺好的，第一次买的时...	0.006074743831211871

其中越靠近1 的评论好评度越高，越靠近 0 好评度越低，从随机抽出的几条评论来看，SNOWNLP 基本能够正确将评论分类。



这是我们对所有评论的情感分析占比的直方图展示，其中好评占比最高，差评占比次高，中间评价占比较少，这跟常识也是相符的，通常我们都倾向于给所购买的东西打好评，我们在网路购物浏览到的也都是好评居多。

如果按照分位数来分的话，整体情感分数偏高，为了方便处理，我们将情感分析结果分为好中差三档评论，直接设置 boundary, 大于 0.8 的才作为好评

- 好评：0.8 ~ 1
- 中评：0.5 ~ 0.8
- 差评：0 ~ 0.5

shop_id	item_id	content	sentiment	sentiment_class
173275708	0	夹克还可以，不薄不厚，适合秋天，刺...	0.9734555	1（好评）
173275708	0	版型超正，立体刺绣，料子厚重，做工...	0.9997616	1
173275708	0	这件衣服在光线下有点发红，稍有点沾...	0.003099763	-1（差评）
173275708	1	其实裤子质量还挺好的，第一次买的时...	0.006074744	-1
173275708	1	太平鸟的衣服还是一如既往的好，双十...	0.99959403	1
173275708	2	28天后追评 扫一...	0.77780575	0（中评）

计算好评率、中评率和差评率

- 首先对每件商品对所有评论进行归类

```
item_sentiment_df = review_df.groupBy(["shop_id",
"item_id"]).agg(collect_list('sentiment_class'))\
.withColumnRenamed("collect_list(sentiment_class)","sentiment_list")
```

- 计算好评和差评率

```
def cal_good_rate(sentiment_list):
    num_good = 0
    for sentiment in sentiment_list:
        if int(sentiment) == 1:
            num_good += 1
    return num_good / len(sentiment_list)

def cal_bad_rate(sentiment_list):
    num_bad = 0
    for sentiment in sentiment_list:
        if int(sentiment) == -1:
            num_bad += 1
    return num_bad / len(sentiment_list)
```

- 生成 item_review_rate 表格

```
item_review_rate =  item_sentiment_df.withColumn("good_rate",
                        cal_good_rate(item_sentiment_df.sentiment_list))\
                       .withColumn("bad_rate",
cal_bad_rate(item_sentiment_df.sentiment_list))\
                                    .drop("sentiment_list")
```

shop_id	item_id	good_rate	bad_rate
1016228978	190	0.6666667	0.0
1016228978	224	1.0	0.0
1023428922	16	0.75	0.0
1026616278	148	1.0	0.0

返回商品好评率、差评率TOP-K

基本思路仍然同之前的销量TOP-K，只需按 good_rate 和 bad_rate 排序，每个商品都得到一个 rank 指标，取出所有rank <= k 的商品返回即可。

```
# 降序排列
good_rate_window = Window.orderBy(desc("good_rate"))
bad_rate_window = Window.orderBy(desc("bad_rate"))
# 生成 good_rank bad_rank
tmp = item_review_rate.cache().withColumn("good_rank",
dense_rank().over(good_rate_window))\
                            .withColumn("bad_rank",

 dense_rank().over(bad_rate_window))
# 取出 top-5
tmp.where("good_rank <= 5")
```

计算店铺好评率、差评率TOP-K

店铺好评率和差评率仅仅是把所有商品的好评率、差评率求均值，最后把所有店铺的好评差评率排序，得到rank，根据给定的 k 返回 TOP-K的店铺。

```
# 按 shop_id 汇总所有的评论
shop_sentiment_df =
item_sentiment_df.groupBy("shop_id").agg(collect_list(item_sentiment_df.sentiment_list))
# 遍历所有的评论计算好评率，差评率类似
def shop_good_rate(collect_list):
    total_count = 0
    good_count = 0
    for sentiment_list in collect_list:
        total_count += len(sentiment_list)
        for sentiment_class in sentiment_list:
            if int(sentiment_class) == 1:
                good_count += 1
    return good_count / total_count

# 创建 dataframe
shop_review_rate_df = shop_sentiment_df.withColumn("shop_good_rate",
shop_good_rate(shop_sentiment_df['collect_list(sentiment_list)']))\
                        .withColumn("shop_bad_rate",
shop_bad_rate(shop_sentiment_df['collect_list(sentiment_list)']))\
                        .drop("collect_list(sentiment_list)")
```

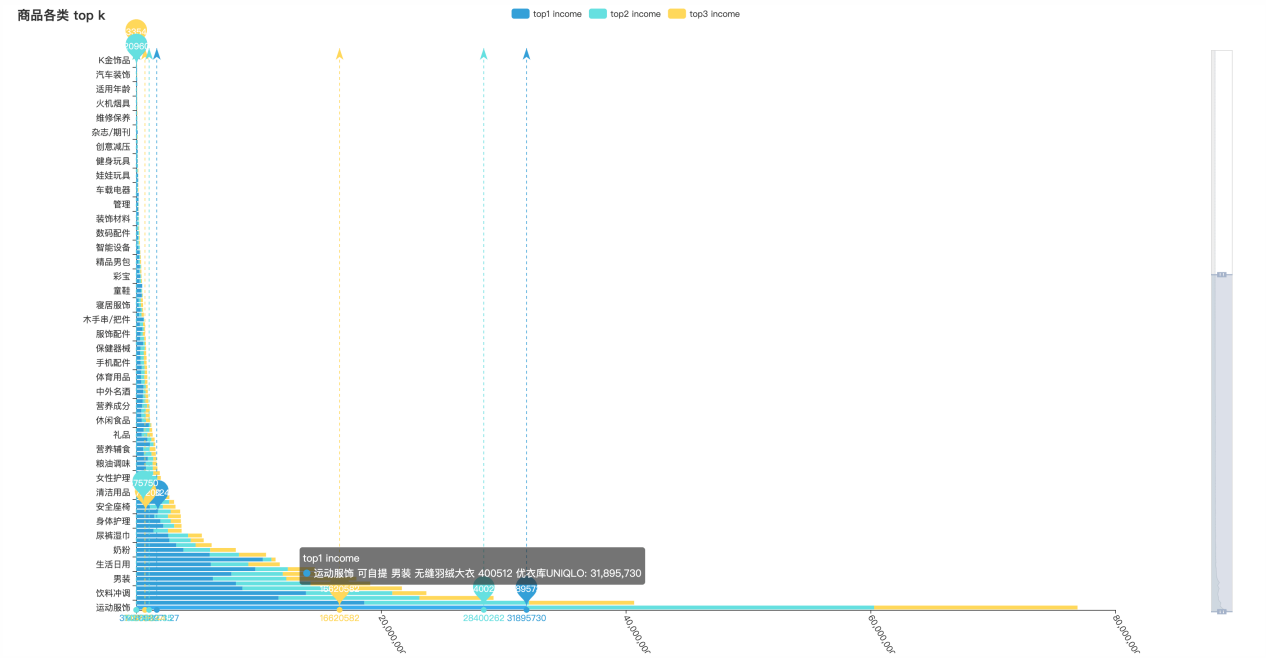
shop_id	shop_good_rate	shop_bad_rate
3081409573	0.68604654	0.18604651
3386085948	0.48076922	0.34615386
2682628788	0.4431818	0.2784091
2973180684	0.55813956	0.22093023

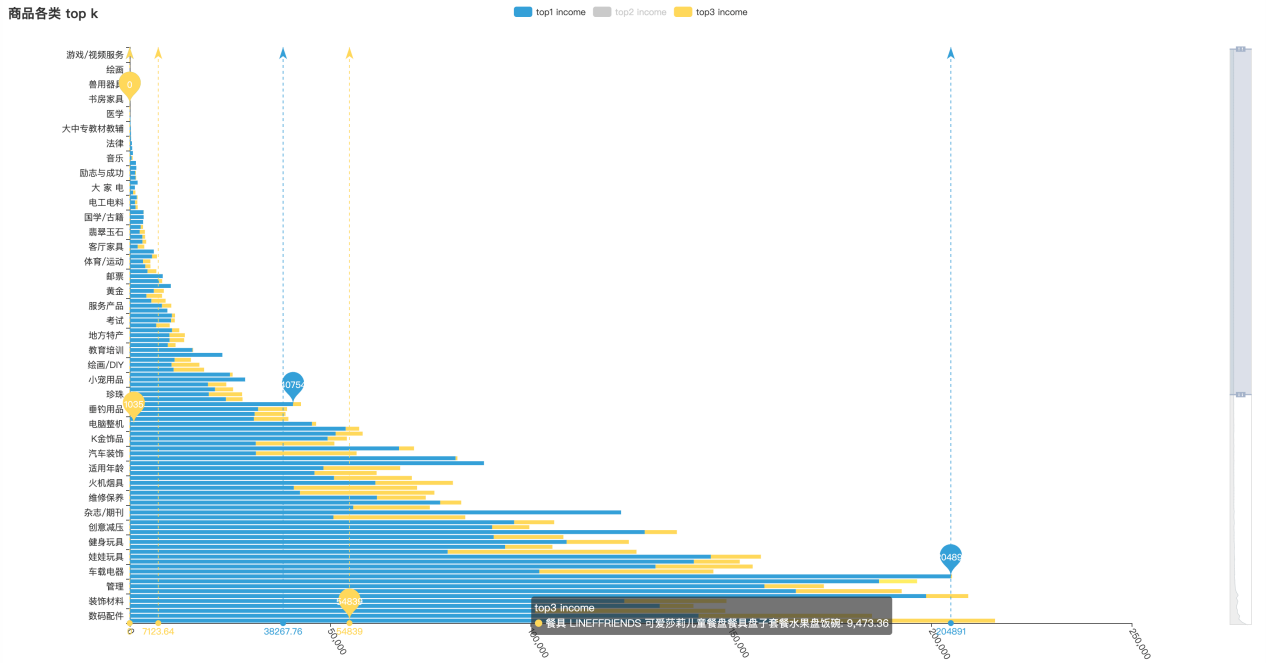
而取出店铺 TOP-K的方法则跟取出商品 TOP-K的方法一模一样，在此不做赘述。

三、大数据看版搭建

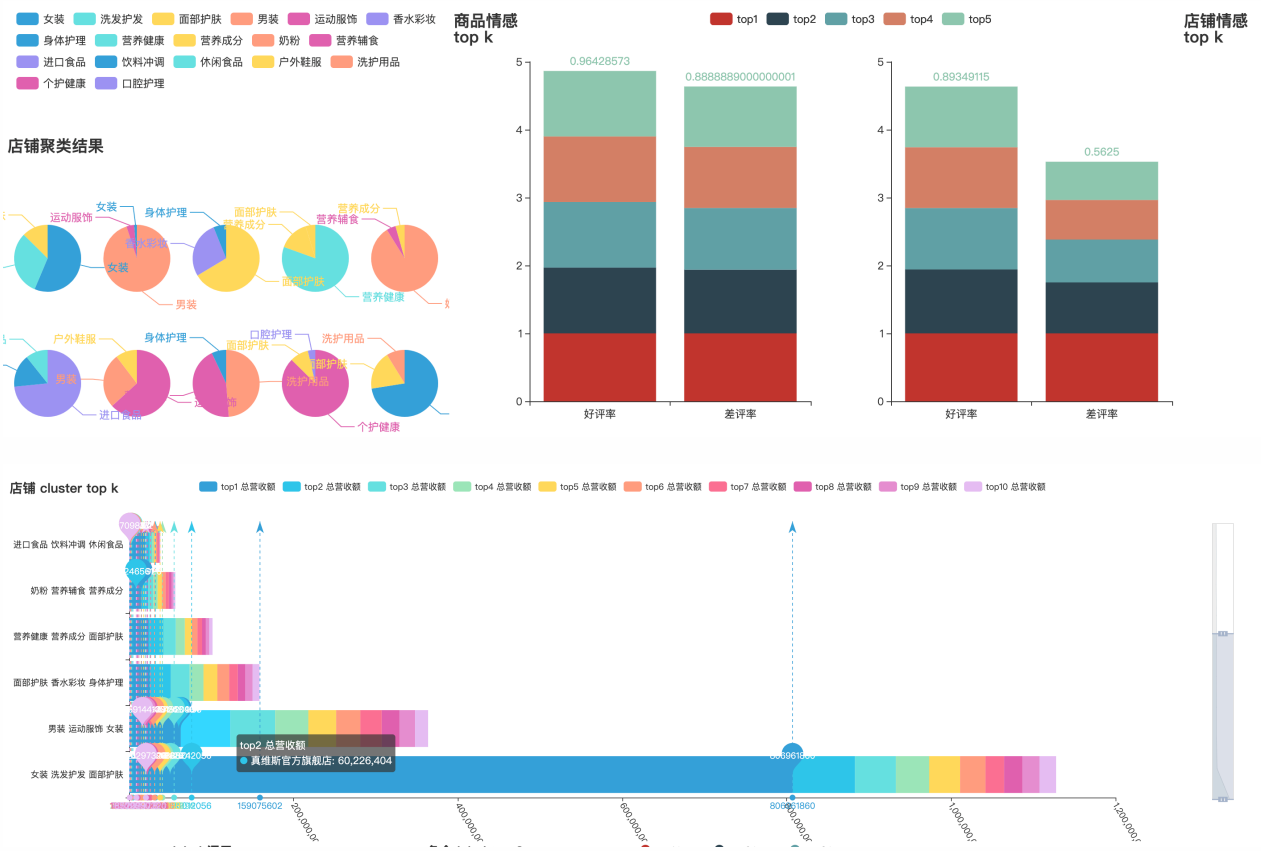
在这一部分中，由于小组成员均缺乏前端经验，我们采用了 pyecharts 作为一个较为简单但不失丰富性的解决方案。pyecharts 作为 echarts 的 python 接口（非官方），为用户提供了更加简单自然的 API，让用户可以在短短几行中设计出展示效果好看，信息量丰富的可视化图片。

在第二题的输出结果中，我们产出了 216 类不同的商品，且每个商品类别均预计算出 top 3 销量的商品，将如此庞大的信息量展现在一张图上不是一件简单的事情。我们最终采用了层叠直方图的解决方案，允许用户选择展现 top k 中的几个，并且可以通过拖动右边 bar 的方式来观看不同类别商品的信息。

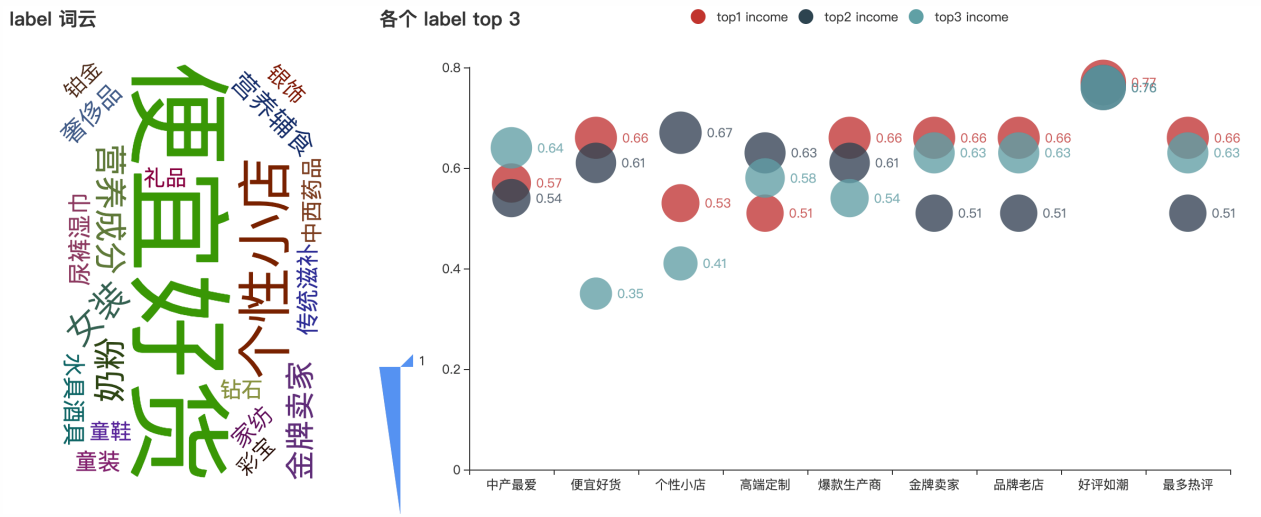




在 4,5,10 的可视化展示中，我们期望可以首先向用户展示不同店铺 cluster 所包含的权重最高的三类商品，以及一个类似于商品 top k 的直方图展示。第一点需求可以通过饼图来解决，第二点需求依照商品 top k 的可视化方案通过层叠的直方图解决，第三点需求则直接通过简单的直方图展现即可。



在 6，7的可视化展示中，我们首先需要展现标签的词云，第二需要展示不同标签的 top k 店铺，不同于之前的 top k，此时每个店铺有销量和好评率两个维度，所以我们决定通过散点图进行展示，y 轴代表好评率，不同的点代表 top 1/2/3 的销量。



四、总结

在本次项目中，我们将处理流程解耦为 offline 与 online 两个过程，将数据映射到一个类 database 的 table 集合，设计了基于天猫商品数据集的大数据分析系统。在算法设计上，我们拓展目光，融合了 naive bayes, kmeans, knn, svd, tf-idf, sentiment analysis 等多种算法，在各题中均取得了高响应、可解释性强的结果；在大数据架构上，我们采用了 hdfs, spark 作为存储、处理的基石，处理并生成了大数据系统内部的一个类 database 的 table 集合，为后续的相似推荐与可视化提供精准高效的信息；在可视化方案上，我们采用了简单易用的 echarts 工具，精心设计了简洁美观且蕴含丰富信息量的大数据看版，且提供一定程度上的交互，让用户可以更快更好的理解数据集中的重要信息。未来，我们希望完善前端，允许用户可以在前端页面中进行实时查询，同时继续优化后端的算法细节，在诸如情感分析、图像聚类的算法中加入神经网络的辅助，得到更加具有说服力的结果，