“句子相似度匹配”项目报告

凌塑奇 2019.08.26

问题的定义

项目概述

“句子相似度匹配”项目属于自然语言处理（NLP）领域。顾名思义，该领域就是利用计算机处理人类语言（自然语言）。所谓“处理”则包括分析、理解自然语言文本，以及理解自然语言文本后生成自然语言文本。本项目仅涉及前者。

自然语言处理被誉为人工智能的明珠。语言作为人类最重要的工具之一，是破解“智能”的关键。然而，纳米学位中相关内容并不多。开展这一项目将有助于了解该领域的研究动态。。

本项目需要解决的问题是判断一对句子的意思是不是相同的。这些句子对来自 Quora 2017 年在 kaggle 举办的 Quora 问题匹配比赛。参赛者需要给出一系列句子对是相同的意思的概率。解决该问题，将帮助 Quora 归并同样问题和构建更好的相关问题推荐系统。

问题陈述

该问题可以看作一个监督学习中的分类问题，通常的解决方案是通过特征工程将文本转变为数值特征，训练出相应的分类器。在自然语言处理领域，通常被归纳为文本语义相似性分析。

简单地说，就是把文本转换为一系列的特征（通常为数值特征），这些特征将文本的词语含义、词语之间的关系、词语的稀缺性、语法等信息隐性地用数字表示。之后，可以利用分类器通过对数字进行分析，获得句子相似的概率。基本的分类器包括线性回归、支持向量机、决策树等。

问题的解决策略将分为三步。第一步，对文本进行编码，提取隐含文本相似性的特征信息。第二步，利用不同的分类器对特征信息进行分析，挑选最合适的分类器。第三部，对最合适的分类器进行优化，建立最终的分类模型。

期望的结果是将预测的句子相似的概率提交到 kaggle 后，需要达到 kaggle private leaderboard 的 top 20%。

评估指标

项目使用 log-loss 指标作为评估指标。该指标可用如下方程描述：

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(p_i) + (1-y_i)\log(1-p_i)\right)$$

(来源: https://www.kaggle.com/c/bnp-paribas-cardif-claims-management/overview/evaluation)

其中，N 表示数据数量，y_i 为 i 数据的真实标签，p_i 为预测的 i 数据标签为 y_i 的概率。

该指标对确切的错误分类的惩罚较大。例如，预测所有数据为真实标签的概率为 1 的 log-loss 为 21.79，预测所有数据为真实标签的概率为 0 的 log-loss 为 12.75，而预测所有数据为真实标签的概率为 0.5 的 log-loss 为 0.69）

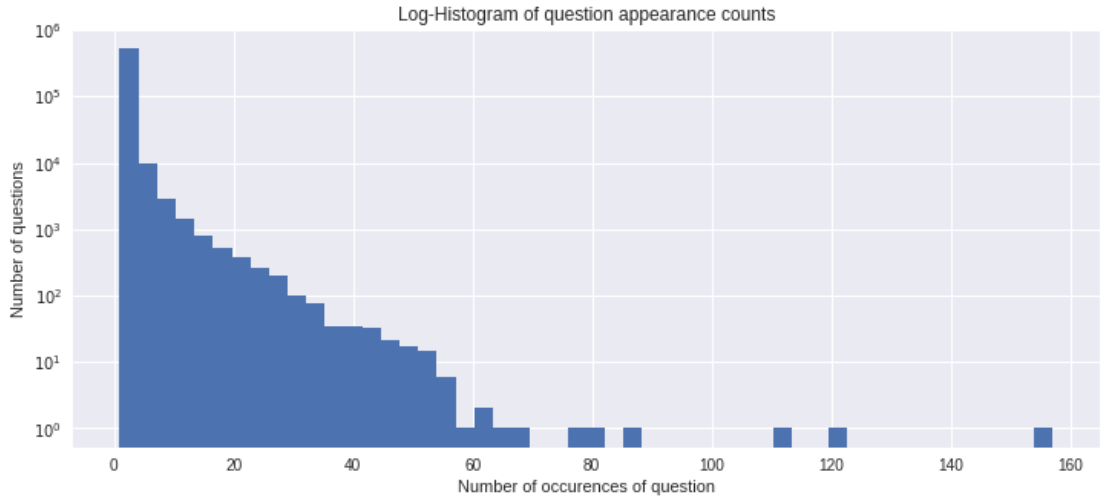
分析

数据的探索

项目的训练数据是约 40 万对句子对，包含句子对编号，每个句子的编号，句子文本和句子对是否相同的标签。其中，约 25 万对为不相同句子对、约 15 万对为相同句子对，相同句子对概率约为 40%。前五个句子对数据如下所示：

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

在数据集中，大部分句子都只出现了一次，少数句子出现了很多次。句子出现次数的直方图如下所示：

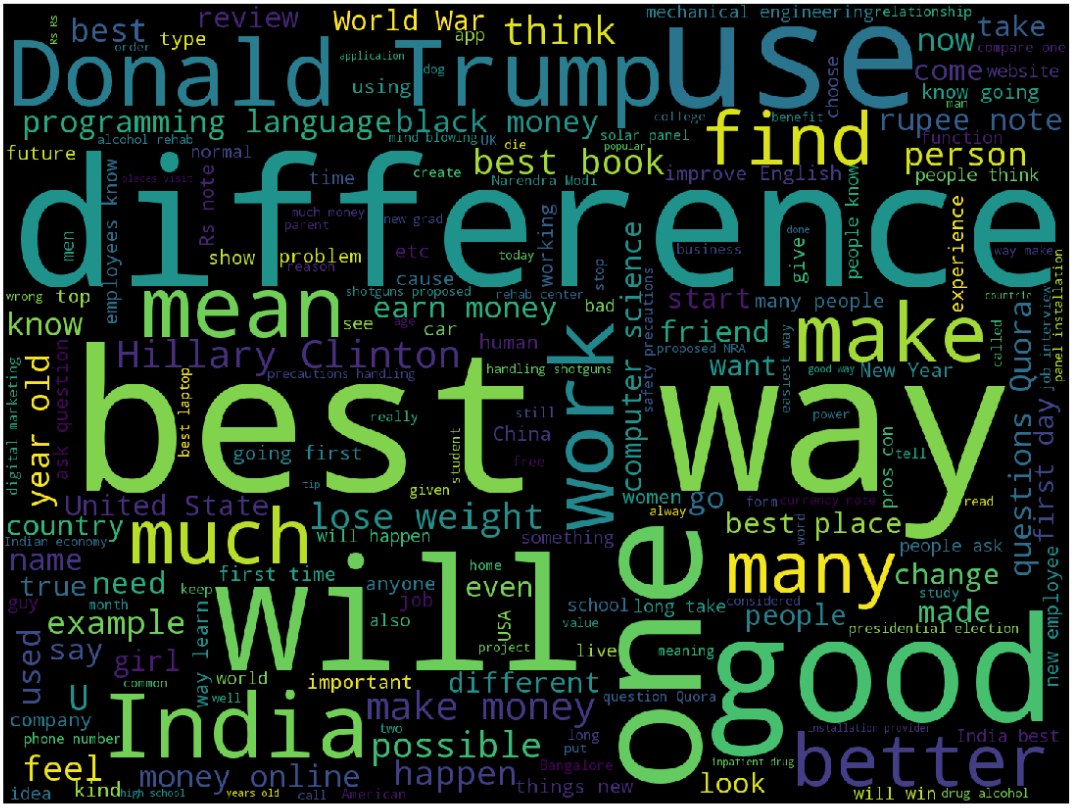


（图片来自 https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb）

数据集中的句子平均长度约 60 个字符，最少字符为 1 个，最长字符为 1169 个。

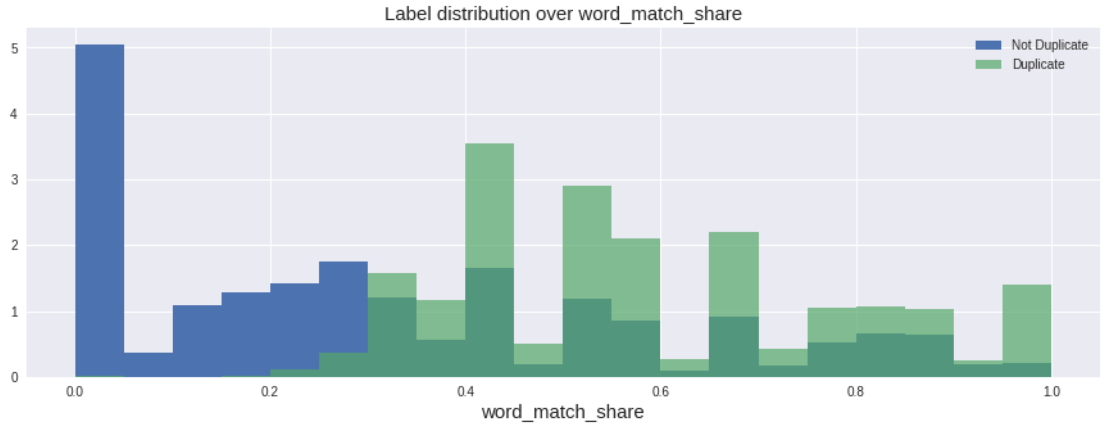
探索性可视化

统计数据集中的单词的出现次数绘制成如下的云图。从图中看出，出现最多的前三个单词是“difference”、“best”、“way”。



（图片来自 https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb）

下图分别统计了被标记为不同的句子对和被标记为相同的句子对之间，包含相同单词占句子对单词数的比率。从图中可以看出，大部分被标记为相同的句子对中相同单词的比率较大。



（图片来自 https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb）

算法和技术

目前常用的文本语义相似性分析有两种方法。

一种是通过特征工程对文本进行数据挖掘，将文本转换为大量的数值特征。目前的数据挖掘方法有三种。一是基于词袋模型的 TF-IDF、SDV 等；二是基于贝叶斯模型的 LDA；三是通过深度学习将文本转换为向量的词嵌入方法，例如 word2vec、GloVe 等。获得文本的数值表示后，可以计算文本向量间的 L1、L2 范数，余弦距离，word mover distance 等。

另一种是在把文本转变到向量空间的基础上，将文本向量输入深度神经网络进行训练，建立神经网络语言模型，从而实现文本相似性的分类。目前的神经网络语言模型大多 Siamese 和 Attention 两种架构。同时，一些预训练的语言模型也在文本语义相似性上得到了良好的结果。例如 Google 推出的 BERT 在多项 NLP 任务上获得了 state-of-art 的表现。

本项目将讨论多种方法的表现，最终选用其中一种进行优化。

特征工程

特征工程方面，本项目对文本进行了统计分析，TF-IDF 转换、universal sentence encoding、GloVe 进行向量化后计算了 cityblock 距离、Euclidean 距离和 cosine 距离。

统计分析主要包括两个句子的单词数差异及相应的比率，字母数差异及对应的比率，共同单词数及对应比率，以及 Jaccard 相似度。

TF-IDF 是 term frequency – inverse document frequency 的缩写。TF 表示文档中某个词出现的次数，这里简单理解为词出现的次数越多，越重要。IDF 表示文档总数除以所有文档中出现这个词的文档数。这两者相乘，就表示某个词在总的文档中不常见，却在某个文档中出现，那么这个词对这个文档就比较重要。

Universal sentence encoding 是 Google 推出的文本 embedding 技术，该技术将句子转换为固定长度的向量。通过神经网络从大量文本中学习转换方法。选择这种 embedding 方法是由于该方法不仅包含了整个句子的语义信息，而且实现非常简单。

GloVe 是 Stanford 推出的词向量 embedding 模型，本项目使用了在通用文本上训练的词向量库 glove.840B.300d.zip 。该库包含 2.2M 个单词向量，向量维度为 300。相比 word2vec，GloVe 在一般性的 NLP 任务的表现更好。GloVe 和 word2vec 一样，都是根据词语在语料中共现（co-accurance）的概率将词语转换为向量。如果两个词语共同出现的次数越多，那么这两个词语相关性更强。GloVe 在考虑词语共现时，利用词语共现的统计信息来训练模型，而 word2vec 则是通过局部窗口的上下文信息预测词语的方式训练模型。在实际运用中，两者并不存在明显的优劣，需要根据语料的实际情况进行取舍。

文本相似度

文本向量化后主要运用 cityblock 距离、Euclidean 距离和 cosine 距离来计算文本的相似度。

Cityblock 距离又称为曼哈顿距离，曼哈顿街区被横平竖直的道路分隔，从一个十字路口到另一个十字路口的驾驶距离可以用横、竖穿越的街区总数来表示。在向量空间，该距离是各维度坐标的差值的绝对值之和，也可以称为向量间的 L1 范数。K 维向量 x，和 y 的 cityblock 距离计算公式如下：

$$\sum_{k=1}^{n}|x_k - y_k|$$

Euclidean 距离是考虑的是两点间的直线距离。该距离是向量空间里两个向量的各维度坐标的差值的平方和的二次方根，也可以称为向量间的 L2 范数。K 维向量 x，和 y 的 Euclidean 距离计算公式如下：

$$\sqrt{\sum_{k=1}^{n}(x_k - y_k)^2}$$

Cosine 距离考虑了向量间的夹角，与 Euclidean 距离相比，该距离更注重向量整体上的差异，而不是各维度上的差异。K 维向量 x，和 y 的 Cosine 距离计算公式如下：

$$\frac{\sum_{k=1}^{n} x_k y_k}{\sqrt{\sum_{k=1}^{n} x_k^2}\sqrt{\sum_{k=1}^{n} y_k^2}}$$

BERT 模型

BERT 全称是 Bidirectional Encoder Representations from Transformers。该模型使用 Transformer 作为算法的主要框架，将预测下一个句子作为训练目标，使用了大规模的训练数据，从而实现在海量语料上进行自监督（没有人工标注）的学习。这使得 BERT 模型能够更好地表达文本的上下文信息，实现对同一个词语在不同语境下不同意思的不同向量表达。

同时，该模型也是一个十分庞大的模型，需要耗费大量的训练成本。幸运的是，Google 开源了该模型的预训练模型。这样，在预训练模型的基础上对特定任务进行微调训练就可以得到针对该任务的模型。

模型对比

完成特征工程后，主要考虑了三种模型进行对比。包括随机森林算法、XGB 和预训练的 BERT 模型。选择这三种算法的理由是：随机森林算法是常用的分类器，XGB 在近来的分类任务表

现优秀，BERT 在多项 NLP 任务获得了最佳表现。
项目中随机森林模型将采用默认参数，XGB 模型采用来自 Zhu Kai 分享的 kaggle kernel 的 XGB 参数，BERT 将文本输入预训练的 BERT 模型进行微调。

运行环境

由于项目对的计算性能需求较高、需要 GPU 支持，因此选择使用云计算平台进行计算。考虑到 CoLab 对 BERT 支持较好，且提供免费 TPU，因此选择在 CoLab 计算，数据储存在 Google Cloud Storage。

Colab 的普通环境，RAM 最大为 25G；GPU 环境 RAM 最大为 12G；TPU 环境，RAM 最大为 12G。

基准模型

BERT 在多项 NLP 任务都有较好的表现，因此对其在本项目的表现非常好奇。本项目选择将训练数据输入 BERT 预训练模型进行微调作为基准模型。训练参数不做改动。

该模型在 CoLab TPU 支持下训练花费 1 小时，在验证集的 logloss 为 0.4958.

方法

数据预处理

首先，是对数据进行清理。包括将文本全部转换为小写，去掉特殊符号，展开常见的缩写，将常见的同义词、短语转换为统一的形式。

然后，将文本分别进行 stemming、lemmatizing、去除 stop words 处理，并分别保存经过处理的文本。在处理过程发现，有些句子是空的，pandas 读取时会转换为 NaN，在处理时会报错。解决方法是使用空字符来填充 NaN 的数据。

执行过程

特征工程

挖掘了三类共 86 个特征，如下表所示：

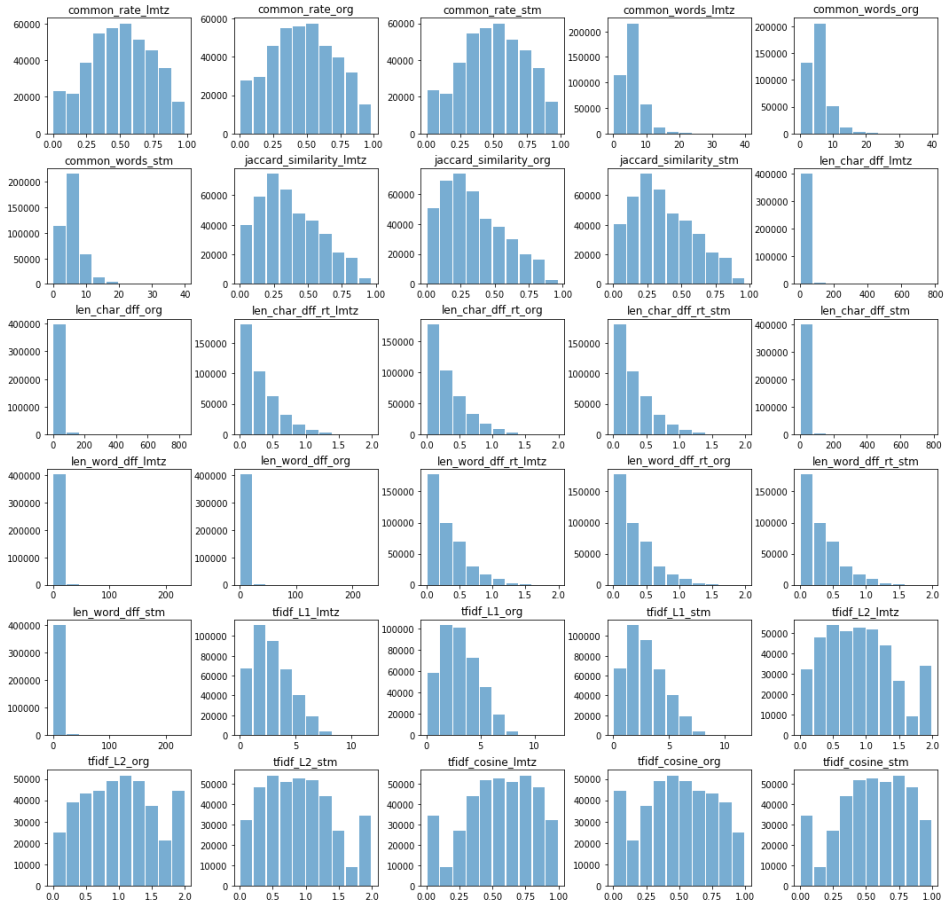
1	基本特征	原文本	首个单词是否相同
2			结尾单词是否相同

3			文本中性情感差异
4			文本正面情感差异
5			文本负面情感差异
6			文本相同情感差异
7			fuzzy_Qratio
8			fuzzy_Wratio
9			fuzzy_partial_ratio
10			fuzzy_partial_token_set_ratio
11			fuzzy_partial_token_sort_ratio
12			fuzzy_token_set_ratio
13			fuzzy_token_sort_ratio
14			句子对重复次数
15			句子 1 重复次数
16			句子 2 重复次数
17			平均重复次数
18			单词个数差异
19			单词个数差异比率
20			字符个数差异
21			字符个数差异比率
22			相同单词数
23			相同单词数比率
24			Jaccard 相似度
25			TFIDF-L1 距离
26			TFIDF-L2 距离
27			TFIDF-cosine 距离
28			单词个数差异
29			单词个数差异比率
30			字符个数差异
31			字符个数差异比率
32			相同单词数
33			相同单词数比率
34			Jaccard 相似度
35			TFIDF-L1 距离
36			TFIDF-L2 距离
37			TFIDF-cosine 距离
38			单词个数差异
39			单词个数差异比率
40			字符个数差异
41			字符个数差异比率
42			相同单词数
43			相同单词数比率
44			Jaccard 相似度
45			TFIDF-L1 距离

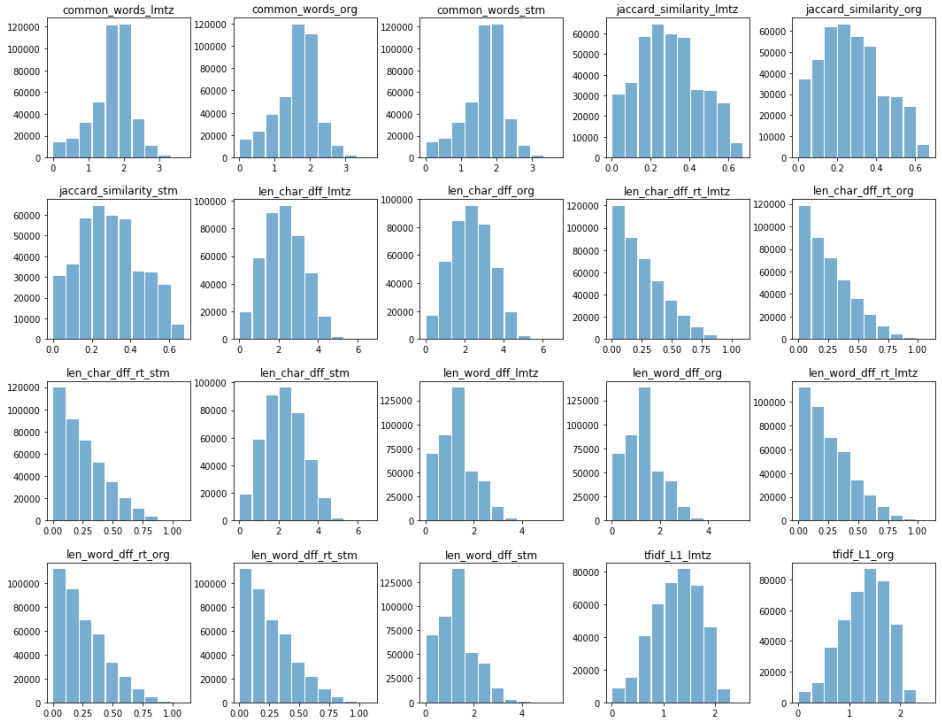
46			TFIDF-L2 距离
47			TFIDF-cosine 距离
48		remove stop words	相同单词数
49			相同单词数比率
50			Jaccard 相似度
51			TFIDF-L1 距离
52			TFIDF-L2 距离
53			TFIDF-cosine 距离
54	embedding 特征	word2vec	word mover distance
55			normalized word mover distance
56			L1 距离
57			L2 距离
58			cosine 距离
59			canberra 距离
60			minkowski 距离
61			braycurtis 距离
62			句子 1 向量的偏度
63			句子 2 向量的偏度
64			句子 1 向量的峰度
65			句子 2 向量的峰度
66		GloVe	L1 距离
67			L2 距离
68			cosine 距离
69			canberra 距离
70			minkowski 距离
71			braycurtis 距离
72			句子 1 向量的偏度
73			句子 2 向量的偏度
74			句子 1 向量的峰度
75			句子 2 向量的峰度
76		universal sentence encoding	L1 距离
77			L2 距离
78			cosine 距离
79			对数处理的 cosine 距离
80			canberra 距离
81			minkowski 距离
82			braycurtis 距离
83			句子 1 向量的偏度
84			句子 2 向量的偏度
85			句子 1 向量的峰度
86			句子 2 向量的峰度

其中一些特征存在较大的偏差，对其进行了对数化处理。部分特征未处理前的分布直方图如

下示：



对数处理后如下图所示：



BERT 模型训练

项目的 BERT 模型采用的是 uncased BERT base model 作为预训练模型。在该模型的基础上，输入句子对数据集，进行微调训练。微调训练的参数如下图所示：

```
TRAIN_BATCH_SIZE = 32
EVAL_BATCH_SIZE = 8
PREDICT_BATCH_SIZE = 8
LEARNING_RATE = 2e-5
NUM_TRAIN_EPOCHS = 3.0
MAX_SEQ_LENGTH = 128
```

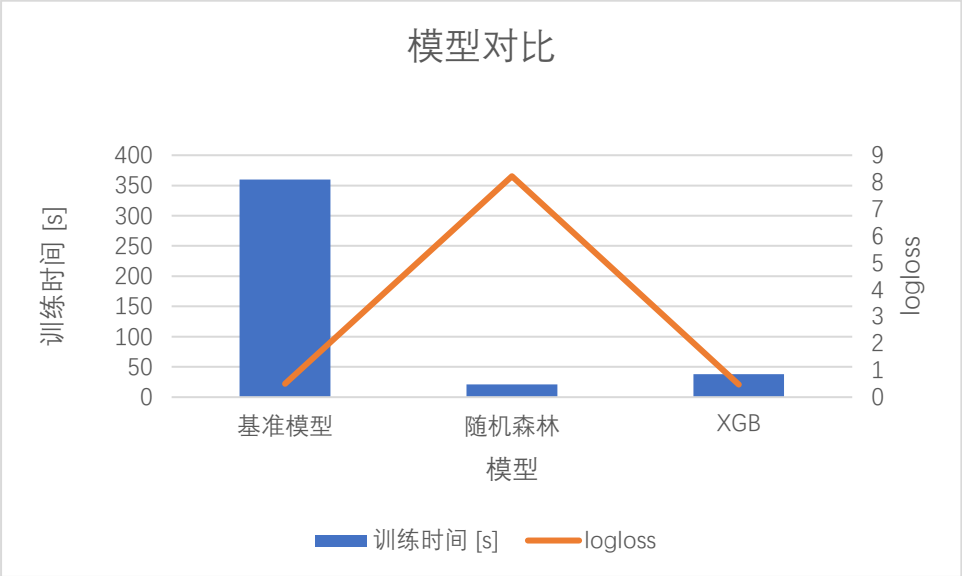
在 colab TPU 环境下的训练时长约一个小时。

模型对比

将训练数据中的 20%作为验证数据，分别对比了随机森林和 XGB 两种模型以及基准模型在验证集的表现。

随机森林模型的参数采用默认值。XGB 模型的参数来自 kaggle kernel。

三种模型的训练时间和 logloss 如下图所示：



从图中可以看出，随机森林模型的训练时间最少，但 logloss 远远不能满足要求。XGB 模型的训练时间远远小于 BERT 模型，并且 logloss 为 0.4679，小于基准模型的 0.4958。因此，选择 XGB 模型处理本项目。

训练难点

1. 测试集过大

这部分面临的最大问题是测试集数据较大。计算测试集 embedding 特征时，常常会由于出现很大的矩阵而导致超过内存使用量。解决方法是分批处理测试集，减小了每次处理的矩阵维度。
代码如下：

```
train_data = train_data.fillna(" ")


batch_size = 300000
batch_num = train_data.shape[0]//batch_size

ft_train = pd.DataFrame({'canberra_glv':0, 'minkowski_glv':0, 'braycurtis_glv':0,
                         'skew_q1vec_glv':0, 'skew_q2vec_glv':0, 'kur_q1vec_glv':0,
                         'kur_q2vec_glv':0}, index=[0])
```

```
def compute_ft_in_batch_tr(start_index, end_index, ft_train):

  data = train_data.iloc[start_index:end_index, :]
  v_q1 = np.zeros((data.shape[0], 300))
  v_q2  = np.zeros((data.shape[0], 300))

  time_str = get_time()
  print('sentance 2 vector for sentances {} to {} started at '.format(start_index, end_index) + time_str)
  for i, q in tqdm(enumerate(data.question1.values)):
      v_q1[i, :] = sent2vec(q)

  for i, q in tqdm(enumerate(data.question2.values)):
      v_q2[i, :] = sent2vec(q)

  time_str = get_time()
  print('sentance 2 vector for sentances {} to {} ended at '.format(start_index, end_index) + time_str)

  data = pd.DataFrame()
  data['canberra_glv'] = [canberra(x, y) for (x, y) in zip(np.nan_to_num(v_q1),
                                                           np.nan_to_num(v_q2))]

  data['minkowski_glv'] = [minkowski(x, y, 3) for (x, y) in zip(np.nan_to_num(v_q1),
                                                            np.nan_to_num(v_q2))]

  data['braycurtis_glv'] = [braycurtis(x, y) for (x, y) in zip(np.nan_to_num(v_q1),
                                                            np.nan_to_num(v_q2))]

  data['skew_q1vec_glv'] = [skew(x) for x in np.nan_to_num(v_q1)]
  data['skew_q2vec_glv'] = [skew(x) for x in np.nan_to_num(v_q2)]
  data['kur_q1vec_glv'] = [kurtosis(x) for x in np.nan_to_num(v_q1)]
  data['kur_q2vec_glv'] = [kurtosis(x) for x in np.nan_to_num(v_q2)]

  return ft_train.append(data, ignore_index=True )
```

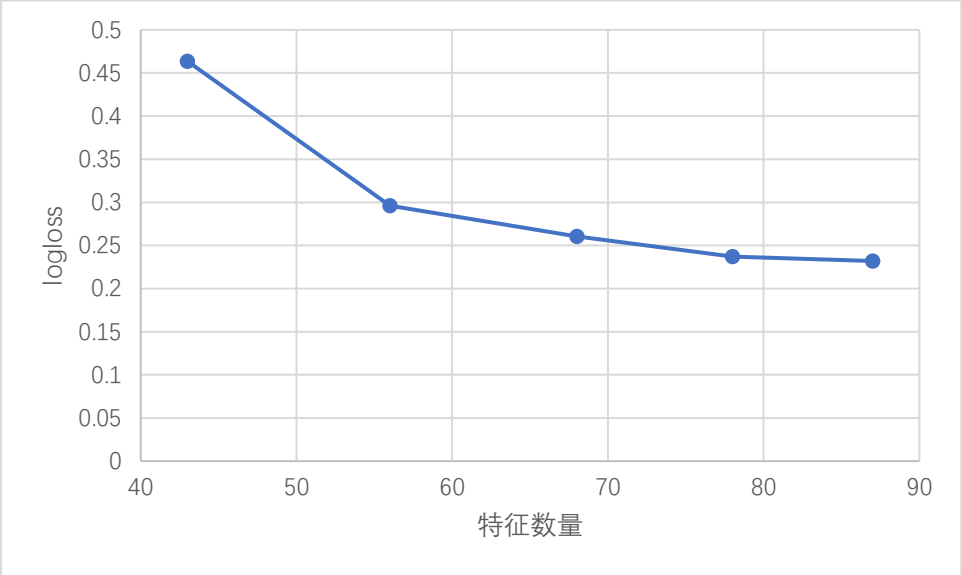
```
for i in range(0, batch_num + 1):
  start_index = i*batch_size
  end_index = (i+1)*batch_size
  if end_index > train_data.shape[0]:
    end_index = train_data.shape[0]
  ft_train = compute_ft_in_batch_tr(start_index, end_index, ft_train)
```

2. Logloss 过大

由于前期采用的参数来自 kaggle kernel，因此需要对参数进行调节。

但是在在调节参数时发现，logloss 总是无法通过调节参数达到要求的 logloss。经过搜集资料发现，对模型表现贡献最大的是特征以及特征数量。因此收集了更多的特征计算方法，扩充了特征的数量。主要包括增加了基本特征、word embedding 增加了 word2vec。

下图示意了特征数量增加后，XGB 在学习率为 0.1 情况下的表现：



3. 参数调节

XGB 模型在许多分类问题都有不错的表现。使用该方法的难点在于确定合适的参数组合。该模型需要调节的参数包括学习率（eta）、基学习器数量、最大树深度、最小叶子节点样本权重和、gamma（节点分裂所需的最小损失函数下降值）、subsample（控制对于每棵树，随机采样的比例）和 colsample_bytree（控制每棵随机采样的列数的占比）。

通常的参数调节方法为 grid search，即历遍所有可能的组合，分别训练模型，选择最佳模型。然而这种方法效率太低，耗时。更常用的是随机的 grid search，即随机产生大量可能的组合，选择最佳模型。这种方法则有可能无法选到最优的参数组合，并且时间成本较高。同时，XGB 的 sklearn 包不支持 GPU 加速的 grid search，因此时间成本更大。

另外还有贝叶斯优化算法，该算法假定寻找最优化参数的过程是一个高斯过程。通过随机遍历一定的数据点并拿到结果之后，可以大致绘制出整个数据的分布曲线，从而确定最优化参数。

考虑到本项目的时间成本，决定采用贝叶斯优化算法。使用的库是 bayes_opt。
代码如下，参考自 kaggle kernel：

```
# 调节 max_depth min_child_weight
def xgb_evaluate(max_depth, min_child_weight):
    params = {'eval_metric': 'logloss',
              'max_depth': int(max_depth),
              'min_child_weight': int(min_child_weight),
              'subsample': 0.8,
              'eta': 0.1,
              'gamma': 0,
              'colsample_bytree': 0.9,
              'tree_method':'gpu_hist',
             'scale_pos_weight':w0,
             'max_bin':256}
    # Used around 1000 boosting rounds in the full model
    cv_result = xgb.cv(params, dtrain3, num_boost_round=150, nfold=10, feval = wt_log_loss)
#     print(cv_result.iloc[-1, :])

    # Bayesian optimization only knows how to maximize, not minimize, so return the negative RMSE
    return -1.0 * cv_result['test-wt_logloss-mean'].iloc[-1]

xgb_bo = BayesianOptimization(xgb_evaluate, {'max_depth': (3, 7),
                                             'min_child_weight':(1, 5)})
# Use the expected improvement acquisition function to handle negative numbers
# Optimally needs quite a few more initiation points and number of iterations
xgb_bo.maximize(init_points=5, n_iter=10, acq='ei')
```

4. 正负例比率与指标计算方法

首次将预测结果提交至 kaggle 时发现，kaggle 的 logloss 得分与模型计算结果相差较大。

经过浏览讨论版发现，测试集与训练集的正负例比率相差较大。如果按照训练集的正负例比率训练模型，模型在测试集的表现与训练集不同。因此需要通过 scale-pos-weight 参数进行调整。

另外，kaggle 使用的是加权的计算方法。为了在调整参数时获得与评分接近的结果，需要使用加权的 logloss 计算方法来评估调参时的模型表现。项目中采用了 kaggle kernel 分享的加权计算方法。

完善

首先，在学习率为 0.1 的情况下确定基学习器的数量。使用 xgb.cv 方法。参数及运行结果如下：

```
params = {}
params["objective"] = "binary:logistic"
params["eval_metric"] = "logloss"
params["eta"] = 0.1
params["max_depth"] = 6
params["min_child_weight"] = 1
params["gamma"] = 0
params["subsample"] = 0.8
params["colsample_bytree"] = 0.9
params["scale_pos_weight"] = 1
params["tree_method"] = "gpu_hist"  # 使用GPU加速的直方图算法
params['max_bin'] = 256
params['seed']=0

model1 = xgb.cv(params, dtrain, num_boost_round = 2000, nfold = 10,
                metrics='logloss', early_stopping_rounds = 200,
                verbose_eval = 50)
```

```
[0]     train-logloss:0.633718+6.64575e-05      test-logloss:0.633831+0.00029447
[50]    train-logloss:0.259569+0.000376819      test-logloss:0.263334+0.00301036
[100]   train-logloss:0.242787+0.000351965      test-logloss:0.250486+0.00304783
[150]   train-logloss:0.234026+0.000254398      test-logloss:0.245843+0.00292573
[200]   train-logloss:0.22739+0.000338739       test-logloss:0.243242+0.00281348
[250]   train-logloss:0.221693+0.000252442      test-logloss:0.241534+0.00277096
[300]   train-logloss:0.216504+0.000317105      test-logloss:0.240384+0.00272041
[350]   train-logloss:0.211827+0.00031869       test-logloss:0.239537+0.00269492
[400]   train-logloss:0.207462+0.000347681      test-logloss:0.238915+0.00261331
[450]   train-logloss:0.203297+0.000340691      test-logloss:0.238475+0.00265159
[500]   train-logloss:0.199215+0.000351405      test-logloss:0.238043+0.00261589
[550]   train-logloss:0.19543+0.000363806       test-logloss:0.237742+0.00263645
[600]   train-logloss:0.191675+0.00036635       test-logloss:0.237506+0.00268936
[650]   train-logloss:0.18818+0.00041828        test-logloss:0.23735+0.00266467
[700]   train-logloss:0.184696+0.000428424      test-logloss:0.23725+0.00257775
[750]   train-logloss:0.181333+0.000438034      test-logloss:0.237205+0.00255697
[800]   train-logloss:0.1781+0.000444679        test-logloss:0.23714+0.00257023
[850]   train-logloss:0.174907+0.000452158      test-logloss:0.237121+0.00260547
[900]   train-logloss:0.171765+0.000435965      test-logloss:0.237098+0.00260204
[950]   train-logloss:0.168761+0.000438468      test-logloss:0.237099+0.00263065
[1000]  train-logloss:0.165764+0.00040425       test-logloss:0.237094+0.00268428
[1050]  train-logloss:0.162849+0.000416107      test-logloss:0.237154+0.00269404
```

在基学习器数达到 950 之后，test-logloss 开始上升，因此在 0.1 的学习率下，基学习器数量为 较为合适。不过考虑到后续需要调整多个参数，因此选择学习率 0.1，基学习器 150 的组合进行后续调参。

接着调节 max-depth 和 min-sample-weight 的组合。这两个参数的一搬取值为 1-6。采用贝叶斯优化算法进行调参。参数设置和调参结果如下所示:

```
xgb_bo = BayesianOptimization(xgb_evaluate, {'max_depth': (3, 7),
                                             'min_child_weight':(1, 5)})
# Use the expected improvement acquisition function to handle negative numbers
# Optimally needs quite a few more initiation points and number of iterations
xgb_bo.maximize(init_points=5, n_iter=10, acq='ei')
```

```
|   iter    |  target   | max_depth | min_ch... |
-------------------------------------------------
|  1        | -0.2935   |  5.596    |  1.408    |
|  2        | -0.2944   |  5.697    |  4.648    |
|  3        | -0.2935   |  5.062    |  1.998    |
|  4        | -0.3084   |  3.361    |  2.678    |
|  6        | -0.2886   |  7.0      |  1.0      |
|  7        | -0.2884   |  6.999    |  3.092    |
|  8        | -0.2906   |  7.0      |  2.425    |
|  9        | -0.2891   |  7.0      |  5.0      |
|  10       | -0.2894   |  6.994    |  1.008    |
|  11       | -0.2893   |  6.995    |  4.988    |
|  12       | -0.2884   |  7.0      |  3.147    |
|  13       | -0.2884   |  6.998    |  3.104    |
|  14       | -0.2884   |  7.0      |  3.202    |
|  15       | -0.2884   |  7.0      |  3.111    |
=================================================
```

从上图可以看出，max-depth 和 min-sample-weight 的组合在 6 和 3 的情况下达到最小 logloss。max-depth 和 min-sample-weight 的组合优化 6 为 3 和 。

接着，调节 gamma。该数值影响模型的拟合情况，过小可能导致过拟合。过大可能导致欠拟合。采用贝叶斯优化算法进行调参。参数设置和调参结果如下所示：

```
xgb_bo = BayesianOptimization(xgb_evaluate, {'gamma': (0, 0.5),
                                             })
# Use the expected improvement acquisition function to handle negative numbers
# Optimally needs quite a few more initiation points and number of iterations
xgb_bo.maximize(init_points=5, n_iter=10, acq='ei')
```

iter	target	gamma
1	-0.2884	0.4616
2	-0.2884	0.3993
3	-0.2884	0.281
4	-0.2884	0.007406
5	-0.2884	0.0376
6	-0.2884	0.3568
7	-0.2884	0.4516
8	-0.2884	0.2733
9	-0.2884	0.07935
10	-0.2884	0.01489
11	-0.2884	0.4546
12	-0.2884	0.4233
13	-0.2884	0.0125
14	-0.2884	0.2834
15	-0.2884	0.4036

从上图可以看出，其他取值的 gama 都没有达到比 0 更小的 logloss，因此维持 gamma = 0。

最后，调节 subsample 和 colsample_bytree 组合。这两个参数的一般取值在 0.5 之 0.9 之间。采用贝叶斯优化算法进行调参。参数设置和调参结果如下所示：

```
xgb_bo = BayesianOptimization(xgb_evaluate, {'subsample': (0.6, 1.0),
                                             'colsample_bytree': (0.6, 1.0)})
# Use the expected improvement acquisition function to handle negative numbers
# Optimally needs quite a few more initiation points and number of iterations
xgb_bo.maximize(init_points=5, n_iter=10, acq='ei')
```

iter	target	colsam...	subsample
1	-0.3054	0.9926	0.8937
2	-0.3062	0.8711	0.8746
3	-0.3097	0.913	0.8092
4	-0.3085	0.6686	0.7479
5	-0.3125	0.7752	0.6055
6	-0.3046	0.6	1.0
7	-0.3057	1.0	1.0
8	-0.3051	0.6	1.0
9	-0.3035	0.6	1.0
10	-0.3048	0.6021	0.9997
11	-0.3024	0.6024	0.9996
12	-0.3032	0.9976	0.9999
13	-0.3042	0.9968	1.0
14	-0.302	0.6009	0.9997
15	-0.3036	0.6003	1.0

从上图可以看出，subsample 和 colsample_bytree 组合在 0.99997 和 0.6009 的情况下达到最小 logloss。subsample 和 colsample_bytree 组合优化为 0.99997 和 0.6009。

确定这些参数后，参考相关文章和项目的经验，选择了 0.02 的学习率和 3600 的基学习器组合。

结果

模型评价与验证

最终选用的模型参数如图所示：

```
params = {}
params["objective"] = "binary:logistic"
params["eval_metric"] = "logloss"
params["eta"] = 0.02
params["max_depth"] = 6
params["min_child_weight"] = 3
params["gamma"] = 0
params["subsample"] = 0.9997
params["colsample_bytree"] = 0.6009
params["scale_pos_weight"] = w0
params["tree_method"] = "gpu_hist"  # 使用GPU加速的直方图算法
params['max_bin'] = 256

start = time.time()
model3 = xgb.train(params, dtrain3, num_boost_round = 3600)
end = time.time()
train_time = end - start

print("train time: {} s".format(train_time))
```

```
train time: 1382.8663144111633 s
```

```
start = time.time()
prediction = model3.predict(dtest)
end = time.time()
predict_time = end - start
print("predict time: {} s".format(predict_time))

subm = pd.DataFrame()
subm['test_id'] = test_raw['test_id']
subm['is_duplicate'] = prediction
subm.to_csv('submission3600_2.csv', index=False)
print('submission file saved')
```

```
predict time: 16.02756929397583 s
submission file saved
```

训练时间为 1382s，预测时间为 16s。

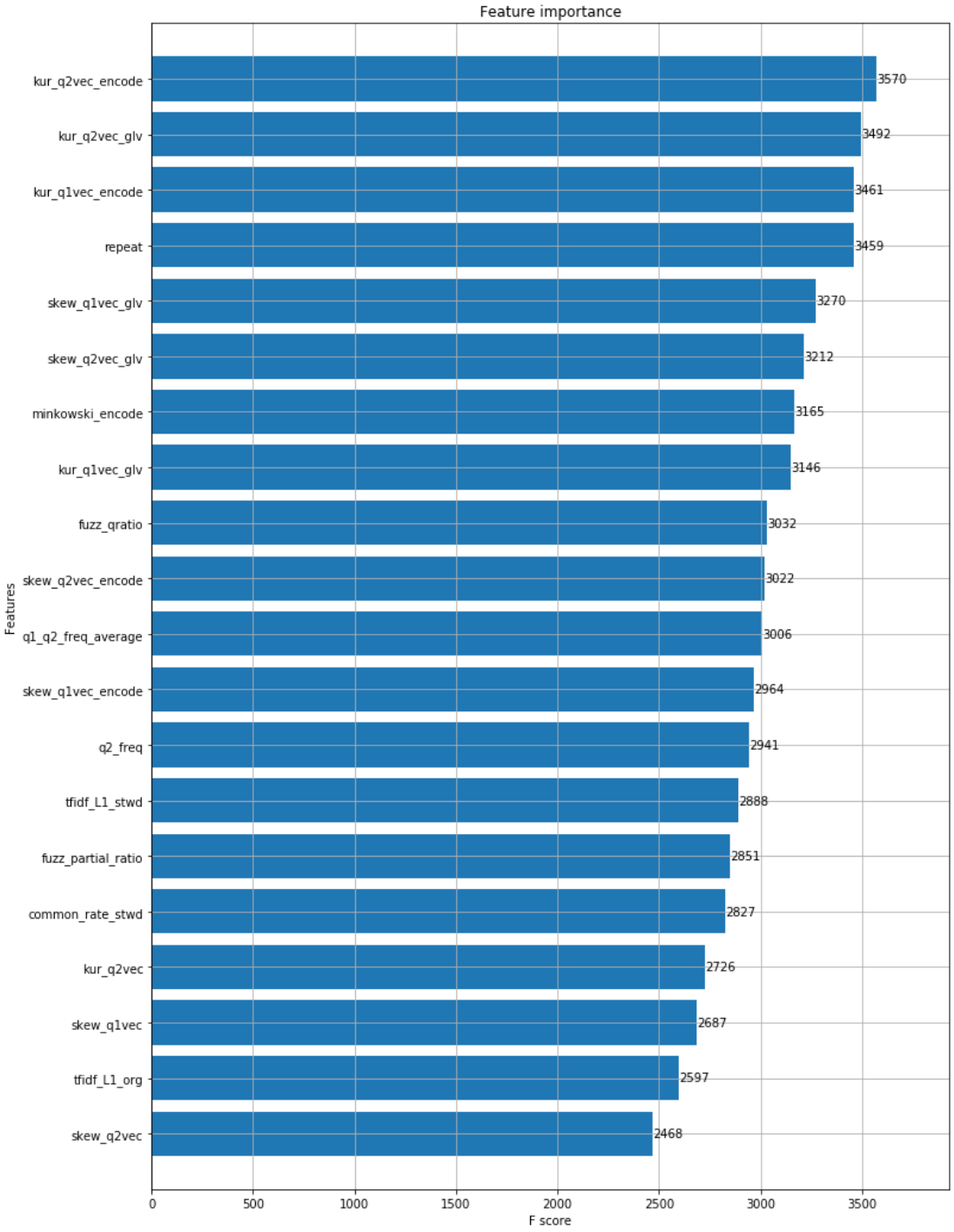
将预测结果提交至 kaggle 进行评分。完善前与完善后的得分如下所示。两次提交的模型主要有三点不同。一是去掉了'qid'这一特征；二是使用调整正负例比率训练模型；三是参数的优化。这三点变化对结果有了显著的提高。由于时间的关系没法进一步研究。

Submission and Description	Private Score	Public Score
submission3600_2.csv a few seconds ago by Shu Lin opt parameter	0.16039	0.15661
submission3600.csv 3 hours ago by Shu Lin initial submission xbg with 87 features, num_boost_bound =3600	0.19261	0.18973

该成绩在 private leaderboard 上是 332/3307，达到了前 10%。由此，可以认为该模型有了不错的表现。

特征重要性

XGB 模型在训练时会对每个特征进行重要性评分。下图展示了前 20 个在训练中评分最高的特征。



这些特征中，除了大部分为文本向量相关的特征外，反映句子对重复次数的特征 repeat、q_1_q2_freq_average、q2_freq，基于编辑距离的 fuzzy_ratio，基于词袋模型的 tfidif 类特征、相同单词率等也排在前列。

值得注意的是，词频统计类的特征（如 tfidf 和 common_rate）都是去除停用词后的结果。可以推断，停用词对词频统计影响较大。

此外，排在前三位的均为句子向量本身的特征，例如峰度 kurtosis 和偏度 skew。而常用的 cityblock 距离、Euclidean 距离和 cosine 距离均未出现在前 20。唯一排在前 20 的距离特征

为 3 阶的 Minkowski Distance。这样的结果对于用什么样的距离来计算文本相似度提出了疑问。是否采用更高阶的距离能更有效衡量文本向量的相似度?

还有一个值得注意的点是，排在前列的向量类特征采用文本向量化方法主要是 universal sentence encoder 和 GloVe 这两种方法都强调文本的全局信息。。从特征重要性图中看出，这样的考虑在文本相似性检验上表现出色。

项目结论

对项目的思考

我原本以为这个项目比较简单，业界已有成熟、简单、有效的解决方案。

实际尝试才发现，许多宣称有效的新方法在实际中表现并不尽如人意。表现优秀的解决方案常常是多种看似简单的方法的堆叠、平均。

比赛中排名前列的团队，大多尝试了非常多的方法，最终选择了几种进行组合。这样的解法，看似只花费了力气，实际上仍然是了不起的工作。很多时候，花费很多力气的方法就是最好的方法。

在完成中，我学到了很多实用的技巧。例如使用 pickle 保存生成的语言模型、训练模型等等，在需要的时候能更快地使用。

需要作出的改进

本项目时间比较仓促，准备不够充分，所以整个流程不够完善。例如开题报告确定的方案没有用上。原本计划主要采用 BERT 模型完成项目，但是由于训练时间的关系放弃了。

在实施过程中也遇到了很多想不到的困难。这些困得都只有动手做才会发现，也由此获得了解决方法，也算因祸得福。

本项目的特征工程部分实际上还有提高的空间。讨论版上分享了许多句子空间关系的特征，由于时间关系没有尝试。

最后，同样由于时间关系，没有调节学习率与基学习器的组合。这方面也有优化的空间。

参考资料

1. Natural language processinghttps://en.wikipedia.org/wiki/Natural_language_processing
2. 自然语言处理-简介 https://www.zhihu.com/topic/19560026/intro
3. Is That a Duplicate Quora Question? https://www.linkedin.com/pulse/duplicate-quora-

question-abhishek-thakur/
4. A Gentle Introduction to the Bag-of-Words Model https://machinelearningmastery.com/gentle-introduction-bag-words-model/
5. What Are Word Embeddings for Text? https://machinelearningmastery.com/what-are-word-embeddings/
6. Gentle Introduction to Statistical Language Modeling and Neural Language Models https://machinelearningmastery.com/statistical-language-modeling-and-neural-language-models/
7. 文本主题模型之 LDA(一) LDA 基础 https://www.cnblogs.com/pinard/p/6831308.html
8. 自然语言处理基础与实战（8）- 主题模型 LDA 理解与应用 https://www.jianshu.com/p/74ec7d5f6821
9. 从 0 到 1，了解 NLP 中的文本相似度 https://www.jianshu.com/p/22afb6d25e74?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation
10. 1st place solution https://www.kaggle.com/c/quora-question-pairs/discussion/34355#latest-572705
11. BNP Paribas Cardif Claims Management, Evaluation https://www.kaggle.com/c/bnp-paribas-cardif-claims-management/overview/evaluation
12. Log Loss Evaluation Section Missing. https://www.kaggle.com/c/quora-question-pairs/discussion/30605#latest-172575
13. Manhattan LSTM model for text similarity https://medium.com/@gautam.karmakar/manhattan-lstm-model-for-text-similarity-2351f80d72f1
14. BERT https://github.com/google-research/bert
15. BERT 和 ULMFIT embedding 比较文本分类结果 https://www.cnblogs.com/demo-deng/p/10797405.html
16. Bert-as-service https://github.com/hanxiao/bert-as-service
17. Predicting Movie Review Sentiment with BERT on TF Hub https://github.com/google-research/bert/blob/master/predicting_movie_reviews_with_bert_on_tf_hub.ipynb
18. Complete Guide to Parameter Tuning in XGBoost with codes in Python https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
19. quora-question-pairs-xgboost https://www.kaggle.com/benjaminkz/quora-question-pairs-xgboost/notebook
20. Bayesian Optimization with XGBoost https://www.kaggle.com/btyuhas/bayesian-optimization-with-xgboost