中山大学数据科学与计算机学院
计算机科学与技术专业-人工智能
本科生实验报告

（2018-2019 学年秋季学期）

学　　号：16337113
姓　　名：劳马东
教学班级：教务 2 班
专　　业：超算

1 实验内容

1.1 算法原理

1.1.1 GBDT 回归树

1、 概述

GBDT(Gradient Boosting Decision Tree) 是一种迭代的决策树算法，该算法由多棵决策树组成，所有树的结论累加起来做最终答案。其核心就在于，每一棵树学的是之前所有树结论累加和的残差，即与真实结论的差，形式化的定义为：

$$Y^i = Y_{true} - \sum_{k=0}^{i-1} Y^k_{predict} \ , \ i \geq 0 \tag{1}$$

比如 A 的真实年龄是 18 岁，但第一棵树的预测年龄是 12 岁，差了 6 岁，即残差为 6 岁。那么在第二棵树里我们把 A 的年龄设为 6 岁去学习，如果第二棵树真的能把 A 分到 6 岁的叶子节点，那累加两棵树的结论就是 A 的真实年龄；如果第二棵树的结论是 5 岁，则 A 仍然存在 1 岁的残差，第三棵树里 A 的年龄就变成 1 岁，继续学，以此类推，直到残差小到一定程度。

2、 训练过程

(1) 初始化残差 E_0 为 Y，i 为 0；

(2) 增加一棵决策树 T_i 在数据集 (X, E_i) 上训练；

(3) T_i 预测 X 得到预测值 $Y^i_{predict}$；

(4) 计算新的残差 $E_{i+1} = E_i - Y^i_{predict}$；

(5) 如果残差收敛，迭代结束，否则 i 增加 1，回到第 2 步。

3、 Shrinkage

Shrinkage（缩减）的思想认为，每次走一小步逐渐逼近结果的效果，要比每次迈一大步很快逼近结果的方式更容易避免过拟合。即它不完全信任每一个棵残差树，它认为每棵树只学到了真理的一小部分，累加的时候只累加一小部分，通过多学几棵树弥补不足，具体到公式，就是：

$$E_{i+1} = E_i - \theta \times Y^i_{predict} \tag{2}$$

其中 θ 是缩减率，也叫学习率、可信度。

4、 预测过程

预测的过程就是把所有决策树的结果累加的过程，即：

$$Y_{predict} = \sum_{i=0}^{n} (\theta \times Y^i_{predict}) \tag{3}$$

1.1.2 逻辑回归

1、 概述

二元逻辑回归是计算在给定 x 的情况下，属于某类的概率，即：

$$f(x) = P(label|x) \in [0, 1] \tag{4}$$

它属于一种线性算法，即 x 的每个维度的最高次幂都是 1。特征向量 x 的每一个维度，都会对结果产生影响，所以我们希望可以给每一个特征赋予一个权重来计算结果：

$$s = w_0 + \sum_{i=1}^{d}(w_i \times x_i) \tag{5}$$

表达式中的 w_i 表示第 i 维特征的权重，$w_i > 0$ 表示该特征对正类别有正面影响，且值越大，正面影响越大，反之亦然。

问题是这样计算出来的 s 取值是 $(-\infty, +\infty)$ 而不是一个概率值，因此需要一种转换——*sigmoid* 函数。

$$g(z) = \frac{1}{1 + e^{-z}} \tag{6}$$

其函数曲线如图 1，在 $z \to -\infty$ 时，其取值为 0，$z \to +\infty$ 时取值为 1，在远离 0 的地方函数值越接近 0 或 1，函数值为 0.5 是一种不可区分的状态。

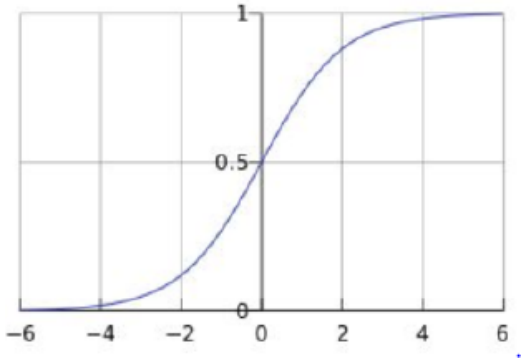


图 1: sigmoid 函数图像

因此，$f(x)$ 实际上是一个复合函数，为：

$$f(x) = g(s) = g(w\widetilde{x}) = \frac{1}{1 + e^{-w\widetilde{x}}} \in (0, 1) \tag{7}$$

其中 $\widetilde{x}$ 表示特征向量 x 加上第 0 维的 1 后的增广向量，即 $\widetilde{x} = (1, x_1, x_2, ..., x_d)$。

2、 训练过程

训练过程就是计算得到 w 的过程。假设有一个基于 sigmoid 函数的损失函数 $e(w)$ 是关于 w 的函数，我们的目标是使预测值与真实值的误差最小，即 $e(w)$ 取最小值，对应的 w 即为所求。在数学上，可以通过求导的方法计算极小值点，进而得到最小值点，但问题是很多损失函数难以求导，因此需要用到梯度下降法。

梯度下降法最终的目标是让 w 走到 $e(w)$ 导数为 0 的地方。假设 $e(w)$ 是一个开口向上的函数，如果当前 w 处在右半边，即 $e^{'}(w) > 0$，那么 w 就应该往左走; 如果 w 在左半边，即 $e^{'}(w) < 0$，那么 w 就应该往右走，不断重复直到 $e^{'}(w) = 0$。

3、 预测过程

预测过程很简单，得到 w 之后，计算 $f(x)$，如果概率大于 0.5，认为属于正类别，小于 0.5 属于负类别，等于 0.5 不可区分或者归到某一类别中。

4、 从二元到多元——oneVSall

假设数据集一共有 n 类，分别记为 $L1, L2, ..., L_n$，oneVSall 的做法是每次取一个类别 L_i 作为正类别，其余类别全部作为负类别，将其输入到逻辑回归模型中训练，这样得到一组权重 W_i。重复这个过程 n 次，就能得到 n 组权重，记为 $W_1, W_2, ..., W_n$，这就是整个训练过程。

在预测时，同样是每次选择一个类别作为正类别，用逻辑回归的方法得到一个预测概率 p_i，特征向量 x 的类别是这些概率的最大值对应的类别，即：

$$f(x) = \arg\max_i \frac{1}{1 + e^{-W_i \widetilde{x}}} \tag{8}$$

1.2 伪代码

1、 GDBT 训练过程

输入: 训练集 X，训练集 y, 学习率 θ

输出: 训练好的多棵决策树

function $fit(X, y, \theta)$
  $T \leftarrow$ *collection of dicision trees*
  $E_{new} \leftarrow y$
  $i \leftarrow 0$
  repeat
    $E \leftarrow E_{new}$
    $T_i \leftarrow$ *new dicision tree*
    $T_i.fit(X, E)$
    $Y_p^i \leftarrow T_i.predict(X)$
    $E_{new} \leftarrow E - \theta \times Y_p^i$
    $i \leftarrow i + 1$
  until E_{new} *is not convergent*
  return T
end function

2、 logistic 回归训练过程

输入: 训练集 $X(m \times n)$，训练集 y, 学习率 θ, 损失函数 E

输出: 收敛的权重

function $fit(X, y, \theta)$
    $a\_X \leftarrow argmented(X)$
    $W_{new} \leftarrow array\ of\ n+1\ random\ value$
    repeat
        $W \leftarrow W_{new}$
        $W_{new} = W - \theta \times \frac{\partial E(W)}{\partial W}$
    until W_{new} *is convergent*
    return W_{new}
end function

3、 oneVSall 训练过程

输入: 训练集 $X(m \times n)$，训练集 y

输出: 对应每个类别训练好的二元分类模型

function $fit(X, y)$
    $clfs \leftarrow\ collection\ of\ binary\ classifiers$
    for $label \in unique(y)$ **do**
        $y\_new \leftarrow to\_binary(y, label)$
        $clfs_i \leftarrow\ new\ binary\ classifier$
        $clfs_i.fit(X, y\_new)$
    end for
    return $clfs$
end function

1.3 关键代码

1、 GBDT 训练

在实际情况下，残差是很难收敛的，因此为了避免无限迭代，需要设置残差的阈值或决策树的数量，二者的效果是等价的。

图 2 的代码中 $n\_estimators$ 是决策树的数量，$RegressionTree$ 是 $CART$ 树，其三个参数用于剪枝。

2、 GBDT 预测

预测是训练的逆过程，训练时每棵树的预测结果只才用了一小部分，因此预测时也应该如此。

```
residual_error = y.copy()
for i in range(self.n_estimators):
    self.weighted_trees.append(RegressionTree(min_samples=self.min_samples,
                                              min_impurity=self.min_impurity,
                                              max_depth=self.max_depth))
    # GDBT模型每棵树训练的label实际上残差
    # 残差Ri = y - sum(y_predict_k), k=0,1,2,...,i-1
    self.weighted_trees[-1].fit(X, residual_error)
    y_pred = self.weighted_trees[-1].predict(X)
    # 对于残差学习出来的结果，只累加一小部分（learning_rate）逐步逼近目标
    # 即每次只纠正一点点错误而不是一步纠正，有效避免过拟合
    residual_error -= self.learning_rate * y_pred
```

图 2: GBDT 训练

图 3 代码中，*map* 函数将每棵树映射为其预测结果乘上学习率，这可以提高并行度，减少预测时间。每棵树的预测结果的累加和组成了最终的预测结果。

```
def predict(self, X):
    y_pred_res = np.zeros((X.shape[0], ))
    # 由于训练的时候只取训练结果的一小部分，预测的时候也要乘learning_rate
    for y_pred in map(lambda t: self.learning_rate * t.predict(X), self.weighted_trees):
        y_pred_res += y_pred
    return y_pred_res
```

图 3: GBDT 预测

3、 逻辑回归训练

判断是否收敛的方法有两种，一种是判断权重向量本身，另一种是判断损失函数值是否达到一定阈值。项目中使用了后者。

代码如图 4。*argmented* 函数在 X 每一行的第一个位置插入一个 1，返回增广的 X，简单起见权重向量 W 被初始化为全 0；以交叉熵 *cross_entropy* 作为损失函数，其导数 *gradient_ce* 作为梯度函数；函数 *fmin_tnc* 使用梯度下降法优化 W 直到收敛。

```
def fit(self, X, y):
    X = self.argmented(X)
    W = np.zeros(X.shape[1])
    result = opt.fmin_tnc(func=cross_entropy, x0=W,
                          fprime=gradient_ce, args=(X, y))
    self.__W = result[0]
```

图 4: 逻辑回归训练

4、 oneVSall 预测

如图 5，在预测过程中需要记录两个量：当前的预测结果 *y_pred* 及其对应的概率 *y_prob*，预测时遍历每个类别 c，用这个类别对应的分类器来预测 X 得到一个结果 *y_p*，

这个结果是概率数组而不是类别数组。有了 y_p 和 y_prob，就能比较他们对应位置的元素，如果 y_p 的概率比 y_prob 大，就应该更新 y_prob 和 y_pred。

```
def predict(self, X):
    y_pred = np.zeros(X.shape[0])
    y_prob = np.zeros(X.shape[0])
    for c in self.__y_unique:
        y_p = self.classifiers[c].predict_prob(X)
        for i in range(X.shape[0]):
            if y_p[i] > y_prob[i]:
                y_prob[i] = y_p[i]
                y_pred[i] = c
    return y_pred
```

图 5: oneVSall 预测

2 创新点 & 优化

2.1 GBDT 回归树

1、 *tag* 处理

(1) 基于频数

不同的 tag 有 1996 个，数据集总共 80000 行，挑选出其中出现次数超过一定阈值的 tag 作为特征。实验中发现阈值为 5000 时最佳，最终挑选出的 tag 如图 (a)。

```
Out[15]: {'Business trip',
          'Couple',
          'Deluxe Double Room',
          'Double Room',
          'Family with older children',
          'Family with young children',
          'Group',
          'Leisure trip',
          'Solo traveler',
          'Standard Double Room',
          'Stayed 1 night',
          'Stayed 2 nights',
          'Stayed 3 nights',
          'Stayed 4 nights',
          'Submitted from a mobile device',
          'Superior Double Room'}
```

(a) 频数超过 5000 的 tag

```
Out[18]: {'Budget Single Room Non Smoking',
          'Business trip',
          'Couple',
          'Deluxe Queen Guestroom',
          'Double Queen Waterfront',
          'Double or Twin Room with Sea View',
          'Duplex',
          'Embassy Suite',
          'Family with young children',
          'Group',
          'King Duplex Suite',
          'King Hilton Sea View',
          'King Room with Balcony',
          'Large Double Room',
          'Leisure trip',
          'Prestige Suite',
          'Privilege Junior Suite with Spa Access',
          'Privilege Room with 1 Queen Size bed',
          'Queen Guestroom',
          'Solo traveler',
          'Stay 3 nights',
          'Studio King Family'}
```

(b) 效果最好的 22 个 tag

(2) 基于效果

最直接的方法就是将 tag 逐个有放回地加入到属性中，观察其对最终验证集相关系数的影响，选出最优的一些。实验中有选择地对一些 tag 做了测试，最终挑选的 22 个最好的 tag 如图 (b)。

(3) 基于类别

仔细分析 tag 可以发现可以大致分为以下几类：

i. Leisure or Business Trip

ii. number of people

iii. room

iv. stay n nights

v. submitted from a mobile device or not

这五类 tag 可以作为五个特征，给每一行数据在该特征上赋予特定的值（如 Leisure Trip 为 0、Business Trip 为 1，solo 为 1、couple 为 2、family with young children 为 3 等）。

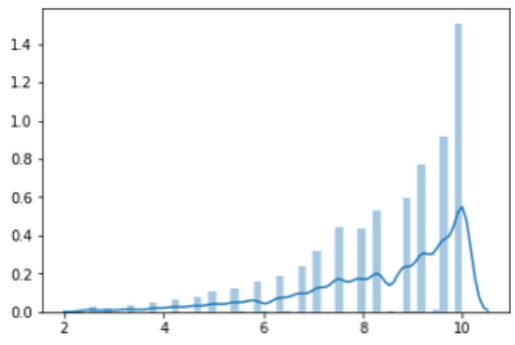
2、 优化数据分布不均的问题

查看 y 的分布，会发现数据是极度右偏的，如下图 (c)。而许多回归模型都假设数据是正态分布的，因此需要将偏态分布转化为正态分布，常用的方法是 box-cox 变换。

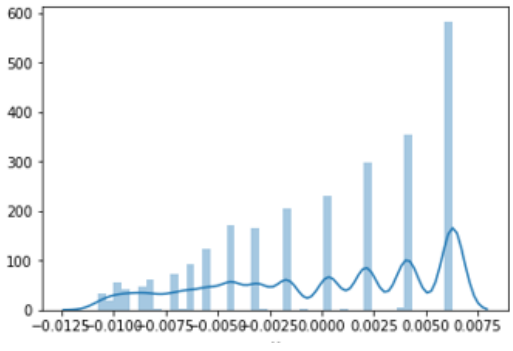
$$y(\lambda) = \begin{cases} \ln(y) & \lambda = 0 \\ \frac{y^{\lambda}-1}{\lambda} & \lambda \neq 0 \end{cases} \tag{9}$$

式中 $y(\lambda)$ 为经 Box-Cox 变换后得到的新变量，y 为原始连续因变量，要求取值为正，λ 为变换参数。项目中使用了 scipy 库的 box-cox 函数，其返回 λ 的值和变换后的 y。利用该 λ 值能得到逆变换后的 y，即：

$$y = \begin{cases} e^{y(\lambda)} & \lambda = 0 \\ (y(\lambda) \times \lambda + 1)^{\frac{1}{\lambda}} & \lambda \neq 0 \end{cases} \tag{10}$$



(c) y 的分布



(d) box-cox+ 标准正态转化-y 的分布

3、 特征筛选

计算各个特征与 y 的 pearson 相关系数（如图 3，已取绝对值），从中删除一些相关性较低的 tag 类别。可以看到 Total_Number_of_Reviews_Reviewer_Has_Given 和 mobile_device 两个特征相关性极低，可以删除。

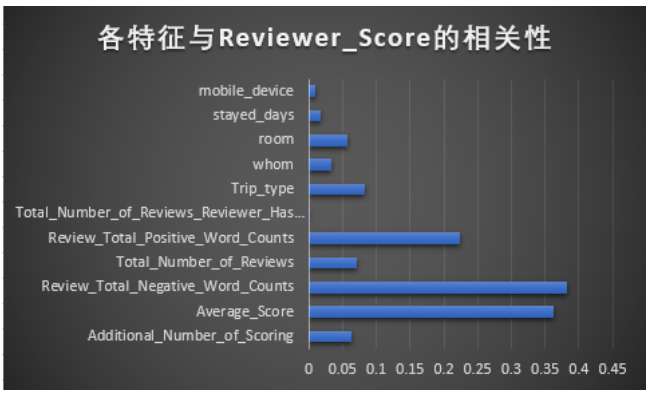


图 6: 各特征与 y 的相关系数

2.2 doc2vec 而不是 word2vec

word2vec 可以将词转化为一个 K 维的向量，而段落/文本的向量就是其含有的词对应的向量的均值，因此段落/文本之间的相似性也就转化为向量之间的距离（如余弦距离、欧氏距离）。

然而，即使上述模型对词向量进行平均处理，我们仍然忽略了单词之间的排列顺序对情感分析的影响。因为 word2vec 只是基于“词”的维度进行语义分析，而并不具有“上下文”的语义分析能力。Quoc Le 和 Tomas Mikolov 提出了 Doc2Vec 方法，该方法在 word2vec 的基础上增加一个段落向量，使用一个 paragraph id 增强了同一段落之间不同句子之间的联系，从而具有“上下文”语义分析的能力。

3 实验结果及分析

3.1 实验结果展示

1、 GBDT 回归树取实验数据训练集的前 16 行作为训练集，16-21 行作为测试集，如下图 (a)、(b)、(d)：

	a	b	c	d	e	f	y
0	255.0	8.1	57.0	1290.0	10.0	1.0	6.3
1	211.0	8.6	0.0	2923.0	19.0	12.0	10.0
2	189.0	9.2	27.0	781.0	29.0	24.0	10.0
3	1258.0	9.4	20.0	4204.0	31.0	3.0	9.2
4	289.0	8.8	0.0	1519.0	3.0	9.0	10.0
5	563.0	8.3	30.0	10842.0	26.0	8.0	9.6
6	728.0	8.9	3.0	3168.0	12.0	21.0	9.6
7	192.0	8.4	4.0	1769.0	8.0	1.0	8.8

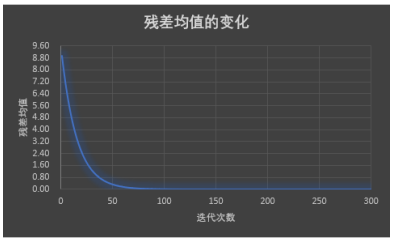
(a) 训练集

	a	b	c	d	e	f	y
8	73.0	9.1	12.0	619.0	2.0	25.0	9.6
9	587.0	7.4	16.0	3869.0	4.0	4.0	8.3
10	728.0	8.9	7.0	3168.0	28.0	1.0	10.0
11	429.0	8.6	106.0	1462.0	12.0	1.0	7.5
12	348.0	8.0	0.0	1710.0	14.0	9.0	9.2
13	995.0	8.1	16.0	3826.0	7.0	4.0	7.1
14	46.0	9.1	5.0	221.0	4.0	2.0	8.8
15	2682.0	7.1	21.0	9086.0	10.0	2.0	8.8

(b) 训练集

使用 300 棵 CART 树，学习率为 0.065，CART 树的最大深度为 3。训练过程中残差均

值的变化趋势如下图 (c)，一开始时残差减小较快，在迭代 100 次之后基本趋于 0。用训练好的模型预测测试集，结果如图 (d) y_pred，计算得相关系数为 0.96：



(c) 训练过程残差均值变化

	a	b	c	d	e	f	y_pred	y_true
0	1322.0	8.4	52.0	6117.0	4.0	1.0	7.583152	5.8
1	282.0	8.6	0.0	2875.0	4.0	2.0	8.882576	8.3
2	180.0	8.7	0.0	1766.0	30.0	6.0	9.641562	10.0
3	252.0	8.1	7.0	1600.0	5.0	1.0	8.241960	7.1
4	570.0	9.3	11.0	2319.0	9.0	16.0	9.648609	8.8
5	87.0	9.2	21.0	579.0	34.0	2.0	9.589217	10.0

(d) 测试集与预测结果

2、 逻辑回归

训练集如下，含有两个特征，分别是两门考试的成绩，y 为是否通过。

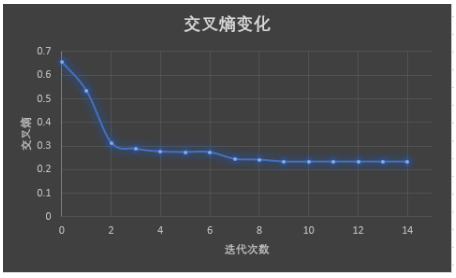
	exam1	exam2	admitted
0	34.623660	78.024693	0
1	30.286711	43.894998	0
2	35.847409	72.902198	0
3	60.182599	86.308552	1
4	79.032736	75.344376	1
5	45.083277	56.316372	0
6	61.106665	96.511426	1
7	75.024746	46.554014	1

(e) 训练集

	exam1	exam2	admitted
8	76.098787	87.420570	1
9	84.432820	43.533393	1
10	95.861555	38.225278	0
11	75.013658	30.603263	0
12	82.307053	76.481963	1
13	69.364589	97.718692	1
14	39.538339	76.036811	0
15	53.971052	89.207350	1

(f) 训练集

使用交叉熵作为损失函数，其导数作为梯度函数，调用 scipy.optimize 模块的 fmin_tnc 函数训练，期间交叉熵的变化情况如下图 (g)，一开始时错误减小较快，在第 9 次迭代之后基本稳定在 0.2327 左右。得到的权重向量 W 为：$(-18.66001375, 0.14838242, 0.14839205)$，预测结果如下图 (h)：



(g) 训练过程交叉熵变化

	exam1	exam2	admitted	admitted_pred
0	69.070144	52.740470	1	0
1	67.946855	46.678574	0	0
2	70.661510	92.927138	1	1
3	76.978784	47.575964	1	0
4	67.372028	42.838438	0	0
5	89.676776	65.799366	1	1

(h) 测试集与预测结果

3.2 评测指标展示及分析

3.2.1 GBDT 回归树

1、 相关系数

使用 300 棵 CART 树，学习率为 0.065，CART 树的最大深度为 3。采用 5 折交叉验证，分别评测基本数据集和各种数据集处理方法在验证集上的相关系数以及上交后在测试集上的相关系数，结果如下表。

不同的优化方法在验证集和测试集上都得到了不同程度的提升。基于频数的处理方法提升最小，这是因为出现次数多的 tag 区分能力不强；基于类别的方法在测试集上达到了一个新的数量级——0.66，因为这种方法给每行数据的赋值是有意义的，值的大小能体现 tag 在某一类 tag 中的重要程度；基于效果的处理方法在测试集上效果最好，这毋庸置疑，它本身就是从模型出发，挑选出对模型提升最有利的 tag；box-cox 变换无论在验证集和测试集没有达到预期的效果，初步认为是测试集上 y 的分布和训练集不一致导致的。

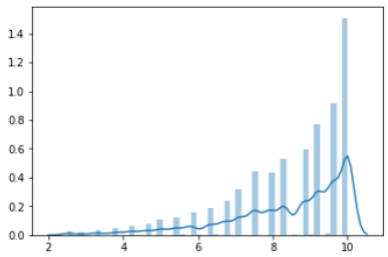
优化方法	验证集相关系数	测试集相关系数
-	0.6538	0.6540
tag 处理-基于频数	0.6575	0.6575
tag 处理-基于类别	0.6574	0.6614
tag 处理-基于效果	0.6572	0.6616
基于效果 +box-cox 变换	0.6575	0.6598

2、 预测 y 与原始 y 的分布

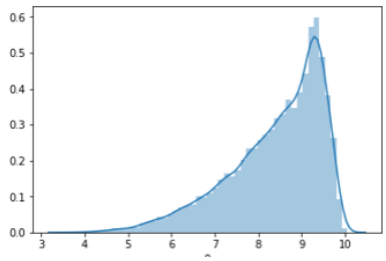
我们的目标是使预测出来的 y（记为 y_p）与训练集上 y（记为 y_t）分布尽可能一致。

在不做任何变换之前，预测的 y（记为 y_p^0）与 y_t 分布差别较大——y_t 是极度右偏的，左边几乎没有分布，而 y_p^0 在左边部分明显过多；

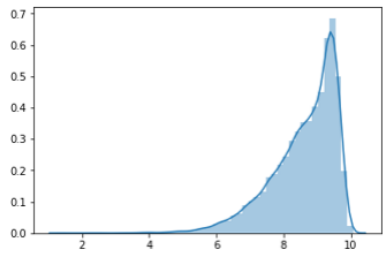
经过 box-cox 变换后，预测的 y（记为 y_p^{bc}）依然极度右偏，但是左边的分布明显减少，与 y_t 基本一致。



(i) y_t 的分布



(j) 无变换-y_p^0 的分布



(k) box-cox 变换-y_p^{bc} 的分布

3.2.2 分类问题——逻辑回归

1、 准确率

实验中对比了两种文本转向量方法——word2vec 和 doc2vec。word2vec 使用的模型是 glove 官网提供的 glove.840B.300d.txt；doc2vec 使用的是自己训练的模型，参数 window=5, vector_size=150, negative=25, epochs=25, alpha=0.05。

最终两者 5 折交叉验证的准确率如下表。在二分类问题中，doc2vec 相对于 word2vec 提高了将近 2%，效果明显；而在五分类问题中，doc2vec 准确率低于 word2vec，猜测原因是类别多了之后段与段之间的联系不再那么紧密，doc2vec 模型转换得到的文本向量与原文本的意思有较大出入。

方法	二分类验证集准确率	五分类验证集准确率
word2vec	0.8714	43.3266
doc2vec	0.8901	43.3101