基于集成学习的 Amazon 用户评论质量预测

实验报告

班级：学堂在线 1期机器学习-训练营

姓名：王雨静

时间：2020年12月23日

写在前面

本实验的最终代码是在Google Colab上运行的。

目录

一、案例简介

随着电商平台的兴起，以及疫情的持续影响，线上购物在我们的日常生活中扮演着越来越重要的角色。在进行线上商品挑选时，评论往往是我们十分关注的一个方面。然而目前电商网站的评论质量参差不齐，甚至有水军刷好评或者恶意差评的情况出现，严重影响了顾客的购物体验。因此，对于评论质量的预测成为电商平台越来越关注的话题，如果能自动对评论质量进行评估，就能根据预测结果避免展现低质量的评论。本案例中我们将基于集成学习的方法对 Amazon 现实场景中的评论质量进行预测。

二、作业说明

本案例中需要大家完成两种集成学习算法的实现（Bagging、AdaBoost.M1），其中基分类器要求使用 SVM 和决策树两种，因此，一共需要对比四组结果（AUC 作为评价指标）：

- Bagging + SVM
- Bagging + 决策树
- AdaBoost.M1 + SVM
- AdaBoost.M1 + 决策树

注意集成学习的核心算法需要**手动进行实现**，基分类器可以调库。

基本要求

- 根据数据格式设计特征的表示
- 汇报不同组合下得到的 AUC
- 结合不同集成学习算法的特点分析结果之间的差异
- （使用 sklearn 等第三方库的集成学习算法会酌情扣分）

扩展要求

- 尝试其他基分类器（如 k-NN、朴素贝叶斯）
- 分析不同特征的影响
- 分析集成学习算法参数的影响

三、赛题说明

数据描述

本次数据来源于 Amazon 电商平台，包含超过 50,000 条用户在购买商品后留下的评论，各列的含义如下：

- reviewerID：用户 ID
- asin：商品 ID
- reviewText：英文评论文本
- overall：用户对商品的打分（1-5）
- votes_up：认为评论有用的点赞数（只在训练集出现）
- votes_all：该评论得到的总评价数（只在训练集出现）
- label：评论质量的 label，1 表示高质量，0 表示低质量（只在训练集出现）

评论质量的 label 来自于其他用户对评论的 votes，votes_up/votes_all ≥ 0.9 的作为高质量评论。此外测试集包含一个额外的列Id，标识了每一个测试的样例。

文件说明

- train.csv：训练集
- test.csv：测试集，用户和商品保证在训练集中出现过，没有关于 votes 和 label 的列

文件使用 \t 分隔，可以使用 pandas 进行读取：

```
import pandas as pd

    train_df = pd.read_csv('train.csv', sep='\t')
```

提交格式

提交文件需要对测试集中每一条评论给出预测为高质量的**概率**，每行包括一个Id（和测试集对应）以及预测的概率Predicted（0-1的浮点数），用逗号分隔。示例提交格式如下：

```
Id,Predicted
0,0.9
1,0.45
2,0.78

    ...
```

提交文件需要命名为result.csv

四、特征提取

4.1 文本算术特征提取

使用“reviewText”列，提取字数计数、字符计数、句子数、平均字数、平均句子长度的信息。
计算方法：

- 字数计数:计算文本中记号的数量(用空格分隔)
- 字符计数:将每个标记的字符数相加计算
- 句子数:计算句子的数量(以句点分隔)
- 平均字数:字数除以字数的总和(字符数/字数)
- 平均句子长度:句子长度的总和除以句子的数量(字数/句子数量)

代码实现：

```
#增加计数列
clean_df['word_count'] = clean_df["reviewText"].apply(lambda x: len(str(x).split(" ")))
clean_df['char_count'] = clean_df["reviewText"].apply(lambda x: sum(len(word) for word in str(x).split(" ")))
clean_df['sentence_count'] = clean_df["reviewText"].apply(lambda x: len(str(x).split(".")))
clean_df['avg_word_length'] = clean_df['char_count'] / clean_df['word_count']
clean_df['avg_sentence_lenght'] = clean_df['word_count'] / clean_df['sentence_count']
```

4.2 文本情绪特征提取

调用TextBlob的函数进行情绪特征的提取。

代码实现：

```
# 新增情绪列
from textblob import TextBlob
clean_df["sentiment"] = clean_df['reviewText'].apply(lambda x: TextBlob(x).sentiment.polarity)
```

4.3 Tfldf文本特征提取

调用sklearn库，对词频-逆向文件频率特征进行提取，ngram_range = (1, 2)。并使用卡方检验，选出其中最优代表性的50个词。
代码实现：

```
#用tfidf提取文本特征，ngram_range = (1, 2)
from time import time
from sklearn.feature_extraction.text import TfidfVectorizer

#将reviewText转化成稀疏矩阵
print("Extracting features from the training data using a sparse vectorizer")
t0 = time()
vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.5,
                                 stop_words='english', ngram_range = (1, 2))
tfidf_train = vectorizer.fit_transform(train_df['reviewText'])
duration = time() - t0
print("done in %fs" % duration)
print("n_samples: %d, n_features: %d" % tfidf_train.shape)
print()

print("Extracting features from the test data using the same vectorizer")
t0 = time()
tfidf_test = vectorizer.transform(test_df['reviewText'])
duration = time() - t0
print("done in %fs " % duration)
print("n_samples: %d, n_features: %d" % tfidf_test.shape)
print()
feature_names = vectorizer.get_feature_names() #所有特征的名字

#选取最好的50个特征
from sklearn.feature_selection import SelectKBest, chi2
print("Extracting 50 best features by a chi-squared test" )
t0 = time()
ch2 = SelectKBest(chi2, k=50)
tfidf_best_train = ch2.fit_transform(tfidf_train, train_df['label'])
tfidf_best_test = ch2.transform(tfidf_test)

new_feature_names = [feature_names[i] for i
                         in ch2.get_support(indices=True)]
print("done in %fs" % (time() - t0))
print()
print(new_feature_names)
```

4.4 清理文本

调用nltk库，删除文本中的“stopword”, 并进行词干化（stemming）/词元化（lemmatisation），最后，只取前20个词返回。储存在‘text_clean’列中。
代码实现：

```
import nltk ## for language detection
import re
'''
Preprocess a string.
:parameter
    :param text: string - name of column containing text
    :param lst_stopwords: list - list of stopwords to remove
    :param flg_stemm: bool - whether stemming is to be applied
    :param flg_lemm: bool - whether lemmitisation is to be applied
    :param max_len: int - max number of words in the text
:return
    cleaned text
'''
nltk.download('stopwords')
nltk.download('wordnet')
def utils_preprocess_text(text, flg_stemm=False, flg_lemm=True, lst_stopwords=None, max_len = 20):
    ## clean (convert to lowercase and remove punctuations and characters and then strip)
    text = re.sub(r'[^\w\s]', '', str(text).lower().strip())

    ## Tokenize (convert from string to list)
    lst_text = text.split()    ## remove Stopwords
    if lst_stopwords is not None:
        lst_text = [word for word in lst_text if word not in
                    lst_stopwords]

    ## Stemming (remove -ing, -ly, ...)
    if flg_stemm == True:
        ps = nltk.stem.porter.PorterStemmer()
        lst_text = [ps.stem(word) for word in lst_text]

    ## Lemmatisation (convert the word into root word)
    if flg_lemm == True:
        lem = nltk.stem.wordnet.WordNetLemmatizer()
        lst_text = [lem.lemmatize(word) for word in lst_text]

    lst_text = lst_text[:max_len]

    ## back to string from list
    text = " ".join(lst_text)
    return text
```

最后将4.1、4.2、4.4节提取到的特征保存到clean.csv，clean_test.csv文件中。

```
clean_df.to_csv('/content/drive/MyDrive/Colab Notebooks/xtzx/hw6/data/clean.csv', sep = '\t')
```

4.5 BERT特征提取

调用transformer库预训练好的'bert-base-uncased'模型对在4.4节清理过的文本进行分词和特征提取。使用GPU进行加速。最后保存到bert_clean.npy，bert_clean_test.npy文件中。

```
[ ]  text = train_df['text_clean'][0]
     encoded_input = tokenizer(text, return_tensors='pt').to('cuda')
     output = model(**encoded_input)
```

```
[ ]  feature = output[0][:,0,:].cpu().detach().numpy()
     feature.shape
```

```
(1, 768)
```

```
     text = train_df['text_clean'][1]
     encoded_input = tokenizer(text, return_tensors='pt').to('cuda')
     output = model(**encoded_input)
     feature2 = output[0][:,0,:].cpu().detach().numpy()
     feature2.shape
```

```
(1, 768)
```

```
[ ]  old = np.vstack((feature, feature2))
     all_bert = old.copy()
```

```
[ ]  all_bert
```

```
array([[-0.4800572 , -0.03585712,  0.23369914, ..., -0.4476912 ,
         0.51827276,  0.46136004],
       [-0.37914237,  0.09411145,  0.05395877, ..., -0.47702298,
         0.60461056,  0.29169947]], dtype=float32)
```

```
t0 = time()
for i in range(2, 57039):
  text = train_df['text_clean'][i]
  encoded_input = tokenizer(text, return_tensors='pt').to('cuda')
  output = model(**encoded_input)
  featurei = output[0][:,0,:].cpu().detach().numpy()
  all_bert = np.vstack((all_bert, featurei))
  if i % 1000 == 0:
    print(i)
    np.save('/content/drive/MyDrive/Colab Notebooks/xtzx/hw6/data/bert_clean.npy', all_bert)

print("done in ", time() - t0, "sec")
```

4.6 label的转化

计算train中的得分。（存在问题：没有把投票数量考虑进去）

```
train_df['score'] = train_df['votes_up'] / train_df['votes_all']
```

```
print("AUC score:", roc_auc_score(train_df['label'], train_df['score'])) #想要最后输出的是这个值 #也可能需要再改进 考虑votes数量
```

```
AUC score: 1.0
```

4.7 tfidf和bert特征降维

利用多层感知机（MLP）对维度较高的tfidf、bert特征进行降维，得到一个值，放入dataframe中。（这个做法可能是不对的，或者不能称之为降维，因为利用到了label）

```
#tfidf + MLPRefressor
clf = MLPRegressor(random_state=1, max_iter=20, verbose = True,
                   early_stopping = True)
clf.fit(tfidf_best_train, train_df['score'])
print("train score:", clf.score(tfidf_best_train, train_df['score']))
train_tfidf_pred = clf.predict(tfidf_best_train)
train_df['tfidf'] = train_tfidf_pred
print("AUC score:", roc_auc_score(train_df['label'], train_tfidf_pred))
```

```
test_pred = clf.predict(tfidf_best_test)
for i in range(len(test_pred)): #把小于零的改成0，大于1的改成1
  if test_pred[i] < 0: test_pred[i] = 0
  elif test_pred[i] > 1: test_pred[i] = 1

test_df['tfidf'] = test_pred
```

```
from sklearn.neural_network import MLPRegressor
clf = MLPRegressor(random_state=1, max_iter=20, verbose = True,
                   early_stopping = True, activation = 'logistic')
clf.fit(bert_train, train_df['score'])
print("train score:", clf.score(bert_train, train_df['score']))
```

```
train_bert_pred = clf.predict(bert_train)
train_df['bert'] = train_bert_pred #将这个特征储存起来
print("AUC score:", roc_auc_score(train_df['label'], train_bert_pred))
```

```
AUC score: 0.7216580568583564
```

4.8 生成特征矩阵

对特征数据进行处理，离散化（如果要进行二分类，使用label作为y值），和归一化。

```
from sklearn.preprocessing import Normalizer
#将数值特征离散化，并归一化
discrete_train_df = train_df.copy()
discrete_test_df = test_df.copy()

def discrete_counts(df, df1,  num = 11): #训练集和测试集的标准要一样 df训练集，df1测试集
    discretedf_train = df.copy()
    discretedf_test = df1.copy()
    for c in df.columns[1:]: # 遍历每一列特征
        if c == 'word_count' or c == 'char_count' or c == 'sentence_count' or c == 'avg_word_length' or c == 'avg_sentence_lenght' or c == 'sentiment' or c == 'bert' or c == 'tfidf':
            # 离散化特征
            # discretedf_train[c], bins = pd.qcut(df[c], q = num,labels = False, retbins=True)
            # discretedf_test[c] = pd.cut(df[c], bins = bins,labels = False)

            #归一化
            maxx = discretedf_train[c].max()
            discretedf_train[c] = discretedf_train[c]/ maxx
            discretedf_test[c] = discretedf_test[c]/ maxx


    return discretedf_train, discretedf_test

classes = 20
discrete_train_df, discrete_test_df = discrete_counts(discrete_train_df, discrete_test_df, classes)
discrete_train_df.head()
```

```
from sklearn.model_selection import train_test_split
#将这些特征取出，生成numpy矩阵
X_train = discrete_train_df[feature_names].values
label_train = discrete_train_df['label'].values
score_train = discrete_train_df['score'].values
x_test = discrete_test_df[feature_names].values
x_train, x_val, y_train, y_val, score_train, score_val = train_test_split(X_train, label_train, score_train, test_size= 0.2, random_state=2020)
x_train.shape, x_val.shape, y_train.shape, y_val.shape, score_train.shape, score_val.shape
```

```
((45631, 9), (11408, 9), (45631,), (11408,), (45631,), (11408,))
```

五、集成学习模型建立

5.1 Bagging模型

按照课堂上教的办法建立Bagging模型。

```
#Bagging 方法
class Bagging(object):
            #replicate_number拔靴采样次数，boot_size每一次采样的训练集的大小，ml选用的分类器
    def __init__(self, replicate_number = 10, boot_size = 0.6, ml = 'DecisionTreeRegressor', RANDOM_SEED = 2020):
        self.replicate_number = replicate_number
        self.boot_size = boot_size
        self.ml = ml
        self.RANDOM_SEED = RANDOM_SEED


    def fit(self, X, y):
        self.models = list(range(self.replicate_number))
        for i in range(self.replicate_number):
            #每次用不同的random_seed采样
            ti = time()
            x_train, x_test, y_train, y_test = train_test_split(X,y, test_size= 1 - self.boot_size, random_state=self.RANDOM_SEED + i)
            if self.ml == 'LinearSVR':
                self.models[i] = LinearSVR()
                self.models[i].fit(x_train, y_train)
            elif self.ml == 'DecisionTreeClassifier':
                self.models[i] = DecisionTreeClassifier(max_depth = 10) #为了避免过拟合，将深度设置为10
                self.models[i].fit(x_train, y_train)
            elif self.ml == 'DecisionTreeRegressor':
                self.models[i] = DecisionTreeRegressor(max_depth = 10)
                self.models[i].fit(x_train, y_train)
            elif self.ml == 'KNeighborsClassifier':
                self.models[i] = KNeighborsClassifier(n_neighbors=3)
                self.models[i].fit(x_train, y_train)
            elif self.ml == 'BernoulliNB':
                self.models[i] = BernoulliNB()
                self.models[i].fit(x_train, y_train)

            elif self.ml == 'LogisticRegression':
                self.models[i] = LogisticRegression()
                self.models[i].fit(x_train, y_train)
```

```
            elif self.ml == 'MLPRegressor':
              self.models[i] = MLPRegressor(random_state=1, max_iter=10,early_stopping = True)
              self.models[i].fit(x_train, y_train)

            else:
                self.models[i] = Guess()
                self.models[i].fit(x_train, y_train)

            print("round ", i, " done in %fs" % (time() - ti), "auc:", roc_auc_score(label_train, self.models[i].predict(X)) )

    def predict(self, X): #平权投票，得出最终的predict
        rst = 0
        for i in range(self.replicate_number):
            pred = self.models[i].predict(X)
            rst += pred
        rst = rst / self.replicate_number

        return rst
```

5.2 AdaBoost模型

按照课堂上教的办法建立AdaBoost.M1模型。

```
#AdaBoost方法

class AdaBoost(object):
    #T 重复次数
    def __init__(self, T = 5, ml = 'DecisionTreeClassifier'):
        self.T = T
        self.ml = ml

    def fit(self, X, y):
        self.models = list(range(self.T)) #分类器
        self.errors = list(range(self.T))
        self.betas = np.ones(self.T) * 2
        self.sample_weight = np.ones(X.shape[0]) / X.shape[0]
        for i in range(self.T):
            #根据权重生成模型
            ti = time()
            if self.ml == 'LinearSVR':
                self.models[i] = LinearSVR()
                self.models[i].fit(X, y, self.sample_weight)
            elif self.ml == 'DecisionTreeClassifier':
                self.models[i] = DecisionTreeClassifier(max_depth = 10)
                self.models[i].fit(X, y, self.sample_weight)
            elif self.ml == 'DecisionTreeRegressor':
                self.models[i] = DecisionTreeRegressor(max_depth = 10)
                self.models[i].fit(X, y, self.sample_weight)
            elif self.ml == 'KNeighborsClassifier':
                self.models[i] = KNeighborsClassifier(n_neighbors=3)
                self.models[i].fit(X, y, self.sample_weight)
            elif self.ml == 'BernoulliNB':
                self.models[i] = BernoulliNB()
                self.models[i].fit(X, y, self.sample_weight)

            elif self.ml == 'LogisticRegressionr':
                self.models[i] = LogisticRegression()
                self.models[i].fit(X, y, self.sample_weight)
```

```
        elif self.ml == 'MLPRegressor':
          self.models[i] = MLPRegressor(random_state=1, max_iter=10,early_stopping = True)
          self.models[i].fit(X, y, self.sample_weight)

        else:
            self.models[i] = Guess()
            self.models[i].fit(X, y)
        #用这个模型进行预测
        pred = self.models[i].predict(X)

        #计算所有错误分类样本权重和 因为计算的不是分类标签，而是一个score，这里改为weight * 差值得绝对值
        self.errors[i] = 0
        for j, pre in enumerate(pred):
            if pre != y[j]:
                self.errors[i] = self.errors[i] + self.sample_weight[j] * np.abs(y[j] - pre)

        print("error:", self.errors[i])
        #计算beta
        self.betas[i] = self.errors[i] / (1 - self.errors[i])

        if self.errors[i] > 0.5:
            print('a better C is required')
            self.T = i + 1
            return
        if self.errors[i] == 0: #如果全做对了，beta等于0.99，返回
            self.betas[i] = 0.99
            print('error = 0')
            self.T = i + 1
            return ;
```

```
            #跟新每个样本的权重，并归一化
            for j, pre in enumerate(pred):
                if pre == y[j]:
                    self.sample_weight[j] = self.sample_weight[j] * self.betas[i]
            sumw = sum(self.sample_weight)
            print("sum of sample_weight:", sumw)
            for j, w in enumerate(self.sample_weight):
                self.sample_weight[j] = w / sumw
            print("T = ", i, " done in %fs" % (time() - ti), "auc:", roc_auc_score(label_train, self.models[i].predict(X),) )

    def predict(self, X): #融合所有假设，各自投票权重为1/beta
        self.pollweight = np.log(1 / self.betas)
        rst = 0
        for i in range(self.T):
            pred = self.models[i].predict(X)
            rst += pred * self.pollweight[i]
        rst = rst / sum(self.pollweight)
        return rst
```

六、基本模型训练

6.1 Bagging + 决策树

```
from time import time
#bagging + 决策树

t0 = time()

model = Bagging(ml = 'DecisionTreeRegressor')
model.fit(x_train, score_train)
print("done in %fs" % (time() - t0))

t0 = time()
pred1 = model.predict(x_train)
print("done in %fs" % (time() - t0))
print(pred1)
print("train AUC score:", roc_auc_score(label_train, pred1))
```

```
round  0  done in 0.277052s auc: 0.8002564794442433
round  1  done in 0.279075s auc: 0.8009581332184988
round  2  done in 0.277151s auc: 0.8012954746500454
round  3  done in 0.271189s auc: 0.7999412441477649
round  4  done in 0.274953s auc: 0.8009118485839195
round  5  done in 0.278196s auc: 0.8023993138786727
round  6  done in 0.273980s auc: 0.8006565445310247
round  7  done in 0.226906s auc: 0.8014592421872067
round  8  done in 0.276077s auc: 0.7983603907873128
round  9  done in 0.252148s auc: 0.7999806410164729
done in 2.897827s
done in 0.062802s
[0.89045115 0.44821936 0.41434243 ... 0.76887281 0.71333494 0.72218691]
train AUC score: 0.8211283342746065
```

```
val_pred1 = model.predict(x_val)
print("val AUC score:", roc_auc_score(label_val, val_pred1))
```

```
val AUC score: 0.80252840813192
```

6.2 Bagging + SVM

```
#bagging + svm
t0 = time()
model2 = Bagging(ml = 'LinearSVR')
model2.fit(x_train, score_train)
print("done in %fs" % (time() - t0))
t0 = time()
pred2 = model2.predict(x_train)
print("done in %fs" % (time() - t0))
print(pred2)
print("train AUC score:", roc_auc_score(label_train, pred2))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
round  0  done in 0.825485s auc: 0.7890381165013343
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
round  1  done in 0.946265s auc: 0.7885303090363073
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
round  2  done in 1.036679s auc: 0.7885822319834108
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
round  3  done in 1.017801s auc: 0.7887067878068604
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
round  4  done in 1.044463s auc: 0.7888805262834648
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
round  5  done in 1.050794s auc: 0.7887938627729367
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
round  6  done in 1.026055s auc: 0.7890804580202466
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
round  7  done in 1.068044s auc: 0.7889254215698109
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
round  8  done in 1.024630s auc: 0.7885789869706565
round  9  done in 1.066048s auc: 0.7891609222671871
done in 10.336176s
done in 0.005111s
[0.9238687  0.53565953 0.45639918 ... 0.82182427 0.59331942 0.75159482]
train AUC score: 0.7888734465050047
```

```
val_pred2 = model2.predict(x_val)
print("val AUC score:", roc_auc_score(label_val, val_pred2))
```

```
val AUC score: 0.7980583205340266
```

6.3 AdaBoost + 决策树

```
#AdaBoost.M1 + 决策树
t0 = time()
model3 = AdaBoost(ml = 'DecisionTreeRegressor')
model3.fit(x_train, score_train)
print("done in %fs" % (time() - t0))

t0 = time()
pred3 = model3.predict(x_train)
print("done in %fs" % (time() - t0))
print(pred3)
print("train AUC score:", roc_auc_score(label_train, pred3))
```

```
error: 0.16978933620928388
sum of sample_weight: 0.9956417433171747
T =  0  done in 0.679250s auc: 0.8122792942154864
error: 0.16788368272359966
sum of sample_weight: 0.9964688403919439
T =  1  done in 0.670001s auc: 0.8168516145639135
error: 0.1672238864377843
sum of sample_weight: 0.9967645513066196
T =  2  done in 0.597006s auc: 0.8170629916225182
error: 0.16641415250457386
sum of sample_weight: 0.9974203423461734
T =  3  done in 0.670431s auc: 0.8160878515747076
error: 0.1661423130865551
sum of sample_weight: 0.9974054356514069
T =  4  done in 0.669971s auc: 0.8157966013936329
done in 3.389236s
done in 0.024664s
[0.90080117 0.41095426 0.51437705 ... 0.78373738 0.83261677 0.72830441]
train AUC score: 0.8248114991840974
```

```
val_pred3 = model3.predict(x_val)
print("val AUC score:", roc_auc_score(label_val, val_pred3))
```

```
val AUC score: 0.7962092558328271
```

6.4 AdaBoost + SVM

```
#AdaBoost.M1 + svm
t0 = time()
model4 = AdaBoost(ml = 'LinearSVR')
model4.fit(x_train, score_train)
print("done in %fs" % (time() - t0))
t0 = time()
pred4 = model4.predict(x_train)
print("done in %fs" % (time() - t0))
print(pred4)
print("train AUC score:", roc_auc_score(label_train, pred4))
```

```
error: 0.20143816397725067
sum of sample_weight: 0.9999999999992506
T =  0  done in 0.274680s auc: 0.7638072289064091
error: 0.2014112946288019
sum of sample_weight: 0.9999999999999749
T =  1  done in 0.292682s auc: 0.7638073303987775
error: 0.20140618846644015
sum of sample_weight: 0.9999836120408779
T =  2  done in 0.291294s auc: 0.7638067625901214
error: 0.20141628824260868
sum of sample_weight: 0.9999836121196284
T =  3  done in 0.289151s auc: 0.7638078543188417
error: 0.20141468370061155
sum of sample_weight: 0.9999999999995853
T =  4  done in 0.279967s auc: 0.7638096948966592
done in 1.544767s
done in 0.004763s
[0.85362041 0.43264735 0.29619046 ... 0.71375637 0.56606314 0.8351714 ]
train AUC score: 0.763807821402398
```

```
val_pred4 = model4.predict(x_val)
print("val AUC score:", roc_auc_score(label_val, val_pred4))
```

```
val AUC score: 0.7691397547174588
```

6.5 实验结果对比

AUC 值

	Bagging		AdaBoost.M1	
	train	val	train	val
决策树	0.8211	0.8025	0.8248	0.7965
SVM	0.7889	0.7981	0.7638	0.7691

经过对比可以发现，决策树集成学习后在train集上的AUC值比使用一个分类器更高，说明集成学习+决策树在训练集上是有效果的。
但是由于决策树本身存在过拟合可能，在测试集上的AUC值略低于训练集上的AUC值。
AdaBoost + 决策树，由于样本权重的更新，使得最终的模型在训练集上更加过拟合了，因此，训练集AUC更高，但验证集上的AUC值却下降了。
但是SVM分类器+集成学习的办法，相比较单个的SVM并没有很大的提升。而且训练集和验证集上的AUC值相差不大，验证集上的AUC甚至超过了训练集上的AUC值。这说明SVM分类器可能更简单，过拟合可能性较小。

七、其他基分类器

7.1 MLP

```
model5 = Bagging(ml = 'MLPRegressor', boot_size = 0.4, replicate_number = 10)
model5.fit(x_train, score_train)
print("done in %fs" % (time() - t0))
t0 = time()
pred5 = model5.predict(x_train)
print("done in %fs" % (time() - t0))
print(pred5)
print("train AUC score:", roc_auc_score(label_train, pred5))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
round  0  done in 0.682487s auc: 0.7939935581970818
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
round  1  done in 0.716042s auc: 0.7937061153513434
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
round  2  done in 0.706617s auc: 0.7941036774168755
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
round  3  done in 0.701932s auc: 0.7942292646222318
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
round  4  done in 0.669422s auc: 0.7933842748218034
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
round  5  done in 0.707352s auc: 0.7942546322282762
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
round  6  done in 0.655141s auc: 0.7928608457611424
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
round  7  done in 0.690790s auc: 0.7949478936808738
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
round  8  done in 0.647835s auc: 0.7951096121694068
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
round  9  done in 0.709762s auc: 0.794896009136288
done in 7.422439s
done in 0.312136s
[0.9030571  0.48427928 0.45903362 ... 0.72674792 0.59210588 0.70962133]
train AUC score: 0.7944097701710802
```

```
val_pred5 = model5.predict(x_val)
print("val AUC score:", roc_auc_score(label_val, val_pred5))
```

```
val AUC score: 0.8027592088199884
```

7.2 Logistic Regression

```
#Bagging + LogisticRegression
t0 = time()
model6 = Bagging(ml = 'LogisticRegression')
model6.fit(x_train, label_train)
print("done in %fs" % (time() - t0))
t0 = time()
pred6 = model6.predict(x_train)
print("done in %fs" % (time() - t0))
print(pred6)
print("train AUC score:", roc_auc_score(label_train, pred6))
```

```
round  0  done in 0.339356s auc: 0.5950741968188835
round  1  done in 0.244109s auc: 0.5992327725550811
round  2  done in 0.234867s auc: 0.5954122020653807
round  3  done in 0.236408s auc: 0.5966501181988352
round  4  done in 0.376844s auc: 0.5990103588871643
round  5  done in 0.326322s auc: 0.5997960263982198
round  6  done in 0.170562s auc: 0.5973883092257413
round  7  done in 0.195942s auc: 0.5928592205117283
round  8  done in 0.125641s auc: 0.5947882544143763
round  9  done in 0.265274s auc: 0.6008488876559853
done in 2.660556s
done in 0.009088s
[1. 0. 0. ... 0. 0. 0.]
train AUC score: 0.6022289534521161
```

```
val_pred6 = model6.predict(x_val)
print("val AUC score:", roc_auc_score(label_val, val_pred6))
```

```
val AUC score: 0.6042972310246759
```

由于是分类器，只能使用label作为y值，最终预测值是1或0，因此，AUC值会低一些。

7.3 BernoulliNB

```
#Bagging + BernoulliNB
t0 = time()
model7 = Bagging(ml = 'BernoulliNB')
model7.fit(x_train, label_train)
print("done in %fs" % (time() - t0))
t0 = time()
pred7 = model7.predict(x_train)
print("done in %fs" % (time() - t0))
print(pred7)
print("train AUC score:", roc_auc_score(label_train, pred7))
```

```
round  0  done in 0.016203s auc: 0.5
round  1  done in 0.015636s auc: 0.5
round  2  done in 0.019522s auc: 0.5
round  3  done in 0.016945s auc: 0.5
round  4  done in 0.015546s auc: 0.5
round  5  done in 0.022813s auc: 0.5
round  6  done in 0.015937s auc: 0.5
round  7  done in 0.014306s auc: 0.5
round  8  done in 0.015578s auc: 0.5
round  9  done in 0.015213s auc: 0.5
done in 0.343661s
done in 0.047335s
[0. 0. 0. ... 0. 0. 0.]
train AUC score: 0.5
```

```
val_pred7 = model7.predict(x_val)
print("val AUC score:", roc_auc_score(label_val, val_pred7))
```

```
val AUC score: 0.4999433876811594
```

伯努利朴素贝叶斯不是这样用的，笑哭.jpg
正确打开方式：传入tfidf的特征值作为X:

```
#Bagging + BernoulliNB

t0 = time()
model7 = Bagging(ml = 'BernoulliNB')
model7.fit(tfidf_best_train, all_label_train)
print("done in %fs" % (time() - t0))
t0 = time()
pred7 = model7.predict(tfidf_best_train)
print("done in %fs" % (time() - t0))
print(pred7)
print("train AUC score:", roc_auc_score(all_label_train, pred7))
```

```
round  0  done in 0.020576s auc: 0.6039107328151292
round  1  done in 0.012975s auc: 0.6032399869001898
round  2  done in 0.012838s auc: 0.6036847704997615
round  3  done in 0.014703s auc: 0.6038749328318981
round  4  done in 0.013291s auc: 0.6038600458057857
round  5  done in 0.013149s auc: 0.604496201665066
round  6  done in 0.013333s auc: 0.6035285695526444
round  7  done in 0.013375s auc: 0.6025400718968115
round  8  done in 0.014893s auc: 0.604573685321412
round  9  done in 0.013690s auc: 0.6046589623864144
done in 0.306085s
done in 0.037997s
[0. 0. 0. ... 0. 1. 0.]
train AUC score: 0.6065152617155697
```

7.4 ComplementNB

```
#Bagging + ComplementNB
t0 = time()
model7 = Bagging(ml = 'ComplementNB')
model7.fit(tfidf_best_train, all_label_train)
print("done in %fs" % (time() - t0))
t0 = time()
pred7 = model7.predict(tfidf_best_train)
print("done in %fs" % (time() - t0))
print(pred7)
print("train AUC score:", roc_auc_score(all_label_train, pred7))
```

```
round  0  done in 0.017011s auc: 0.6316575645635879
round  1  done in 0.012526s auc: 0.6266465485505986
round  2  done in 0.012769s auc: 0.633800250588117
round  3  done in 0.012482s auc: 0.630818471012104
round  4  done in 0.012894s auc: 0.6306086056504208
round  5  done in 0.016086s auc: 0.6372227335507651
round  6  done in 0.012757s auc: 0.6237250519915009
round  7  done in 0.012510s auc: 0.6267227397444001
round  8  done in 0.016992s auc: 0.6313332293416098
round  9  done in 0.013049s auc: 0.6343956245127561
done in 0.283554s
done in 0.022007s
[0.  0.  0.  ... 0.  1.  0.6]
train AUC score: 0.6496830128490408
```

7.5 小结

基分类器	Train AUC score
MLP	0.7944
Logistic Regression	0.6022
BernoulliNB	0.6051
ComplementNB	0.6497

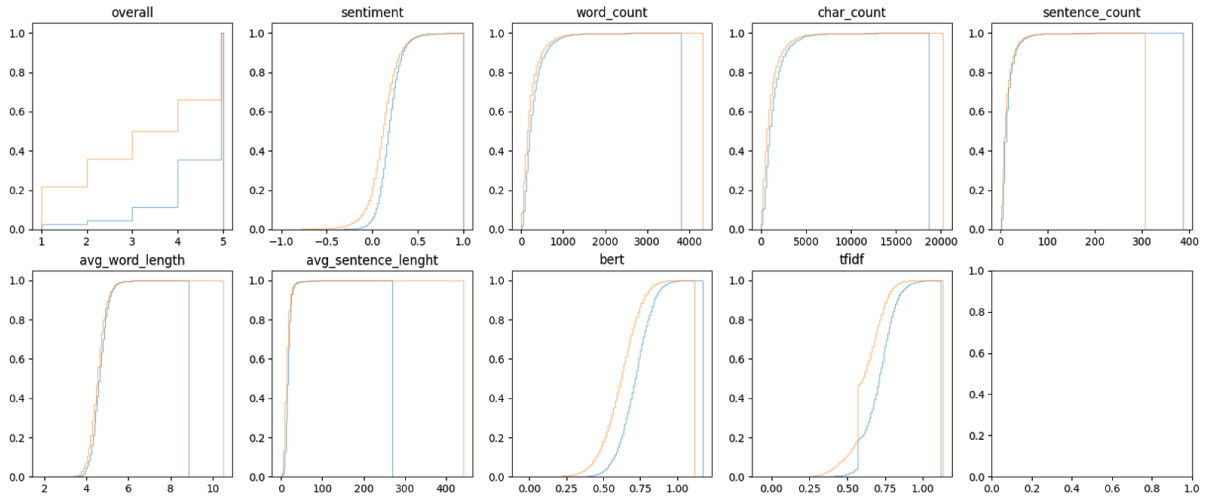
因为使用的X（选取的特征），和target value (label or score) 有不同，这些AUC值并不能用来做对比。
KNN分类器速度太慢，这里没有进行测试。

八、不同特征的影响

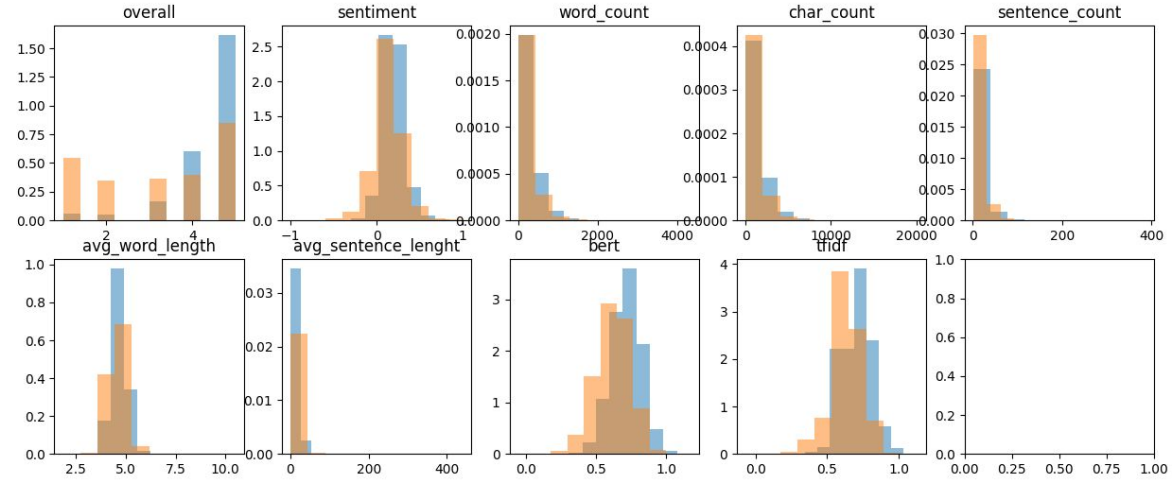
8.0 图说特征

将各个特征的累计柱状图画出来看看。（蓝线代表高质量评论）

```
import matplotlib.pyplot as plt
figure, axes = plt.subplots(2, 5, figsize = (20, 8), dpi = 100)
ax = axes.flatten()
for i, c in enumerate(feature_names):
    ax[i].hist(train_df[c][train_df['label'] == 1], bins = 100, alpha = 0.5, histtype='step', cumulative = True,density=True, label = '1')
    ax[i].hist(train_df[c][train_df['label'] == 0], bins= 100, alpha = 0.5, histtype='step', cumulative = True,density=True, label = '0')
    ax[i].set_title(c, fontsize=15)
    ax[i].set_title(c)

#plt.savefig("features.png") #保存作的图
plt.show()
```




```
import matplotlib.pyplot as plt
figure, axes = plt.subplots(2, 5, figsize = (15, 6), dpi = 100)
ax = axes.flatten()
for i, c in enumerate(feature_names):
    ax[i].hist(train_df[c][train_df['label'] == 1], bins = 10, alpha = 0.5, cumulative = False,density=True, label = '1')
    ax[i].hist(train_df[c][train_df['label'] == 0], bins= 10, alpha = 0.5, cumulative = False,density=True, label = '0')
    ax[i].set_title(c, fontsize=15)
    ax[i].set_title(c)

#plt.savefig("features.png") #保存作的图
plt.show()
```



从图中可以发现， overall, sentiment, bert, tfidf对文本最终的高质量与否贡献更大。

8.1 情绪特征

```
#仅情绪特征

feature_names = ['sentiment']
X_train = discrete_train_df[feature_names].values
all_label_train = discrete_train_df['label'].values
all_score_train = discrete_train_df['score'].values
x_test = discrete_test_df[feature_names].values
x_train, x_val, label_train, label_val, score_train, score_val = train_test_spli
            all_label_train, all_score_train, test_size= 0.2, random_state=2020)
model = Bagging(ml = 'MLPRegressor')
model.fit(x_train, score_train)
pred = model.predict(x_train)
print(pred1)
print("train AUC score:", roc_auc_score(label_train, pred))
val_pred = model.predict(x_val)
print("val AUC score:", roc_auc_score(label_val, val_pred))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_percep
  % self.max_iter, ConvergenceWarning)
round  0  done in 0.903374s auc: 0.6465790858845704
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_percep
  % self.max_iter, ConvergenceWarning)
round  1  done in 0.910787s auc: 0.6460108932033387
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_percep
  % self.max_iter, ConvergenceWarning)
round  2  done in 0.924314s auc: 0.6469728515869141
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_percep
  % self.max_iter, ConvergenceWarning)
round  3  done in 0.807997s auc: 0.6463869580880133
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_percep
  % self.max_iter, ConvergenceWarning)
round  4  done in 0.886816s auc: 0.645987311314372
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_percep
  % self.max_iter, ConvergenceWarning)
round  5  done in 0.922318s auc: 0.6468477608712522
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_percep
  % self.max_iter, ConvergenceWarning)
round  6  done in 0.936371s auc: 0.6458152611770323
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_percep
  % self.max_iter, ConvergenceWarning)
round  7  done in 0.913854s auc: 0.6465416338290851
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_percep
  % self.max_iter, ConvergenceWarning)
round  8  done in 0.850847s auc: 0.6465006377698144
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_percep
  % self.max_iter, ConvergenceWarning)
round  9  done in 0.907583s auc: 0.6464482704507192
[0.89045115 0.44821936 0.41434243 ... 0.76887281 0.71333494 0.72218691]
train AUC score: 0.6464245760972361
val AUC score: 0.647584447294986
```

8.2 计数特征

```
#仅计数特征
feature_names = ['word_count',  'char_count', 'sentence_count', 'avg_word_length',  'avg_sentence_lenght']
X_train = discrete_train_df[feature_names].values
all_label_train = discrete_train_df['label'].values
all_score_train = discrete_train_df['score'].values
x_test = discrete_test_df[feature_names].values
x_train, x_val, label_train, label_val, score_train, score_val = train_test_split(X_train,
            all_label_train, all_score_train, test_size= 0.2, random_state=2020)
model = Bagging(ml = 'MLPRegressor')
model.fit(x_train, score_train)
pred = model.predict(x_train)
print(pred1)
print("train AUC score:", roc_auc_score(label_train, pred))
val_pred = model.predict(x_val)
print("val AUC score:", roc_auc_score(label_val, val_pred))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarn
  % self.max_iter, ConvergenceWarning)
round  0  done in 0.936319s auc: 0.618845004228803
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarn
  % self.max_iter, ConvergenceWarning)
round  1  done in 0.939298s auc: 0.6188589361136536
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarn
  % self.max_iter, ConvergenceWarning)
round  2  done in 0.936372s auc: 0.6201351944180953
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarn
  % self.max_iter, ConvergenceWarning)
round  3  done in 0.707695s auc: 0.6200209222403053
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarn
  % self.max_iter, ConvergenceWarning)
round  4  done in 0.955296s auc: 0.6208322193174375
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarn
  % self.max_iter, ConvergenceWarning)
round  5  done in 0.938718s auc: 0.6186279916007111
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarn
  % self.max_iter, ConvergenceWarning)
round  6  done in 0.835160s auc: 0.620529953099828
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarn
  % self.max_iter, ConvergenceWarning)
round  7  done in 0.839651s auc: 0.6195499318177011
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarn
  % self.max_iter, ConvergenceWarning)
round  8  done in 0.949557s auc: 0.6194263744597143
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarn
  % self.max_iter, ConvergenceWarning)
round  9  done in 0.942991s auc: 0.6212309005420598
[0.89045115 0.44821936 0.41434243 ... 0.76887281 0.71333494 0.72218691]
train AUC score: 0.6199902111982125
val AUC score: 0.6282361983737847
```

8.3 bert

```
#bert + MLP
x_train, x_val, label_train, label_val, score_train, score_val = train_test_split(bert_train, #bert
            all_label_train, all_score_train, test_size= 0.2, random_state=2020)
model = Bagging(ml = 'MLPRegressor')
model.fit(x_train, score_train)
print("done in %fs" % (time() - t0))

t0 = time()
pred = model.predict(x_train)
print("done in %fs" % (time() - t0))
print(pred1)
print("train AUC score:", roc_auc_score(label_train, pred))
val_pred = model.predict(x_val)
print("val AUC score:", roc_auc_score(label_val, val_pred))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Converge
  % self.max_iter, ConvergenceWarning)
round  0  done in 6.738817s auc: 0.7104639399946395
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Converge
  % self.max_iter, ConvergenceWarning)
round  1  done in 6.843150s auc: 0.701787864875694
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Converge
  % self.max_iter, ConvergenceWarning)
round  2  done in 6.926196s auc: 0.7100605846349936
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Converge
  % self.max_iter, ConvergenceWarning)
round  3  done in 6.916078s auc: 0.7134310117831817
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Converge
  % self.max_iter, ConvergenceWarning)
round  4  done in 6.506691s auc: 0.6973536687831665
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Converge
  % self.max_iter, ConvergenceWarning)
round  5  done in 6.885368s auc: 0.7174002603635847
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Converge
  % self.max_iter, ConvergenceWarning)
round  6  done in 6.939979s auc: 0.7153492175473283
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Converge
  % self.max_iter, ConvergenceWarning)
round  7  done in 7.166987s auc: 0.7094424741666948
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Converge
  % self.max_iter, ConvergenceWarning)
round  8  done in 6.841785s auc: 0.7142524388959024
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Converge
  % self.max_iter, ConvergenceWarning)
round  9  done in 7.411280s auc: 0.7168434019111781
done in 261.005581s
done in 3.302174s
[0.89045115 0.44821936 0.41434243 ... 0.76887281 0.71333494 0.72218691]
train AUC score: 0.7201629978409829
val AUC score: 0.7076707538299464
```

8.4 tfidf

```
#tfidf + MLP
x_train, x_val, label_train, label_val, score_train, score_val = train_test_split(tfidf_best_train,
            all_label_train, all_score_train, test_size= 0.2, random_state=2020)
model = Bagging(ml = 'MLPRegressor')
model.fit(x_train, score_train)
print("done in %fs" % (time() - t0))

t0 = time()
pred = model.predict(x_train)
print("done in %fs" % (time() - t0))
print(pred1)
print("train AUC score:", roc_auc_score(label_train, pred))
val_pred = model.predict(x_val)
print("val AUC score:", roc_auc_score(label_val, val_pred))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Convergenc
  % self.max_iter, ConvergenceWarning)
round  0  done in 1.606556s auc: 0.6961753876453011
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Convergenc
  % self.max_iter, ConvergenceWarning)
round  1  done in 1.683364s auc: 0.6965729922279064
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Convergenc
  % self.max_iter, ConvergenceWarning)
round  2  done in 1.628068s auc: 0.6959557046707034
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Convergenc
  % self.max_iter, ConvergenceWarning)
round  3  done in 1.677003s auc: 0.6964874835359491
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Convergenc
  % self.max_iter, ConvergenceWarning)
round  4  done in 1.591139s auc: 0.6968633618941087
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Convergenc
  % self.max_iter, ConvergenceWarning)
round  5  done in 1.699989s auc: 0.6966545084295309
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Convergenc
  % self.max_iter, ConvergenceWarning)
round  6  done in 1.529948s auc: 0.6965802941923624
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Convergenc
  % self.max_iter, ConvergenceWarning)
round  7  done in 1.653477s auc: 0.6972010914653888
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Convergenc
  % self.max_iter, ConvergenceWarning)
round  8  done in 1.562695s auc: 0.6965075653097219
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: Convergenc
  % self.max_iter, ConvergenceWarning)
round  9  done in 1.649590s auc: 0.6963237269709358
done in 124.396138s
done in 0.323533s
[0.89045115 0.44821936 0.41434243 ... 0.76887281 0.71333494 0.72218691]
train AUC score: 0.6968426944819395
val AUC score: 0.7074934667274282
```

8.5 特征小结

上面的几次测试佐证了bert > tfidf > sentimetn > counts对最终得分的影响。

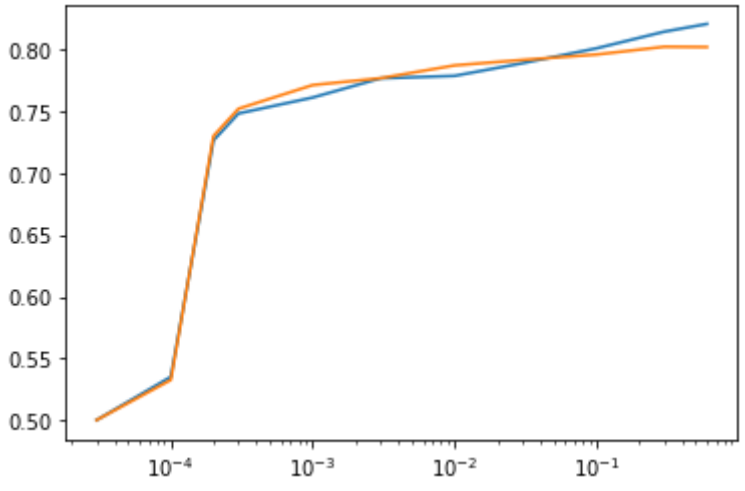
九、集成学习算法参数的影响

9.1 Bagging

9.1.1 Boot size

```
#Bagging boot size对结果的影响 原始 replicate_number = 10, boot_size = 0.6

feature_names = ['overall', 'sentiment',  'word_count', 'char_count', 'sentence_count', 'avg_word_ler

X_train = discrete_train_df[feature_names].values
all_label_train = discrete_train_df['label'].values
all_score_train = discrete_train_df['score'].values
x_test = discrete_test_df[feature_names].values
x_train, x_val, label_train, label_val, score_train, score_val = train_test_split(X_train,
            all_label_train, all_score_train, test_size= 0.2, random_state=2020)

boot_sizes = [0.00003, 0.0001, 0.0002, 0.0003, 0.001, 0.003, 0.01, 0.03,  0.1, 0.3, 0.6]
train_auc = []
val_auc = []

for size in boot_sizes:
  model = Bagging(ml = 'DecisionTreeRegressor', boot_size = size)
  model.fit(x_train, score_train)
  pred = model.predict(x_train)
  print(pred1)
  print("boot size = ", size)
  tr_auc = roc_auc_score(label_train, pred)
  train_auc.append(tr_auc)
  print("train AUC score:",tr_auc)
  val_pred = model.predict(x_val)
  v_auc = roc_auc_score(label_val, val_pred)
  val_auc.append(v_auc)
  print("val AUC score:", v_auc)
```

```
plt.plot(boot_sizes, train_auc)
plt.plot(boot_sizes, val_auc)
plt.xscale('log')
```



Fix replicate_number = 10

极端情况：boot_size = 0.0003（单词使用样本数量为1个），使用决策树模型建模，预测值也只有一个，就是样本本身的score，所以最后输出的值都是一样的，AUC为0.5。

Boot_size非常小时，从boot_size = 0.001(单词样板数量约为5个）到boot_size = 0.003（单次样本数量约为16个），每一次拔靴采样的auc值浮动大，但最终的情况优于平均每次的效果，Bagging方法收益较大。且在这个区间上，验证集上的auc值从0.53提升到了0.75，升幅巨大。

```
round  9  done in 0.007005s auc: 0.5
[0.7361 0.7361 0.7361 ... 0.7361 0.7361 0.7361]
boot size =  3e-05
train AUC score: 0.5
val AUC score: 0.5
round  0  done in 0.006003s auc: 0.4505347674040416
round  1  done in 0.006211s auc: 0.5946441983409948
round  2  done in 0.006741s auc: 0.4995663423107722
round  3  done in 0.006579s auc: 0.4975484477249704
round  4  done in 0.006438s auc: 0.46130479740984887
round  5  done in 0.006447s auc: 0.46357775603280926
round  6  done in 0.006806s auc: 0.47108846423201833
round  7  done in 0.008235s auc: 0.46354258892713324
round  8  done in 0.007120s auc: 0.5660511291670606
round  9  done in 0.008492s auc: 0.6123339958590565
[0.57630592 0.66090736 0.5720243  ... 0.60639768 0.6320202  0.68127615]
boot size =  0.0001
train AUC score: 0.5350506157528281
val AUC score: 0.5325041738399046
round  0  done in 0.006408s auc: 0.6658177639239781
round  1  done in 0.006284s auc: 0.6775394234075909
round  2  done in 0.007267s auc: 0.6501082566976802
round  3  done in 0.006717s auc: 0.41857391262653665
round  4  done in 0.006609s auc: 0.6697229516186153
round  5  done in 0.007166s auc: 0.5017918367932493
round  6  done in 0.010046s auc: 0.6495081199792307
round  7  done in 0.006295s auc: 0.42603317379555916
round  8  done in 0.007090s auc: 0.604734041579997
round  9  done in 0.006908s auc: 0.618446830466023
[0.75634202 0.62658196 0.44560275 ... 0.7097233  0.57902979 0.71271721]
boot size =  0.0002
train AUC score: 0.7267496934587592
val AUC score: 0.730297924086045
round  0  done in 0.006661s auc: 0.6167901348759515
round  1  done in 0.009878s auc: 0.694788958003363
round  2  done in 0.011167s auc: 0.547927222730187
round  3  done in 0.007063s auc: 0.4546752240094296
round  4  done in 0.006656s auc: 0.7295335975810583
round  5  done in 0.006165s auc: 0.6046880038187463
round  6  done in 0.006567s auc: 0.6434422242211647
round  7  done in 0.006110s auc: 0.580630920725326
round  8  done in 0.006646s auc: 0.6326461147912152
round  9  done in 0.007237s auc: 0.6265186063078932
[0.8390873  0.5935348  0.52749944 ... 0.73430403 0.64504274 0.76642153]
boot size =  0.0003
train AUC score: 0.7485373591899681
val AUC score: 0.7523961559532248
```

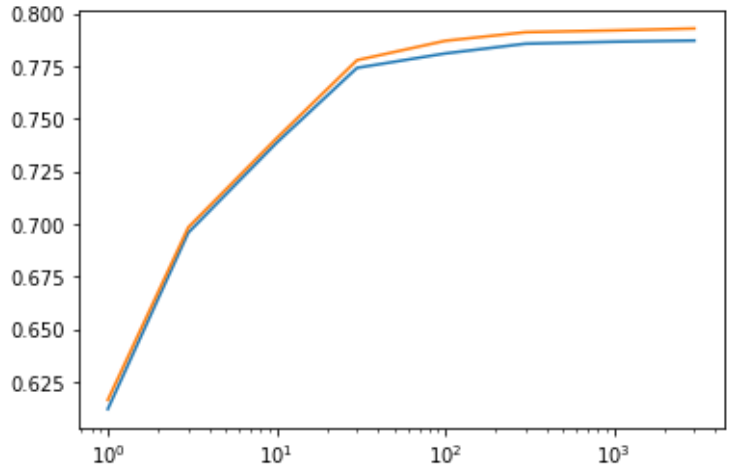
当boot_size继续以指数级增大，训练集和验证集上的auc都略有提升，直到其可能在训练集上过拟合（训练集auc值大于验证集）。

9.2.2 Replicate number

```
#Bagging replicate_number对结果的影响 原始 replicate_number = 10, boot_size = 0.6

boot_size = 0.0003 #将boot_size设置为train集auc浮动较大,但集成效果不错的值
replicate_numbers = [1, 3, 10, 30, 100, 300, 1000, 3000]
train_auc = []
val_auc = []

for number in replicate_numbers:
  model = Bagging(ml = 'DecisionTreeRegressor', boot_size = boot_size, replicate_number = number)
  model.fit(x_train, score_train)
  pred = model.predict(x_train)
  print(pred)
  print("replicate number = ", number)
  tr_auc = roc_auc_score(label_train, pred)
  train_auc.append(tr_auc)
  print("train AUC score:",tr_auc)
  val_pred = model.predict(x_val)
  v_auc = roc_auc_score(label_val, val_pred)
  val_auc.append(v_auc)
  print("val AUC score:", v_auc)
```

```
round  0  done in 0.011224s auc: 0.5885120609827594
[0.86363636 0.76923077 0.33333333 ... 0.90909091 0.94736842 0.86666667]
replicate number =  1
train AUC score: 0.5885120609827594
val AUC score: 0.5918196649746263
round  0  done in 0.006504s auc: 0.5885536193646121
round  1  done in 0.007427s auc: 0.6648674373887407
round  2  done in 0.007477s auc: 0.5573387938422001
[0.87712418 0.6547619  0.24338624 ... 0.65705931 0.5443609  0.84343434]
replicate number =  3
train AUC score: 0.6957916943747691
val AUC score: 0.7011973021944482
```

```
plt.plot(replicate_numbers, train_auc)
plt.plot(replicate_numbers, val_auc)
plt.xscale('log')
```



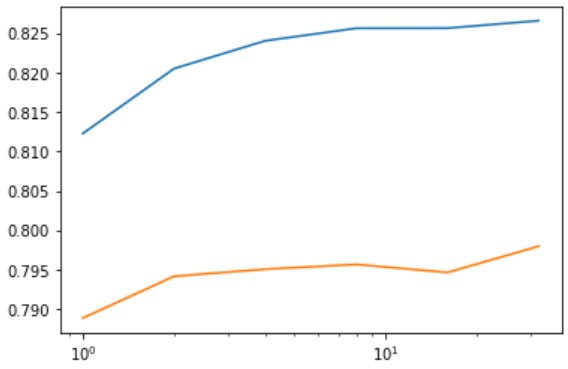
随着replicate_number的指数级别增加，auc值先呈快速上升趋势，当replicate number达到30及以上时，auc值升幅放缓，停留在约0.79的高度。
增加replicate_number，前期收益巨大，后期收益较小，且可能在训练集上过拟合。

9.2 AdaBoost.M1

9.2.1 T

```
#AdaBoost.M1, T对结果的影响

Ts = [1, 2, 4, 8, 16, 32]
train_auc = []
val_auc = []

for T in Ts:
  model = AdaBoost(ml = 'DecisionTreeRegressor', T = T)
  model.fit(x_train, score_train)
  pred = model.predict(x_train)
  print(pred)
  print("T = ", T)
  tr_auc = roc_auc_score(label_train, pred)
  train_auc.append(tr_auc)
  print("train AUC score:",tr_auc)
  val_pred = model.predict(x_val)
  v_auc = roc_auc_score(label_val, val_pred)
  val_auc.append(v_auc)
  print("val AUC score:", v_auc)
```

```
error: 0.16978933620928294
sum of sample_weight: 0.9956417433171747
T =  0  done in 0.524841s auc: 0.8122792928439679
[0.89177056 0.41095426 0.51688611 ... 0.78754345 0.81751665 0.73307843]
T =  1
train AUC score: 0.8122792928439679
val AUC score: 0.7888821141641912
error: 0.16978933620928321
sum of sample_weight: 0.9956417433171747
T =  0  done in 0.530471s auc: 0.8122793147882638
error: 0.16789039657681
sum of sample_weight: 0.996528779346918
T =  1  done in 0.517797s auc: 0.8170136443871443
[0.90009114 0.41095426 0.51494597 ... 0.78677503 0.82697008 0.73119023]
T =  2
train AUC score: 0.8205067524930023
val AUC score: 0.7941737616670606
```



当T值从1增加到2时，auc值增幅明显，但之后就几乎维持不变，且在训练集上的auc值一直高于在验证集上的auc值，说明AdaBoost.M1算法过拟合了训练集。T大于2后增加T，收益不大。（但这里使用的是较深，较复杂的决策树，可能有失偏妥，（基分类期本身容易过拟合）应使用更多基分类器进行比较）

十、总结

通过这次案例，学习和实践了Bagging和AdaBoost.M1两种集成学习的方法，使用了多种不同的基分类器，对机器学习的各种算法又有了更深入的了解。在特征提取方面，尝试调用各种库对各方面的特征进行提取。
但仍然有很多不足方面，比方说放弃了使用pytorch或者keras或者TensorFlow搭建自己的神经网络或者bert模型，希望将来可以有机会再进行练习。