```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np


# In[2]:


train_x = np.load("hw2p2_train_x.npy")
train_y = np.load("hw2p2_train_y.npy")


# In[3]:


train_x.shape


# In[7]:


train_x.shape[0]


# In[4]:


train_x.dtype


# In[5]:


train_y.shape


# In[6]:


train_y.dtype


# In[57]:


# (c)(i) Use the training data to estimate log pkj for each class k = 0, 1 and for each feature
# j = 1, . . . , d.
log_pkj = np.zeros([2, train_x.shape[1]])
label_feature_words = np.zeros([2, train_x.shape[1]])
for i in range(train_x.shape[0]):
    for j in range(train_x.shape[1]):
        feature_i_j = train_x[i,j]
        label_i = train_y[i]
        if label_i == 0:
            label_feature_words[0,j] += feature_i_j
        if label_i == 1:
            label_feature_words[1,j] += feature_i_j
label_total_words = [sum(label_feature_words[0]),sum(label_feature_words[1])]


# In[58]:
```

```python
for i in range(label_feature_words.shape[0]):
    for j in range(label_feature_words.shape[1]):
        log_pkj_value =
np.log((label_feature_words[i,j]+1)/(label_total_words[i]+label_feature_words.shape[1]))
        log_pkj[i,j] = log_pkj_value


# In[59]:


log_pkj


# In[44]:


log_pkj.shape


# In[60]:


# (c)(ii) Also estimate the log-priors log πk for each class.
label_total_number = [0,0]
for i in range(train_y.shape[0]):
    if train_y[i] == 0:
        label_total_number[0] += 1
    if train_y[i] == 1:
        label_total_number[1] += 1
log_πk =
[np.log(label_total_number[0]/train_y.shape[0]),np.log(label_total_number[1]/train_y.shape[0])]



# In[61]:


log_πk


# In[ ]:





# In[31]:


test_x = np.load("hw2p2_test_x.npy")
test_y = np.load("hw2p2_test_y.npy")


# In[36]:


test_y.shape


# In[62]:


# (d)Use the estimates for log pkj and log πk to predict labels for the testing data. That is,
# apply the decision rule with the samples in test x.npy and test y.npy. Report the
# test error.

pred_labels = []
```

```
for i in range(test_x.shape[0]):
    log_prob_class_0 = log_πk[0]
    log_prob_class_1 = log_πk[1]
    for j in range(test_x.shape[1]):
        xj = test_x[i,j]
        log_p0j_value = log_pkj[0,j]
        log_p1j_value = log_pkj[1,j]
        log_prob_class_0 += xj*log_p0j_value
        log_prob_class_1 += xj*log_p1j_value
    if log_prob_class_0 > log_prob_class_1:
        pred_label = 0
    else:
        pred_label = 1
    pred_labels.append(pred_label)


# In[63]:


len(pred_labels)


# In[64]:


misclassified = 0
for i in range(test_y.shape[0]):
    if pred_labels[i] != test_y[i]:
        misclassified += 1
misclassified


# In[65]:


# The test error is 0.1259
misclassified/test_x.shape[0]


# In[39]:


# (e) What would the test error be if we always predicted the same class, namely, the majority
# class from the training data? (Use this result as a sanity check for the error in part (d))
sum(train_y)
# We can see that there are 598 training data with label 1 and 594 training data with label 0
# label 1 is the majority class in the traing data


# In[40]:


train_x.shape[0]-sum(train_y)


# In[41]:


sum(test_y)
# We can see that there are 398 training data with label 1 and 396 training data with label 0


# In[42]:


test_x.shape[0]-sum(test_y)
```

```
# In[43]:


# If we always predict class 1, the test error would be
(test_x.shape[0]-sum(test_y))/test_x.shape[0]


# In[ ]:
```