學號：B06902125 系級：資工三 姓名：黃柏瑋

1.(2%)

請比較實作的 generative model 及 logistic regression 的準確率，何者較佳？請解釋為何有這種情況？

在兩者x和y一樣的前提下，成績如下：

	GENERATIVE MODEL	LOGISTIC REGRESSION
training	0.8745	0.8820
public test	0.8836	0.8883

由上可知，logistic regression在預測的表現上優於generative model。這樣的情形可能是因為generative model認為資料來自機率模型，為資料添了一些假設，適用於資料量少或雜訊高的題目；而此次作業提供的資料量相當充足，利用discriminative model(如logistic regression)能達到更好的分類結果。

2. (2%)

請實作 logistic regression 的正規化 (regularization)，並討論其對於你的模型準確率的影響。接著嘗試對正規項使用不同的權重 (lambda)，並討論其影響。

實作和講義一樣的L2-regularization：

(without regularization)

```
iteration = 92, loss = 0.2663, acc = 0.8858, val_loss = 0.2737, val_acc = 0.8835
iteration = 93, loss = 0.2663, acc = 0.8855, val_loss = 0.2738, val_acc = 0.8832
iteration = 94, loss = 0.2663, acc = 0.8857, val_loss = 0.2737, val_acc = 0.8837
iteration = 95, loss = 0.2663, acc = 0.8857, val_loss = 0.2738, val_acc = 0.8831
iteration = 96, loss = 0.2663, acc = 0.8858, val_loss = 0.2737, val_acc = 0.8837
iteration = 97, loss = 0.2663, acc = 0.8856, val_loss = 0.2737, val_acc = 0.8836
iteration = 98, loss = 0.2663, acc = 0.8856, val_loss = 0.2737, val_acc = 0.8833
iteration = 99, loss = 0.2663, acc = 0.8857, val_loss = 0.2737, val_acc = 0.8833
iteration = 100, loss = 0.2663, acc = 0.8857, val_loss = 0.2737, val_acc = 0.884
iteration = 101, loss = 0.2663, acc = 0.8856, val_loss = 0.2738, val_acc = 0.8837
iteration = 102, loss = 0.2663, acc = 0.8856, val_loss = 0.2737, val_acc = 0.8835
```

(with L2-regularization, λ=1e-2)

```
iteration = 92, loss = 0.266, acc = 0.8856, val_loss = 0.2733, val_acc = 0.8832
iteration = 93, loss = 0.266, acc = 0.8852, val_loss = 0.2734, val_acc = 0.883
iteration = 94, loss = 0.266, acc = 0.8856, val_loss = 0.2733, val_acc = 0.8832
iteration = 95, loss = 0.266, acc = 0.8856, val_loss = 0.2734, val_acc = 0.8833
iteration = 96, loss = 0.266, acc = 0.8856, val_loss = 0.2733, val_acc = 0.8827
iteration = 97, loss = 0.266, acc = 0.8855, val_loss = 0.2733, val_acc = 0.8833
iteration = 98, loss = 0.266, acc = 0.8858, val_loss = 0.2733, val_acc = 0.883
iteration = 99, loss = 0.266, acc = 0.8857, val_loss = 0.2734, val_acc = 0.8828
iteration = 100, loss = 0.266, acc = 0.8856, val_loss = 0.2733, val_acc = 0.8836
iteration = 101, loss = 0.2659, acc = 0.8856, val_loss = 0.2734, val_acc = 0.8831
iteration = 102, loss = 0.2659, acc = 0.8856, val_loss = 0.2734, val_acc = 0.8832
```

由於Logistic regression模型簡單，訓練的時候沒有嚴重的overfitting產生，加入regularization後對於validation data的準確率影響不大，但如果多著眼於loss的話，可以發現有加regularization的training和valid loss會稍微降低，代表regularization還是有稍微優化模型。

再嘗試一些不同的λ：

(with L2-regularization, λ=1e-1)

```
iteration = 92, loss = 0.2675, acc = 0.8845, val_loss = 0.2743, val_acc = 0.8804
iteration = 93, loss = 0.2675, acc = 0.8841, val_loss = 0.2744, val_acc = 0.8803
iteration = 94, loss = 0.2675, acc = 0.8847, val_loss = 0.2743, val_acc = 0.8809
iteration = 95, loss = 0.2675, acc = 0.8843, val_loss = 0.2743, val_acc = 0.8808
iteration = 96, loss = 0.2675, acc = 0.8847, val_loss = 0.2743, val_acc = 0.8809
iteration = 97, loss = 0.2675, acc = 0.8845, val_loss = 0.2743, val_acc = 0.8806
iteration = 98, loss = 0.2675, acc = 0.8846, val_loss = 0.2743, val_acc = 0.8809
iteration = 99, loss = 0.2675, acc = 0.8846, val_loss = 0.2743, val_acc = 0.8806
iteration = 100, loss = 0.2675, acc = 0.8844, val_loss = 0.2743, val_acc = 0.8813
iteration = 101, loss = 0.2675, acc = 0.8844, val_loss = 0.2744, val_acc = 0.8806
iteration = 102, loss = 0.2675, acc = 0.8845, val_loss = 0.2743, val_acc = 0.8806
```

當λ=1e-1時，訓練的成效不佳，出現了underfitting的現象，在training和valid的表現都比不加regularization還差。

(with L2-regularization, λ=2e-2)

```
iteration = 92, loss = 0.2659, acc = 0.8856, val_loss = 0.2732, val_acc = 0.883
iteration = 93, loss = 0.2659, acc = 0.8853, val_loss = 0.2733, val_acc = 0.8823
iteration = 94, loss = 0.2659, acc = 0.8855, val_loss = 0.2731, val_acc = 0.8829
iteration = 95, loss = 0.2659, acc = 0.8855, val_loss = 0.2732, val_acc = 0.8828
iteration = 96, loss = 0.2659, acc = 0.8856, val_loss = 0.2732, val_acc = 0.8827
iteration = 97, loss = 0.2658, acc = 0.8854, val_loss = 0.2732, val_acc = 0.8827
iteration = 98, loss = 0.2658, acc = 0.8856, val_loss = 0.2732, val_acc = 0.883
iteration = 99, loss = 0.2658, acc = 0.8854, val_loss = 0.2732, val_acc = 0.8825
iteration = 100, loss = 0.2658, acc = 0.8855, val_loss = 0.2731, val_acc = 0.8833
iteration = 101, loss = 0.2658, acc = 0.8856, val_loss = 0.2732, val_acc = 0.8824
iteration = 102, loss = 0.2658, acc = 0.8856, val_loss = 0.2732, val_acc = 0.8832
```

當λ=2e-2時，準確率和λ=1e-2時差不多，但loss方面又更降低許多，效果更好一些。

(with L2-regularization, λ=1e-3)

```
iteration = 92, loss = 0.2663, acc = 0.8855, val_loss = 0.2737, val_acc = 0.8833
iteration = 93, loss = 0.2663, acc = 0.8854, val_loss = 0.2738, val_acc = 0.8832
iteration = 94, loss = 0.2663, acc = 0.8857, val_loss = 0.2736, val_acc = 0.8837
iteration = 95, loss = 0.2663, acc = 0.8856, val_loss = 0.2737, val_acc = 0.8832
iteration = 96, loss = 0.2663, acc = 0.8857, val_loss = 0.2737, val_acc = 0.8834
iteration = 97, loss = 0.2663, acc = 0.8856, val_loss = 0.2737, val_acc = 0.8835
iteration = 98, loss = 0.2662, acc = 0.8855, val_loss = 0.2737, val_acc = 0.8833
iteration = 99, loss = 0.2662, acc = 0.8856, val_loss = 0.2737, val_acc = 0.8832
iteration = 100, loss = 0.2662, acc = 0.8855, val_loss = 0.2737, val_acc = 0.8839
iteration = 101, loss = 0.2662, acc = 0.8856, val_loss = 0.2737, val_acc = 0.8836
iteration = 102, loss = 0.2662, acc = 0.8856, val_loss = 0.2737, val_acc = 0.8833
```

當λ=1e-3時，regularization對模型的penalty有點太小，成效不彰，結果和沒有加regularization的差不多。

總而言之，regularization在validation準確率上影響不算大，而λ在1e-2附近的效果會比較好。

3. (1%)

請說明你實作的 best model，其訓練方式和準確率為何？

我的best model主要是建立在basic的logistic regression之上，僅對input做了feature engineering而已。

首先先把training data中完全沒出現過的feature移除："other rel < 18 ever marr not in subfamily"和"grandchild < 18 never marr rp of subfamily"。

接著對連續數值的feature進行分群(binning)，分群原則是將鄰近且分布相近的區間歸在同一群，舉"age"為例：0到10歲與10到18歲大約都只有0.5%以下的人年薪大於50000，而18到25歲大約有2%，因此"age"的第一群便由0到18歲的人組成。
分群結果如下：

FEATURE	BIN BOUNDARIES (A, B]
age	-1, 18, 25, 35, 45, 55, 65, 75, np.inf

FEATURE	BIN BOUNDARIES (A, B]
capital gains	-np.inf, 4600, 7600, 15000, np.inf
capital losses	-np.inf, 1400, 2000, 2200, 3200, np.inf
dividends	-np.inf, 0, 5000, np.inf
num persons worked for employer	-1, 0, 1, 2, 3, 4, 5, 6
working weeks	-np.inf, 25, 45, np.inf
wage per hour	-np.inf, 0, 1200, 1800, 2200, np.inf

此外，我也用Keras架設Deep Neural Network，試了一些不同的架構，而最後用了以下的模型架構：

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 388)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                24896
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               8320
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 129
=================================================================
Total params: 33,345
Trainable params: 33,345
Non-trainable params: 0
_________________________________________________________________
```

接著我利用logistic regression with L2-regularization挑掉權重小於0.1的feature，又因為data中label=0比label=1的多，於是在訓練模型時給上class weight(1:1.2)，平衡在計算loss時倒向label=0的窘境。

以下表格為逐步優化模型時的準確率：

MODEL	VALID ACC	PUBLIC TEST ACC
Logistic Regression with feature binning	0.8869	0.8943
DNN with feature binning	0.8879	0.8953
DNN with feature binning, selection and class weight	0.8884	0.8958

4. (1%)

請實作輸入特徵標準化 (feature normalization)，並比較是否應用此技巧，會對於你的模型有何影響。

在logistic regression的basic models中，若不用normalization會使訓練過程中的loss及accuracy不穩定地跳動，也不好收斂(如下圖所示)；若使用standardization或min-max normalization可以改善這個問題，其中standardization的表現較佳。

(without normalization)

```
iteration = 22, loss = 3.6805, acc = 0.7891, val_loss = 3.7655, val_acc = 0.7864
iteration = 23, loss = 4.2549, acc = 0.7728, val_loss = 4.3194, val_acc = 0.7716
iteration = 24, loss = 3.3811, acc = 0.7785, val_loss = 3.5165, val_acc = 0.7756
iteration = 25, loss = 5.1539, acc = 0.7226, val_loss = 5.1639, val_acc = 0.723
iteration = 26, loss = 4.1486, acc = 0.7734, val_loss = 4.1946, val_acc = 0.7735
iteration = 27, loss = 4.312, acc = 0.782, val_loss = 4.4304, val_acc = 0.7796
iteration = 28, loss = 3.4342, acc = 0.7689, val_loss = 3.4807, val_acc = 0.7684
iteration = 29, loss = 3.7935, acc = 0.7774, val_loss = 3.8989, val_acc = 0.7751
iteration = 30, loss = 4.0558, acc = 0.7437, val_loss = 4.1448, val_acc = 0.7437
iteration = 31, loss = 4.2159, acc = 0.7799, val_loss = 4.2661, val_acc = 0.7777
iteration = 32, loss = 4.1257, acc = 0.7797, val_loss = 4.2003, val_acc = 0.7764
iteration = 33, loss = 4.0599, acc = 0.7842, val_loss = 4.1502, val_acc = 0.7812
iteration = 34, loss = 3.5409, acc = 0.7786, val_loss = 3.6568, val_acc = 0.7758
iteration = 35, loss = 3.8929, acc = 0.767, val_loss = 3.9085, val_acc = 0.7658
iteration = 36, loss = 4.1794, acc = 0.762, val_loss = 4.2557, val_acc = 0.7613
iteration = 37, loss = 3.7247, acc = 0.7855, val_loss = 3.7238, val_acc = 0.7873
iteration = 38, loss = 3.8828, acc = 0.7829, val_loss = 3.9975, val_acc = 0.7795
iteration = 39, loss = 3.9414, acc = 0.7635, val_loss = 3.9676, val_acc = 0.7626
```

(min-max)

```
iteration = 22, loss = 0.3261, acc = 0.8567, val_loss = 0.3301, val_acc = 0.8526
iteration = 23, loss = 0.3251, acc = 0.8561, val_loss = 0.3291, val_acc = 0.8516
iteration = 24, loss = 0.3242, acc = 0.8566, val_loss = 0.3281, val_acc = 0.8522
iteration = 25, loss = 0.3234, acc = 0.8569, val_loss = 0.3273, val_acc = 0.8531
iteration = 26, loss = 0.3224, acc = 0.8581, val_loss = 0.3262, val_acc = 0.8533
iteration = 27, loss = 0.3216, acc = 0.858, val_loss = 0.3254, val_acc = 0.8539
iteration = 28, loss = 0.3209, acc = 0.8583, val_loss = 0.3246, val_acc = 0.8537
iteration = 29, loss = 0.3201, acc = 0.8587, val_loss = 0.3239, val_acc = 0.8538
iteration = 30, loss = 0.3195, acc = 0.859, val_loss = 0.3231, val_acc = 0.8546
iteration = 31, loss = 0.3188, acc = 0.8592, val_loss = 0.3224, val_acc = 0.8539
iteration = 32, loss = 0.3183, acc = 0.8594, val_loss = 0.3219, val_acc = 0.8544
iteration = 33, loss = 0.3176, acc = 0.8599, val_loss = 0.3212, val_acc = 0.855
iteration = 34, loss = 0.317, acc = 0.8599, val_loss = 0.3206, val_acc = 0.855
iteration = 35, loss = 0.3165, acc = 0.8604, val_loss = 0.32, val_acc = 0.8555
iteration = 36, loss = 0.3159, acc = 0.8605, val_loss = 0.3194, val_acc = 0.8563
iteration = 37, loss = 0.3154, acc = 0.8609, val_loss = 0.3189, val_acc = 0.8561
iteration = 38, loss = 0.3149, acc = 0.8609, val_loss = 0.3184, val_acc = 0.8565
iteration = 39, loss = 0.3144, acc = 0.8612, val_loss = 0.3179, val_acc = 0.8562
```

(standardization)

```
iteration = 22, loss = 0.2721, acc = 0.8858, val_loss = 0.2882, val_acc = 0.8788
iteration = 23, loss = 0.2717, acc = 0.8855, val_loss = 0.2876, val_acc = 0.878
iteration = 24, loss = 0.2713, acc = 0.8856, val_loss = 0.287, val_acc = 0.8788
iteration = 25, loss = 0.271, acc = 0.8854, val_loss = 0.2867, val_acc = 0.8783
iteration = 26, loss = 0.2706, acc = 0.8854, val_loss = 0.2866, val_acc = 0.8784
iteration = 27, loss = 0.27, acc = 0.8857, val_loss = 0.2863, val_acc = 0.8785
iteration = 28, loss = 0.2694, acc = 0.8857, val_loss = 0.286, val_acc = 0.8785
iteration = 29, loss = 0.2687, acc = 0.886, val_loss = 0.2858, val_acc = 0.8786
iteration = 30, loss = 0.268, acc = 0.8858, val_loss = 0.2857, val_acc = 0.8785
iteration = 31, loss = 0.2674, acc = 0.8859, val_loss = 0.2855, val_acc = 0.8785
iteration = 32, loss = 0.267, acc = 0.8858, val_loss = 0.2852, val_acc = 0.8786
iteration = 33, loss = 0.2667, acc = 0.8859, val_loss = 0.2847, val_acc = 0.8789
iteration = 34, loss = 0.2665, acc = 0.8862, val_loss = 0.2844, val_acc = 0.8793
iteration = 35, loss = 0.2664, acc = 0.8863, val_loss = 0.2841, val_acc = 0.8776
iteration = 36, loss = 0.2663, acc = 0.8863, val_loss = 0.2837, val_acc = 0.8785
iteration = 37, loss = 0.2661, acc = 0.8865, val_loss = 0.2834, val_acc = 0.8791
iteration = 38, loss = 0.266, acc = 0.8864, val_loss = 0.2831, val_acc = 0.879
iteration = 39, loss = 0.2659, acc = 0.8866, val_loss = 0.2828, val_acc = 0.8785
```

在best model中，由於所有的feature都是one-hot形式，因此沒有使用nomalization(也可以說使用min-max)的效果最好。

(min-max)

```
iteration = 22, loss = 0.2658, acc = 0.8862, val_loss = 0.2713, val_acc = 0.8818
iteration = 23, loss = 0.2655, acc = 0.8864, val_loss = 0.2711, val_acc = 0.8817
iteration = 24, loss = 0.2652, acc = 0.8866, val_loss = 0.2708, val_acc = 0.8816
iteration = 25, loss = 0.265, acc = 0.8868, val_loss = 0.2706, val_acc = 0.8817
iteration = 26, loss = 0.2647, acc = 0.8869, val_loss = 0.2704, val_acc = 0.8825
iteration = 27, loss = 0.2645, acc = 0.8872, val_loss = 0.2702, val_acc = 0.882
iteration = 28, loss = 0.2642, acc = 0.8872, val_loss = 0.27, val_acc = 0.882
iteration = 29, loss = 0.264, acc = 0.8875, val_loss = 0.2699, val_acc = 0.882
iteration = 30, loss = 0.2638, acc = 0.8875, val_loss = 0.2697, val_acc = 0.8822
iteration = 31, loss = 0.2636, acc = 0.8876, val_loss = 0.2696, val_acc = 0.8825
iteration = 32, loss = 0.2636, acc = 0.8878, val_loss = 0.2695, val_acc = 0.8825
iteration = 33, loss = 0.2633, acc = 0.8878, val_loss = 0.2694, val_acc = 0.8825
iteration = 34, loss = 0.2632, acc = 0.8881, val_loss = 0.2693, val_acc = 0.8826
iteration = 35, loss = 0.263, acc = 0.888, val_loss = 0.2692, val_acc = 0.8824
iteration = 36, loss = 0.2629, acc = 0.8882, val_loss = 0.2691, val_acc = 0.8824
iteration = 37, loss = 0.2628, acc = 0.8882, val_loss = 0.2689, val_acc = 0.8824
iteration = 38, loss = 0.2626, acc = 0.8881, val_loss = 0.2689, val_acc = 0.8824
iteration = 39, loss = 0.2625, acc = 0.8884, val_loss = 0.2688, val_acc = 0.8822
```

(standardization)

```
iteration = 22, loss = 0.4059, acc = 0.8752, val_loss = 0.4321, val_acc = 0.8664
iteration = 23, loss = 0.3981, acc = 0.8763, val_loss = 0.4243, val_acc = 0.8669
iteration = 24, loss = 0.3911, acc = 0.8775, val_loss = 0.4171, val_acc = 0.8681
iteration = 25, loss = 0.3849, acc = 0.878, val_loss = 0.4109, val_acc = 0.8696
iteration = 26, loss = 0.3792, acc = 0.8789, val_loss = 0.4051, val_acc = 0.8709
iteration = 27, loss = 0.3741, acc = 0.8793, val_loss = 0.3997, val_acc = 0.8719
iteration = 28, loss = 0.3692, acc = 0.8795, val_loss = 0.3944, val_acc = 0.8721
iteration = 29, loss = 0.3646, acc = 0.8801, val_loss = 0.3895, val_acc = 0.8726
iteration = 30, loss = 0.3602, acc = 0.8804, val_loss = 0.3846, val_acc = 0.8728
iteration = 31, loss = 0.3561, acc = 0.8806, val_loss = 0.3798, val_acc = 0.8733
iteration = 32, loss = 0.3522, acc = 0.881, val_loss = 0.3752, val_acc = 0.8734
iteration = 33, loss = 0.3485, acc = 0.8813, val_loss = 0.3708, val_acc = 0.8744
iteration = 34, loss = 0.3451, acc = 0.8818, val_loss = 0.3667, val_acc = 0.8752
iteration = 35, loss = 0.3419, acc = 0.8819, val_loss = 0.3628, val_acc = 0.8751
iteration = 36, loss = 0.3388, acc = 0.8821, val_loss = 0.3591, val_acc = 0.8757
iteration = 37, loss = 0.336, acc = 0.8825, val_loss = 0.3557, val_acc = 0.876
iteration = 38, loss = 0.3334, acc = 0.8828, val_loss = 0.3525, val_acc = 0.8758
iteration = 39, loss = 0.331, acc = 0.8832, val_loss = 0.3495, val_acc = 0.8758
```