food-recommandation

Initial page

This project aims to undertand and implement a ML-based food recommandation system, inorder to get familar to a ML-Based project and develop similar systems in the future

Introduction

1.概述

1.1问题描述

根据Yelp开放数据库竞赛， 我们设计了一个餐馆推荐系统：

我们的目标是预测某一个用户会不会喜欢某个餐馆。 我们主要探讨的因素有餐馆特性，用户特性（从用户以前发表的评分与评价）， 类似用户的观点（对类似参观给出类似观点的用户）， 和评分与评价的可靠性。

为此，这些论文被用来作为参考：

1. Restaurant Recommendation System, Ashish Gandhe ;
2. Machine Learning and Visualization with Yelp Dataset, Zhiwei Zhang (with her repo);
3. Recommendation for yelp users itself, Wenqi Hou, Gauravi Saha, Manying Tsang

这个项目使用了 3 的方法进行数据清理。 然后它应用 Zhang 提出的SVM模型用于识别假评论，并为每个评论分配“真实分数”（该评价是否可信赖的指标），以便将此分数作为权重 Gandhe 描述的使用历史特征的计算模型。

经过进一步的预处理后，我们将三个机器学习模型应用于获得的数据集， 得到了我们的预测。

1.2 数据

此项目使用的Yelp数据集由五个JSON文件组成：

1. ***business.json*** ：包含有关商家的数据（我们的搜索范围仅限于餐馆，但也有酒店和商店），包括位置数据，属性和类别；
2. ***review.json：*** 包含完整的评论文本，包括撰写该评论的用户的ID和该餐馆的ID大约，以及通过投票获得的星星数（我们仅将文字用于检测欺骗性内容评论）；

3. ***user.json：***包含有关用户的数据，例如受欢迎程度，朋友和姓名；
4. ***checkin.json：***包含商家的签到（我们丢弃了此数据集）；
5. ***tip.json***：包含用户在企业上撰写的tips（删除了文字，仅保留了“表扬次数”，作为用户可靠性的标志。

最重要的特征从***review.json***（2）中获得的或是写入的，因为我们要预测的喜欢/不喜欢，是根据用户分配给餐厅的星级数量来计算的。 ***review.json*** 也是最大的数据集，从2014年到2014年，共有超过100万条评论 2018年，大小超过5GB。

1.3 使用的工具与库

我们用于该项目的工具：

•***Python 3***：我们用于机器学习最广泛使用的编程语言 与Tensorflow GPU兼容;

•***Jupyter Notebook***：编写主要脚本；

•***Pandas***：我们使用此库来读取和管理数据集；

•***Scikit-Learn***：它提供了许多ML非神经模型的现成实现；

•***TensorFlow GPU***：它提供了神经网络。

2.数据处理

2.1数据清理

首先，我们对数据集应用了Hou，Saha和Tsang 进行的数据清理，并做了一些调整以适应我们的项目。（例如，我们要考虑所有可用城市中的餐厅，而不仅仅是拉斯维加斯。）

根据他们的算法，我们要做的是：

1.使用适当的索引将json文件转换为Pandas数据；

2.提取包括所有餐厅的数据；

3.替换包含错误地区和邮政编码的垃圾数据；

4.日期转换和标准化；

5.根据初始特征创建新的解释性特征；

6.因此删除不必要的数据列，因为它们可能会基于新特征增加不确定性；

7.删除重复的餐厅条目并合并其评论；

8.将获得的数据集以pickle格式保存，以腾出内存空间。

特征**Categories， attributes** 和**hours**是我们主要要看的特征。

Catergories 中包含了各种不同商家， 所以只有标签为‘餐馆’的数据被流了下来；然后， 其中最常见和有意义的餐馆类型被收集到一个新的特征，‘**cuisine**‘， 被用来推测用户的喜好，并通过他计算历史特征。

Attributes特征包含了一个关于一个餐馆特点的标签库， 比如说这个参观卖不卖酒，或者有没有wifi。但是这些标签刚开始有一些混乱， 所以作者们给每个意思一样但看上去不同的标签设置了唯一的新的值，比如说wifi标签的值原先有"No", "u’no’", "’no’", ’None’ 和’No' ， 在新的标签中只有‘No'被用来标识

Hours有一个格式为{"day of the week": "opening hour - closing hour"}的数据库，但是这样的数据不是很容易用，所以作者把hours分成了两个特征：开门和关门时间

2.2 虚假数据检测与排查

现在我们的数据经过了清理和规范化，用了Zhang [4]中描述的一些模型来检测欺骗性评论并关联了一个
可信度评分 给他们每个人。

我们训练了他们的所有模型：一个***卷积神经网络（CNN)***，由从Keras预处理库获得的tokenizer序列进行标记；以及一个***Support Vector Machine（SVM),***，由Scikit Learn的TfdfV ectorizer安装在向量Computer 上。

事实证明，和***SVM***相比，***CNN***的性能较差，如表[1]和[2]所示。

因此，我们使用SVM以二进制方式计算预测并确定哪些评论可靠且具有欺骗性，并向评论数据集添加了一个bin_truth_score的特征， 拥有1 表示真实评论，而-1表示假评论。

为了获得概率分布而不是二进制结果，将SVM作为参数传递给Scikit Learn的CalibratedClassifierCV，该模型的预测结果是'评论是真实'的概率 ，而不是一个二进制结果，我们使用它们来添加特征real_truth_score 到评论数据集。

完成此操作后，我们删除了评论文本。 在图[1]和[2]中，我们可以看到两种标签的分布。

	precision	recall	f1-score	support
-1	0.18	0.18	0.18	24070
+1	0.87	0.88	0.88	158468
micro avg	0.78	0.78	0.78	182538
macro avg	0.53	0.53	0.53	182538
weighted avg	0.78	0.78	0.78	182538
accuracy	78.3%			

Table 1: Report for CNN model for fake review detection

	precision	recall	f1-score	support
-1	0.84	0.95	0.89	161016
+1	0.96	0.86	0.91	211105
micro avg	0.90	0.90	0.90	372121
macro avg	0.90	0.91	0.90	372121
weighted avg	0.91	0.90	0.90	372121
accuracy	89.99%			

Table 2: Report for SVM model for fake review detection

表1 表2

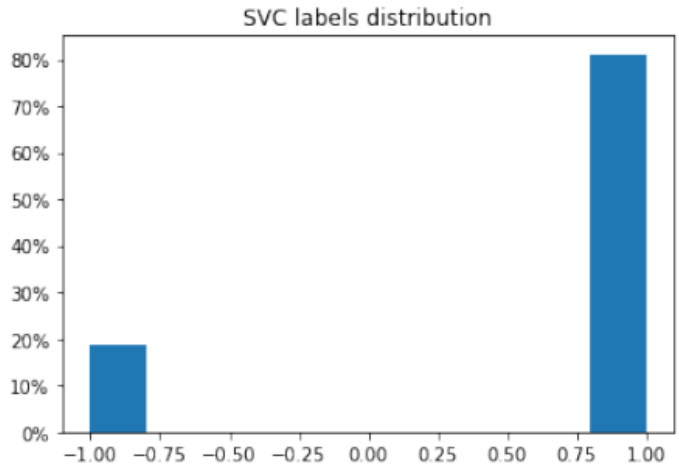


Figure 1: SVC truth labels

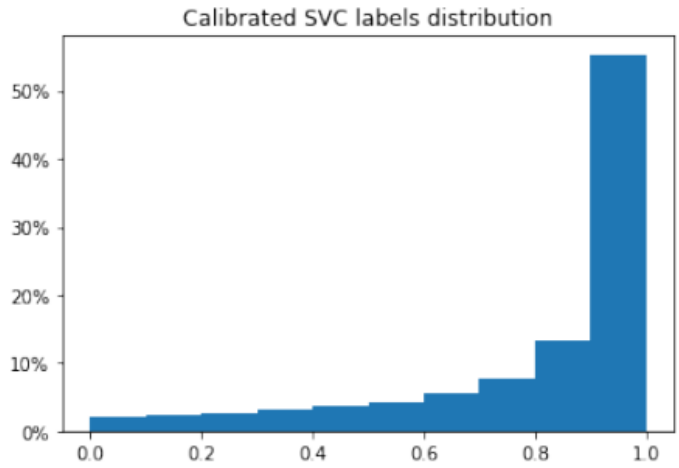


Figure 2: Calibrated truth labels

图1和图2

2.3 历史特征

根据甘德（Gandhe）的论文[3]，我们添加了一些 ***historical features*** 到我们的数据集：

1.用户特征：

1.1 某个用户给出的平均评分，

1.2 某个用户撰写的评论数量；

2.商家特征：

2.1 某餐厅的平均评分，

2.2 有关某家餐厅的评论数量；

3.用户-商家特征：

3.1 某位用户对每种料理（cuisine）的平均评分（在“数据清理”部分中定义的特征），也就是说，对于每种美食，用户对列表中包含该cuisine的所有餐厅给予的投票的平均值，

3.2 某用户对某餐厅的美食给出的评分的平均值，即用户对某餐厅列表中的美食给出的投票的平均值，如特征（3.1。）中所计算。

为此，我们必须将评论数据集分为三个部分：

• ***测试集(Test set)***, 从数据集中考虑的最后一天到m 个月前;

• ***训练集 (Training set)***, 从测试集开始的前一天到 n个月前；

• ***历史(History)***， 数据集的其余部分，用于计算历史特征

我们选择了 m = 3和 n = 8，因此测试集从9/1/2018到11/30/2018，训练集从1/1/2018到8/31/2018，历史记录包含剩余数据，从10/12/2004至12/ 31/2017。

虽然其他特征计算起来很迅速并且很容易实现，但是我们不得不优化（3.1）部分的，以防止计算花费太多时间，因此我们尝试了两个通过在更多处理器上运行来自动加速python的库，而且无需程序员太多的干预。第一个是Modin，专门用于Pandas，第二个是Numba，专门用于Python数学运算，但实际上没有一个成功运行了，前一个是由于与Windows OS不兼容，而另一个则是因为它“喜欢NumPy”（如文档中所述），但不喜欢Pandas。因此，我们决定手动拆分数据集，并把每部分分开来，传递给一个在多个处理器上面同时运行的方程， 这样我们运算的时间就减少到几个小时之内了。

我们决定对评论的’可信度‘（在“虚假数据检测与排查”中描述过）应用两个估算器，来计算’可行度‘的两个附加版本，因此对于这些特征我们有三中’味道‘（’flavours‘）：

1. **标准（standard)：** 每个评论的处理方式相同（每个评论的权重均为1），

2. **二进制(Binary)：** 真实评论的权重为1，欺骗性评论的权重为0，也就是他们的 **bin_truth_score**， 因此，假评论不在总数或平均值中的计算中被用到。

3. **真实的(real)：** 每个评论都以其 **real_truth_score为准,** 无论是总数还是平均值。

2.4基于用户的协作方法

然后，我们决定添加基于*协同过滤（collaborative filtering)*的 一个特征： 我们有很多用户撰写的很多评论，因此我们可以使用同一类用户给同一类餐厅的评分来预测某个用户对从未尝试过的餐厅的看法。因此，我们应用了以下公式：

$$pred(u,r) = a_u + \frac{\sum_{u_i \in U} sim(u,u_i) * (a_{u_i,r} - a_r)}{\sum_{u_i \in U} sim(u,u_i)}$$

其中：

- au 是用户u给所有同一种类别（cuisine）的餐厅 r的平均投票数 (即3.2。用户之间 u 和餐厅 r 在“历史特征”部分中定义的特征）
- ‘aui，r’是用户ui∈U对餐馆 r的所有投票的平均值；
- ar 是餐馆r收到的平均评分（在2.1中的特征）
- sim(u, ui) 是通过计算得出的，基于两个分别代表用户u和ui的向量的“美食相似性(cuisine similarity)" ， 有3.1部分的特征组成。

我们也为该功能提供了三个版本，其含义与上一节中所述的含义相同

2.5降维和进一步数据预处理

现在我们有了所需要的所有相关特征，我们可以开始准备把数据输入模型了：

1.我们添加了标签 “喜欢”。 如果该用户在该评论中给了4或5颗星，则该特征为1；如果该用户给出了1、2或3颗星，则为0 （结果标签分布显示在图[5]和[6]中）；

2. 我们将分类特征转换为数字特征，以方便模型的数据读取：

- 对于特征：**OutdoorSeating, BusinessAcceptsCreditCards, RestaurantsDelivery, RestaurantsReservations, WiFi, Alcohol, city，** 我们用**Panda**的 **get dummies()**来实施 **one-hot 编码**
- 对于特征：**Monday Open, Tuesday Open, Wednesday Open, Thursday Open, Friday Open, Saturday Open, Sunday Open, Monday Close, Tuesday Close, Wednesday Close, Thursday Close,Friday Close, Saturday Close, Sunday Close, postal code** 我们用 **Scikit learn** 的 **OrdinalEncoder.fit transform(),** 把分类特征转换成一个整数array；这回引进一些不准确性， 但是所需要的记忆空间少了很多。 所以我们在一些不那么重要的，却有着很大容量特征使用了这个方程。

3. 我们将 **city**和 **category**中频率不超过阈值θ （θ =100）的标签全部标记为了‘others'. （如图3 和图4 所示）

city	# occurrences
Las Vegas	208437
Phoenix	71126
Toronto	57047
Charlotte	40190
Scottsdale	36579
Pittsburgh	26891
Henderson	21518
Montreal	18531
Tempe	18354
Mesa	17235
...	...
Paw Creek	1
Tottenham	1
springdale	1
Coteau-du-Lac	1
Huntingdon	1
Fabreville	1
De Winton	1
Napierville	1
Laval, Ste Dorothee	1
Les Coteaux	1

Table 3: City frequencies

category	# occurrences
Food	192841
Nightlife	170259
Bars	165959
American (Traditional)	123975
Breakfast & Brunch	120310
American (New)	117437
Sandwiches	82264
Mexican	73726
Burgers	71598
Pizza	67538
...	...
Beer Hall	1
Banks & Credit Unions	1
University Housing	1
Pet Groomers	1
Gardeners	1
Home Health Care	1
Campgrounds	1
Holiday Decorations	1
Roofing	1
Senegalese	1

Table 4: Category frequencies

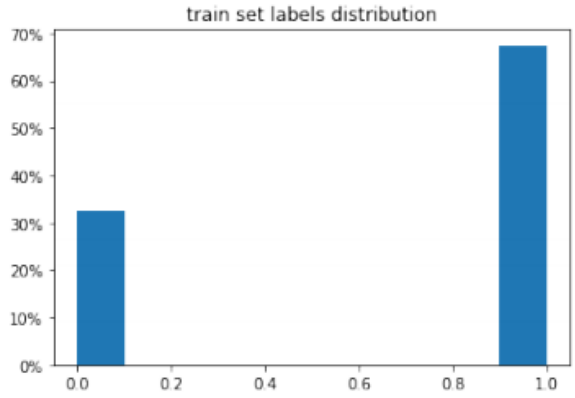


Figure 5: Labels distribution in the train set

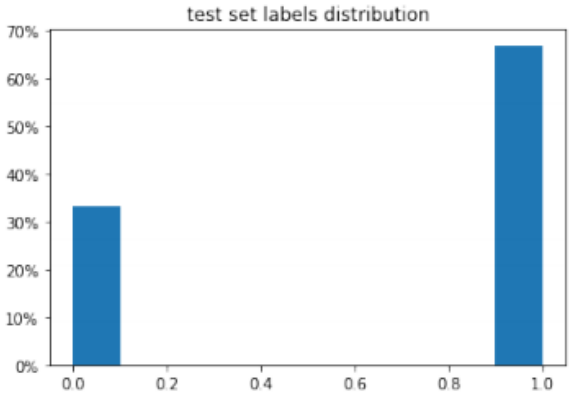


Figure 6: Labels distribution in the test set

3.模型应用

3.1 线性SVM(Linear Support Vector Machine)

我们用***LSVM(Linear Support Vector Machine)***的方法来尝试预测， 一个指定的用户， ***u，***会不会喜欢一个指定的餐馆，**r** 。 ***LSVM*** 是一个有着的线性算法监督式学习模型，被用来分析数据，来达到分类或者其他目的。

在有了一系列的被表示为种类1或者2的训练数据后， 通过***SVM***算法我们可以建立一个可以将新的例子归类到两个种类之一的模型。 这代表着***SVM*** 是一个非概率性而精致线性分类算法。（non-probalistic binary linear classifier.)

SVM模型是将数据表示为空间（地图）中的点，以使各个类别的数据点被尽可能明显地被间隙分开。然后，将新数据点映射到相同的空间中 ，并根据它们落入的分隔线的哪一边来判断他的种类。

SVM 可以通过使用不同的*kernal*（一些数学转换方程）来把数据映射入不同的空间内， 来执行线性和非线性的数据分类。 在我们的项目中， 我们决定使用线性的*kernel*。

3.2随机森林

我们用来进行预测的第二种方法是随机森林法。随机森林是一种用于分类的整体学习方法（在我们的项目中），该方法通过在训练模型时构造多个*“决策树”*， 并采用这些决策树所产出的结果中的*众数（mode）*，来决定输入的数据属于哪一个类别。

{没搞清楚的：

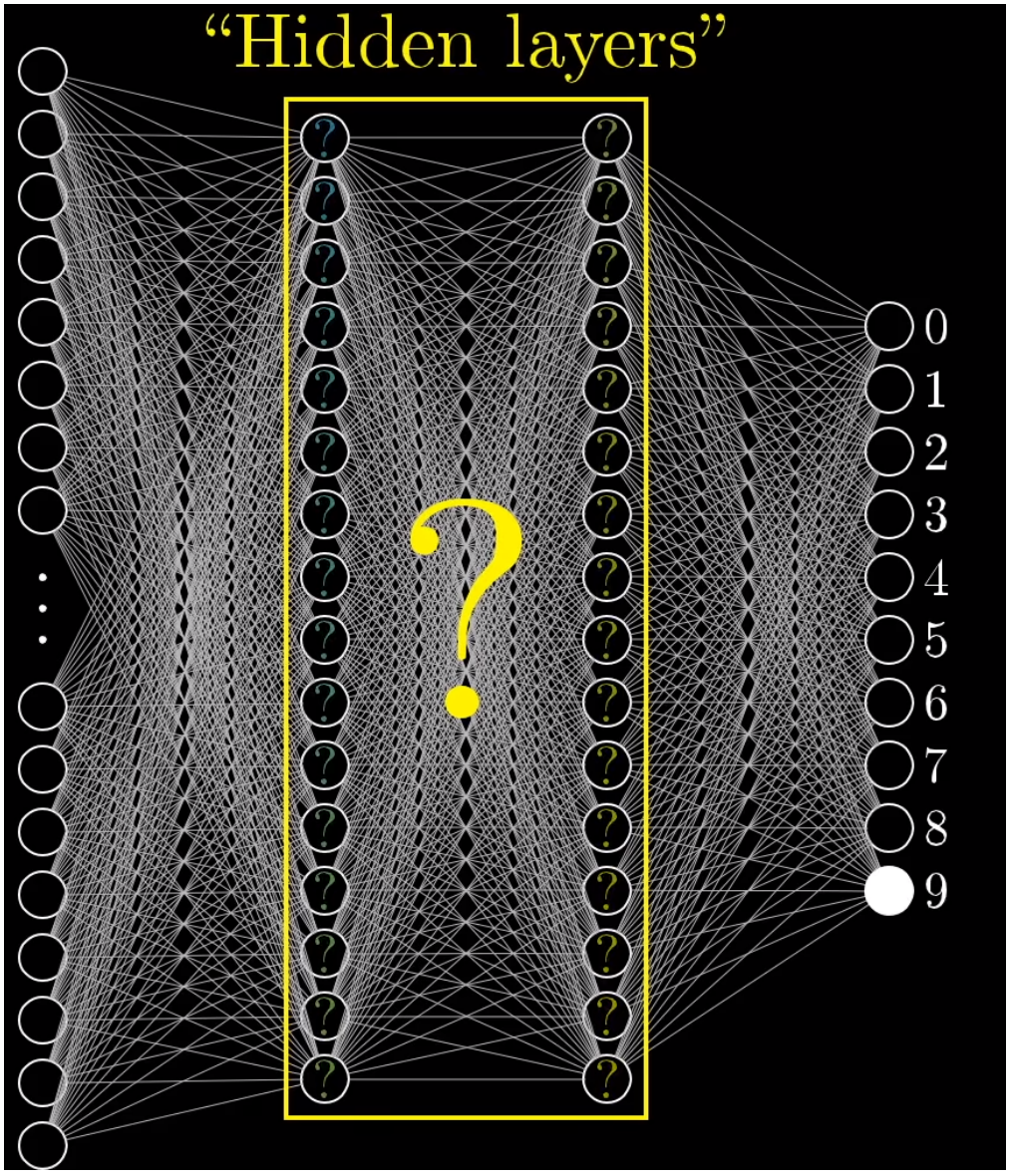
Each tree is grown using a bootstrap sample (sampling with replacement) of training data by choosing n times from all N available training cases, and uses the rest of the examples to estimate its error. At each decision node, best attribute to test is chosen from a random sample of m attributes (where m < M and M is the number of features), rather than from all attributes.

}

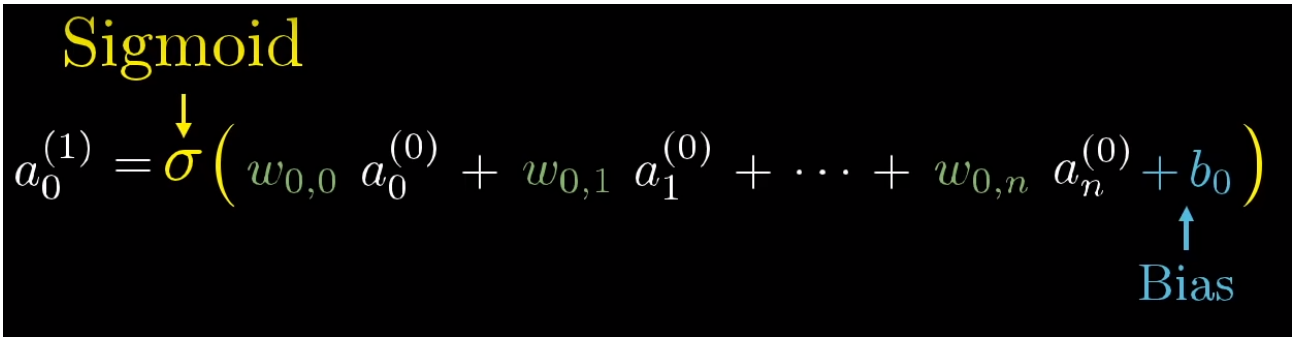
3.3 深度学习的approach

我们选择了通过具有多个隐藏层的前馈神经网络（feedforward neuroal network) 来实现预测

这种神经网络由一个输入层，一个输出层和多个隐藏的层组成。通过矩阵运算和激活函数（activation function），它可以将通过这个神经网络得到的输出与“真实情况”进行比较，使用梯度下降（gradient descend）将误差降至最低，并进行反向传播。



一个关于NN的图示



用activation function, weight 和bias计算通过所有第0层的node的值一个在第1层的node的值

由于存在许多隐藏层，因此必须注意对各个隐藏层的激活函数（activation function）的选择，以免碰到 ”梯度下降消失“ 问题：

- **输入层：** 该层没有激活，因此没有必要选择任何激活功能；
- **隐藏层：** 在各个隐藏层中都有激活函数，但由于其不稳定性无法使用Sigmoid激活函数，因此有必要采用ReLu函数；
- **输出层：** 在输出层中，我们使用了Sigmord函数，以便获得一个事件或另一个事件发生的概率，从而得出概率向量

4. 实验和数据

4.1 LSVM

4.1.1模型实施

为了实现线性SVM方法，我们使用了Scikit Learn的LinearSVC。

对于LSVM来说， 我们主要需要决定的是如何调试以及配置模型的参数。 所以我们决定使用网格搜索（grid search）(也是来自于 Scikit Learn）来找到主要参数的最佳配置， C. 于是我们创造了一个LVSM 模型的instance(这是一个副本的意思吗？） ， 然后使用网格搜索找到了最佳的估计量（estimator）。 也就是说， 这个instance的主要参数， C, 被设置成了网格搜索给出的最佳值。

在执行网格搜索的时候， 因为每个并行的实instance都需要自己的数据集副本：为了提高过程的可并行性，我们只使用了一部分训练数据：

```
sub_train_set = train_set[:round(train_set.shape[0]/x)]
```

以下是对LSVM的instance进行网格搜索时的代码：

```
# Linear SVM model
svc_classifier = LinearSVC ( random_state = 0 , max_iter = 50000)
# Grid Search model
param_grid = {'C':[0.001 ,0.01 ,0.1 ,0.25 ,0.5 ,0.75 ,1 ,10 ,100 ,1000]}
grid = GridSearchCV ( estimator = svc_classifier ,
param_grid = param_grid ,
refit = True ,
verbose = 2 ,
cv = 3 ,
error_score = np . nan ,
n_jobs = -1 ,
pre_dispatch = 6)
grid . fit ( sub_train_set . drop ( columns =[ 'likes ', ' stars_review ',
```

```
14  'user_id ', 'business_id ']) ,
15  sub_train_set ['likes '])
16  Listing 1: Lin
```

最后，我们在整个数据集上用 获得的估计量（estimator）进行了训练，以了解目标标签，并在测试集（test set)上进行了预测。相关代码如下：

```
1   # Estimator to train and predict the label
2   best_model = grid . best_estimator_
3   # Best estimator training
4   best_model . fit ( train_set . drop ( columns =[ 'likes ', ' stars_review
5   'review_id ', 'user_id ', 'business_id ']) ,
6   train_set ['likes '])
7   # Prediction of the target label
8   predic = best_model . predict ( test_set . drop ( columns =[ 'likes ', ' s
9   'review_id ', 'user_id ',
10  'business_id ']))
```

4.1.2 结果

为了在合理的时间内找到好的结果，我们尝试了不同的设置：

• 在应用降维之前，对训练集的1/2进行5000次迭代训练：最佳训练得分0.732，测试得分0.701；

• 在应用降维之前，对训练集的1/3进行10000次迭代训练：最佳训练得分0.725，测试得分0.736；

• 在训练集的1/3上进行训练，经过10000次迭代，阈值为 市 θ 1 = 100，阈值 分类 θ 2 = 200：最佳训练成绩0.722，测试成绩0.706；

• 在整个训练集上进行训练，迭代次数为50000，阈值为 市 和 分类 θ= 100最好的训练分数0.743，测试分数0.737;

但是我们在过去30个小时的长期培训中都未能实现收敛。

	precision	recall	f1-score	support
0	0.70	0.36	0.48	50930
1	0.74	0.92	0.82	103063
macro avg	0.72	0.64	0.65	153993
weighted avg	0.73	0.74	0.71	153993
accuracy	73.727%			

Table 5: Report for SVM model with dimensionality reduction

These are the best-results that we got

- 0.6 percent of overfitting.

4.2 随机森林

4.2.1实施

为了实现整体**随机森林**方法，我们使用了***Scikit Learn***的***RandomForestClassifiner***。

对于**随机森林**来说， 我们主要需要决定的是如何调试以及配置模型的参数。 所以我们决定使用网格搜索（grid search）(也是来自于 Scikit Learn）来找到主要参数们的最佳配置。我们受到knernel ***“Titanic survival prediction”***的启发， 创造了我们的参数网格。

我们使用了RandomForestClassifiner的两个instance：一个在一半的训练集上执行网格搜索，另一个在整个训练集上训练已经获得的模型，并做出预测。

以下是对于第一个instace的网格搜索：

```
# First Random Forest model
random_forest = RandomForestClassifier ( n_jobs = -1 , random_state = 0)
# Grid Search model
param_grid = {'bootstrap ': [ True , False ] ,
'max_depth ': [10 , 30 , 50] ,
```

```
6  ' min_samples_leaf ': [1 , 2 , 4] ,
7  ' min_samples_split ': [2 , 5 , 10] ,
8  ' n_estimators ': [200 , 500 , 1000] ,
9  'criterion ': ['gini ', 'entropy ']}
10 grid = GridSearchCV ( estimator = random_forest ,
11 param_grid = param_grid ,
12 refit = False ,
13 verbose = 5 ,
14 cv = 3 ,
15 error_score = _np . nan ,
16 n_jobs = -1 ,
17 pre_dispatch = 6)
18 sub_train_set = train_set [: round ( train_set . shape [0]/2)]
19 grid . fit ( sub_train_set . drop ( columns =[ 'likes ', ' stars_review ',
20 'user_id ', 'business_id ']) ,
21 sub_train_set ['likes '])
```

网格搜索完成后，我们将拥有一个库，其关键字是网格中使用的参数，而他们对应的值是为相应参数找到的最佳值。

这之后， 我们把随机森林分类算法的参数设置成得出的最佳值。然后在整个训练集上训练模型，并对整个测试集进行预测：

```
1  # Second Random Forest instances with the best value of the params
2  params = grid . best_params_
3  params ['n_jobs '] = -1
4  params ['verbose '] = 5
5  best_model = RandomForestClassifier (** params )
6  # Random Forest training
7  best_model . fit ( train_set . drop ( columns =[ 'likes ',
8  ' stars_review ',
9  'review_id ',
10 'user_id ',
11 'business_id ']) ,
12 train_set ['likes '])
13 # Random Forest prediction
14 predic = best_model . predict ( test_set . drop ( columns =[ 'likes ', ' s
15 'review_id ', 'user_id ',
```

4.2.2 结果

经过三天多的运行，我们获得的最佳测试集分数是0.745，然后我们在整个数据集上面训练了由 GridSearchCV得出的最佳估计量（estimator），并把它用在测试集上面进行预测，得出的对应测试得分为0.741。

The details of those results are shown in table [6].

	precision	recall	f1-score	support
0	0.69	0.40	0.51	50930
1	0.75	0.91	0.83	103063
macro avg	0.72	0.66	0.67	153993
weighted avg	0.73	0.74	0.72	153993
accuracy	74.168%			

Table 6: Report for Random Forest model with dimensionality reduction

From 73.727% to 74.168% the improvement is modest, but not negligible.

4.3前馈神经网络

4.3.1实施

为了在我们的项目中实施神经网络，我们使用**Tensorflow**，利用**Keras' Sequential model**和**Dense layer**，并基于本文开发了该模型：***Building Neural Network using Keras for Classification, Renu Khandelwal***

下面是神经网络的代码：

```
classifier = Sequential ()
# First Hidden Layer
classifier . add ( Dense ( number_hidden_neurons ,
activation = 'relu ',
kernel_initializer = ' random_normal ',
input_dim = number_features ))
# Second Hidden Layer
classifier . add ( Dense ( number_hidden_neurons ,
```

```
activation = 'relu',
kernel_initializer = 'random_normal'))
# Third Hidden Layer
classifier.add(Dense(number_hidden_neurons,
activation = 'relu',
kernel_initializer = 'random_normal'))
# Output Layer
classifier.add(Dense(1,
activation = 'sigmoid',
kernel_initializer = 'random_normal'))
# Compiling the neural network
classifier.compile(optimizer = 'adam',
loss = 'binary_crossentropy',
metrics = ['accuracy'])
```

第一层隐藏层被加入模型通过了一个参数， input dim=n features； 它代表了第一层需要的神经元数量（每个特征都有一个）。

第二层和第三层隐藏层被加到模型中的时候，我们用了以下公式来决定这两层中神经元的数量：

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))} \tag{2}$$

where:

- N_h is the number of hidden neurons.
- N_i is the number of input neurons.
- N_o is the number of output neurons.
- N_s is the number of train samples.
- α an arbitrary scaling factor, usually between 5 and 10.

*来自于***StackExchange***上的 ***How to choose the number of hidden layers and nodes in a feedforward neural network?***

对于输出层，我们仅配置一个神经元：该任务需要二进制分类（是/否），因此我们表达的几率为：

P（是）= 1个 - P（否）

我们可以在输出层中使用两个神经元，但是它仍然代表相同的信息。

关于训练模型， 我们依旧使用Keras：

```
# Fitting the data to the training dataset
classifier . fit ( train_set . drop ( columns = ['likes ', ' stars_review
'review_id ', 'user_id ',
'business_id ']) ,
train_set ['likes '] ,
validation_split = 0.3 ,
batch_size = 100 ,
epochs = 100)
```

在训练过程中，我们的模型执行100次迭代，每次迭代的batch大小为100。训练测试的一部分将用作验证测试，以使比率达到70-30。

关于概率预测， 我们再次使用Keras模型：

```
prediction = classifier . predict ( test_set . drop ( columns = ['likes ',
' stars_review ',
'review_id ',
'user_id ',
'business_id ']))
# Result binarization
binary_prediction = binarize ( prediction , threshold = 0.5)
得到
```

这样以来， 我们得到了一个概率向量。 这时我们使用Scikit Learns 的binarize函数来把这个结果变成一个二进制数， 阈值为0.5/

4.3.2 结果

对于这个模型，我们从***StackExchange***上的 ***How to choose the number of hidden layers and nodes in a feedforward neural network?*** 中提出的架构和开始，并使用了值为α= 6的hyperoarameter。我们观察到， 降低α（ 同时也增加隐藏层的数量）回事结果变差，而使用α= 7和5个隐藏层（而不是3个） 会带来更好的效果。当α继续上升的时候，结果不会再有提升。

以下是我们的实验结果：

	precision	recall	f1-score	support
0	0.69	0.43	0.53	50930
1	0.76	0.90	0.83	103063
macro avg	0.72	0.67	0.68	153993
weighted avg	0.74	0.75	0.73	153993
accuracy	74.679%			

Table 7: Report for Neural Network model with dimensionality reduction

5.总结

目前为止我们所做的：

1.我们下载了Yelp数据集挑战的数据集，其中包含有关许多不同城市的餐馆，用户和评论的信息，以预测某个用户是否会喜欢某个餐馆；

2.我们使用了Recommendation for yelp users itself, Wenqi Hou, Gauravi Saha, Manying Tsang的步骤，执行了第一次数据清理（请参阅第[2.1]节）；

3.我们应用了在Machine Learning and Visualization with Yelp Dataset, Zhiwei Zhang中的一个模型，为每个评论分配两个 真实分数 二进制版本 指出评论是真是假，以及代表评论可信的可能性的真实版本（请参见[2.2]节）；

4.我们将评论数据集分为三个部分：历史，培训和测试；

5.我们使用第一部分来计算 在Restaurant Recommendation System, Ashish Gandhe中描述的 历史特征（historical features)，但我们为每个 新功能根据我们的真实分数提供了两个版本（请参见[2.3]节）；

6.我们添加了一个基于协作过滤的新功能，即我们使用了类似用户的投票来预测某个用户将对某家餐厅进行的投票（请参阅[2.4]节）；

7.我们限制了功能的可能值 市 和 分类 减小我们的最终尺寸 删除出现次数少于固定阈值的数据集（请参阅[2.5]节）；

8.我们实施了三种模型来进行预测：LSVM，随机森林和神经网络（请参阅[3]节）；

9.我们使用不同版本的数据集训练和测试了这三个模型（请参见第[4]节）。

6.References

1. Yelp dataset on Kaggle https://www.kaggle.com/yelp-dataset/yelp-dataset;
2. Yelp Dataset presentation https://www.yelp.com/dataset;
3. Restaurant Recommendation System, Ashish Gandhe https://www.semanticscholar.org/paper/Restaurant-Recommendation-System-Gandhe/093cecc3e 53f2ba4c0c466ad3d8294ba64962050;
4. Machine Learning and Visualization with Yelp Dataset, Zhiwei Zhang https://medium.com/@zhiwei zhang/final-blog-642fb9c7e781 (with her repo https://github.com/zzhang83/Yelp Sentiment Analysis);
5. Recommendation for yelp users itself, Wenqi Hou, Gauravi Saha, Manying Tsang https://www.kaggle.com/wenqihou828/recommendation-for-yelp-users-itself;
6. Titanic Survival Predicton https://www.kaggle.com/sociopath00/random-forest-using-gridsearchcv;
7. Building Neural Network using Keras for Classification, Renu Khandelwal https://medium.com/datadriveninvestor/building-neural-network-using-keras-for-classification-3a 3656c726c1;
8. How to choose the number of hidden layers and nodes in a feedforward neural network? https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-andnodes-in-a-feedforward-neural-net
9. Recommendation system for restaurants, Giovanni Ficarra, Leonardo Picchiami