如何从一个人的名字判断他是不是中国人?

1. 简介

机器学习，人工智能，数据挖掘等等这些词汇在当下这个信息不断膨胀的时代，其热度也逐渐从传统的计算领域延伸到各个商业，工业领域，从最早的手机人脸识别，email的垃圾邮件分类，图片压缩，推荐系统，到后来的自动驾驶，AlphaGo，无一不是当时最前沿的数据科学的产物。并且随着GPU计算性能的提升，深度学习，这一些曾经学界研究已久的模型框架，也逐渐进入了实际的生产中：语音识别（如siri），音乐生成，图像识别等等过去棘手的问题似乎有了一条更强大的解决方案。不过了解这些对身处工业界的我们并无实质性的帮助，事实上我们需要有足够的积累才能处理上述问题。而对于第一次接触该技术原理的人，如果想要了解一个智能模型是如何工作的，最简单的一个方法就是从一个案例入手体会其设计。

为了摆脱实验数据（hypothetical data）的无趣，同时为了控制问题的难度，本文在此采用的是一个**分类**问题，即给出一个人名(作为输入)，让程序判断他**是否**是中国人（即输出是 `True` 或者 `False` ）。本文将以Python为实现语言，构造一个模型，并通过一些简单的概率运算，来实现该功能。

注释：以下是通过网络爬虫从CMU CEE研究生校友名录中获得的人员名单，并通过正则匹配加上人工校验得到的学习数据集，分别存储于 `chinese.txt` 和 `foreigner.txt` 之中。通过以下代码可以查看这两个文件，及文件中姓名的存储格式。

In [1]:

```
cnames = open('chinese.txt').readlines()
print('='*40+'\nThere are %d names in "chinese.txt"\nFirst 5 names are:'%(len(cn
ames)))
for i in range(5):
    print(cnames[i].strip())

fnames = open('foreigner.txt').readlines()
print('='*40+'\nThere are %d names in "foreigner.txt"\nFirst 5 names are:'%
(len(fnames)))
for i in range(5):
    print(fnames[i].strip())
print('='*40)
```

```
========================================
There are 135 names in "chinese.txt"
First 5 names are:
Cai, Lisheng
Chao, Shucheng
Chen, Dihong
Chen, Fangyan
Chen, Shu
========================================
There are 138 names in "foreigner.txt"
First 5 names are:
Adilla, Fatimah
Agarwal, Ajay
Ahn, Jungho
Al Abbas, Ridha
Al Arfaj, Abdullah
========================================
```

2. 贝叶斯与机器学习

在正式进入判别器的设计之前，我们必须介绍的是一个概率论上赫赫有名的公式，即**贝叶斯公式**:

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)}.$$

该公式虽然表达简单，但却是绝大多数智能模型的理论基础，其意义可以如下阐释：若 $X \in \mathbb{R}^n, Y \in \{0, 1\}$ 分别代表输入和输出，$P(Y \mid X)$ 则是我们需要的目标，即在知道 X 的情况下，Y 发生的概率是多少，如果该值比较大，那么我们可以认为 Y 是对的（虽然不是严格意义上的正确，但在概率上很可能是对的）。比如，如果我们知道某人名字的拼写是`Sam`，那么他是中国人的概率，通过我们的计算，得到的结果 $P(\text{有这个名字的是一个中国人} \mid \text{名字是} Sam)$ 应该很低，反之，如果我们输入一个中国人的名字，比如`Xiao Ming`这个计算得到的概率应该很高。

但现实是很无情的，我们通常是无法直接计算 $P(Y \mid X)$ 的，很大程度是因为我们无法计算 $p(X)$ 。但好在，虽然无法计算，但在给定 X 的情况下，该值是一个定值，我们可以忽略，于是

$$P(Y \mid X) \propto P(X \mid Y)P(Y).$$

这时我们就可以通过计算 $P(X \mid Y)P(Y)$ 来估计 $P(Y \mid X)$ 了。

但此时我们仍然遇到了一些问题，那就是 $P(X \mid Y)$ 该如何表达。首先我们来考虑其实际含义 —— “知道这个人是中国人（或者不是）的前提下，他的名字是 X 的概率”，于是如果我们能设计出一种计算模型使得在 Y 不同的时候，该概率会有明显区别，这种计算模型就可以用来估算 $P(X \mid Y)$，而这个模型也常被称为**假设（hypothesis）**，记作 $\mathcal{H}(\theta) : \mathbb{R}^n \to \mathbb{R}$ 。而这个模型中可能会拥有一些参数，用来计算输出，这些参数是不知道的，因此需要通过**样本**进行估计。于是我们接下来要做的就是设计一个“假设”，并且估计其中的参数。（这一估计的过程也称为**训练**或者**学习**）再得到最优估计后，我们就可以利用已有的模型对 $P(Y \mid X)$ 进行估计了。

3. 设计我们的判别器

3.1 直观的理解问题

由于中国人的姓名一定是由拼音拼写成的，因此如果满足拼音拼写规则，则可以认为是中国人；而且拼音拼写规则有限，我们事实上是可以利用该性质对这个问题进行求解的。但这个性质同时也说明了另一个语音上的规律：不同的语言中由于发音习惯的差别，其拼写也会出现一些相应的差别，比如拼音中就会有ing,ong,eng...这样的后鼻音，而且拼音是单音节，多由辅音和原音组成，这也造成了中国人的首字母，不大可能是原音，再加上一些文化上的因素，比如一些大姓，进一步收束了这个分布：如果说中国人名字的拼写存在规律，那么一定会和外国人的存在差别。（接下来我们就能看到）

那么我们该如何设计呢，此处我们做出两个假定，用来简化计算：

- 我们仅对首字母 X_0 及相邻字母 (X_i, X_{i+1}) 进行建模，更复杂的组合规律（如-ing / -ong...）予以忽略；且相邻字母对分布不受位置影响（*比如，开始是ng的概率和结束是ng的概率相同，虽然明显这个假设有误，但是是可以帮助简化计算*）。
- 假定每对相邻字母对的分布相互独立，即 $(X_{i-1} \to X_i) \perp (X_i \to X_{i+1})$ ，这样 $P(X \mid Y)$ 就可以表达成：

  $$P(X \mid Y) \approx P(X_0) \prod_{i=0}^{n-1} P(X_i \to X_{i+1})$$

  如此我们只需要分别统计 $P(X_0)$ 和 $P(X_i \to X_{i+1})$ 即可计算 $P(X \mid Y)$ 了。

于是我们首先对已有的**训练资料（training data）**（`chinese.txt` 和 `foreigner.txt`）进行统计，统计范围为 `A-Z` 外加空格的27个字符。首先我们给予每个字母一个编号：

In [2]:

```
from string import ascii_uppercase as ucase
# all the upper case English characters
# print ucase
# all the characters we concern
chars = ' ' + ucase

# build a dictionary between characters(whit-space and A-Z) and index (0-26)
c2i = dict(zip(chars, range(27)))

print('the characters considered in this program:\n%s'%(chars,))
print('the index of "%s" is %d'%(' ',c2i[' ']))
print('the character at index %d is "%s"'%(13, chars[13]))
```

```
the characters considered in this program:
 ABCDEFGHIJKLMNOPQRSTUVWXYZ
the index of " " is 0
the character at index 13 is "M"
```

接下来我们就可以读入文件进行统计了，其思路如下，给定的名字格式如“姓氏,(white space)名字”，我们可以读入每一行，分别对首字母和相邻字母进行统计，并分别存入一个 $(1, 27)$ 的向量和 $(27, 27)$ 的矩阵中：

In [3]:

```
import numpy as np
first = np.zeros((2,27))
pairs = np.zeros((2,27,27))

for c in range(2):  # c means class label, 0-false(non-chinese), 1-true(chinese)
    names = [fnames, cnames][c]
    for i in range(len(names)):  # i is the record index
        # Remove '\n', convert into upper case and split into two parts
        name = names[i].strip().upper().split(', ')
        for n in name:  #  n in one part of the name
            # count the first character
            first[c, c2i[n[0]]]+=1
            # count the neighboring pairs
            for j in range(len(n)-1):
                pairs[c, c2i[n[j]], c2i[n[j+1]]]+=1
```

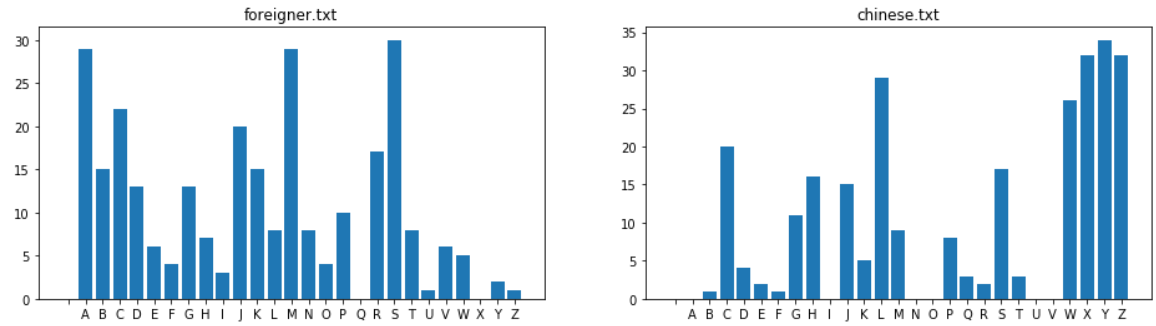
我们前面说到由于发音习惯的不同，不同语言在拼写上也存在着一些差异，如果这个差异切实存在，那么一定可以从我们上面统计的两个分布当中看出来，比如首字母分布的统计结果：

In [4]:

```
from matplotlib import pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (16.0, 4.0) # set default size of plots

global charnum, indxchar
plt.close('all')
ax1 = plt.subplot(121)
ax2 = plt.subplot(122)
axes = [ax1,ax2]

for i in [0,1]:
    ax = axes[i]
    plt.sca(ax)
    plt.title(['foreigner.txt','chinese.txt'][i])
    plt.bar(range(27), first[i,:])
    ax.set_xticks(np.arange(27))
    ax.set_xticklabels(list(chars))
plt.show()
```



In [5]:

```
most5 = (np.argsort(first))[:,-5:]
print('The first 5 most frequent initials in foreigner\'s name:\n%s'
      %(', '.join(map(lambda x: chars[x],most5[0,::-1]))))
print('The first 5 most frequent initials in Chinese\'s name:\n%s'
      %(', '.join(map(lambda x: chars[x],most5[1,::-1]))))
```

```
The first 5 most frequent initials in foreigner's name:
S, M, A, C, J
The first 5 most frequent initials in Chinese's name:
Y, Z, X, L, W
```

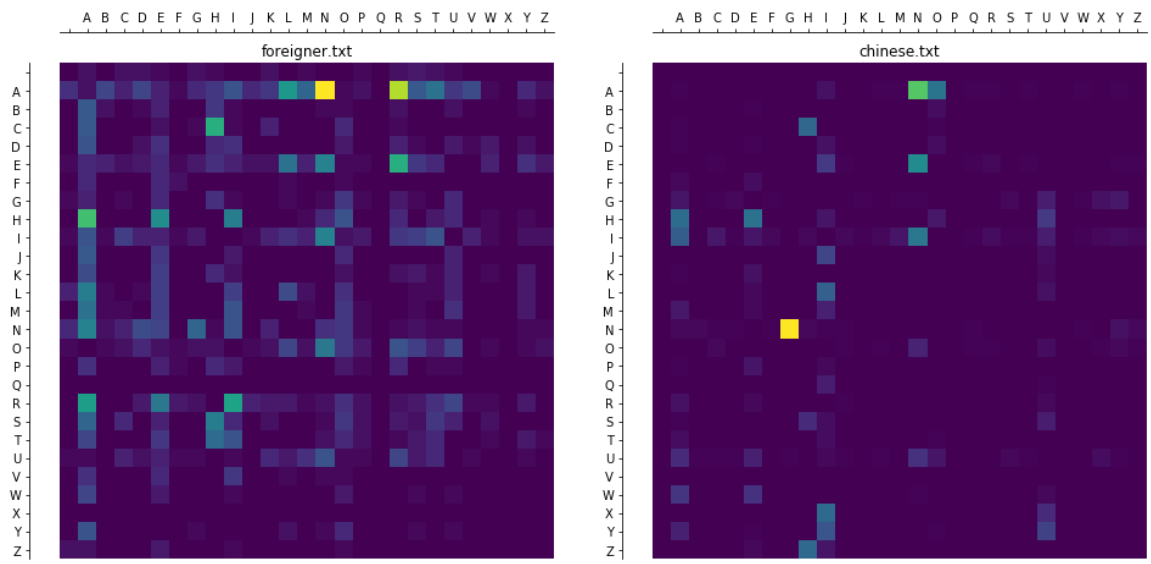
其中我们可以看出，对于首字母而言，中国人和外国人的最常见首字母在分布上是有很大差异的，而且最频繁的5个首字母甚至完全不一样。这对于我们来说是一件非常好的事情，因为这样我们就能过通过这个区别来区分中国人的姓名了。除了首字母，相邻字母对的分布也应该有一些差别：

In [6]:

```
%matplotlib inline
plt.rcParams['figure.figsize'] = (16.0, 8.0) # set default size of plots

global charnum, indxchar
plt.close('all')
ax1 = plt.subplot(121)
ax2 = plt.subplot(122)
axes = [ax1,ax2]

for i in [0,1]:
    ax = axes[i]
    plt.sca(ax)
    plt.title(['foreigner.txt','chinese.txt'][i])
    plt.imshow(pairs[i,:,:], interpolation='nearest')
    ax.set_xticks(np.arange(27))
    ax.set_xticklabels(list(chars))
    # Move left and bottom spines outward by 25 points
    ax.spines['left'].set_position(('outward', 25))
    ax.spines['top'].set_position(('outward', 25))
    # Hide the right and top spines
    ax.spines['right'].set_visible(False)
    ax.spines['bottom'].set_visible(False)
    # Only show ticks on the left and bottom spines
    ax.yaxis.set_ticks_position('left')
    ax.yaxis.set_ticks(range(27))
    ax.yaxis.set_ticklabels(chars)
    ax.xaxis.set_ticks_position('top')
    ax.xaxis.set_ticks(range(27))
    ax.xaxis.set_ticklabels(chars)
plt.show()
```



In [7]:

```
m = np.argsort(pairs[0,:,:].flatten())[:-6:-1]
print('The 5 most frequent pairs in "foreigner.txt":')
print(map(lambda x: '->'.join((chars[x[0]],chars[x[1]])) ,(map(lambda m: (m/27,
m%27), m))))
m = np.argsort(pairs[1,:,:].flatten())[:-6:-1]
print('The 5 most frequent pairs in "chinese.txt":')
print(map(lambda x: '->'.join((chars[x[0]],chars[x[1]])) ,(map(lambda m: (m/27,
m%27), m))))
```

```
The 5 most frequent pairs in "foreigner.txt":
['A->N', 'A->R', 'H->A', 'E->R', 'C->H']
The 5 most frequent pairs in "chinese.txt":
['N->G', 'A->N', 'E->N', 'I->N', 'A->O']
```

可以看出，在相邻的字母对上，中国人和外国人的名字仍然有一定的区别，这为我们接下来构造判别器提供了可靠的依据。

3.2 判别函数

注意：由于两个文件中姓名数目接近，不妨假设 $P(Y=1)=0.5$ 。

我们现在已经知道了每个首字母以及每个字母对在数据集中出现的次数，根据我们的假设

$$P(X \mid Y) \approx P(X_0 \mid Y) \prod_{i=0}^{n-1} P(X_i \to X_{i+1} \mid Y),$$

我们估算 $P(X \mid Y)$ 时只需将给定的这些概率进行乘积就行；那么我们最直接的想法是：将频数除以总数就可以估算到频率了，并可以用它近似替代概率：

$$P(X_i \to X_{i+1} \mid Y) \approx \frac{N(X_i \to X_{i+1} \mid Y)}{N},$$

但这样做会有一个潜在的漏洞，如果有一个字母对 $(\alpha \to \beta)$ 不存在于数据集中，那么对该字母对的频率估计就是0，这样如果 X 含有该字母对，我们对其的估计 $P(X \mid Y)=0; \forall Y$ 。那么我们便无法对 $P(Y \mid X)=\frac{0}{0+0}$ 进行估计了。一个简单的类比如下：

一个球队和对方比赛在N场比赛中一场没赢，那么下一次比赛该球队的获胜概率根据最或然估计(MLE)为0，而事实上这个却说不通，因为就算是一次没赢，该球队应该仍然有获胜机会（即便不大），于是贝叶斯给出贝叶斯估计如下

$$\hat{p}=\frac{n+\sigma}{N+2\sigma}$$

其中 σ 为一个给定的参数，在未有观测的情况下，该值为0.5，满足我们的常识，这种方式称为**拉普拉斯光滑化(Laplace Smoothing)**。

In [8]:

```
# Laplace Smoothing(sigma=1)
first +=1; pairs +=1
for c in range(2):
    first[c,:] /= first[c,:].sum()
    pairs[c,:,:] /= pairs[c,:,:].sum()
```

进行光滑化后，我们便可以对 $P(X \mid Y)$ 进行估计了，值得注意的是，由于 $p \in (0,1)$ ，多个小数不断的连乘会使得积越来越小，直到 `float` 无法准确存储，变成 `0.0` 。这个数值问题称为**“下溢（underflow）”**，为了避免这个问题，我们最好将 $\prod p_i$ 变成 $\exp[\sum \log(p_i)]$ ，于是预测函数可以写作如下所示：

In [9]:

```
def predict(full_name):
    names = full_name.strip().upper().split(', ')
    logp = np.zeros(2)
    for n in names:
        logp += np.log(first[:,c2i[n[0]]])
        for i in range(len(n)-1):
            logp += np.log(pairs[:,c2i[n[i]],c2i[n[i+1]]])
    return 1/(1+np.exp(logp[0]-logp[1]))
```

有了该函数我们就可以预测一个名字是不是中国人的名字了，比如：

In [10]:

```
for name in ['Jim, Rodhen','Harry, Potter','Cao, Cao','Cai, Wenji']:
    print(name,predict(name)>0.5)
```

```
('Jim, Rodhen', False)
('Harry, Potter', False)
('Cao, Cao', True)
('Cai, Wenji', True)
```

我们也可以对该分类器进行一些性能上的测试：

In [11]:

```
names = open('test.txt').readlines()
nright = 0
print('Correct?'+' '*12+'Prediction'+' '*15+'Probability\n'+' '*47+'P(Y|X)')
for name in names:
    name, chinese = name.split('; ')
    chinese = int(chinese)
    prob = predict(name)
    pred = int(prob>0.5)
    right = (pred==chinese)
    nright += int(right)
    print(' [%s]   %s is %sa Chinese%s %.4f'
          %(right, name.strip(),'NOT '*(1-pred),' '*(19-
len(name.strip())+4*pred), prob))

print(">>> test accuracy: %.1f%%"%(nright*100./len(names)))
```

```
Correct?            Prediction                Probability
                                                P(Y|X)
 [True]   Alex, Bruce is NOT a Chinese          0.0000
 [True]   Jiang, Zhemin is a Chinese            1.0000
 [True]   Xi, Jingping is a Chinese             1.0000
 [True]   Bush, George is NOT a Chinese         0.0002
 [True]   Clinton, Hilary is NOT a Chinese      0.0039
 [True]   Smith, Jim is NOT a Chinese           0.1358
 [True]   Haeyoung, Noh is NOT a Chinese        0.0115
 [True]   Patrick Price is NOT a Chinese        0.0000
 [True]   Sean, Qian is a Chinese               0.9632
 [True]   Wang, Weilong is a Chinese            0.9999
 [True]   Cheng, Zhaoqi is a Chinese            1.0000
 [True]   Tang, Jiayi is a Chinese              0.9992
 [True]   Lei, Tian is a Chinese                0.9118
 [True]   Matteo, Pozzi is NOT a Chinese        0.0014
 [True]   David, Vey is NOT a Chinese           0.0000
 [True]   Liu, Xuesong is a Chinese             1.0000
 [True]   Mario, Bergers is NOT a Chinese       0.0000
 [True]   Mao, Yisheng is a Chinese             1.0000
 [True]   Jodi, Russo is NOT a Chinese          0.0002
 [True]   Leffard, Maxine is NOT a Chinese      0.0368
>>> test accuracy: 100.0%
```

从以上结果可以看到：尽管做出若干假设和限制，该分类起在测试集上的表现还是很不错的。

4. 总结

机器学习的基础就是概率和概率推理，通过构造出一个概率分布，使得不同种类的数据在不同标签上分布不一样，从而区别不同的数据，本质看就是贝叶斯公式的应用，然而由于不同模型持有不同假设（hypothesis），比如认为线性可分 $\hat{y} = sign(w^\top x)$，或者一些条件独立等等，由于这些假设可以构建不同的分布，对目标进行估计，因此初学机器学习，第一眼就会觉得其枝繁叶茂，十分复杂。然而其从本质上看，无非是：

- 提出一个假设，构造一个分布 $P(Y \mid X; \theta)$；
- 根据该分布计算当前数据集的或然性（likelihood） $lh(\theta \mid \mathcal{D}) = \prod_{i=1}^{M} P(Y_i \mid X_i; \theta)$；
- 根据最或然估计或者凸优化对模型参数进行估计 $\hat{\theta} = \arg\max_\theta(lh(\theta))$；
- 利用该分布预测目标 $P(Y \mid X = x; \hat{\theta})$。

掌握以上要领，拿到任何一个模型，我们都有能力将其分解开来学习，而且把握它和其他模型之间的相似性。在这里，我们会发现，上文并没给出目标分布在数据集上的或然性，以及最或然估计的步骤，如果有能力，你可以试试看假定 $p(\alpha \to \beta \mid c) = \theta_{\alpha \to \beta|c}$，则其最或然估计值为：

$$\hat{\theta}_{\alpha \to \beta|c} = \frac{\sum_{i=1}^{M} \sum_{j=1}^{N-1} \mathbf{1}[C_i = c] N_{X_i = \alpha \to X_{i+1} = \beta}}{\sum_{i=1}^{M} \sum_{j=1}^{N-1} \mathbf{1}[C_i = c]}$$

后记

虽然这里讲的是分类，但是回归问题也是这样一个套路，比如最小二乘法求解线性回归：

- 提出假设 $\mathcal{H}_\theta : y \sim \mathcal{N}(w^\top x + b, \sigma^2)$；
- 计算数据集或然性 $lh(w, b \mid \mathcal{D}) = \prod_{i=1}^{M} \frac{1}{\sqrt{2\pi}\sigma} \exp[-\frac{[y_i - (w^\top x_i + b)]^2}{2\sigma^2}] \propto \exp[-\frac{\sum_{i=1}^{M} [y_i - (w^\top x_i + b)]^2}{2\sigma^2}]$；
- 最或然估计 $\hat{w}, \hat{b} = \arg\max_{w,b} lh(w, b \mid \mathcal{D}) = \arg\min_{w,b} \sum_{i=1}^{M} [y_i - (w^\top x_i)]^2$，即最小二乘估计。
- 预测目标 $\hat{y} = \hat{w}^\top x + \hat{b}$。

其原理和本文几乎一致。