Intro to Machine Learning

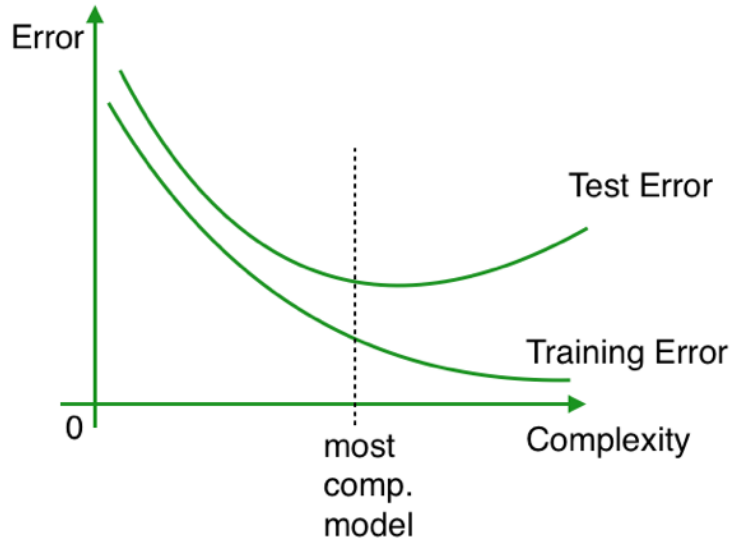
Assignment 2

Student Name: Song Han
Instructor Name: Graham Taylor

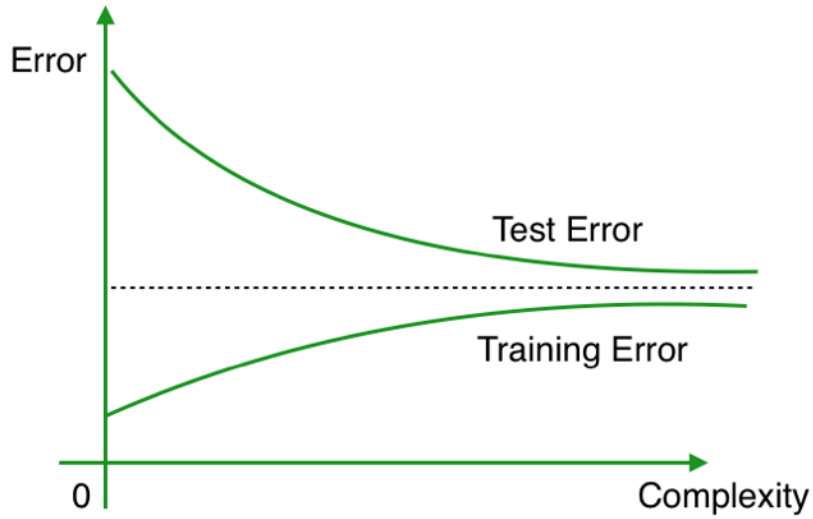
1. Theory(Discuss with Jingjing Wang)

Part 1:

(a)



(b)



Part 2:

	Dataset 1	Dataset 2
KNN (K=1)	Works very well! Since the data in the same class are very close. And there are no noise in each class data. So whenever we want to classify a new object, its nearest 5 data will belongs to the same class. Thus, K=5 works well.	Work poor. K = 1 is very complex in this dataset since there are four noise data in each class and boundary will separate every noise from each class. When we classify a new object, maybe it can be classify as a noise even it is not since the new object may very close to the noise.
KNN (K=5)	Works very well! Since the data in the same class are very close. And there are no noise in each class data. So whenever we want to classify a new object, its nearest 5 data will belongs to the same class. Thus, K=5 works well.	Work well. From the dataset, we can see that each class only mix 4 noise data. Since K=5 now, if the new object is close to a noise, there still another 4 non-noise data around it. So the new object won't be classified as a noise.
Logistic Regression	Cannot separate two classes in this Dataset since the spread of these two classes data("。" and "+") cannot separate by one line.	Work well. Only few noise in each class, so Logistic Regression still works.

Part 3:

(a)

$$\text{Bayes Rule: } P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

$$\text{Posterior: } P(y_0|\vec{x}) = P(Y = y_0|X = \vec{x}) = \frac{P(X=\vec{x}|Y=y_0)P(Y=y_0)}{P(X=\vec{x})}$$

(b)

$$P(y_0|\vec{x}) = \frac{P(X=\vec{x}|Y=y_0)P(Y=y_0)}{P(X=\vec{x})} = \frac{P(X=\vec{x}|Y=y_0)P(Y=y_0)}{P(X=\vec{x}|Y=y_0)P(Y=y_0)+P(X=\vec{x}|Y=y_1)P(Y=y_1)}$$

$$= \frac{\frac{P(X=\vec{x}|Y=y_0)P(Y=y_0)}{P(X=\vec{x}|Y=y_0)P(Y=y_0)}}{\frac{P(X=\vec{x}|Y=y_0)P(Y=y_0)}{P(X=\vec{x}|Y=y_0)P(Y=y_0)} + \frac{P(X=\vec{x}|Y=y_1)P(Y=y_1)}{P(X=\vec{x}|Y=y_0)P(Y=y_0)}}$$

$$= \frac{1}{1+\frac{P(X=\vec{x}|Y=y_1)P(Y=y_1)}{P(X=\vec{x}|Y=y_0)P(Y=y_0)}} = \frac{1}{1+exp[ln(\frac{P(X=\vec{x}|Y=y_1)P(Y=y_1)}{P(X=\vec{x}|Y=y_0)P(Y=y_0)})]}$$

$$= \frac{1}{1+exp[-ln(\frac{P(X=\vec{x}|Y=y_0)P(Y=y_0)}{P(X=\vec{x}|Y=y_1)P(Y=y_1)})]}$$

$$= \frac{1}{1+exp(-z)} \quad where\ z = ln(\frac{P(X=\vec{x}|Y=y_0)P(Y=y_0)}{P(X=\vec{x}|Y=y_1)P(Y=y_1)})$$

(c)

By part b)

$P(y_0|\vec{x}) = \frac{1}{1+exp(-z)}$ $where$ $z = ln(\frac{P(X=\vec{x}|Y=y_0)P(Y=y_0)}{P(X=\vec{x}|Y=y_1)P(Y=y_1)})$

$z = ln(\frac{P(X=\vec{x}|Y=y_0)P(Y=y_0)}{P(X=\vec{x}|Y=y_1)P(Y=y_1)})$

$= ln[P(X=\vec{x}|Y=y_0)] + ln[P(Y=y_0)] - ln[P(X=\vec{x}|Y=y_1)] - ln[P(Y=y_1)]$

$= ln[\frac{P(X=\vec{x}|Y=y_0)}{P(X=\vec{x}|Y=y_1)}] + ln[\frac{P(Y=y_0)}{P(Y=y_1)}]$

$= ln[\frac{\frac{1}{(2\pi)^{\frac{d}{2}}|\sum|^{\frac{1}{2}}} e^{-\frac{1}{2}(X-\mu_0)^T\sum^{-1}(X-\mu_0)}}{\frac{1}{(2\pi)^{\frac{d}{2}}|\sum|^{\frac{1}{2}}} e^{-\frac{1}{2}(X-\mu_1)^T\sum^{-1}(X-\mu_1)}}] + ln[\frac{P(Y=y_0)}{P(Y=y_1)}]$

$where$ $-\frac{1}{2}(X-\mu_i)^T\sum^{-1}(X-\mu_i) = -\frac{1}{2}(X^T\Sigma^{-1}X - \mu_i^T\Sigma X - X^T\Sigma^{-1}\mu_i + \mu_i^T\Sigma^{-1}\mu_i)$

$= -\frac{1}{2}(X^T\Sigma^{-1}X) - \frac{1}{2}(-\mu_i^T\Sigma^{-1}X - X^T\Sigma^{-1}\mu_i + \mu_i^T\Sigma^{-1}\mu_i)$

$\therefore z = ln[\frac{e^{-\frac{1}{2}(X^T\Sigma^{-1}X)}e^{-\frac{1}{2}(-\mu_0^T\Sigma^{-1}X - X^T\Sigma^{-1}\mu_0 + \mu_0^T\Sigma^{-1}\mu_0)}}{e^{-\frac{1}{2}(X^T\Sigma^{-1}X)}e^{-\frac{1}{2}(-\mu_1^T\Sigma^{-1}X - X^T\Sigma^{-1}\mu_1 + \mu_1^T\Sigma^{-1}\mu_1)}}] + ln[\frac{P(Y=y_0)}{P(Y=y_1)}]$

$= ln[e^{-\frac{1}{2}(-\mu_0^T\Sigma^{-1}X - X^T\Sigma^{-1}\mu_0 + \mu_0^T\Sigma^{-1}\mu_0 + \mu_1^T\Sigma^{-1}X + X^T\Sigma^{-1}\mu_1 - \mu_1^T\Sigma^{-1}\mu_1)}] + ln[\frac{P(Y=y_0)}{P(Y=y_1)}]$

$\because X^T \in \mathbb{R}^{1\times d}, X \in \mathbb{R}^d, \Sigma and \Sigma^{-1} \in \mathbb{R}^{d\times d}, \mu_i \in \mathbb{R}^d$

$\therefore \mu_0^T\Sigma^{-1}X \in \mathbb{R}^d, X^T\Sigma^{-1}\mu_0 \in \mathbb{R}^d$

$\therefore z = ln[e^{-\frac{1}{2}(-2\mu_0^T\Sigma^{-1}X - 2\mu_1^T\Sigma^{-1}X + \mu_0^T\Sigma^{-1}\mu_0 - \mu_1^T\Sigma^{-1}\mu_1)}] + ln[\frac{P(Y=y_0)}{P(Y=y_1)}]$

$= ln[e^{\mu_0^T\Sigma^{-1}X - \mu_1^T\Sigma^{-1}X - \frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0 + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1}] + ln[\frac{P(Y=y_0)}{P(Y=y_1)}]$

$= (\mu_0^T\Sigma^{-1} - \mu_1^T\Sigma^{-1})X - \frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0 + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + ln[\frac{P(Y=y_0)}{P(Y=y_1)}]$

$\therefore \vec{w} = \mu_0^T\Sigma^{-1} - \mu_1^T\Sigma^{-1}$

$b = -\frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0 + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + ln[\frac{P(Y=y_0)}{P(Y=y_1)}]$

2. Practice (Discuss with Jingling Wang)

Part 1:

1. By doing the main check, if statement can be run directly. Firstly, some variables are defined in this if statement.
2. Then set d equals function “read_evergreen_csv”.
   When “read_evergreen_csv” being called, “_read_csv” will be called as well since “_read_csv” is inside “read_evergreen_csv”
   Function”_read_csv” will first open training file and manipulate data. This is function returns four things: X_list is a list, with each element inside X-list is a list of floats which are the numerical features (from “alchemy_category_score” to “label”) for each corresponding row. y_list is the

label of training data. Text contains all the words in boilerplate but eliminate the punctuation. Categories is a list of all samples' categories.

3. After calling "_read_csv", category dictionary will be made and then convert each sample's category to a binary list, "one-hot", with 1 stands for the sample belongs to that category and 0 is not. Then append this binary list to X-list.
4. Vectorizer texts and transform to a sparse matrix.
5. Function "read_evergreen_csv"return a dictionary "ret" so that training data and test data that have been manipulated above are more manageable.
6. Split data into training, validation and test sets.
7. Normalize the data and write "data", "bag_of_words" and "word_names" into their corresponding pickle files.

(2).

What do the discovered topics tell you about the types of web pages in the training data?

Each discovered topic contains 15 tops words with the probability of that topic generated each word according to all the training data that Kaggle provided. By reading the top words for each topic, we can have a general idea what is that topic about. For example, from the topics that my computer generated, I saw there is a topic with most of its 15 top words are related food and health, then I can predict that there are some web pages are related to food or how to eat healthy.

Since the "num_topics" defined in "train_lda.py" is 30, then LDA model will generate 30 topics depending on the documents provided. By increasing or decreasing the number of topics, more or less topics can be generated; in another word, more topics generated, more specially classified for documents.

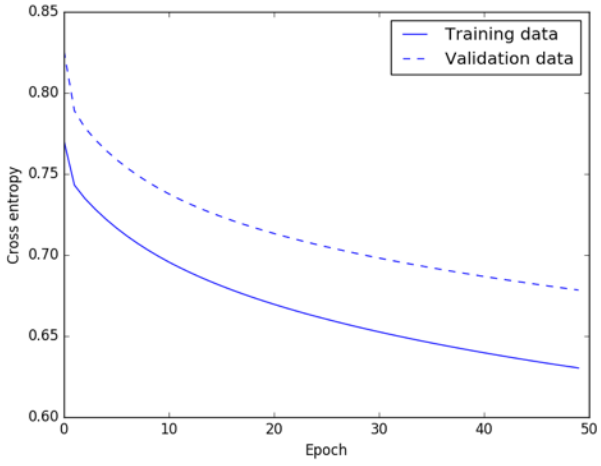
In addition, each web page may belong to more than one topics that we generated since different topics may share same top words which appear in the web page.

Do any topics stand out?

No, Since we can only get top 15 words for each topic. In addition, these topics are just random generated by LDA model. If we set number of topics a larger number, then LDA model will random generate more topics. However, we don't know whether there is any topic stand out/

(3).

***Default value*:** epochs=50, learning_rate = 0.01, n_hidden = 200, batch_size = 500



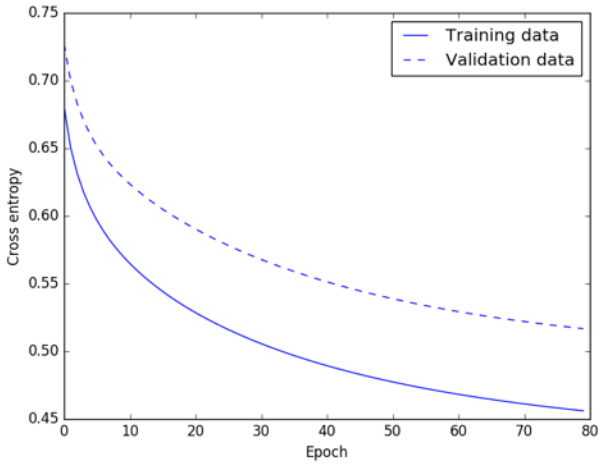
Results:

```
train accuracy: 0.6510
train cross entropy: 0.6303
validation accuracy: 0.6253
validation cross entropy:
0.6784
```

```
training took: 12.8868479729 sec
```

Comment: When using default values, we can see that cross entropy do not converge and accuracy is not that good. But it takes a short time to run. So I will try to increase epochs and decrease batch_size to increase the accuracy but at the same time, I predict that running time will raise as well.

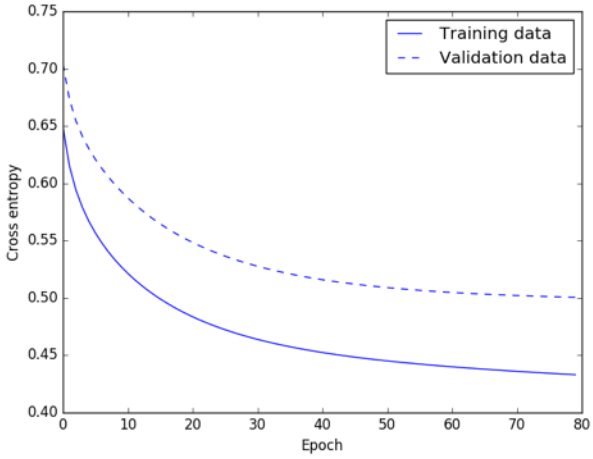
Try: epochs=80, learning_rate = 0.01, n_hidden = 200, batch_size = 30



Result:
train accuracy: 0.7963
train cross entropy: 0.4561
validation accuracy: 0.7494
validation cross entropy: 0.5168
training took: 61.7885961533 sec

Comment: cross entropy converges, but running time increase a lot, and my validation accuracy increase to 74.94%. I think my epoch is ok now since larger epochs takes a longer time. So I try to manipulate other hyper-paramters' value to increase accuracy but decrease training time. Next time I will try larger batch_size and larger learning rate in decrease my running time and hope my accuracy won't drop too much.

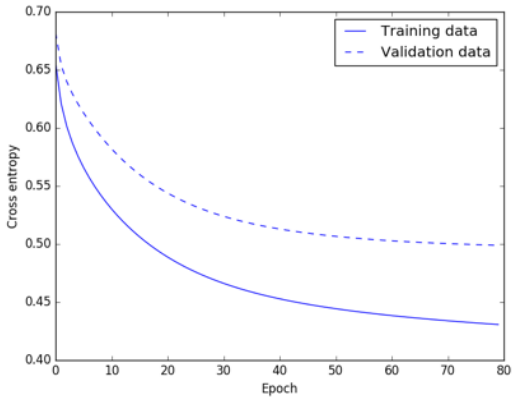
Try: epochs=80, learning_rate = 0.03, n_hidden = 200, batch_size = 40



Result:
train accuracy: 0.8109
train cross entropy: 0.4331
validation accuracy: 0.7595
validation cross entropy: 0.5006
training took: 47.1236009598 sec

Comment: It works really good! training time decrease a lot and accuracy increase as well. Next time try less number of hidden layer and see how it will work.

Try: epochs=80, learning_rate = 0.03, n_hidden = 160, batch_size = 40



Result:
train accuracy: 0.8109
train cross entropy: 0.4307
validation accuracy: 0.7696
validation cross entropy: 0.4989
training took: 47.4107480049 sec

Comment: Training time decrease and my accuracy increase 1% and time increase 0.3s.

Do the additional features help classification accuracy? Why or why not?
The additional features DO help classification accuracy. By adding these additional features, my validation accuracy can reach 76.96%. Then I commanded code which append the LDA topic features to my training data. I use the same hyper-parameters to run my model, then my validation accuracy decrease to 61.01%.

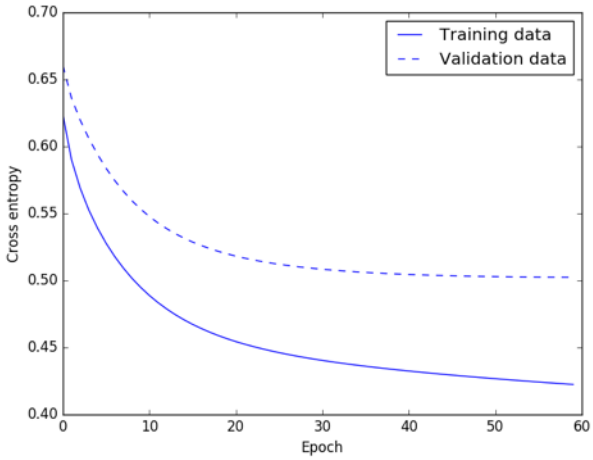
```
    # append the lda topic features
#    with open('lda_features.pkl') as f:
#        train_w, val_w, test_w = pickle.load(f)
#    train_X = np.concatenate((train_X, train_w), axis=1)
#    val_X   = np.concatenate((val_X, val_w), axis=1)
#    test_X  = np.concatenate((test_X, test_w), axis=1)
```

command "append the LDA topic features

```
After training, epochs:100, n_hidden: 160, learning_rate: 0.03
train accuracy: 0.7053
train cross entropy: 0.5660
validation accuracy: 0.6101
validation cross entropy: 0.6498
training took: 56.5484261513 sec
```

Results WITHOUT topic features.

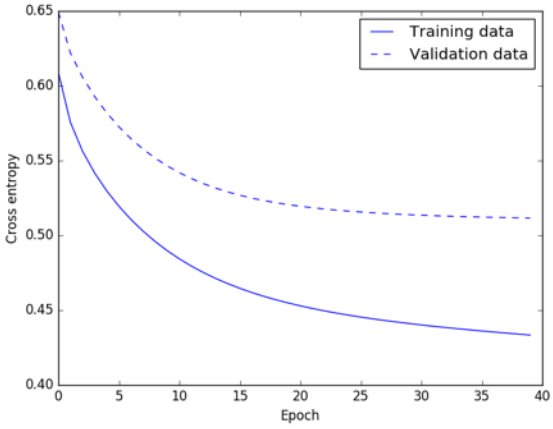
(4).
Choose "Momentum" Method
Try: momentum = 0.5, epochs=60, learning_rate = 0.03, n_hidden = 160, batch_size = 40



Result:
train accuracy: 0.8150
train cross entropy: 0.4224
validation accuracy: 0.7646
validation cross entropy: 0.5024
training took: 32.4788088799 sec

Comment:I first choose momentum equals 0.5, and at the same time, I decrease epochs to 60. My result is really good. I got validation accuracy is 76.46% which did not change over 1%, however, my training time decrease a lot!

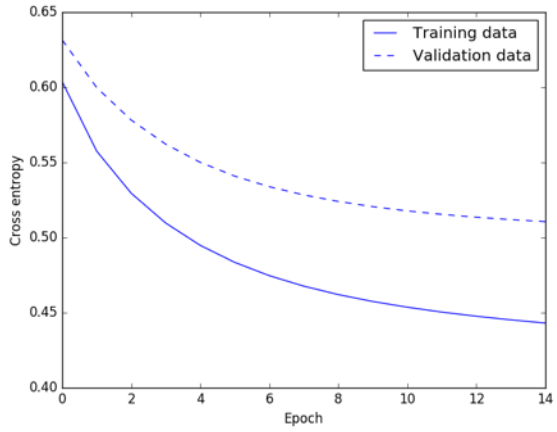
Try: momentum = 0.5, epochs=40, learning_rate = 0.05, n_hidden = 130, batch_size = 60



Result:
train accuracy: 0.8081
train cross entropy: 0.4334
validation accuracy: 0.7722
validation cross entropy: 0.5116
training took: 15.5437159538 sec

Comment: I changed my hyper-parameters, and get shorter training time

Choose "Nesterov Accelerated Gradient":
Try: momentum = 0.5, epochs=15, learning_rate = 0.015, n_hidden = 100, batch_size = 40



Result:
train accuracy: 0.8003
train cross entropy: 0.4431
validation accuracy: 0.7595
validation cross entropy: 0.5106
training took: 7.79454803467 sec

Comment: I tried multiple set of hyper-parameters, and find this one works very well and training time is the shortest. I found that when my epochs is larger, my model easily get overfitting. It is sensitive.

How do the training and validation accuracies and/or convergence times compare to what you found in question 3?
Training and validation accuracies do not change a lot compared to question 3. By setting the proper hyper-parameters, both training and validation accuracies in question 3 and this question can reach around 76%. However, by using "Momentum" or "Nesterov Accelerated Gradient" method, training time can be highly reduced. Our training can be more efficient.

Do the cross entropy and/or accuracy differ between training and validation data as training progresses? Why or why not?
Yes, the differs between training and validation data in question 2 are similar with this question. In both question 3 and this question, training accuracy is always bigger than validation accuracy, and training cross entropy is always smaller than validation cross entropy. The reason is that we use training data to generate model but we use validation set to measure how well our model by predicting validation labels. That's why training error is bigger than validation error and training cross entropy is smaller than validation entropy.