南开大学

Nankai University

南　开　大　学

网络空间安全学院

Python 大作业实验报告

Fake news detection

姓名：袁铭泽

学号：2012777

专业：信息安全

目录

Fake news detection

(一) 项目说明 1

1.问题描述 1

2.数据处理 1

(1)数据读取 1

(2)文本断词 2

(3)数据合并和删除 2

(4)过采样处理真假数据不平衡 3

(二) 方法选择 5

1.传统方法 5

(1)特征构建 5

(2)分类器 a. LogisticRegression 逻辑回归 6

b. DecisionTree 决策树 7

c. GradientBoosting 梯度提升 7

d. RandomForest 随机森林 8

e. RidgeRegression 岭回归 8

(3)ROC 曲线 9

2.深度学习方法 9

(1)处理数据 9

(2)搭建 RNN 11

(3)模型训练 11

(4)loss 和 accuracy 曲线 12

(5)混淆矩阵绘图 13

(三) 指标分析 13

1.是否平衡真假数据的指标对比 13

2.loss 曲线分析拟合效果 14

Fake news detection

（一）项目说明

1. 问题描述

微信上有许多新闻，这些新闻有真有假。通过 Python 建立模型，用给定的 train 集(包含一系列真假新闻)训练模型，再用 test 集检测模型，模型判断新闻真伪，与真实真伪值作对比，得到一系列指标，从而判断模型的优良。本实验将从传统方法和深度学习方法两方面进行实现。

2. 数据处理

（1） 数据读取

先导出各种包，按照 utf-8 编码的方式来读取数据，且以字符串形式读取。（后面会将 label 转为数字以便后续操作）

查看 train 集的前 5 条新闻。

Fake news detection problem

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import re
import string
import jieba
from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score,roc_curve, auc
```

Read the data

In [2]:
```
df_train = pd.read_csv("train.csv", encoding='utf-8').astype(str)
df_test = pd.read_csv("test.csv", encoding='utf-8').astype(str)
```

In [3]:
```
df_train.head(5)
```

Out[3]:

	Ofiicial Account Name	Title	News Url	Image Url	Report Content	label
0	环球人物	中国反腐风刮到阿根廷，这个美到让人瘫痪的女总统，因为8个本子摊上大事了	http://mp.weixin.qq.com/s?__biz=MTAzNDI4MDc2MQ...	http://mmbiz.qpic.cn/mmbiz_jpg/hpcO6kWnPm6cX3M...	内容不符	0
1	西湖之声	腾讯为《如懿传》道歉？这部3亿大剧上映第一天遭网友狂吐槽：愣是拍成村头恋曲...	http://mp.weixin.qq.com/s?__biz=MTA2Mjk0MTE2MA...	http://mmbiz.qpic.cn/mmbiz_jpg/vQCGoQzHAbaAXRr...	满口胡言	0
2	厦门晚报	顺风车司机奸杀20岁女乘客，落网视频曝光！滴滴道歉...	http://mp.weixin.qq.com/s?__biz=MTA3NzI1Mzg4MQ...	http://mmbiz.qpic.cn/mmbiz_jpg/TxqQX9BtmOMpwDZ...	？	0
3	腾讯娱乐	偶遇鹿晗关晓彤旅行过七夕，小情侣是真滴甜...	http://mp.weixin.qq.com/s?__biz=MTA5NTIzNDE2MQ...	http://mmbiz.qpic.cn/mmbiz_jpg/9Ju9PZ1NxhfkIZ3...	领个屁证，过你妹的七夕，几天前的图在今天拿来博眼球	0
4	腾讯娱乐	赵丽颖和冯绍峰即将公布恋情？网友：曝不曝没区别啊	http://mp.weixin.qq.com/s?__biz=MTA5NTIzNDE2MQ...	http://mmbiz.qpic.cn/mmbiz_jpg/9Ju9PZ1NxhdTkXb...	事件不实。	0

（2） 文本断词

下载中文常用停用词表，用 jieba 分词，删除停用词。
尝试 re 匹配中文字符（即只留下中文字符，其余删除），但经后续检验，模型训练效果不好，故将其注释掉。

In [4]:
```
#中文文本断词
def process(s):
    with open('stop_words.txt', 'r', encoding='utf-8') as f: # 打开停用词表
        stop_words = [line.strip() for line in f.readlines()]
    a = jieba.lcut(s)                                        # 分词
    b = [w for w in a if w not in stop_words]                # 删除停用词及标点符号
    s = " ".join(b)
    return s
```

In [5]:
```
#数据清洗
#def find_chinese (file):
#    pattern = re.compile(r'[^\u4e00-\u9fa5]')
#    chinese_txt = re.sub(pattern,'',file)
#    return chinese_txt
```

（3） 数据合并和删除

认为 News Url 和 Image Url 对判断新闻真假作用不大，删除这两列。将 Ofiicial Account Name 和 Report Content 合并到 Title 列中，将其一并作为特征加入 TF-IDF 计算，再删除 Ofiicial Account Name 和 Report Content 这两列。最后将合并好的特征调用分词函数，一同处理。

In [6]:
```
df_train=df_train.drop(["News Url","Image Url"],axis=1)
df_test=df_test.drop(["News Url","Image Url"],axis=1)
```

In [7]:
```
df_train['Title'] = df_train['Title'] + df_train['Ofiicial Account Name']+df_train['Report Content']
#df_train['Title'] = df_train['Title'] + df_train['Ofiicial Account Name']
df_train=df_train.drop(["Ofiicial Account Name","Report Content"],axis=1)
df_test['Title'] = df_test['Title'] + df_test['Ofiicial Account Name']+df_test['Report Content']
df_test=df_test.drop(["Ofiicial Account Name","Report Content"],axis=1)
```

In [8]:
```
df_train["Title"] = df_train["Title"].apply(lambda x: process(x))
df_test["Title"] = df_test["Title"].apply(lambda x: process(x))
#df_train["Title"] = df_train["Title"].apply(lambda x: find_chinese(x))
#df_test["Title"] = df_test["Title"].apply(lambda x: find_chinese(x))
```

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\dell\AppData\Local\Temp\jieba.cache
Loading model cost 1.686 seconds.
Prefix dict has been built successfully.
```

处理效果如下：

```
In [9]: df_test.shape, df_train.shape
Out[9]: ((10141, 2), (10587, 2))
```

```
In [10]: df_test.head(10)
```

Out[10]:

	Title	label
0	国务院 生 孩子 补助 明年 月 实施 浙江 这档 私家车 第一 广播 国务院 发布 类似 信息	0
1	年轻 帅小伙 没 身亡 真相 惊呆 所有人！太 可惜 杭州 交通 918	0
2	迪丽 热巴 时装周 走 秀气 场 一米 八 病态 妆容 挡不住 高级 感 腾讯 娱乐 泰国人 模特	0
3	李晨 北京 四合院 内景 曝光 妈妈 吃饺子 画面 hin 温馨 腾讯 娱乐 造谣生事	0
4	唾液 测 天赋 饭后 剧烈运动 阑尾炎 月 科学 流言 中招 央视 新闻 唾液 基因 检...	0
5	昆山 龙哥 之死 虎 嗅网 全民 热议 话题 事件 里 美化 犯罪分子 不忍 直视	0
6	粉丝 暴民 堕入 魔 道 虎 嗅网 雪崩 雪花 无辜 ## 平安 重庆 发表 申明 魔 道 ...	0
7	120 天后 范冰冰 终于 现身 8.84 亿 洗白 事 虎 嗅 APP 八个 亿 人民日报 查清	0
8	国务院 生 孩子 补助 明年 月 实施 浙江 这档 浙江 之声 标题 党 没 补助 政策	0
9	爱情 公寓 粉丝 请 放过 动感 101	0

（4） 过采样处理真假数据不平衡

查看 train 集中真假数据的分布

Check out the distribution of fake news compare to real news

```
In [17]: import matplotlib.pyplot as plt
```

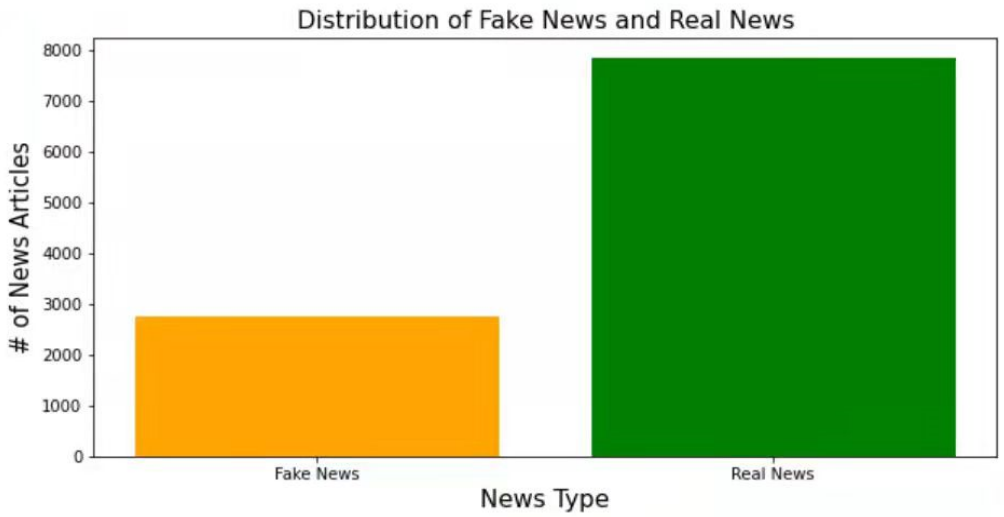
```
In [18]: len1=0#train真新闻个数
         len2=0#train假新闻个数
         for i in range(len(df_train)):
             if df_train["label"][i]=="0":
                 len1+=1
             elif df_train["label"][i]=="1":
                 len2+=1
```

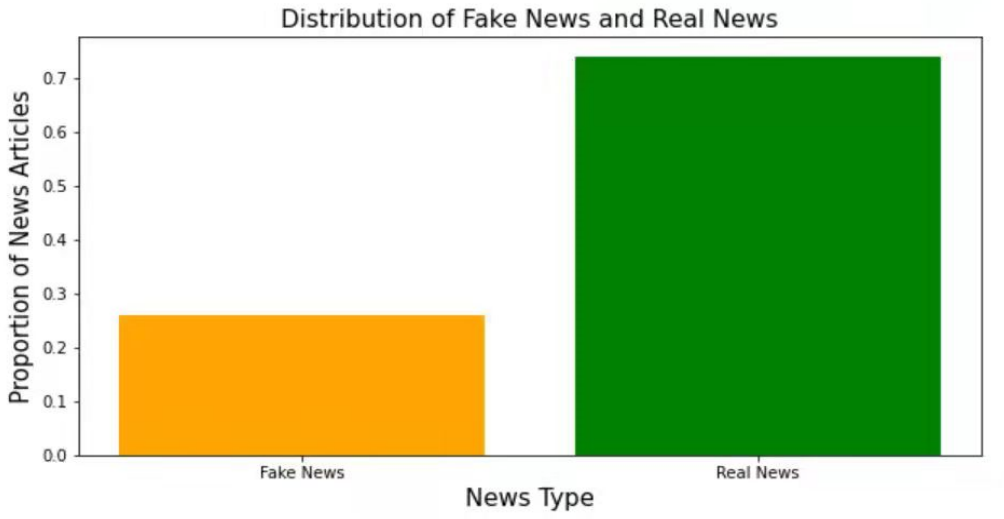
画出柱状图

```
In [19]: plt.figure(figsize=(10, 5))
         plt.bar('Fake News', len2, color='orange')
         plt.bar('Real News', len1, color='green')
         plt.title('Distribution of Fake News and Real News', size=15)
         plt.xlabel('News Type', size=15)
         plt.ylabel('# of News Articles', size=15)

         total_len = len1 + len2
         plt.figure(figsize=(10, 5))
         plt.bar('Fake News', len2 / total_len, color='orange')
         plt.bar('Real News', len1 / total_len, color='green')
         plt.title('Distribution of Fake News and Real News', size=15)
         plt.xlabel('News Type', size=15)
         plt.ylabel('Proportion of News Articles', size=15)
```

图像如下:

```
Out[19]: Text(0, 0.5, 'Proportion of News Articles')
```






```
In [21]: df_train["label"].value_counts()
Out[21]: 0    7844
         1    2743
         Name: label, dtype: int64
```

可以看到真假数据数量不太平衡，处理方法有过采样和欠采样，以及用多个分类器进行分类。由于 fake news 数量较少，这里采取过采样。后续使用多个分类器。

过采样使用的是朴素随机过采样，优点是简单，缺点是过采样后的数据集会反复出现一些样本，可能产生过拟合现象。但在本实验后续的 loss 函数图中可以看到，由于真假数据相差不是很大，几乎不存在过拟合现象。

```
In [23]: x_train=df_train["Title"]
         y_train=df_train["label"]
         x_test=df_test["Title"]
         y_test=df_test["label"]
```

```
In [27]: # 随机采样函数
         import imblearn
         from imblearn.over_sampling import RandomOverSampler
         ros = RandomOverSampler(random_state=0, sampling_strategy=1)
         xv_train, y_train = ros.fit_resample(xv_train, y_train)
```

```
In [28]: y_train.value_counts()
```

```
Out[28]: 0    7844
         1    7844
         Name: label, dtype: int64
```

实现了真假数据平衡。后面数据对比可发现，此步骤对各参数提升有一定作用，但影响不大。

（二）方法选择

1. 传统方法

（1） 特征构建

```
In [25]: from sklearn.feature_extraction.text import TfidfVectorizer
```

```
In [26]: vectorization=TfidfVectorizer()
         xv_train=vectorization.fit_transform(x_train)
         xv_test=vectorization.transform(x_test)
```

TfidfVectorizer 可以把原始文本内容变换为以 TF-IDF 组成的特征矩阵。TF（词频）：一个词在文档中出现的次数就是它的词频。IDF（逆文档频率）：在一个文档中多次出现但在许多其他文档中也出现多次的词可能无关紧要。TF-IDF = TF * IDF

在训练集上调用 fit_transform()，先拟合数据，然后转化它将其转化为标准形式。测试集调用 transform()，为通过训练集我们已经找到了转换规则，可以直接将其运用到测试集上。所以在测试集上的处理，我们只需要标准化数据而不需要再次拟合数据。

（2） 分类器

fit(xv_train,y_train)对测试数据进行回归预测；score :评价模型，逻辑回归模型返回平均准确度，用测试数据检验已建立的模型好不好；predict()预测方法，返回 xv_test 的预测值。

a. LogisticRegression 逻辑回归

LogisticRegression分类器

```
In [32]: from sklearn.linear_model import LogisticRegression

In [33]: LR=LogisticRegression()
         LR.fit(xv_train,y_train)

Out[33]: LogisticRegression()

In [34]: LR.score(xv_test,y_test)

Out[34]: 0.8925155310127206

In [35]: pred_LR=LR.predict(xv_test)

In [36]: print(classification_report(y_test,pred_LR))
```

```
              precision    recall  f1-score   support

           0       0.93      0.95      0.94      8659
           1       0.65      0.58      0.61      1482

    accuracy                           0.89     10141
   macro avg       0.79      0.76      0.77     10141
weighted avg       0.89      0.89      0.89     10141
```

```
In [37]: pred_LR=LR.predict_proba(xv_test)[:,1]
         fp_LR, tp_LR, thresholds = roc_curve(y_test, pred_LR)
         ra_LR = auc(fp_LR, tp_LR)
```

classification_report 的指标分析：

precision：精度=正确预测的个数(TP)/被预测正确的个数(TP+FP)；也就是模型预测的结果中有多少是预测正确的

recall:召回率=正确预测的个数(TP)/预测个数(TP+FN)；也就是某个类别测试集中的总量中有多少样本预测正确了；

f1-score:f1 = 2*精度*召回率/(精度+召回率)

support：当前行的类别在测试数据中的样本总量

macro avg：每个类别评估指标未加权的平均值

weighted avg：加权平均，测试集中样本量大的，认为它更重要，给它设置的权重大

使用 roc_auc_score()的时候，输入为真实标签与预测概率，如果输入为预测标签，得到的结果会变差，所以提前将预测标签用 predict_proba 函数改为预测概率。roc_curve 得到混淆矩阵中的 FP 和 TP 的值，auc 函数计算 auc 值（即 ROC 曲线下的面积）

b. DecisionTree 决策树

DecisionTree分类器

In [44]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [45]:
```
DT=DecisionTreeClassifier()
DT.fit(xv_train,y_train)
```

Out[45]:
```
DecisionTreeClassifier()
```

In [46]:
```
DT.score(xv_test,y_test)
```

Out[46]:
```
0.8810768168819643
```

In [47]:
```
pred_DT=DT.predict(xv_test)
```

In [48]:
```
print(classification_report(y_test,pred_DT))
```

```
              precision    recall  f1-score   support

           0       0.94      0.92      0.93      8659
           1       0.59      0.63      0.61      1482

    accuracy                           0.88     10141
   macro avg       0.76      0.78      0.77     10141
weighted avg       0.89      0.88      0.88     10141
```

In [49]:
```
pred_DT=DT.predict_proba(xv_test)[:,1]
fp_DT, tp_DT, thresholds = roc_curve(y_test, pred_DT)
ra_DT = auc(fp_DT, tp_DT)
```

c. GradientBoosting 梯度提升

GradientBoostingClassifier分类器

In [50]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [51]:
```
GBC=GradientBoostingClassifier(random_state=0)
GBC.fit(xv_train,y_train)
```

Out[51]:
```
GradientBoostingClassifier(random_state=0)
```

In [52]:
```
GBC.score(xv_test,y_test)
```

Out[52]:
```
0.8350261315452125
```

In [53]:
```
pred_GBC=GBC.predict(xv_test)
```

In [54]:
```
print(classification_report(y_test,pred_GBC))
```

```
              precision    recall  f1-score   support

           0       0.92      0.89      0.90      8659
           1       0.45      0.52      0.48      1482

    accuracy                           0.84     10141
   macro avg       0.68      0.71      0.69     10141
weighted avg       0.85      0.84      0.84     10141
```

In [55]:
```
pred_GBC=GBC.predict_proba(xv_test)[:,1]
fp_GBC, tp_GBC, thresholds = roc_curve(y_test, pred_GBC)
ra_GBC = auc(fp_GBC, tp_GBC)
```

d. RandomForest 随机森林

RandomForestClassifier分类器

In [56]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [57]:
```
RFC=RandomForestClassifier(random_state=0)
RFC.fit(xv_train,y_train)
```

Out[57]:
```
RandomForestClassifier(random_state=0)
```

In [58]:
```
RFC.score(xv_test,y_test)
```

Out[58]:
```
0.9117444039049404
```

In [59]:
```
pred_RFC=RFC.predict(xv_test)
```

In [60]:
```
print(classification_report(y_test,pred_RFC))
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      8659
           1       0.82      0.51      0.63      1482

    accuracy                           0.91     10141
   macro avg       0.87      0.74      0.79     10141
weighted avg       0.91      0.91      0.90     10141
```

In [61]:
```
pred_RFC=RFC.predict_proba(xv_test)[:,1]
fp_RFC, tp_RFC, thresholds = roc_curve(y_test, pred_RFC)
ra_RFC = auc(fp_RFC, tp_RFC)
```

e. RidgeRegression 岭回归

RidgeClassifier分类器

In [62]:
```
from sklearn.linear_model import RidgeClassifier
```

In [63]:
```
RCF=RidgeClassifier(random_state=0)
RCF.fit(xv_train,y_train)
```

Out[63]:
```
RidgeClassifier(random_state=0)
```

In [64]:
```
RCF.score(xv_test,y_test)
```

Out[64]:
```
0.90760280051277
```

In [65]:
```
pred_RCF=RCF.predict(xv_test)
```

In [66]:
```
print(classification_report(y_test,pred_RCF))
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      8659
           1       0.75      0.56      0.64      1482

    accuracy                           0.91     10141
   macro avg       0.84      0.76      0.79     10141
weighted avg       0.90      0.91      0.90     10141
```

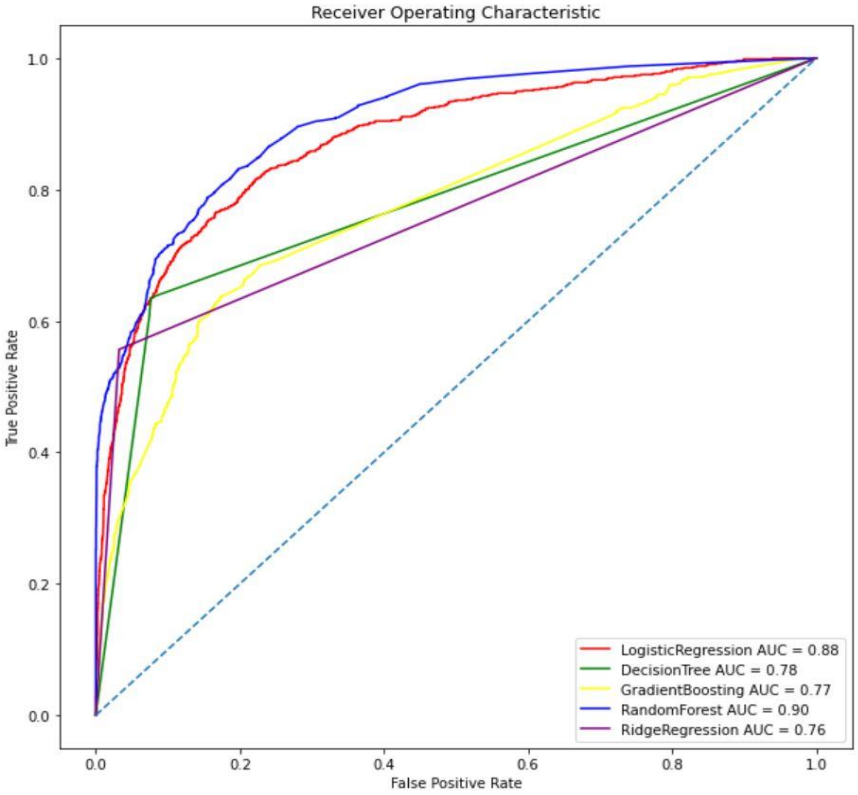
In [67]:
```
#pred_RCF=RCF.predict_proba(xv_test)[:,1]
fp_RCF, tp_RCF, thresholds = roc_curve(y_test, pred_RCF)
ra_RCF = auc(fp_RCF, tp_RCF)
```

（3） ROC 曲线

ROC 曲线可以查出一个分类器在某个阈值时对样本的识别能力。ROC 曲线越是靠近左上角，试验的 TPR 越高和 FPR 越低，即灵敏度越高，误判率越低，则诊断方法的性能越好。

In [68]:
```
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(fp_LR,tp_LR, color='red',label = 'LogisticRegression AUC = %0.2f' % ra_LR)
plt.plot(fp_DT,tp_DT, color='green',label = 'DecisionTree AUC = %0.2f' % ra_DT)
plt.plot(fp_GBC,tp_GBC, color='yellow',label = 'GradientBoosting AUC = %0.2f' % ra_GBC)
plt.plot(fp_RFC,tp_RFC, color='blue',label = 'RandomForest AUC = %0.2f' % ra_RFC)
plt.plot(fp_RCF,tp_RCF, color='purple',label = 'RidgeRegression AUC = %0.2f' % ra_RCF)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

Out[68]:
```
Text(0.5, 0, 'False Positive Rate')
```



其中纵轴 True Positive Rate = TP/(TP + FN) 横轴 False Positive Rate = FP/(FP + TN)

图中可以看出，在此实验中，随机森林的性能最好，性能最差的是岭回归。

2. 深度学习方法

(1) 处理数据

首先，将中文文本数据转换为数字。有很多方法，前面的 TF-IDF 就是其中一种方法。这里换一种方法：使用 Tokenizer 按照分词出现的次数排序建立字典，排序前 10000 的列入字典中。

Convert text to vectors, our classifier only takes numerical data.

In [57]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
x1=df_train["Title"]
x2=df_train["label"]
```

In [58]:
```
X_train, X_test, Y_train,Y_test = train_test_split(x1, x2, test_size=0.20, random_state=18)
```

In [59]:
```
tokenizer=Tokenizer(num_words=10000)
tokenizer.fit_on_texts(x1)
tokenizer.word_index
```

Out[59]:
```
{'信息': 1,
 '实': 2,
 '内容': 3,
 '标题': 4,
 '请': 5,
 '明星': 6,
 '公众': 7,
 '文章': 8,
 '不符': 9,
 '不实': 10,
 '恶意': 11,
 '事实': 12,
 '中国': 13,
 '造谣': 14,
 '艺人': 15,
 '号': 16,
 '虚假': 17,
 '事件': 18,
 '谢谢': 19,
```

再将其转化为列表，这样就得到数字信息了。

In [54]:
```
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
```

In [73]:
```
X_train
```

Out[73]:
```
[[630,
  129,
  1044,
  4673,
  543,
  6770,
  8226,
  226,
  4772,
  1142,
  3242,
  800,
  1360,
  4663,
  3103,
  265],
 [9110, 2544, 2374, 480, 2697, 3649, 8, 231, 22, 9110],
 [701,
  3372,
```

再统一长度，保证每个文本都是同样的长度，避免不必要的错误。

In [62]:
```
X_train = tf.keras.preprocessing.sequence.pad_sequences(X_train, padding='post', maxlen=256)
X_test = tf.keras.preprocessing.sequence.pad_sequences(X_test, padding='post', maxlen=256)
```

（2）搭建 RNN

Embedding 层将数字列表转化为向量列表

In [63]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(10000, 128),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(16)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1)
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 128)         1280000

 bidirectional (Bidirectiona  (None, None, 128)        98816
 l)

 bidirectional_1 (Bidirectio  (None, 32)               18560
 nal)

 dense (Dense)               (None, 64)                2112

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65
```

（3）模型训练

EarlyStopping 当监测数量停止改善时停止训练。monitor：要监测的数据；val_loss，验证集的损失函数（误差）；patient：对于设置的 monitor，可以忍受在多少个 epoch 内没有改进，不宜设置过小，防止因为前期抖动导致过早停止训练，也不宜设置的过大；restore_best_weights：当发生 EarlyStopping 时，模型的参数未必是最优的，将其设置为 True，则自动查找最优的 monitor 指标时的模型参数。model.compile(optimizer = 优化器，loss = 损失函数，metrics = ["准确率"])

In [64]:
```
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True)
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])

history = model.fit(X_train, Y_train, epochs=10,validation_split=0.1, batch_size=30, shuffle=True, callbacks=[early_stop])
```

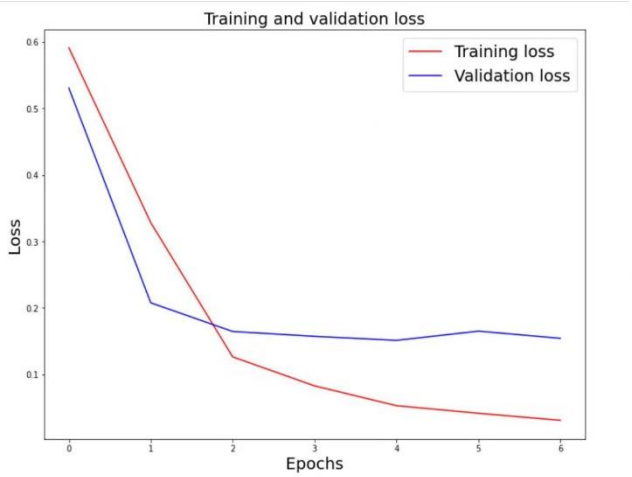
```
Epoch 1/10
255/255 [==============================] - 98s 363ms/step - loss: 0.5821 - accuracy: 0.7390 - val_loss: 0.5026 - val_accuracy: 0.7450
Epoch 2/10
255/255 [==============================] - 109s 427ms/step - loss: 0.2771 - accuracy: 0.8636 - val_loss: 0.2089 - val_accuracy: 0.9268
Epoch 3/10
255/255 [==============================] - 110s 432ms/step - loss: 0.1222 - accuracy: 0.9631 - val_loss: 0.1749 - val_accuracy: 0.9398
Epoch 4/10
255/255 [==============================] - 102s 398ms/step - loss: 0.0752 - accuracy: 0.9772 - val_loss: 0.1688 - val_accuracy: 0.9492
Epoch 5/10
255/255 [==============================] - 111s 436ms/step - loss: 0.0537 - accuracy: 0.9848 - val_loss: 0.1906 - val_accuracy: 0.9410
Epoch 6/10
255/255 [==============================] - 116s 456ms/step - loss: 0.0375 - accuracy: 0.9902 - val_loss: 0.1735 - val_accuracy: 0.9587
```

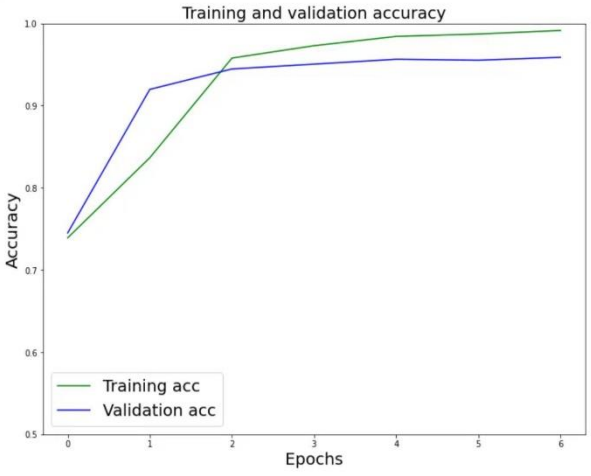
model.fit()方法来执行训练过程，每一批 batch 的大小为 30，迭代次数 epochs 为 10，从测试集中划分 90%给训练集。

(4)loss 和 accuracy 曲线

In [79]:
```
history_dict = history.history

acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']
epochs = history.epoch

plt.figure(figsize=(12,9))
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss', size=20)
plt.xlabel('Epochs', size=20)
plt.ylabel('Loss', size=20)
plt.legend(prop={'size': 20})
plt.show()

plt.figure(figsize=(12,9))
plt.plot(epochs, acc, 'g', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy', size=20)
plt.xlabel('Epochs', size=20)
plt.ylabel('Accuracy', size=20)
plt.legend(prop={'size': 20})
plt.ylim((0.5,1))
plt.show()
```

fit 函数返回 History 对象，调出 history 中记录的各指标画出 loss 函数和 accuracy 函数。





也能看到深度学习的各指标是较好的:

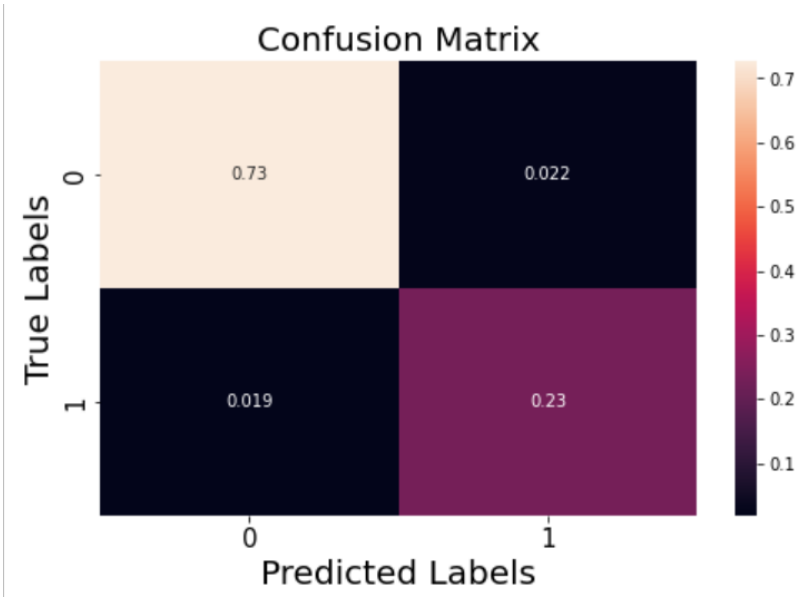
In [79]:
```
print('Accuracy on testing set:', accuracy_score(binary_predictions, Y_test))
print('Precision on testing set:', precision_score(binary_predictions, Y_test))
print('Recall on testing set:', recall_score(binary_predictions, Y_test))
```

```
Accuracy on testing set: 0.9589235127478754
Precision on testing set: 0.912639405204461
Recall on testing set: 0.9246704331450094
```

(5)混淆矩阵绘图

各块显示的是概率

In [80]:
```
import seaborn as sns
matrix = confusion_matrix(binary_predictions, Y_test, normalize='all')
plt.figure(figsize=(8, 5))
ax= plt.subplot()
sns.heatmap(matrix, annot=True, ax = ax)

# labels, title and ticks
ax.set_xlabel('Predicted Labels', size=20)
ax.set_ylabel('True Labels', size=20)
ax.set_title('Confusion Matrix', size=20)
ax.xaxis.set_ticklabels([0,1], size=15)
ax.yaxis.set_ticklabels([0,1], size=15)
```



（三）指标分析

1. 是否平衡真假数据的指标对比（以 LogisticRegression 为例）

LogisticRegression分类器

```
from sklearn.linear_model import LogisticRegression

LR=LogisticRegression()
LR.fit(xv_train,y_train)
```

```
LogisticRegression()
```

```
LR.score(xv_test,y_test)
```

```
0.8861059067153141
```

```
pred_LR=LR.predict(xv_test)
```

```
print(classification_report(y_test,pred_LR))
```

	precision	recall	f1-score	support
0	0.90	0.97	0.94	8659
1	0.70	0.38	0.49	1482
accuracy			0.89	10141
macro avg	0.80	0.68	0.71	10141
weighted avg	0.87	0.89	0.87	10141

LogisticRegression分类器

```
from sklearn.linear_model import LogisticRegression

LR=LogisticRegression()
LR.fit(xv_train,y_train)
```

```
LogisticRegression()
```

```
LR.score(xv_test,y_test)
```

```
0.8948821615225323
```

```
pred_LR=LR.predict(xv_test)
```

```
print(classification_report(y_test,pred_LR))
```

	precision	recall	f1-score	support
0	0.93	0.95	0.94	8659
1	0.67	0.55	0.61	1482
accuracy			0.89	10141
macro avg	0.80	0.75	0.77	10141
weighted avg	0.89	0.89	0.89	10141

可以看到各指标略有提升，但总体差距不大。

2. loss 曲线分析拟合效果

（1）当模型**欠拟合**时，loss 函数表现为下图 1 或图 2。图一 training loss 下降的非常平缓以至于好像都没有下降，这说明模型没有从训练集学到什么东西。图二特点是在训练结束时候 training loss 还在继续下降，这说明还有学习空间，模型还没来得及学就结束了。

图 1

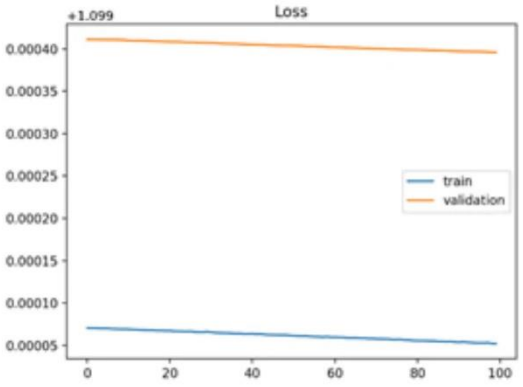
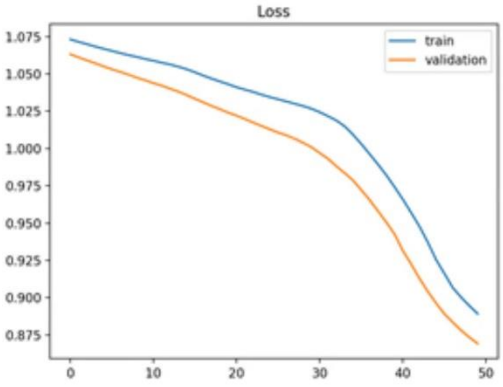


图 2



（2）当模型**过拟合**时，loss 函数表现为下图 3。模型把训练集学的有点过了，以致于把一些噪音和随机波动也学进来了。过拟合有时候是因为训练太久造成的。training loss 一直在不断地下降，而 validation loss 在某个点开始不再下降反而开始上升了，这就说明我们应该在这个拐点处停止训练。

图 3

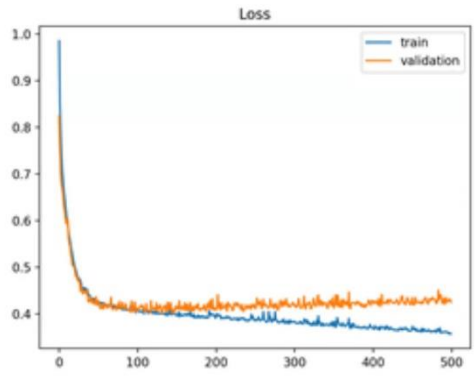
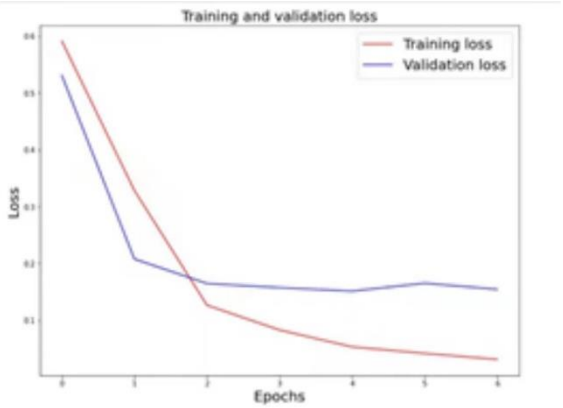


图 4



（3）当模型**完美拟合**时，特点是 training loss 和 validation loss 都已经收敛并且之间相差很小。可以看到我们的函数（图 4）training loss 和 validation loss 相差虽然不是很小，不能算是完美的拟合，但也不存在过拟合和欠拟合等问题，总体上拟合状态较好。