Junhao Wang
jw3668
EECS E6720
HW1

#1) let X_i be the door that is selected
let Y_i be the door that is open by the host
let D_i be the door that contain the prize

Consider X_1, Y_3

$$P(D_1 \mid X_1, Y_3) = \frac{P(D_1, X_1, Y_3)}{P(X_1, Y_3)} = \frac{P(D_1, X_1, Y_3)}{P(D_1, X_1, Y_3) + P(D_2, X_1, Y_3)}$$

$$= \frac{P(Y_3 \mid D_1, X_1) \cdot P(D_1, X_1)}{P(Y_3 \mid D_1, X_1) \cdot P(D_1, X_1) + P(Y_3 \mid D_2, X_1) \cdot P(D_2, X_1)}$$

$$= \frac{\frac{1}{2} \cdot \frac{1}{3}}{\frac{1}{2} \cdot \frac{1}{3} + 1 \cdot \frac{1}{3}} = \frac{\frac{1}{6}}{\frac{3}{6}} = \frac{1}{3}$$

$\Rightarrow P(D_1^c \mid X_1, Y_3) = \frac{2}{3}$
$\Rightarrow P(D_2 \mid X_1, Y_3) = \frac{2}{3}$ since Y_3 is open.
$\Rightarrow$ He/she should switch.

#2) $\vec{\pi} \sim \text{Dirichlet}(\alpha_i)$
$\vec{x} \sim \text{multi}(\vec{\pi})$

$$P(\vec{\pi} \mid \vec{x}_{1:n}) \frac{P(\vec{x}_{1:N} \mid \vec{\pi}) \cdot P(\vec{\pi})}{\int P(\vec{x}_{1:N} \mid \vec{\pi}) \cdot P(\vec{\pi}) \, d\vec{\pi}} \propto P(\vec{x}_{1:N} \mid \vec{\pi}) \cdot P(\vec{\pi})$$

$$\Rightarrow \propto \prod_{i=1}^{N} \frac{\Gamma(\sum_{j=1}^{k} x_{ij} + 1)}{\prod_{j=1}^{k} \Gamma(x_{ij} + 1)} \prod_{j=1}^{k} \pi_j^{x_{ij}} \cdot \frac{\Gamma(\sum_{j=1}^{k} \alpha_j)}{\prod_{j=1}^{k} \Gamma(\alpha_j)} \prod_{j=1}^{k} \pi_j^{\alpha_j - 1}$$

Since we only care about terms are function form of π.

$$\Rightarrow \propto \prod_{i=1}^{N} \prod_{j=1}^{k} \pi_j^{x_{ij}} \prod_{j=1}^{k} \pi_j^{\alpha_j - 1} = \prod_{j=1}^{k} \pi_j^{\sum_{i=1}^{N} x_{ij} + \alpha_j - 1}$$

From this term we can see this can be normalized to a Dirichlet distribution. let $\sum_{i=1}^{N} x_{ij}$ denote x_j

$$\Rightarrow P(\vec{\pi} \mid \vec{x}_{1:n}) = \frac{\Gamma(\sum_{j=1}^{k} \alpha_j + x_j)}{\prod_{j=1}^{k} \Gamma(\alpha_j + x_j)} \prod_{j=1}^{k} \pi^{x_j + \alpha_j - 1}$$

The parameter of each α_j is updated to $x_j + \alpha_j$
note x_i is total # of observations that fall in category j
The new weights for each π_i is updated by x_i in the above fashion

#3) $x_i \sim \text{Poisson}(\lambda)$

$\lambda \sim \text{Gamma}(a, b)$

a) $P(\lambda | x_{1:N}) = \frac{P(x_{1:N}|\lambda) \cdot P(\lambda)}{\int P(x_{1:N}|\lambda) \cdot P(\lambda) \, d\lambda} \propto P(x_{1:N}|\lambda) \cdot P(\lambda)$

$$\propto \prod_{i=1}^{N} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!} \cdot \frac{b^a}{\Gamma(a)} \lambda^{a-1} e^{-b\lambda}$$

$$\propto \frac{\lambda^{\sum x_i} e^{-N\lambda}}{\prod_{i=1}^{N} x_i!} \cdot \frac{b^a}{\Gamma(a)} \lambda^{a-1} e^{-b\lambda}$$

$$\propto \lambda^{\sum x_i} e^{-N\lambda} \lambda^{a-1} e^{-b\lambda}$$

$$\propto \lambda^{a+\sum x_i - 1} e^{-\lambda(N+b)}$$

notice this can normalized to a Gamma distribution with a update to $a + \sum_{i=1}^{N} x_i$, b update to $b+N$.

$\Rightarrow P(\lambda | x_{1:N}) \sim \text{Gamma}(a + \sum_{i=1}^{N} x_i, b+N)$

b) $P(x^* | x_1, \dots, x_n) = \int_0^\infty P(x^* | \lambda) \cdot P(\lambda | \vec{x}_{1:N}) \, d\lambda$

$$= \int_0^\infty \frac{\lambda^{x^*} e^{-\lambda}}{x^*!} \frac{(N+b)^{\sum x_i + a}}{\Gamma(\sum x_i + a)} \lambda^{\sum x_i + a - 1} e^{-\lambda(N+b)} \, d\lambda$$

$$= \frac{1}{x^*!} \frac{(N+b)^{\sum x_i + a}}{\Gamma(\sum x_i + a)} \int_0^\infty \lambda^{x^* + \sum x_i + a - 1} e^{-\lambda(N+b+1)} \, d\lambda$$

$$= \frac{1}{x^*!} \frac{(N+b)^{\sum x_i + a}}{\Gamma(\sum x_i + a)} \frac{\Gamma(\sum x_i + x^* + a)}{(N+b+1)^{\sum x_i + x^* + a}} \int_0^\infty \frac{(N+b+1)^{\sum x_i + x^* + a}}{\Gamma(\sum x_i + x^* + a)} \cdot \lambda^{x^* + \sum x_i + a - 1} e^{-\lambda(N+b+1)} \, d\lambda$$

$$= \frac{1}{x^*!} \frac{(N+b)^{\sum x_i + a}}{\Gamma(\sum x_i + a)} \frac{\Gamma(\sum x_i + x^* + a)}{(N+b+1)^{\sum x_i + x^* + a}}$$

$$= \frac{1}{x^*!} \frac{(\sum x_i + x^* + a - 1)!}{(\sum x_i + a - 1)!} \left(\frac{N+b}{N+b+1}\right)^{\sum x_i + a} \left(\frac{1}{N+b+1}\right)^{x^*}$$

let $k = x^*$ $r = \sum x_i + a$ $p = \frac{1}{N+b+1}$

$\Rightarrow \binom{k+r-1}{k} \cdot (1-p)^r p^k$

$P(x^* | x_1, \dots, x_n) \rightarrow$ follow Negative Binomial $\left(\sum x_i + a, \frac{1}{N+b+1}\right)$

In [27]:

```
%matplotlib inline
import numpy as np
import math
from matplotlib import pyplot as plt
import pandas as pd
from scipy.stats import nbinom
from matplotlib.pyplot import figure
plt.rcParams['figure.figsize'] = [18, 5]
```

In [28]:

```
def predict(Xtest,X0,X1,gamma_pars,e,f):

    a,b =  gamma_pars
    n0 = np.shape(X0)[0]
    n1 = np.shape(X1)[0]

    logXpred0 = np.sum(nbinom.logpmf(Xtest, a + np.sum(X0[:,:-1],axis = 0), (n0 + a)
    logXpred1 = np.sum(nbinom.logpmf(Xtest, a + np.sum(X1[:,:-1],axis = 0), (n1 + a)


    y0Haty = logXpred0 + math.log((e + n0)/(n0 + n1 + e + f))
    y1Haty = logXpred1 + math.log((f + n1)/(n0 + n1 + e + f))


    return y0Haty,y1Haty
```

In [29]:

```
def calculateConfusionMtrix(Ytest,predX0,predX1,numX):
    FF = 0
    TT = 0
    FT = 0
    TF = 0
    Xlabel = np.zeros(numX)

    for i in range (numX):
        if predX0[i] < predX1[i]:
            Xlabel[i] = 1
        else:
            Xlabel[i] = 0

    for i in range(numX):
        if Xlabel[i] == Ytest[i] and Ytest[i] == 0:
            FF += 1

        if Xlabel[i] == Ytest[i] and Ytest[i] == 1:
            TT += 1

        if Xlabel[i] != Ytest[i] and Ytest[i] == 0:
            TF += 1

        if Xlabel[i] != Ytest[i] and Ytest[i] == 1:
            FT += 1


    return FF,TT,TF,FT,Xlabel
```

In [30]:

```
def confusion_Matrix(FF,FT,TF,TT):
    table = pd.DataFrame([[FF,FT],[TF,TT]])
    table.index.name = ["predict not spam","predct spam"]
    table.columns = ["actually not spam","actually spam"]
    return table

def expected_lemda(gamma_pars,X0,X1):
    a,b = gamma_pars
    n0 = np.shape(X0)[0]
    n1 = np.shape(X1)[0]
    expected_lemda0 = (a + np.sum(X0[:,:-1],axis = 0))/(b + n0)
    expected_lemda1 = (a + np.sum(X1[:,:-1],axis = 0))/(b + n1)
    return expected_lemda0,expected_lemda1

def findErrorPre(Xpre,Ytest,numX):
    errorIdx = np.zeros(0)
    for i in range(numX):
        if Xpre[i] != Ytest[i]:
            errorIdx = np.append(errorIdx,np.array([i]))

    return errorIdx

def find_smallest_3indices(Xpred0,Xpred1,numX):
    Xdiff = np.zeros(numX)
    Xdiff = np.abs(Xpred1 - Xpred0)
    Indices = Xdiff.argsort()[:3]
    return Indices
```

In [31]:

a)

```
#load data
df_Xtrain = pd.read_csv("X_train.csv")
df_Xtest = pd.read_csv("X_test.csv")
df_Ytrain = pd.read_csv("label_train.csv")
df_Ytest = pd.read_csv("label_test.csv")

#append label to X
df_Xtrain['Y'] = df_Ytrain

#select label X0 and X1
df_Xtrain0 = df_Xtrain[df_Xtrain['Y'] == 0]
df_Xtrain1 = df_Xtrain[df_Xtrain['Y'] == 1]

#put data into array
Xtrain0 = np.array(df_Xtrain0)
Xtrain1 = np.array(df_Xtrain1)
Xtest = np.array(df_Xtest)
Ytest = np.array(df_Ytest)

#compute parameters
gamma_paras = (1,1)
e = 1
f = 1
numX = np.shape(Xtest)[0]

#initialize prediction array
predX0 = np.zeros(numX)
predX1 = np.zeros(numX)

#make prediction
predX0, predX1 = predict(Xtest, Xtrain0, Xtrain1, gamma_paras,e,f)
```

In [32]:

b)

```
#b
#initialize predction count
#first index is what the prediction return
#second index is what the email actually is
Ytest.reshape(numX,)
Xlabel = np.zeros(numX)

FF, TT, TF, FT, Xlabel= calculateConfusionMtrix(Ytest,predX0,predX1,numX)
table = confusion_Matrix(FF,FT,TF,TT)
print(table)
accuracy = (FF + TT)/(numX)
print("accuracy",accuracy)
```

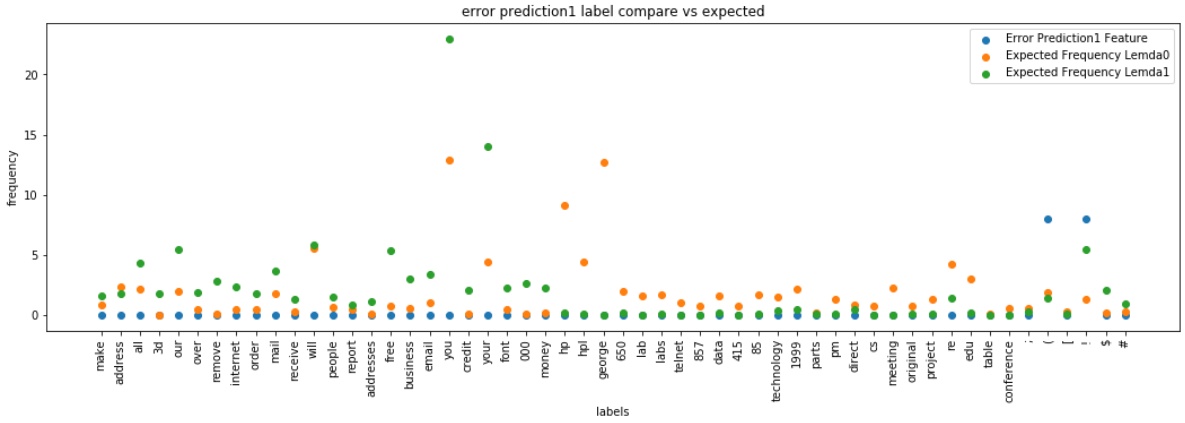
```
                                actually not spam  actually spam
[predict not spam, predct spam]
0                                             231             10
1                                              48            171
accuracy 0.8739130434782608
```

In [33]:

c)

```
#load the coloum name
df_readMe = pd.read_csv("README")
df_readMe = df_readMe.iloc[1:]

#put data in array
readMe = np.array(df_readMe)

#calculate expected lemda0 and lemda1
lemda0,lemda1 = expected_lemda(gamma_paras,Xtrain0, Xtrain1)

#find the index of error prediction
errorIdx = findErrorPre(Xlabel,Ytest,numX)
print("Error Index",errorIdx)

#pack the items we need in df frame for plotting graph latter
#useing 1st X, 24th X, 49th X
readMe = readMe.reshape(54,).tolist()
Xerror1 = Xtest[1:2:1,::1].reshape(54,).tolist()
Xerror2 = Xtest[24:25:1,::1].reshape(54,).tolist()
Xerror3 = Xtest[49:50:1,::1].reshape(54,).tolist()
df = pd.DataFrame({"lemda0":lemda0,"lemda1":lemda1,"Xerror1":\
    Xerror1,"readMe":readMe,"Xerror2":Xerror2,"Xerror3": Xerror3})

#calculate the prediction probability
error1PredP = math.exp(predX0[1])/ (math.exp(predX0[1]) + math.exp(predX1[1]))

#plot first error prediction with expected lemda0 and lemda1
fig1, ax = plt.subplots()
ax.scatter(np.arange(len(df['readMe'])), df['Xerror1'],label = "Error Prediction1 Fe
ax.scatter(np.arange(len(df['readMe'])), df['lemda0'],label = "Expected Frequency Le
ax.scatter(np.arange(len(df['readMe'])), df['lemda1'],label = "Expected Frequency Le
ax.xaxis.set_ticks(np.arange(len(df['readMe'])))
ax.xaxis.set_ticklabels(df['readMe'], rotation = 90)
plt.xlabel("labels")
plt.ylabel("frequency")
plt.title("error prediction1 label compare vs expected")
plt.legend()
plt.show()

print("nonSpam predict probability", error1PredP)
```

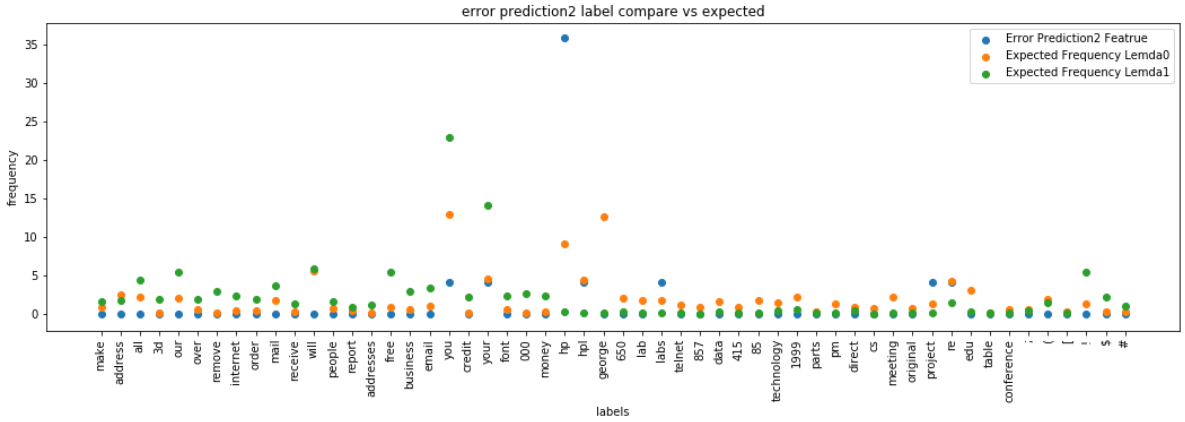
```
Error Index [  1.  24.  49.  86. 128. 153. 167. 168. 169. 170. 183. 18
9. 203. 224.
 227. 233. 236. 248. 249. 250. 257. 267. 272. 290. 303. 326. 330. 343.
 344. 369. 375. 396. 402. 406. 410. 412. 413. 414. 416. 418. 420. 421.
 422. 424. 425. 426. 429. 433. 434. 435. 436. 439. 442. 445. 447. 448.
 452. 458.]
```




```
nonSpam predict probability 0.9280563527178559
```

In [34]:

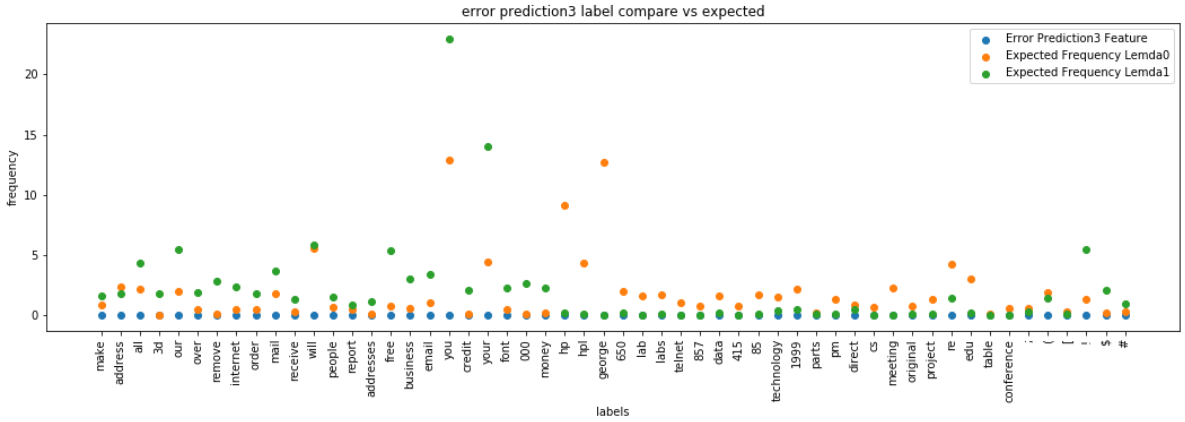
```
#plot the second error prediction with expected lemda0 and lemda1
fig2, ax = plt.subplots()
ax.scatter(np.arange(len(df['readMe'])), df['Xerror2'],label = "Error Prediction2 Fe
ax.scatter(np.arange(len(df['readMe'])), df['lemda0'],label = "Expected Frequency Le
ax.scatter(np.arange(len(df['readMe'])), df['lemda1'],label = "Expected Frequency Le
ax.xaxis.set_ticks(np.arange(len(df['readMe'])))
ax.xaxis.set_ticklabels(df['readMe'], rotation = 90)
plt.xlabel("labels")
plt.ylabel("frequency")
plt.title("error prediction2 label compare vs expected")
plt.legend()
plt.show()

#calculate the prediction probability
error1PredP = math.exp(predX0[24])/ (math.exp(predX0[24]) + math.exp(predX1[24]))
print("nonSpam predict probability", error1PredP)
```




```
nonSpam predict probability 1.0
```

In [35]:

```
#plot the thrid error prediction with expected lemda0 and lemda1
fig3, ax = plt.subplots()
ax.scatter(np.arange(len(df['readMe'])), df['Xerror3'],label ="Error Prediction3 Fea
ax.scatter(np.arange(len(df['readMe'])), df['lemda0'],label = "Expected Frequency Le
ax.scatter(np.arange(len(df['readMe'])), df['lemda1'],label = "Expected Frequency Le
ax.xaxis.set_ticks(np.arange(len(df['readMe'])))
ax.xaxis.set_ticklabels(df['readMe'], rotation = 90)
plt.xlabel("labels")
plt.ylabel("frequency")
plt.title("error prediction3 label compare vs expected")
plt.legend()
plt.show()

#calculate the prediction probability
error3PredP = math.exp(predX0[49]) / (math.exp(predX0[49]) + math.exp(predX1[49]))
print("nonSpam predict probability",error3PredP)
```




```
nonSpam predict probability 0.9999945898376463
```

In [36]:

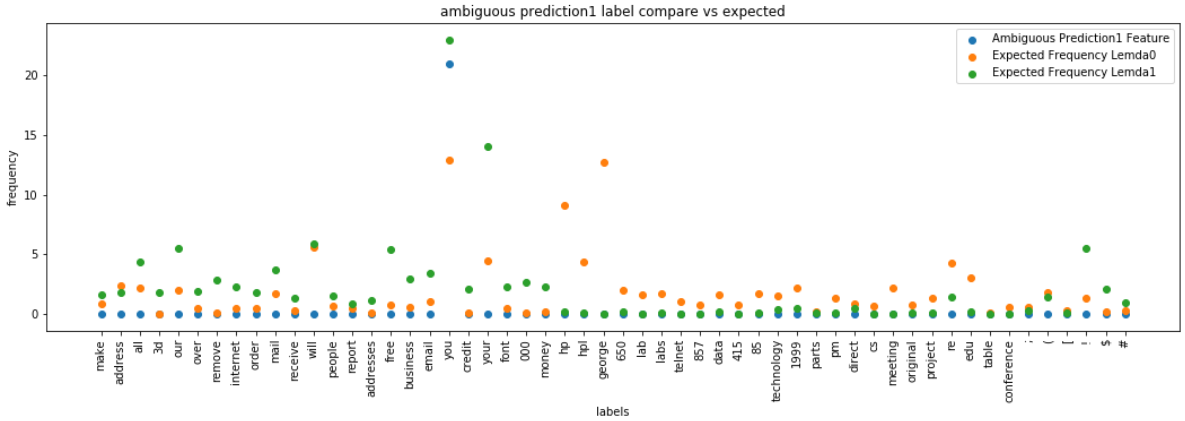
d)

```
indces = find_smallest_3indices(predX0,predX1,numX)
print("most ambigous",indces)
```

```
most ambigous [391 430 396]
```

In [37]:

```
df['Ambiguous1'] = Xtest[391:392:1,::1].reshape(54,).tolist()
df['Ambiguous2'] = Xtest[430:431:1,::1].reshape(54,).tolist()
df['Ambiguous3'] = Xtest[396:397:1,::1].reshape(54,).tolist()

#plot the first ambigous predictions label agains the expected label
fig4, ax = plt.subplots()
ax.scatter(np.arange(len(df['readMe'])),df['Ambiguous1'],label ="Ambiguous Predictic
ax.scatter(np.arange(len(df['readMe'])), df['lemda0'],label = "Expected Frequency Le
ax.scatter(np.arange(len(df['readMe'])), df['lemda1'],label = "Expected Frequency Le
ax.xaxis.set_ticks(np.arange(len(df['readMe'])))
ax.xaxis.set_ticklabels(df['readMe'], rotation = 90)
plt.xlabel("labels")
plt.ylabel("frequency")
plt.title("ambiguous prediction1 label compare vs expected")
plt.legend()
plt.show()

#calculate the prediction probability
ambig1PredP = math.exp(predX0[391])/ (math.exp(predX0[391]) + math.exp(predX1[391]))
print("nonSpam predict probability", ambig1PredP)
```

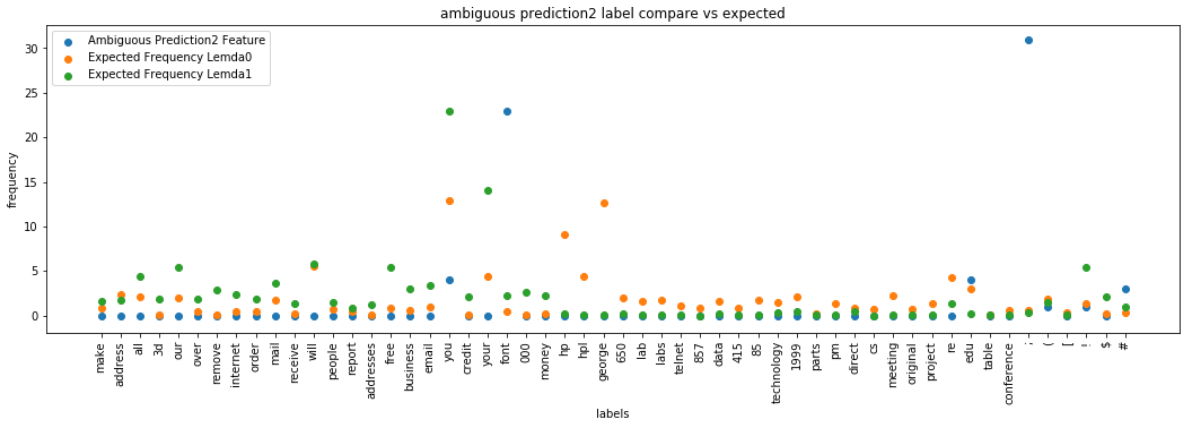


nonSpam predict probability 0.509470278181714

In [38]:

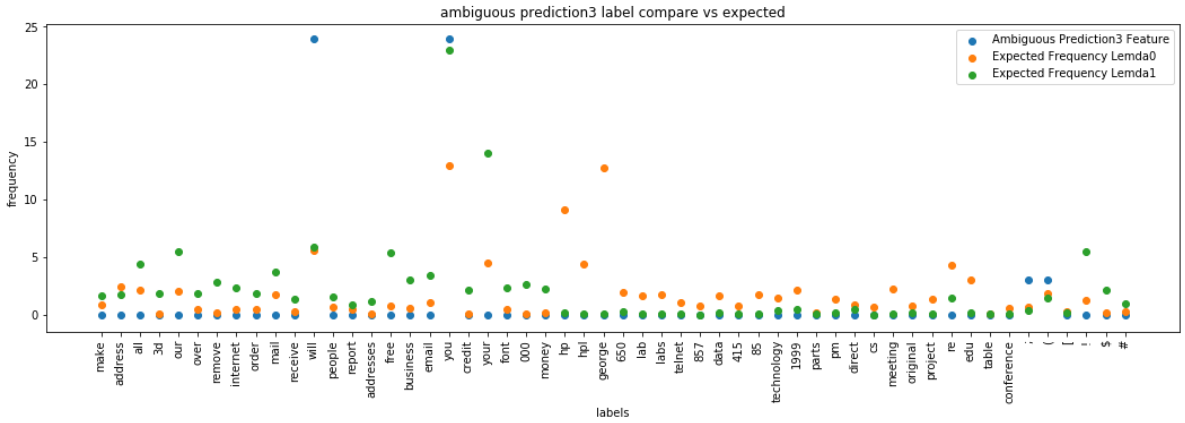
```
#plot the first ambigous predictions label agains the expected label
fig5, ax = plt.subplots()
ax.scatter(np.arange(len(df['readMe'])),df['Ambiguous2'],label ="Ambiguous Predictic
ax.scatter(np.arange(len(df['readMe'])), df['lemda0'],label = "Expected Frequency Le
ax.scatter(np.arange(len(df['readMe'])), df['lemda1'],label = "Expected Frequency Le
ax.xaxis.set_ticks(np.arange(len(df['readMe'])))
ax.xaxis.set_ticklabels(df['readMe'], rotation = 90)
plt.xlabel("labels")
plt.ylabel("frequency")
plt.title("ambiguous prediction2 label compare vs expected")
plt.legend()
plt.show

#calculate the prediction probability
ambig2PredP = math.exp(predX0[430]) / (math.exp(predX0[430]) + math.exp(predX1[430])
print("nonSpam predict probability",ambig2PredP)
```

```
nonSpam predict probability 0.511725204449384
```



In [39]:

```
fig6, ax = plt.subplots()
ax.scatter(np.arange(len(df['readMe'])),df['Ambiguous3'],label ="Ambiguous Predictic
ax.scatter(np.arange(len(df['readMe'])), df['lemda0'],label = "Expected Frequency Le
ax.scatter(np.arange(len(df['readMe'])), df['lemda1'],label = "Expected Frequency Le
ax.xaxis.set_ticks(np.arange(len(df['readMe'])))
ax.xaxis.set_ticklabels(df['readMe'], rotation = 90)
plt.xlabel("labels")
plt.ylabel("frequency")
plt.title("ambiguous prediction3 label compare vs expected")
plt.legend()
plt.show()

#calculate the prediction probabilityA
ambig3PredP = math.exp(predX0[396]) / (math.exp(predX0[396]) + math.exp(predX1[396])
print("nonSpam predict probability",ambig3PredP)
```



nonSpam predict probability 0.4859533826964165