## Coding A/B test

### Import

import numpy as np

import pandas as pd

import seaborn as sns #for plotting

import matplotlib.pyplot as plt #for plotting

from scipy.stats import ttest\_ind #for t-test

### Generate new features

Generate a new feature 'is\_new' representing a new user or an old user.

data['interval'] = (data['c2'] - data['c1']).apply(lambda x: x.days)

data['is\_new'] = (data['interval'] == 0).astype(int)

### Target values distribution on control and test

#Print the target values distribution on control and test.

data.groupby('if\_testgroup')['target'].mean()

#Plot the target values distribution on control and test.

fig, ax = plt.subplots(figsize=(8, 5))

sns.barplot(x='if\_testgroup', y='target', data=data, ax=ax)

ax.set\_xlabel('device', fontsize=12) #横坐标label

ax.set\_ylabel('Number of devices', fontsize=12)

plt.tight\_layout()

plt.show()

### For each feature, feature distribution on control and test + target distribution on control and test

Take the feature device for example, 一行两幅图

#For each feature, feature distribution on control and test

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(18, 6))

#横轴是data中的feature’device’，纵轴是count(device),hue表示double bar每个bar对应'if\_testgroup'一个值

sns.countplot(x='device', hue='if\_testgroup', data=data, ax=ax[0])

ax[0].set\_title('Device distribution on test and target', fontsize=16)

#target distribution on control and test for each feature

sns.barplot(x='device', y='target', hue='if\_testgroup', data=data, ax=ax[1])

ax[1].set\_title('Pages Visited vs. device', fontsize=16)

plt.tight\_layout()

plt.show()

两行两幅图

fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(18, 12))

sns.countplot(x='device', data=data, ax=ax[0])

ax[0].set\_ylabel('Device distribution on test and target', fontsize=12)

sns.barplot(x='device', y='target', data=data, ax=ax[1])

ax[1].set\_ylabel('Pages Visited vs. device', fontsize=12)

plt.tight\_layout()

plt.show()

### T-test for all features

#test这一列表示是否属于control group or test group,target variable is target

control\_target = data[data['if\_testgroup'] == 0]['target'].values

test\_target = data[data[' if\_testgroup'] == 1]['target'].values

print('T-test:\t', ttest\_ind(a=control\_target, b=test\_target, equal\_var=False)) #output t\_value, p\_value

### T-test for each feature

Take the feature ‘device’ for example

def run\_ttest(df):

test\_data = df[df['if\_testgroup'] == 1]['target'].values #或者df.loc[df. if\_testgroup == 0, 'target']

test\_mean = test\_data.mean()

ctrl\_data = df[df['if\_testgroup'] == 0]['target'].values

ctrl\_mean = test\_data.mean()

result = ss.ttest\_ind(ctrl\_data, test\_data, equal\_var=False)

conclusion = 'Significant' if result.pvalue < 0.05 else 'Not Significant'

return pd.Series({'number\_test': len(test\_data),

'number\_ctrl': len(ctrl\_data),

'mean\_test': test\_mean,

'mean\_ctrl': ctrl\_mean,

'mean\_diff': test\_mean - ctrl\_mean,

'pvalue':result.pvalue,

'conclusion':conclusion})

tests.groupby('device').apply(data). .reset\_index()

## Parses a JSON string and converts it to a Pandas DataFrame

Common examples of unstructured data are text, picture, vedio

Common examples of semi-structured data are XML、JSON.

Common examples of structured data are Excel files or SQL databases. They have structured rows and columns that can be sorted.

格式[{ }, { }, ….{ }]

每个{ }里的内容如下：

Use two method pd.read\_json(). The josn file string format有特定的格式要求

json.load 需要自己解析

{"session\_id":["D258NVMV202LS"],

"unix\_timestamp":[1442640552],

"cities":["San Jose CA, Montreal QC"],

"user":[[{"user\_id":5749,"joining\_date":"2015-04-02","country":"FR"}]]}

session\_id timestamp cities user\_id joining\_date country

**import json**

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

#将从.josn文档读入的data，parse data read from a josn file into dataframe

def parse\_json(data): # function to parse json data

session\_id = [] #解析后有几个feature，就初始化几个如session\_id, timestamp, cities…

timestamp = []

cities = []

user\_id = []

joining\_date = []

country = []

for item in data:

session\_id.append(item['session\_id'][0])

timestamp.append(item['unix\_timestamp'][0])

cities.append(item['cities'][0])

user\_id.append(item['user'][0][0]['user\_id'])

joining\_date.append(item['user'][0][0]['joining\_date'])

country.append(item['user'][0][0]['country'])

# create DataFrame

maps = {'session\_id': session\_id, 'timestamp': timestamp, 'cities': cities,

'user\_id': user\_id, 'joining\_date': joining\_date, 'country': country}

columns=['session\_id', 'timestamp', 'cities', 'user\_id', 'joining\_date', 'country']

return pd.DataFrame(maps, columns=columns)

with open('./data/city\_search.json', 'r') as f:

data = json.load(f)

data = parse\_json(data)

data['joining\_date'] = pd.to\_datetime(data['joining\_date'])

### hackerrank

#### 输入

#输入多个整数n, m

n, m = map(int, input().split())

#输入1个整数n

n=int(input())

#输入1个浮点数f

f=float(input())

#输入n行float数据，存入array(如果不让用numpy就不能用这个）

train = np.array([input().split() for \_ in range(n)], float) #训练数据，每一行最后一个数是y，前面是attributes

#输入n行float数据，存入list

mylist = []

for \_ in range(n):

mylist.append( map(float, input().split()) )

## Git

创建（**Repository）**增加内容，提交修改 ，并把修改同步到远程库，常用的命令是git clone、git checkout、git commit、git push、git pull等。

## Deep Learning LSTM GRU MLP

### 安装TF，Keras

注意anaconda里python的 版本、 TF版本, keras版本的兼容性

1.构建TF虚拟环境

prompt中输入python，查看python的版本，比如是3.7.6

prompt---conda create -n tfjane python=3.7.6

等待一段时间，可能还要输入y，安装完成后输入activate tfjane或者conda activate tfjane，回车，如果下一行开头出现(tfjane)，则表示环境设置成功，即进入TF环境

2.在TF中安装TF和Keras

TF环境中---conda install tensorflow

完成后，输入python---import tensorflow as tf如果不报错，则继续安装

conda install ipython

conda install jupyter notebook

conda install nb\_conda

继续安装keras

TF环境中--- pip install keras==2.3.1 ---注意这里指定了keras的版本号，一开始可以不指定，可能会报错，因为keras的版本和tensorflow不兼容

完成后，输入python---import keras如果出现‘using tensorflow backend’则成功

### 启动tensorflow环境下的jupyter

anaconda >anaconda prompt---activate tfjane—打开jupyter(tfjane)---new---选择tfjane

不用了，关闭jupyter notebook

anaconda >anaconda prompt---deactivate

### 如何运行TF，Keras

anaconda里Jupyter里运行tensorflow，

1.打开 Anaconda Prompt 终端

2.输入：conda activate tensorflow

3.等待，输入：jupyter notebook

4.不想要写程序的时候，Anaconda Prompt 终端输入deactivate

### MLP (Multiple Layer Perception)

* **MLP with 3 hidden layers for binary classification**

from keras.layers import Dense,LSTM,Dropout

from keras.models import Sequential

from keras import optimizers

#用归一化后的training data训练，接收的是2维数据，LSTM接收3维数据

#2维数据和LightGBM的training data是一样的

#one input layer, three hidden layers, one output layer，2 dropout layers也可以不要。

#如果修改了model.add(layers.Dense...),重新执行model=keras.Sequential()

#网络构建

model = Sequential()

model.add(Dense(16, input\_dim = 11, activation = 'relu'))

model.add(Dropout(0.2)) #deal with overfitting，本层输入32个节点,丢弃20%

model.add(Dense(32, activation = 'relu'))

model.add(Dropout(0.2)) #dropout layer

model.add(Dense(16, activation = 'relu')) #hidden layer 16 nodes

model.add(Dense(1, activation = 'sigmoid')) #output layer，1 node二分类

model.summary()#查看构建的网络，此时并未训练网络

#网络compile

model.compile(optimizer ='adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

#training

#history 记录model训练过程，可以从下面两种中选择

#Training dataset中取20%作为validation dataset

#Test dataset作为validation dataset

选history = model.fit(train\_x, train\_y, batch\_size =256, epochs=10, validation\_split=0.2)

选history = model.fit(x\_train, y\_train, batch\_size =256, epochs=10, validation\_data=(x\_test, y\_test) )

* **MLP 多分类**

与MLP二分类，两个方面不同

1.网络构建中，output layer的参数，

model.add(Dense(3, activation = 'softmax')) #三分类问题

2.compile中的loss参数

#target variable 做one hot encoding，100,010,001, loss='categorical\_crossentropy'

#target variable 做顺序编码，比如用0,1,2表示3个类别,loss='sparse\_categorical\_crossentropy'

model.compile(optimizer ='adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

* **MLP Regression**

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

* **L2 deal with overfitting**

#自己设置的L2参数0.005，对weights惩罚的参数，这个对性能影响大。

from keras import regularizers

model.add(Dense(32, input\_dim = 11, kernel\_regularizer=regularizers.l2(0.005), activation = 'relu'))

### Prediction

#使用训练好的模型在test data上进行预测

y\_pred\_keras = model.predict(X\_test)

### Evaluation

#使用训练好的模型在training data上的性能评价

loss, accuracy = model.evaluate(X\_train, y\_train, verbose=0)

### LSTM (Long Short Term Memory)

是RNN的一种

* **单层LSTM 网络, binary classification**

#LSTM输入是3维数据

#32个节点(128, 64个节点也行)，input\_shape是数据的后两维，数据的shape是(10000,120,11)

model = keras.Sequential()

model.add(layers.LSTM(32, input\_shape=(train\_x.shape[1:]), activation='tanh'))

model.add(layers.Dense(1)) #output layer

* **3层LSTM网络,binary classification**

3 hidden layers，为什么return\_sequences因为LSTM输入是3维，只有output layer前一层可以不用return\_sequences

model = keras.Sequential()

model.add(layers.LSTM(32, input\_shape=(train\_x.shape[1:]), return\_sequences=True))

model.add(layers.LSTM(32, return\_sequences=True))

model.add(layers.LSTM(32))

model.add(layers.Dense(1)) #output layer

* **3层LSTM网络,,binary classification**

model.compile(。。。。。) #回归问题

#训练中不断降低Learning Rate，连续3个val\_loss没有降低，则学习速率\*factor,当LR降到min\_lr时，不能再降了

LR\_reduction = keras.callbacks.ReduceLROnPlateau(monitor='val\_loss', patience=3, factor=0.5, min\_lr=0.00001)

history = model.fit(train\_x, train\_y, batch\_size = 128, epochs=200, validation\_data=(test\_x, test\_y), callbacks=[LR\_reduction])

### Bidirectional LSTM+recurrent\_dropout文本处理

主要用于text, audio处理，从两个方向读取信息

from keras.layers import Bidirectional

model=keras.Sequential()

model.add(layers.Embedding(10000, 16, input\_length=200))

model.add(layers.Bidirectional(layers.LSTM(64, dropout=0.2, recurrent\_dropout=0.5))) #the first hidden layer

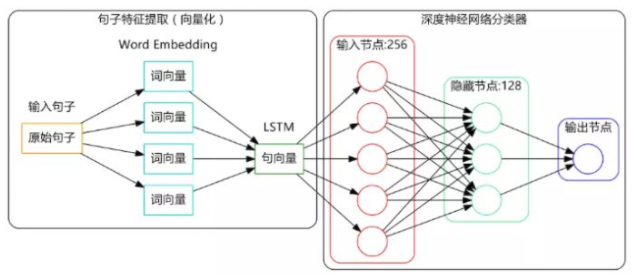
model.add(layers.Dropout(0.5))

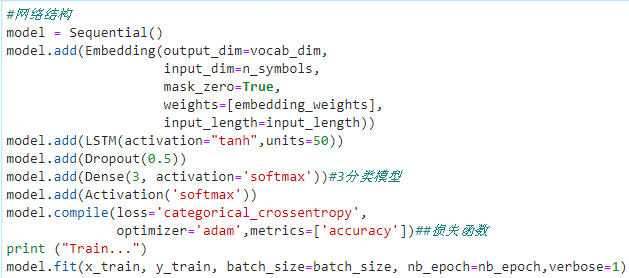
model.add(layers.Dense(1,activation='sigmoid')) #the output layer

recurrent\_dropout，每次删除相同位置的node，我也不太懂

### GRU文本处理

GRU相比LSTM结构更简单，参数更少，因此应用也比较广泛





注意

 inspect imbalanced classes----过采样或者欠采样



## Attention+LSTM文本处理

### 1.安装NTLK, gensim, word2vec, spacy

在TF虚拟环境下装，才能在tfjane中使用即在tensorflow中使用

anaconda >anaconda prompt---tfjane---输入pip install --upgrade gensim

anaconda >anaconda prompt--- tfjane---输入conda install word2vec 或pip…

anaconda >anaconda prompt--- tfjane---输入pip install nltk

anaconda >anaconda prompt--- tfjane---输入**pip install spacy**

conda install -c conda-forge spacy-model-en\_core\_web\_sm #Spacy的英文包

pip install neuralcoref

pip install textacy

注意，anaconda版本新，python版本过高，可能安装不上

测试安装成功 否？

Anaconda prompt---python---import gensim

### 2.启动tensorflow环境下的jupyter

anaconda >anaconda prompt---activate tfjane—打开jupyter---new---选择tfjane

不用了，关闭jupyter notebook

anaconda >anaconda prompt---deactivate

n-gram模型，n=1表示计算一个句子的概率的时候，只考虑每个word与前面的1个word相关，n=2表示。。。通常选择n=2或3或4



### 3.短信数据预处理

delete: 标点符号、括号、问号，只留下字母、数字和字符

将大写字母转化为小写

### 4.每个word转化为vector

从所有短信数据中学习，为每个word产生一个vector

Word2Vec

Glove更优

idea case: Google已经为300万word训练构建了vector,每个vector维度为300, 300-dimensional vectors for 3 million words

这个单词向量矩阵太大了（3.6G）

GloVe 进行训练得到包含 400,000 个word向量，00-dimensional vectors for 400,000 words

### 5. RoBERTa

6.XLNet

7.BiLSTM+Attention

### 8.评价方法

accuracy+Recall+F1-score



### 

### Spacy

### Data preprocessing

初始同一类别的数据在一起，数据预处理的时候，打乱，有利于model training

index=np.random.permutation(len(data)) #乱序化(0，len(data))的数字

data=data.iloc[index] #乱序后的index给data

features=data[data.columns[1:-3]] #取第一列到倒数第4列为features

target=data.iloc[:, 3:]取最后3列为target，target有3个值，多分类