|  |
| --- |
| **机器学习** |
| **罗斯曼商店销售** |
| 主 研 人：刘一阳  参 研 人： |
| 审 核 人：  方 向：工业算法案例研究  版 本 号：A |
|  |

**版本更新**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **更新描述** | **作者** |
| 2019/04/04 | A | 初稿 | 刘一阳 |

**选题表格**

|  |  |  |  |
| --- | --- | --- | --- |
| 时间 | 竞赛名 | 竞赛背景描述（50字以内） | 类型（分类/回归） |
| 2019/04/04 | Home Depot产品搜索相关性（Home Depot Product Search Relevance） | 购物者依靠Home Depot的产品权威来寻找和购买最新产品，并及时获得满足其家居装修需求的解决方案。在本场比赛中，Home Depot要求Kagglers通过开发能够准确预测搜索结果相关性的模型来帮助他们改善客户的购物体验。 | 分类 |

目录

[1. 背景描述 4](#_Toc25575_WPSOffice_Level1)

[1.1 竞赛赛题描述 4](#_Toc5940_WPSOffice_Level2)

[1.2 评估指标描述 5](#_Toc12357_WPSOffice_Level2)

[2. 数据来源及描述性统计分析 5](#_Toc5940_WPSOffice_Level1)

[2.1 大赛数据来源 5](#_Toc22683_WPSOffice_Level2)

[2.2 数据的描述性统计 5](#_Toc16174_WPSOffice_Level2)

[2.2.1 数据基本情况描述： 5](#_Toc5940_WPSOffice_Level3)

[2.2.2 数据字段介绍： 5](#_Toc12357_WPSOffice_Level3)

[2.2.3 数据描述性统计 6](#_Toc22683_WPSOffice_Level3)

[3. 优秀算法思路 6](#_Toc12357_WPSOffice_Level1)

[3.1 方案一 6](#_Toc32293_WPSOffice_Level2)

[3.1.1 方案一数据预处理及特征工程部分方案 6](#_Toc16174_WPSOffice_Level3)

[3.1.2 方案一模型设计、建立部分方案 6](#_Toc32293_WPSOffice_Level3)

[3.1.3 方案一结果、排名等 6](#_Toc29922_WPSOffice_Level3)

[3.1.4 方案一算法流程图 6](#_Toc4787_WPSOffice_Level3)

[3.2 方案二 7](#_Toc29922_WPSOffice_Level2)

[… 8](#_Toc4787_WPSOffice_Level2)

[4. 算法比较 8](#_Toc22683_WPSOffice_Level1)

[表4-1 算法比较 8](#_Toc7253_WPSOffice_Level2)

[5. 总结与展望 8](#_Toc16174_WPSOffice_Level1)

[5.1 总结 8](#_Toc23198_WPSOffice_Level2)

[5.2 建模思路 8](#_Toc16619_WPSOffice_Level2)

1. 背景描述

家得宝即美国家得宝公司。为全球领先的家居建材用品零售商，美国第二大零售商，家得宝遍布美国、加拿大、墨西哥和中国等地区，连锁商店数量达2234家。家得宝连续9年被美国《财富》杂志评为“最受欢迎的专业零售商”。并在2007年美国财富500强中排名第17位，2006年全球财富500强排名第43位，同年被美国《财富》杂志评为“最受仰慕的专业零售商”第1位及“最受仰慕的公司”第13位。在美国共有1200多家店铺，海外有133家店铺。《财富》杂志于北京时间2017年6月7日晚发布了最新的美国500强排行榜，家得宝排名23。2017年6月，《2017年BrandZ最具价值全球品牌100强》公布，家得宝排名第24位。2018年7月19日，《财富》世界500强排行榜发布，家得宝位列57位。

**1.1 竞赛赛题描述**

购物者依靠Home Depot的产品权威来寻找和购买最新产品，并及时获得满足其家居装修需求的解决方案。从安装新的吊扇到改造整个厨房，只需点击鼠标或点击屏幕，客户就可以快速获得正确的查询结果。速度，准确性和无摩擦的客户体验至关重要。

在本次比赛中，Home Depot要求Kagglers通过开发能够准确预测搜索结果相关性的模型来帮助他们改善客户的购物体验。搜索相关性是Home Depot用于衡量客户获得正确产品的速度的隐含指标。目前，人类评估者评估潜在变化对其搜索算法的影响，这是一个缓慢而主观的过程。通过删除或最小化搜索相关性评估中的人工输入，Home Depot希望增加其团队可以对当前搜索算法执行的迭代次数。

<https://www.kaggle.com/c/home-depot-product-search-relevance>

**1.2 评估指标描述**

竞赛中评估模型优劣的指标。

选手的提交的评估指标是均方根误差（RMSE）。

1. 数据来源及描述性统计分析

**2.1 大赛数据来源**

描述大赛数据的来源

大赛数据提供方家得宝公司，为大赛提供历史销售数据，从软硬件环境诸多方面提供大赛支撑。

这里是数据的超链接。

链接：<https://pan.baidu.com/s/1z8mT2-StvW91Z3CPQ-PrbQ>

提取码：w53w

**2.2 数据的描述性统计**

* + 1. **数据基本情况描述：**

此数据集包含Home Depot网站上的许多产品和真实客户搜索字词。挑战在于预测所提供的搜索项和产品组合的相关性得分。为了创建真实标签，Home Depot已将搜索/产品对众包给多个人类评估者。

相关性是1（不相关）到3（高度相关）之间的数字。例如，搜索“AA电池”将被认为与一组AA电池（相关性= 3）高度相关，与无绳电钻（相关性= 2）轻微相关，与雪铲无关（相关性） = 1）。

每对由至少三名人类评估者评估。提供的相关性分数是评级的平均值。关于评级，还有三件事需要了解：

1. 给出评估者的具体说明在relevant\_instructions.docx
2. 评估者无法访问这些属性
3. 评分这可以访问产品图像，而竞争对手不包括图象

选手的任务是预测测试集中列出的每对的相关性。请注意，测试集包含已查看和未查看的搜索项。

* + 1. **数据字段介绍：**

**文件说明：**

train.csv - 训练集

test.csv – 测试集

product\_descriptions.csv - 包含每个产品的文字说明。您可以通过product\_uid将此表加入培训或测试集。

attributes.csv - 提供有关产品子集的扩展信息（通常代表详细的技术规范）。并非每个产品都具有属性。

sample\_submission.csv - 显示正确提交格式的文件

relevant\_instructions.docx - 向人类评估者提供的说明

**数据字段：**

id - 表示（search\_term，product\_uid）对的唯一Id字段

product\_uid - 产品的ID

product\_title - 产品名称

product\_description - 产品的文字说明（可能包含HTML内容）

search\_term - 搜索查询

relevance - 给定ID的相关性评级的平均值

name - 属性名称

value - 属性的值

**表2-1 train.csv数据表字段介绍**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 变量含义 | 变量类型 | 缺失率 |
| Id | 唯一Id字段 | 连续 | 0% |
| product\_uid | **产品的ID** | **连续** | **0%** |
| product\_title | 产品名称 | 名称 | 0% |
| search\_term | **搜索查询** | **名称** | **0%** |
| relevance | 给定ID的相关性评级的平均值 | 离散 | 0% |

**表2-2 test.csv数据表字段介绍**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 变量含义 | 变量类型 | 缺失率 |
| Id | 唯一Id字段 | 连续 | 0% |
| product\_uid | **产品的ID** | **连续** | **0%** |
| product\_title | 产品名称 | 名称 | 0% |
| search\_term | **搜索查询** | **名称** | **0%** |

**表2-3 attributes.csv数据表字段介绍**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 变量含义 | 变量类型 | 缺失率 |
| product\_uid | **产品的ID** | **连续** | **0%** |
| product\_description | 产品的文字说明（可能包含HTML内容） | 说明 | 0% |

**表2-4 product\_descriptions.csv数据表字段介绍**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 变量含义 | 变量类型 | 缺失率 |
| product\_uid | **产品的ID** | **连续** | **0%** |
| product\_description | 产品的文字说明（可能包含HTML内容） | 说明 | 0% |

**表2-5 sample\_submission.csv数据表字段介绍**

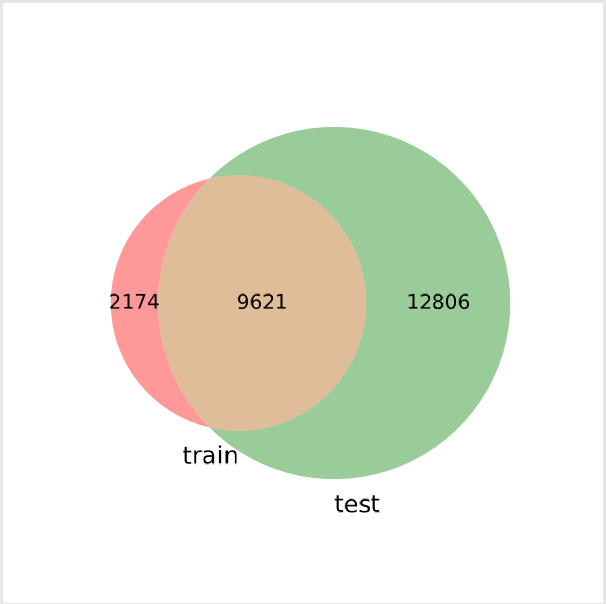
|  |  |  |  |
| --- | --- | --- | --- |
|  | 变量含义 | 变量类型 | 缺失率 |
| Id | 唯一Id字段 | 连续 | 0% |
| relevance | 给定ID的相关性评级的平均值 | 离散 | 0% |

* + 1. **数据描述性统计**

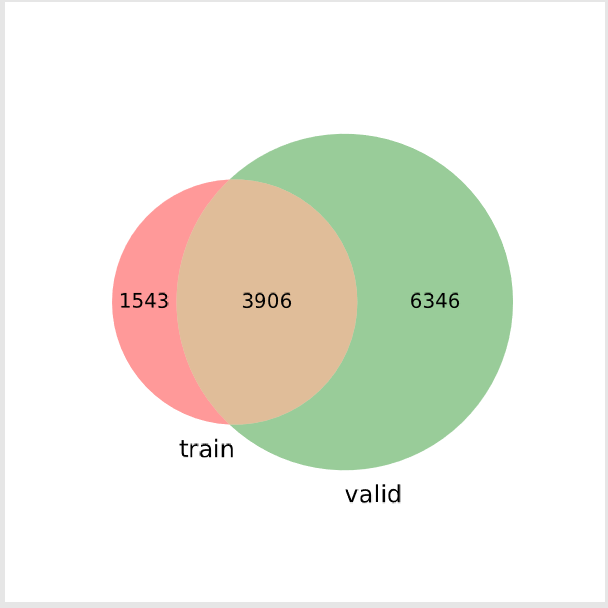
以下的分享来自于第三名的分享：



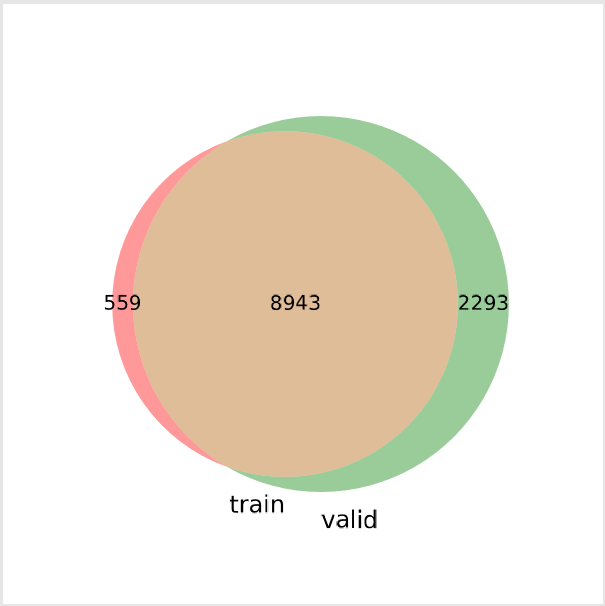
**表2-2-3-1 actual\_product\_uid**



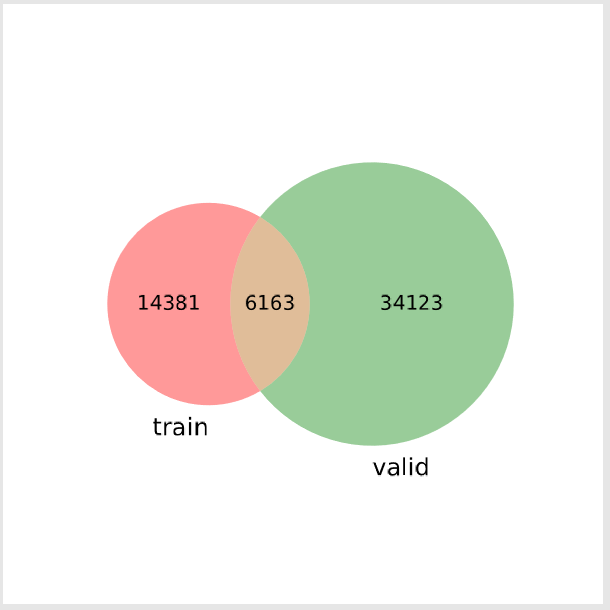
**表2-2-3-2 actual\_search\_term**



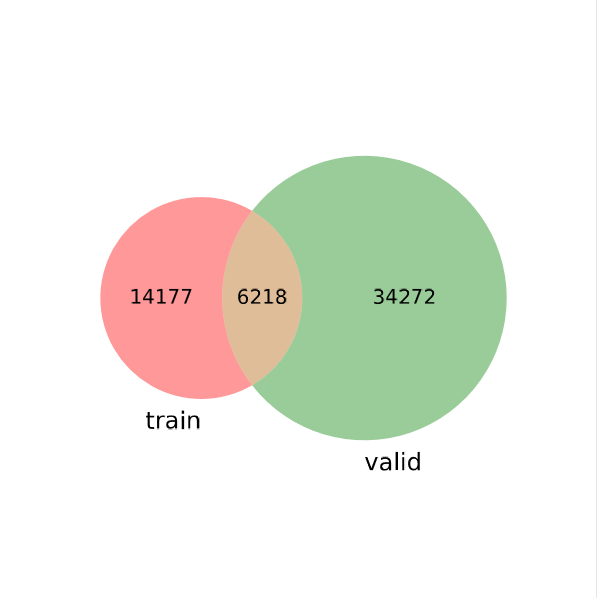
**表2-2-3-3 proposed\_search\_term**



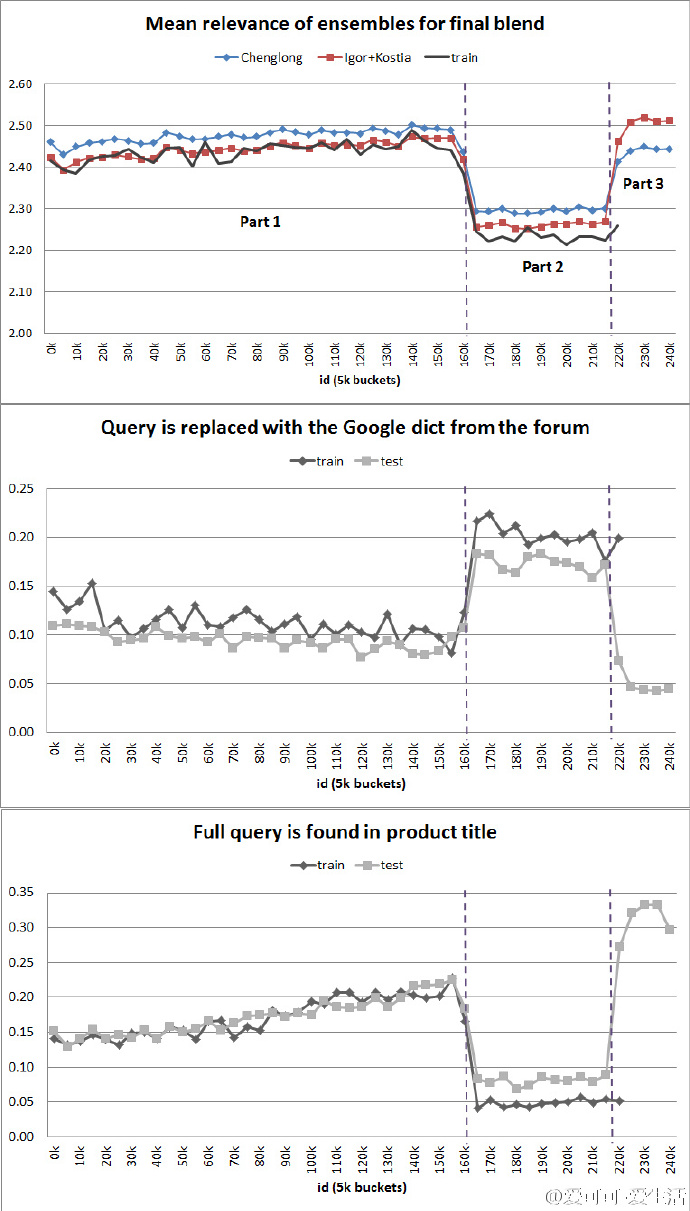
**表2-2-3-3 naive\_search\_term**



**表2-2-3-4 proposed\_product\_uid**



**表2-2-3-5 naive\_product\_uid**



3. 优秀算法思路

**3.1 方案一**

**3.1.1** 方案一数据预处理及特征工程部分方案

数据预处理：

主要步骤：

1. Pattern-replace 基类

初始化: 传一个pattern\_replace\_pair\_list, 把满足pattern的部分转换成replace

转换: 按每一对pattern, replace(tuple格式)做替换, 去掉开头结尾空格(strip())

1. 首字母大写变小写

覆盖父类transform方法

1. 本应该分开的两个单词，第二个单词首字母大写
2. 单词替换

给一个词典, 一一对应替换, 用于改正typo

子类初始化时只积累pattern\_replace\_pair\_list, 并没有完成替换.

调用transform方法才替换

1. 分割连接符连接的单词
2. 数字加子母
3. 去掉数字中的逗号
4. 单词的数字换成阿拉伯数字
5. 同一单位
6. 去掉html标签
7. 标点乱码替换成原始标点
8. 提取词干Lemma和词型归一stemmer
9. Query扩展
10. 处理商品名称
11. 处理商品属性
12. 处理不同数据结构的输入输出

把上面的代码都用户合并到一起就是main函数咯。总结一下大神的代码有几点很值得学习：

1. 重复的内容用类，和类的继承减少代码重复（各种replacer），并且放到一起使得逻辑清晰。

2. 用config文件中的字段控制部分代码是否运行（各种if config.大写单词）， 比反复注释，取消注释节省时间。

3. 各种设置统一到一个config文件中，避免调试的时候到处找，浪费时间。

def main():

###########

## Setup ##

###########

logname = "data\_processor\_%s.log"%now

logger = logging\_utils.\_get\_logger(config.LOG\_DIR, logname)

# put product\_attribute\_list, product\_attribute and product\_description first as they are

# quite time consuming to process

columns\_to\_proc = [

# # product\_attribute\_list is very time consuming to process

# # so we just process product\_attribute which is of the form

# # attr\_name1 | attr\_value1 | attr\_name2 | attr\_value2 | ...

# # and split it into a list afterwards

# "product\_attribute\_list",

"product\_attribute\_concat",

"product\_description",

"product\_brand",

"product\_color",

"product\_title",

"search\_term",

]

if config.PLATFORM == "Linux":

config.DATA\_PROCESSOR\_N\_JOBS = len(columns\_to\_proc)

# clean using a list of processors

processors = [

LowerCaseConverter(),

# See LowerUpperCaseSplitter and UnitConverter for why we put UnitConverter here

UnitConverter(),

LowerUpperCaseSplitter(),

WordReplacer(replace\_fname=config.WORD\_REPLACER\_DATA),

LetterLetterSplitter(),

DigitLetterSplitter(),

DigitCommaDigitMerger(),

NumberDigitMapper(),

UnitConverter(),

QuartetCleaner(),

HtmlCleaner(parser="html.parser"),

Lemmatizer(),

]

stemmers = [

Stemmer(stemmer\_type="snowball"),

Stemmer(stemmer\_type="porter")

][0:1]

## simple test

text = "1/2 inch rubber lep tips Bullet07"

print("Original:")

print(text)

list\_processor = ListProcessor(processors)

print("After:")

print(list\_processor.process([text]))

#############

## Process ##

#############

## load raw data

dfAll = pkl\_utils.\_load(config.ALL\_DATA\_RAW)

columns\_to\_proc = [col for col in columns\_to\_proc if col in dfAll.columns]

## extract product name from search\_term and product\_title

ext = ProductNameExtractor()

dfAll["search\_term\_product\_name"] = dfAll["search\_term"].apply(ext.transform)

dfAll["product\_title\_product\_name"] = dfAll["product\_title"].apply(ext.transform)

if config.TASK == "sample":

print(dfAll[["search\_term", "search\_term\_product\_name", "product\_title\_product\_name"]])

## clean using GoogleQuerySpellingChecker

# MUST BE IN FRONT OF ALL THE PROCESSING

if config.GOOGLE\_CORRECTING\_QUERY:

logger.info("Run GoogleQuerySpellingChecker at search\_term")

checker = GoogleQuerySpellingChecker()

dfAll["search\_term"] = dfAll["search\_term"].apply(checker.correct)

## clean uisng a list of processors

df\_processor = DataFrameParallelProcessor(processors, config.DATA\_PROCESSOR\_N\_JOBS)

df\_processor.process(dfAll, columns\_to\_proc)

# split product\_attribute\_concat into product\_attribute and product\_attribute\_list

dfAll["product\_attribute"] = dfAll["product\_attribute\_concat"].apply(\_split\_attr\_to\_text)

dfAll["product\_attribute\_list"] = dfAll["product\_attribute\_concat"].apply(\_split\_attr\_to\_list)

if config.TASK == "sample":

print(dfAll[["product\_attribute", "product\_attribute\_list"]])

# query expansion

if config.QUERY\_EXPANSION:

list\_processor = ListProcessor(processors)

base\_stopwords = set(list\_processor.process(list(config.STOP\_WORDS)))

qe = QueryExpansion(dfAll, ngram=3, stopwords\_threshold=0.9, base\_stopwords=base\_stopwords)

dfAll["search\_term\_alt"] = qe.build()

if config.TASK == "sample":

print(dfAll[["search\_term", "search\_term\_alt"]])

# save data

logger.info("Save to %s"%config.ALL\_DATA\_LEMMATIZED)

columns\_to\_save = [col for col in dfAll.columns if col != "product\_attribute\_concat"]

pkl\_utils.\_save(config.ALL\_DATA\_LEMMATIZED, dfAll[columns\_to\_save])

## auto correcting query

if config.AUTO\_CORRECTING\_QUERY:

logger.info("Run AutoSpellingChecker at search\_term")

checker = AutoSpellingChecker(dfAll, exclude\_stopwords=False, min\_len=4)

dfAll["search\_term\_auto\_corrected"] = list(dfAll["search\_term"].apply(checker.correct))

columns\_to\_proc += ["search\_term\_auto\_corrected"]

if config.TASK == "sample":

print(dfAll[["search\_term", "search\_term\_auto\_corrected"]])

# save query\_correction\_map and spelling checker

fname = "%s/auto\_spelling\_checker\_query\_correction\_map\_%s.log"%(config.LOG\_DIR, now)

checker.save\_query\_correction\_map(fname)

# save data

logger.info("Save to %s"%config.ALL\_DATA\_LEMMATIZED)

columns\_to\_save = [col for col in dfAll.columns if col != "product\_attribute\_concat"]

pkl\_utils.\_save(config.ALL\_DATA\_LEMMATIZED, dfAll[columns\_to\_save])

## clean using stemmers

df\_processor = DataFrameParallelProcessor(stemmers, config.DATA\_PROCESSOR\_N\_JOBS)

df\_processor.process(dfAll, columns\_to\_proc)

# split product\_attribute\_concat into product\_attribute and product\_attribute\_list

dfAll["product\_attribute"] = dfAll["product\_attribute\_concat"].apply(\_split\_attr\_to\_text)

dfAll["product\_attribute\_list"] = dfAll["product\_attribute\_concat"].apply(\_split\_attr\_to\_list)

# query expansion

if config.QUERY\_EXPANSION:

list\_processor = ListProcessor(stemmers)

base\_stopwords = set(list\_processor.process(list(config.STOP\_WORDS)))

qe = QueryExpansion(dfAll, ngram=3, stopwords\_threshold=0.9, base\_stopwords=base\_stopwords)

dfAll["search\_term\_alt"] = qe.build()

if config.TASK == "sample":

print(dfAll[["search\_term", "search\_term\_alt"]])

# save data

logger.info("Save to %s"%config.ALL\_DATA\_LEMMATIZED\_STEMMED)

columns\_to\_save = [col for col in dfAll.columns if col != "product\_attribute\_concat"]

pkl\_utils.\_save(config.ALL\_DATA\_LEMMATIZED\_STEMMED, dfAll[columns\_to\_save])

**特征：**

为了避免重复，我在这里只注意一些重要发现。使用和不使用初始预处理的文本用于生成功能。我们还分别处理'with'，'for'，'without'之后的单词。

1. 每次我们想要生成计数特征时，我们实际上生成了一组最多5个特征：交集中不同单词的数量，交集中总单词数，交集中不同单词中的字母数，交集中单词的比例查询的单词数，交叉词中的字母与查询中的字母数的比率。我相信这有助于从数据中捕获信息。实际上，我们最具预测性的功能之一是在查询和产品标题之间的交集中共享字母。
2. 在difflib.SequenceMatcher（）的帮助下，距离特征包括或结合了Jaccard系数，Damerau-Levenshtein距离和序列比较（以字符表示）。后者给了我们top10的两个功能。
3. TFIDF具有不同的组合。其中最有价值的是我们使用标题或项目符号生成要素权重，然后将它们乘以相应单词中的字母数，然后计算查询中单词的权重总和。总的来说，我们观察到，在这场比赛中，不仅单词作为一个信息单元很重要，而且单词中的字母数量也非常重要。确切地说，我必须说我们在TFIDF中使用了词干。
4. 另一组重要功能是直接针对与每个唯一搜索词相关的产品计算的TFIDF（和一些其他功能）。
5. Word2vec：我和Kostia训练了可用文本的模型，成龙也使用了预训练模型。事实证明，这些功能非常丰富。成龙也用过doc2vec。
6. WordNet与查询和产品标题中最重要的三元组之间的不同对词之间的相似之处。我们使用路径相似性，Leacock-Chodorow相似性和Resnik相似性。它给了我们一些前20个功能。
7. 查询扩展：查询按顶三元组中的两个最后一个单词分组，然后我们假设大多数相关性更接近于3而不是1.然后我们在产品描述中找到20个最常用的单词并计算出多少的百分比这些词汇在匹配产品的产品说明中。这是我们的十大功能之一。
8. 成龙使用LSA。
9. 来自顶级三卦的一些品牌和产品名称的假人。
10. 我和Kostia最终在第2部分中使用了id作为id（参见附件pdf）。

**3.1.2** 方案一模型设计、建立部分方案

模型设计:

Xgboost, gradient boost, random forest, SVR, extratrees.

Although SVR model 0.01 higher RMSE than the best model, it was beneficial for the ensemble due to low correlation with other models.

部分方案：

合奏

我们准备了两个独立的合奏。来自成龙的一个更好的人也使用了我们的功能。我和Kostia没有使用成龙的功能，并且正如我们所想的那样，在数据集的不同部分（第1部分：id221473）中有一个显着不同的行为。由于我们也观察到数据中的常规模式，我们认为其中一个集合可能特别容易在某些部分过度拟合。因此，在将我们的合奏混合到最终提交时，我们实际上做了不同的赌注，假设在某些部分中，其中一个模型在私下比在公共场合表现得更差。事实上，我们发现我们的公共和私人模型表现没有重大差异。

文本处理是关键

在本次比赛中，我和Kostia尝试了不同的文本处理方法，然后使用不同的输入生成相同的功能。公共RMSE正在不断改进。然后，在某些时候，我们意识到我们有机会赢得并且必须正确记录我们的工作并遵守获胜解决方案的要求。因此，我们开始根据统一的文本处理算法重新计算功能，并且仅观察公共分数中的微小（如果有的话）进展。

我们现在意识到，其中一个关键原因是不同的预处理引入了我们的特征变化，从而产生了更好的模型性能。当我们简化我们的特征生成时，我们失去了一些差异，因此也失去了性能。我相信我们可以生产出更好的模型，但是我们没有抓住失去的变化

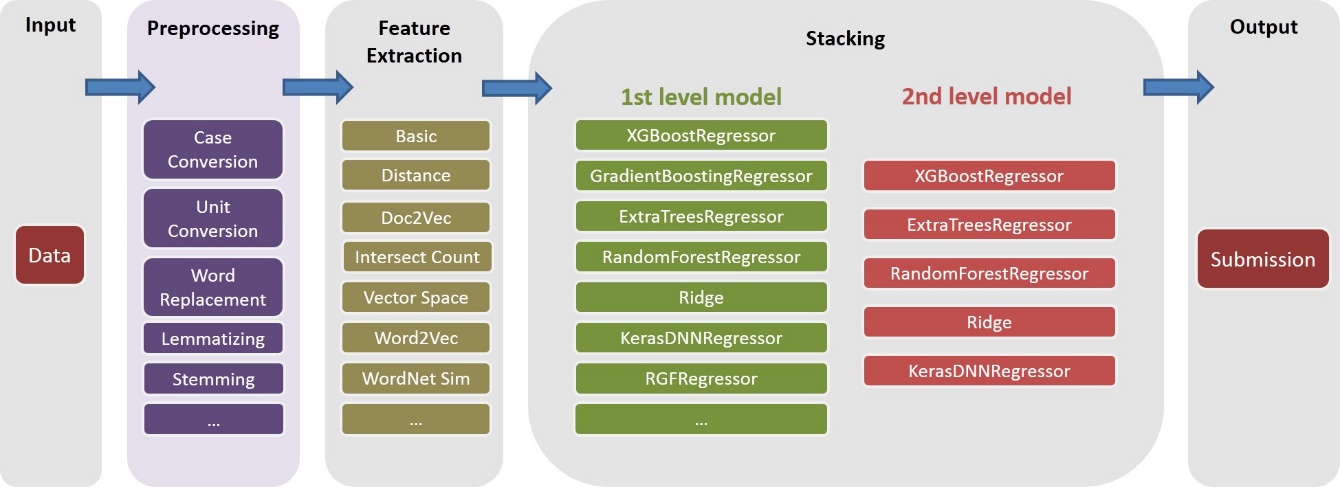
实际上，如果我们的团队已经提交了我们的顶级排行榜模型（成龙的模型+来自我和Kostia的旧模型的混合），我们将赢得私人RMSE 0.43188的第二（或第一）位置。我们没有提交它，因为我们既没有将旧的替代词发布到论坛，也没有机会逐步重现旧的结果（该集合包括在竞赛的不同时间产生的数十个模型）。说实话，由于数据中的模式，我预计会出现重大的排行榜重组，并相信新模型对于这次重组会更加强大。

**3.1.3** 方案一结果、排名等

最后结果：0.43270

排名：1/2125

**3.1.4** 方案一算法流程图



**图3-1 流程图**

[https://github.com/ChenglongChen/Kaggle\_HomeDepot](https://github.com/ChenglongChen/Kaggle_HomeDepot%20)

**3.2 方案二**

**3.2.1** 方案二数据预处理及特征工程部分方案

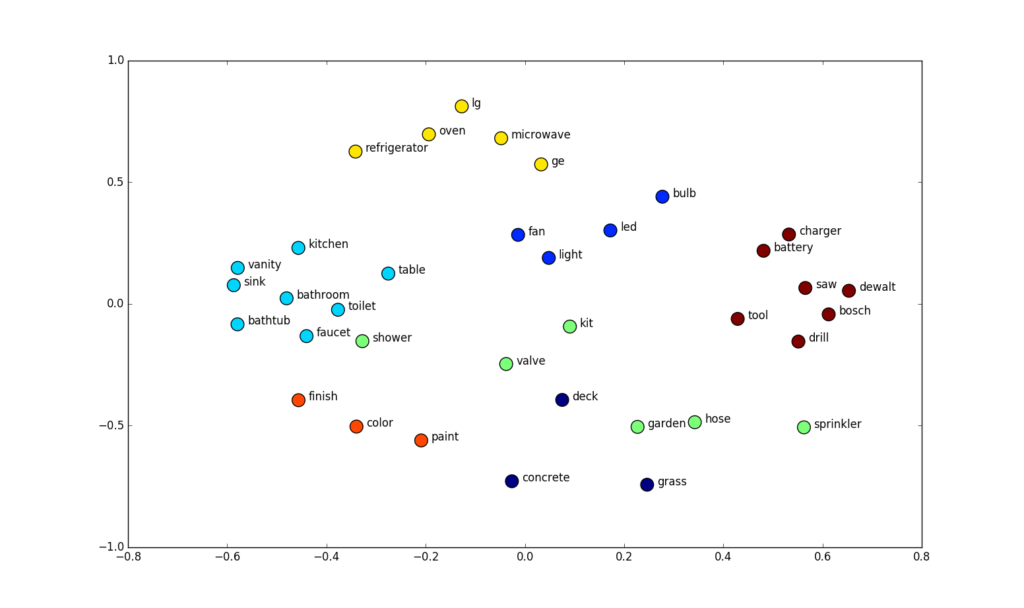
预处理和特征工程方法可以分为五类：关键字匹配，语义匹配，实体识别，词汇扩展和聚合特征。

1. 关键字匹配

在关键字匹配中，我们计算了搜索字词与产品信息的不同部分之间的匹配字词数，并且还存储了匹配的字词位置。为了克服拼写错误，我们使用了模糊匹配，我们计算了字符n-gram匹配而不是完整项。我们还计算了匹配项的tf-idf标准化分数，以针对非特定术语匹配进行标准化。

1. 语义匹配

为了捕获语义相似性（例如淋浴与浴室），我们使用潜在语义分析（LSA）和非负矩阵分解（NMF）进行矩阵分解。为了进一步了解在家得宝语料库中训练过的LSA或NMF没有捕捉到的相似之处，我们使用了经过预训练的word2vec和GloVe字嵌入，这些嵌入是在各种外部语料库上训练的。在LSA，NMF，GloVe和word2vec中，GloVe字嵌入提供了最佳性能。请参见下图，它如何捕获类似的实体。



**图3-2：在产品描述和标题上训练的单词嵌入向量的可视化 - 在单词嵌入空间中的相关单词聚类（使用余弦距离矩阵的多维缩放的二维投影，K均值聚类）**

1. 实体识别

主要动机是提取被搜索的主要实体，并分别在查询和产品标题中进行描述。我们的主要方法是包括匹配项的位置信息，但是oob错误分析显示它还不够。我们还尝试了POS标记，但我们注意到许多代表实体属性和规范的术语也被捕获为名词，没有明显的模式可以将它们与主要实体术语区分开来。相反，每当我们看到诸如“for”，“with”，“in”等介词时，我们决定在颠倒术语的顺序后提取最后N个术语作为潜在的主要实体，这些介词通常后跟实体属性/规范。

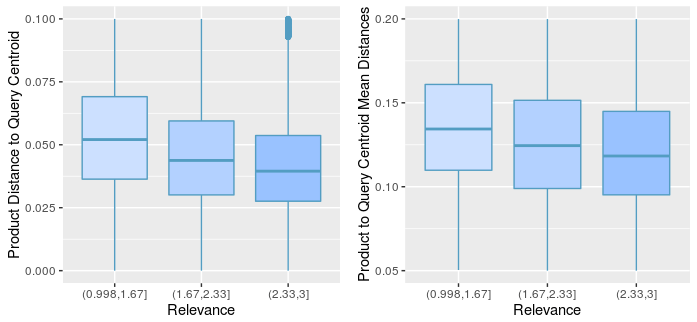
1. 词汇扩展

为了捕捉“宠物”与“狗”类型的关系，我们对从搜索词和产品标题中提取的主要实体进行了词汇扩展。词汇扩展包括从WordNet中提取同义词，下位词和hypernym。

1. 聚合特征

在开始时，我们通过不同的方式测量搜索项与产品的相似性，但搜索项非常嘈杂（即拼写错误）。由于检索到的大多数产品都是相关的，我们为每个查询聚类产品，然后计算聚类质心并使用此质心作为参考。计算产品与查询质心的相似性提供了强大的信息（见下图，左图）。

除此之外，一些查询是模棱两可的（例如“手动”而不是“窗口锁定”），这些模糊的术语对于人类评估者来说也不清楚，并且可能导致相关得分较低。我们还决定通过计算每个查询的产品与查询质心的平均相似度来包含此信息。下图（右图）显示了这种关系



**图3-2.通过聚合为每个查询检索的产品，提取有关产品与查询的相关性以及查询歧义量化的信息提取。**

**3.2.2** 方案二模型设计、建立部分方案

正如许多人所注意到的，这个数据集奖励各种功能和建模方法的混合，这是团队中最容易实现的。无论我们尝试过什么奇特的工程和建模，几乎没有任何案例对CV得分产生负面影响，因此我们不断添加模型和功能，直到我们用完为止。我们的简历策略运作良好，让我们有信心创建大型L1和L2合奏并添加第三层线性堆叠。

我们没有使用productuid或id作为一个功能，所以也许这有帮助。我们注意到不同id范围的趋势和平均相关性的跳跃，但我们只使用二进制变量id> 160,000来解释它。

数据清理/预处理

* 使用已发布的Google更正进行拼写更正，并对根据标题，说明和reddit注释构建的语料库使用Norvig更正。
* 使用在论坛中发布的手动创建的词典对查询进行拼写更正
* 从标题中删除括号中的所有单词（不包括电池）
* 通过介词（in，for，with等）和反转顺序拆分标题和查询，以获得重要的单词到最后
* 单位标准化

功能组

* 单词匹配计数和Wordnet同义词，下位词和hypernym扩展标题和查询的位置
* 产品标题/属性和查询的属性提取和匹配
* 牌
* 颜色
* 材料
* 尺寸，数量
* 功率，电压，电流，btu
* 查询，标题，描述之间的Jaccard和Dice距离
* Char n-gram TF-IDF，SVD，基于NMF的余弦距离和总共计项计数
* 在查询，产品标题（使用名词位置）和产品描述中的所有名词（使用nltk pos tagger）中提取1,2和3个重要名词，并对重要名词相似性（L1，L2，余弦距离）进行量化。在论坛中发表了重要且不重要的词典
* 标题，品牌和查询之间最常见的子序列
* 平均查询和产品标题，描述，属性，重要名词，Wordnet使用预训练的word2vec模型和GloVe字嵌入扩展了syn / hypo / hypernym的相似性
* 基于各种语料库的手套词嵌入质心和距离（预训练，从查询+标题+描述生成，富含Reddit DIY和Home Improvement评论，富含Wordnet同义词）
* 轻松的字推动者距离
* 轻松的单词移动距离（由查询/标题单词位置加权）
* 余弦距离
* 欧氏距离
* 查询，标题，描述的质心向量作为特征
* BOW，TF-IDF，基于LSA的功能
* 字N-gram TF-IDF余弦距离（N = 1,2,3）
* SVD余弦距离（50 dim，100 dim，300 dim）
* NMF余弦距离
* 原始SVD向量作为特征
* 按查询和产品聚合
* 标准偏差，等级变换，距离特征的百分位数
* L1，L2，余弦距离查询和SVD / NMF /手套的产品质心
* 统计功能
* Num在整个数据集中出现查询
* 在整个数据集中出现产品数量
* 利用id异常
* 布尔指标ID> 160,000
* 功能互动
* 排名靠前的特征之间的双向多项式相互作用

交叉验证

我们使用了3次交叉验证的3次运行：2次运行带有析取查询（查询不会出现在验证集中），1次运行带有析取产品ID，大致匹配列车和测试之间看不见的查询和产品ID的比例。

建模和整合

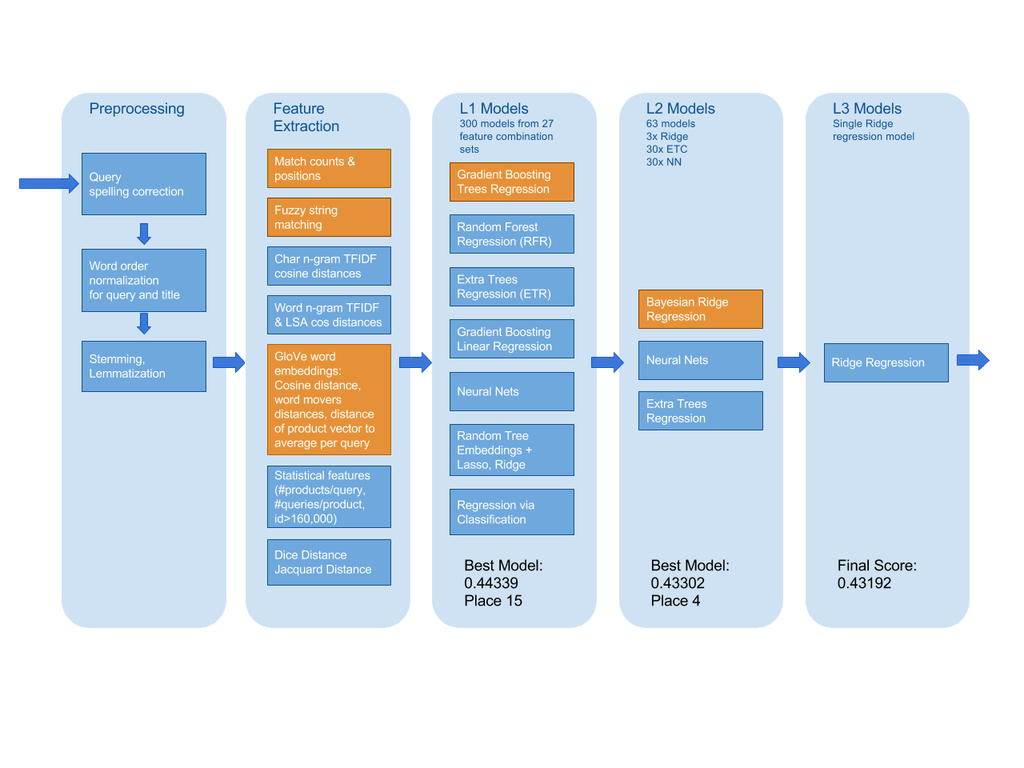
* 基于各种特征子集的约200个L1模型，使用XGB，RFR，ETR，神经网络，随机树嵌入+ Lasso / Ridge，ETC（通过分类回归）进行训练
* L2堆叠贝叶斯岭，10X袋装神经网络和10X袋装ETR，为每个L1 CV运行进行训练
* L3堆叠与岭回归

**3.2.3** 方案二结果、排名等

最后结果：0.43192

排名：1/2125

**3.2.4** 方案二算法流程图



**表3-2 流程图**

4. 算法比较

**表4-1 算法比较**

|  | **评估指标** | **特征工程** | **基础算法** | **基本库** |
| --- | --- | --- | --- | --- |
| **算法1** | **0.43270** | **Word2vec,TFIDF, LSA** | **Xgboost，梯度增强，随机森林，SVR，extratrees。** | **Sklearn, xsboost** |
| **算法2** | **0.43192** | **聚合特征** | **贝叶斯，xgboost，随机森林，额外树木和神经网络，无监督特征变换和广义线性模型** | **Python（scikit-learn，nltk，pandas，numpy，scipy，xgboost，keras，hyperopt，matplotlib）。有时R也用于可视化（ggplot）** |

制胜之处还是在参赛者的特征工程，排名第一的参赛者显然做了更加独特的特征工程。

5. 总结与展望

**5.1 总结**

在查询过程中，我了解到这是kaggle最早也是最少欢迎的几个比赛之一，因而上面提供的两个方案能够在三千多支队伍中脱颖而出就显得格外不容易。对数据细致的处理是第三名的制胜法宝，而第一名的奥妙则是更好的使用了XGboost

在收集整理的过程中也学习到了这些优秀参赛者的一些方法和经验，为今后的学习和工作提供了宝贵的经验。

**5.2 建模思路**

在学习了两位选手的思路后，我倾向于采用第三名 的方案，对数据进行尽可能详细的分析。

