學號：B04303006 系級： 經濟四 姓名：劉傳筠

1. (0.5%) 請比較你實作的 generative model、logistic regression 的準確率，何者較佳?

*用全部的 training data train *Logistic update 2000 times, batch size=10000

	Training data	Public	Private
generative	0.842327	0.84348	0.84363
logistic	0.850619	0.85737	0.84879

=> 用 Logistic regression 的準確率較佳（此為有做標準化的版本）

2. (0.5%) 請實作特徵標準化(feature normalization)並討論其對於你的模型準確率的影響

調整了數值為連續的欄位(age, fnlweight, capital_gain, capital_loss, hours_per_week)。

(1) without normalization

In [313]:
```
# run regression
w = train_logistic(x_train, y_train, epoch=2000, batch=10000, test_size=0.3)
np.save("weight.npy", w)
```

```
Update 100  train acc:  0.7936995436995437 test acc:  0.7964991299007064
Update 200  train acc:  0.24039136539136538 test acc:  0.2417852390213942
Update 300  train acc:  0.7794840294840295 test acc:  0.7798136963865289
Update 400  train acc:  0.7852755352755353 test acc:  0.7848295629030607
Update 500  train acc:  0.7942699192699193 test acc:  0.7964991299007064
Update 600  train acc:  0.7891804141804142 test acc:  0.7935305558399017
Update 700  train acc:  0.7843541593541593 test acc:  0.7841130105435562
Update 800  train acc:  0.3954896454896455 test acc:  0.4036237076466373
Update 900  train acc:  0.24232186732186733 test acc:  0.24321834374040333
Update 1000  train acc:  0.7892242892242892 test acc:  0.7940423789538336
Update 1100  train acc:  0.7795717795717796 test acc:  0.780427884123247
Update 1200  train acc:  0.7478939978939979 test acc:  0.7488995803050466
Update 1300  train acc:  0.24157599157599158 test acc:  0.2427065206264715
Update 1400  train acc:  0.7770709020709021 test acc:  0.7763332992117924
Update 1500  train acc:  0.7916812916812916 test acc:  0.7954754836728427
Update 1600  train acc:  0.7911547911547911 test acc:  0.7917903572525335
Update 1700  train acc:  0.7596525096525096 test acc:  0.7578053024874604
Update 1800  train acc:  0.7778167778167778 test acc:  0.7776640393080152
Update 1900  train acc:  0.7951912951912952 test acc:  0.7956802129184154
Update 2000  train acc:  0.2643032643032643 test acc:  0.26625038386733546
```

(2) with normalization

In [315]:
```
# run regression
w = train_logistic(x_train, y_train, epoch=2000, batch=10000, test_size=0.3)
np.save("weight.npy", w)
```

```
Update 100  train acc:  0.8478413478413478 test acc:  0.8472719828027434
Update 200  train acc:  0.8504299754299754 test acc:  0.8501381922407616
Update 300  train acc:  0.8501228501228502 test acc:  0.8507523799774798
Update 400  train acc:  0.8509126009126009 test acc:  0.8509571092230526
Update 500  train acc:  0.8510881010881011 test acc:  0.8513665677141979
Update 600  train acc:  0.8512636012636012 test acc:  0.8514689323369843
Update 700  train acc:  0.8514829764829764 test acc:  0.8519807554509161
Update 800  train acc:  0.8514391014391014 test acc:  0.8521854846964889
Update 900  train acc:  0.8513952263952264 test acc:  0.8522878493192753
Update 1000  train acc:  0.8513952263952264 test acc:  0.8521854846964889
Update 1100  train acc:  0.8513074763074763 test acc:  0.8519807554509161
Update 1200  train acc:  0.8510881010881011 test acc:  0.8519807554509161
Update 1300  train acc:  0.8511319761319761 test acc:  0.8518783908281298
Update 1400  train acc:  0.851044226044226 test acc:  0.8517760262053434
Update 1500  train acc:  0.8510881010881011 test acc:  0.8517760262053434
Update 1600  train acc:  0.8511319761319761 test acc:  0.8517760262053434
Update 1700  train acc:  0.8511319761319761 test acc:  0.8516736615825571
Update 1800  train acc:  0.851044226044226 test acc:  0.8516736615825571
Update 1900  train acc:  0.851044226044226 test acc:  0.8516736615825571
Update 2000  train acc:  0.851000351000351 test acc:  0.8516736615825571
```

=> 有加 normalization 之後 training accuracy 較高，準確率也不會跳來跳去

=> update 2000 之後 test acc 就一直下去了，所以後來沒有 update 那麼多次

3. (1%) 請說明你實作的 best model，其訓練方式和準確率為何?

這次實作的 model 是用 keras, 結果是 training(70% training data): 0.8659, testing(30% training data): 0.8468, private: 0.85603, and public: 0.86056

訓練方式就是試用不同的疊法還有試用不同的參數。像是加 regularization term=0.001, 每層 dropout rate = 0.5, activation 用 relu 和其他的看看之類的

但因為各種試來試去最後就沒時間去探索別種更好的 model 了...

最後用的模型長這樣

```
model = Sequential()
model.add(Dense(30, input_dim=x_train.shape[1], kernel_regularizer=regularizers.l2(0.001), init='uniform', activation='relu'))
for i in range(6):
    model.add(Dense(100, kernel_regularizer=regularizers.l2(0.001), init='uniform', activation='relu'))
    Dropout(rate=0.5, seed=0)
model.add(Dense(1, init='uniform', activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.fit(X_TRAIN, Y_TRAIN, nb_epoch=100, batch_size=100, verbose=1)
```

有另外加一項就是這個人是否有賺錢(capital_gain > captital_loss)，其他的像是是否成年、年齡的二次項等等似乎沒有太大的幫助。

ML HW2 written b04303006 劉傳鎮 ◆ date/ ◆ page/

1. $\max \prod_{n=1}^{N} \prod_{k=1}^{K} [p(x_n|C_k)\pi_k]^{t_{nk}}$

取log $\Rightarrow \max \log \prod_{n=1}^{N} \prod_{k=1}^{K} [p(x_n|C_k)\pi_k]^{t_{nk}}$

$= \max \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} [\ln P(x_n|C_k) + \ln \pi_k]$

s.t. $\sum_k \pi_k = 1$

use lagrange multiplier $\Rightarrow \ln \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} [\ln P(x_n|C_k) + \ln \pi_k] + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right)$

F.O.C with regard to π_k: $\sum_{n=1}^{N} \frac{t_{nk}}{\pi_k} + \lambda = 0.$

$-\lambda \cdot \pi_k = \sum_{n=1}^{N} t_{nk} = N_k$

$\begin{cases} t_{nk} = 1 & \text{if observation } n \text{ belongs to class } k \\ t_{nk} = 0 & \text{if } // \quad n \times \text{belongs to class } k \end{cases}$

$-\lambda \cdot \pi_1 = N_1$

$-\lambda \cdot \pi_2 = N_2$

$\vdots \quad \vdots$

+) $-\lambda \cdot \pi_k = N_k$

$-\lambda \cdot \underbrace{\sum_{i=1}^{k} \pi_i}_{=1} = N \Rightarrow \lambda = -N \Rightarrow N \cdot \pi_k = N_k \Rightarrow \pi_k = \frac{N_k}{N}$ #

2. $\Sigma = \begin{bmatrix} a & \vdots & b \\ \cdots & \sigma_{ij} & \cdots \\ c & \vdots & d \end{bmatrix}$ (i th, j th)

let $\Delta = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ (all σ_{kl}, $k \neq i$, $l \neq j$)

$\det \Sigma = \sigma_{i1} \cdot \det(\square) - \sigma_{i2} \det(\cdot) + \cdots + \sigma_{ij} \det(\Delta) + \cdots$

↳ 餘因子不會跟 σ_{ij} 有關

$\Rightarrow \frac{\partial \log(\det \Sigma)}{\partial \sigma_{ij}} = \frac{(-1)^{i+j} \det(\Delta)}{\det \Sigma} =$ 不含 j i 的 element of inverse matrix $(\Sigma^{-1}) = e_j \cdot \Sigma^{-1} \cdot e_i^T$

PAGE
DATE

3. ① $p(x|C_k) = N(x|\mu_k, \Sigma) = \frac{1}{(2\pi)^{p/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left\{\frac{-1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right\}$

$\prod_{n=1}^{N}\prod_{k=1}^{K}\left[p(x_n|C_k)\frac{N_k}{N}\right]^{t_{nk}}$

$\Rightarrow \max \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \log\left[\frac{1}{(2\pi)^{p/2}|\Sigma|^{\frac{1}{2}}} \exp\left\{\frac{-1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right\}\frac{N_k}{N}\right]$

$(*) = \max \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\left\{\log\left[\frac{1}{(2\pi)^{p/2}|\Sigma|^{\frac{1}{2}}}\frac{N_k}{N}\right] + \frac{-1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right\}$

$\frac{\partial \sum_n \sum_k \square}{\partial \mu_k} = \sum_{n=1}^{N} t_{nk}\left[\Sigma^{-1}x - \Sigma^{-1}\mu_k\right] = 0.$

展开，取和 μ_k 有关的部分

$-x^T\Sigma^{-1}\mu_k - \mu_k^T\Sigma^{-1}x + \mu_k^T\Sigma^{-1}\mu_k$

$= -2(\mu_k^T\Sigma^{-1}x) + \mu_k^T\Sigma^{-1}\mu_k$ △

$\frac{\partial \triangle}{\partial \mu_k} = -2\Sigma^{-1}x + 2\Sigma^{-1}\mu_k$

$\sum_{n=1}^{N} t_{nk}\left[x_n - \mu_k\right] = 0.$

$\sum_{n=1}^{N} t_{nk}x_n = \sum_{n=1}^{N} t_{nk}\cdot\mu_k$ → 有 N_k 个是 1

$\Rightarrow \sum_{n=1}^{N} t_{nk}\cdot x_n = N_k\cdot\mu_k.$

$\Rightarrow \mu_k = \frac{1}{N_k}\sum_{n=1}^{N} t_{nk}\cdot x_n.$

② 上题 (*): $\frac{\partial \sum_{n=1}^{N}\sum_{k=1}^{K} \square}{\partial \Sigma^{-1}}$

$\therefore t_{nk}\log\frac{1}{|\Sigma|^{\frac{1}{2}}} = \frac{\partial}{\partial \Sigma^{-1}}\left\{\sum_{n=1}^{N}\sum_{k=1}^{K}\left[t_{nk}\log\frac{1}{(2\pi)^{p/2}|\Sigma|^{\frac{1}{2}}}\frac{N_k}{N} + t_n\frac{1}{2}(x-\mu_k)^T\cdot\Sigma^{-1}(x-\mu_k)\right]\right\}$

$-\frac{1}{2}t_{nk}\log|\Sigma|^{-1} = \sum_{n=1}^{N}\sum_{k=1}^{K}\begin{bmatrix} e_1\Sigma e_1 & & \\ & e_1\Sigma e_2 & \\ & & \ddots \\ & & e_j\Sigma e_i \end{bmatrix}\cdot\frac{1}{2}t_{nk} + \frac{-1}{2}t_{nk}(x-\mu_k)(x-\mu_k)^T$ (i th, j th)

$-\frac{1}{2}t_{nk}\log|\Sigma^{-1}|$

$\therefore \frac{\partial \frac{1}{2}t_{nk}\log|\Sigma^{-1}|}{\partial \Sigma^{-1}}$ → by 第2题 $= \frac{1}{2}\Sigma^T = \Sigma$

$\frac{\partial \frac{-1}{2}t_{nk}\,tr[\sim]}{\partial \Sigma^{-1}} = \frac{\partial}{\partial \Sigma^{-1}} tr\left[(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right]$

$= tr\left[(x-\mu_k)(x-\mu_k)^T\Sigma^{-1}\right]$ (CHA SHIN symmetric)

$= \frac{-1}{2}t_{nk}(x-\mu_k)(x-\mu_k)^T$

$\frac{\partial}{\partial A}\log|A| = (A^{-1})^T$

PAGE
DATE

F.O.C. $\frac{\partial \square}{\partial \Sigma^{-1}} = 0.$

(sigma)

$$\Rightarrow N \cdot \Sigma = \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} (x_n - \mu_k)(x_n - \mu_k)^T.$$

$$\Sigma = \sum_{k=1}^{K} \frac{1}{N} \sum_{n=1}^{N} t_{nk} (x_n - \mu_k)(x_n - \mu_k)^T$$

$$= \sum_{k=1}^{K} \frac{N_k}{N} \underbrace{\frac{1}{N_k} \sum_{n=1}^{N} t_{nk} (x_n - \mu_k)(x_n - \mu_k)^T}$$

$$= \sum_{k=1}^{K} \frac{N_k}{N} S_k, \text{ where } S_k = \frac{1}{N_k} \sum_{n=1}^{N} (x_n - \mu_k)(x_n - \mu_k)^T \quad \#$$