# ECE2013 Fall 2022

## Assignment 3: Naive Bayes/Logistic Regression Classification

In this assignment you will apply machine learning techniques for image and text classification task, and apply logistic regression classifier to do binary classification.

**Programming language**

You may only use modules from the Python standard library and numpy.

## Contents


- Part 1: Image Classification
- Part 2: Text Classification
- Part 3: Linear Classifier
- Extra Credit
- Provided Code Skeleton
- Deliverables
- Report checklist


## Part 1: Digit image classification

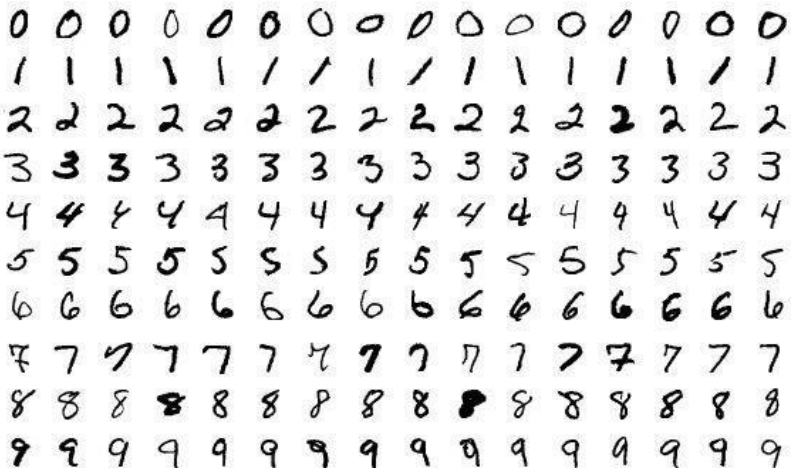

Image from Baidu

**Data:** You are provided with part of the Digit Mnist dataset. There are 55000 training examples and 10000 test examples. The labels are from 0 to 9, representing digits of 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. In this section, you will apply Naïve Bayes model for this task.

**Naive Bayes model**

- **Features:** Each image consists of 28*28 pixels which we represent as a flattened array of size 784, where each feature/pixel $F_i$ takes on intensity values from 0 to 255 (8 bit grayscale).
- **Training:** The goal of the training stage is to estimate the **likelihoods** **$P(F_i \mid class)$** for every pixel location i and for every digit class (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). The likelihood estimate is defined as

  **$P(F_i = f \mid class)$ = (# of times pixel i has value f in training examples from this class) / (Total # of training examples from this class)**

  In addition, as discussed in the lecture, you have to **smooth** the likelihoods to ensure that there are no zero counts. *Laplace smoothing* is a very simple method that increases the observation count of every value f by some constant k. This corresponds to adding k to the numerator above, and k*V to the denominator (where V is the number of possible values the feature can take on). The higher the value of k, the stronger the smoothing. Experiment with different values of k (say, from 0.1 to 10) and find the one that gives the highest classification accuracy.

  You should also estimate the **priors** **$P(class)$** by the empirical frequencies of different classes in the training set.

- **Testing:** You will perform **maximum a posteriori (MAP)** classification of test digit class according to the learned Naive Bayes model. Suppose a test image has feature values $f_1, f_2, \ldots, f_{784}$. According to this model, the posterior probability (up to scale) of each class given the digit is given by

  **$P(class) \cdot P(f_1 \mid class) \cdot P(f_2 \mid class) \cdot \ldots \cdot P(f_{784} \mid class)$**

  Note that in order to avoid underflow, it is standard to work with the log of the above quantity:

  **$\log P(class) + \log P(f_1 \mid class) + \log P(f_2 \mid class) + \ldots + \log P(f_{784} \mid class)$**

  After you compute the above decision function values for all ten classes for every test image, you will use them for MAP classification.

- **Evaluation:** Report your performance in terms of **average classification rate** and the **classification rate for each digit** (percentage of all test images of a given item correctly classified). Also report your **confusion matrix**. This is a 10x10 matrix whose entry in row r and column c is the percentage of test images from class r that are classified as class c. In addition, for each class, show the test examples from that class that have the highest and the lowest posterior probabilities according to your classifier. You can think of these as the most and least "prototypical" instances of each digit class (and the least "prototypical" one is probably misclassified).

- **Likelihood visualization:** When using classifiers in real domains, it is important to be able to inspect what they have learned. One way to inspect a naive Bayes model is to look at the most likely features for a given label. Another tool for understanding the parameters is to visualize the feature likelihoods

for high intensity pixels of each class. Here high intensities refer to pixel values from 128 to 255. Therefore, the likelihood for high intensity pixel feature $F_i$ of class $c_1$ is sum of probabilities of the top 128 intensities at pixel location i of class $c_1$.

$$\text{Feature likelihood } (F_i, c_1) = \sum_{k=128}^{255} P(F_i = k \mid c_1)$$

For each of the ten classes, plot their trained likelihoods for high intensity pixel features to see what likelihood they have learned.

## Part 2: Text Classification

You are given a dataset consisting of texts which belong to 14 different classes. We have split the dataset into a training set and a development dataset. The training set consists of 3865 texts and their corresponding class labels from 1-14, with instances from each of the classes and the development set consists of 483 test instances and their corresponding labels. We have already done the preprocessing of the dataset and extracted into a Python list structure in text_main.py. Using the training set, you will learn a Naive Bayes classifier that will predict the right class label given an unseen text. Use the development set to test the accuracy of your learned model. Report the accuracy, recall, and F1-Score that you get on your development set. We will have a separate (unseen) train/test set that we will use to run your code after you turn it in. No other outside non-standard python libraries can be used.

## Unigram Model

The bag of words model in NLP is a simple unigram model which considers a text to be represented as a bag of independent words. That is, we ignore the position the words appear in, and only pay attention to their frequency in the text. Here each text consists of a group of words. Using Bayes theorem, you need to compute the probability of a text belonging to one of the 14 classes given the words in the text. Thus you need to estimate the posterior probabilities:

$$P(\text{ Class } = C_i \mid \text{ Words }) = \frac{P(\text{ Class } = C_i)}{P(\text{ Words })} \prod_{\text{All words}} P(\text{ Word } \mid \text{ Class } = C_i)$$

It is standard practice to use the log probabilities so as to avoid underflow. Also, P(words) is just a constant, so it will not affect your computation.

## Training and Development

- **Training:** To train the algorithm you are going to need to build a bag of words model using the texts. After you build the model, you will need to estimate the log-likelihoods $\log P(\text{Word}|\text{Type} = C_i)$. The variable $C_i$ can only take on 14 values, 1-14. Additionally, you will need to make sure you

smooth the likelihoods to prevent zero probabilities. In order to accomplish this task, use Laplace

- **Development:** After you have computed the log-likelihoods, you will have your model predict class labels of the text from the development set. In order to do this, you will do MAP estimation classification using the equation shown above.

Use only the training set to learn the individual probabilities. The following results should be put in your report:

1. Plot your confusion matrix. This is a 14x14 matrix whose entry in row r and column c is the percentage of test text from class r that are classified as class c.
2. Accuracy, recall, and F1 scores for each of the classes on the development set.
3. Top 20 feature words of each of the classes.
4. Calculate your accuracy without including the class prior into the Naive Bayes equation i.e. Only computing the ML inference of each instance. Report the change in accuracy numbers, if any. Also state your reasoning for this observation. Is including the class prior always beneficial? Change your class prior to a uniform distribution. What is the change in result?

## Part 3: Linear Classifier

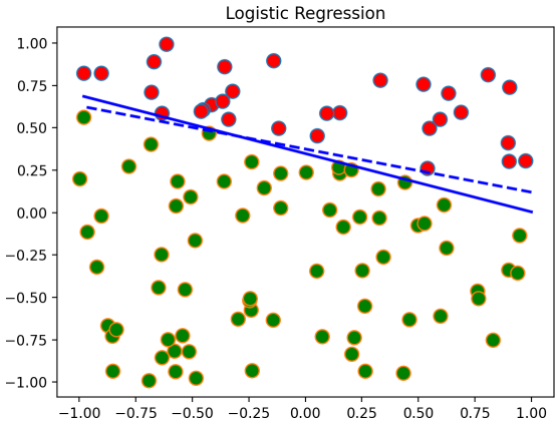


Image by TA

There are some points on the 2-D plane, some of which are labeled as 1, and others are labeled as 0. Your task is to find the boundary line that can correctly separate these two categories of points. As in the image shown above, the solid line is the real boundary, and the dashed line is the boundary found by a logistic regression classifier.

**Note**:

a) You need to achieve a logistic regression classifier for this task. The *logistic.py* are the only file you need to modify in this section.

b) Although we only do classification on 2-D points in this task, your code should be working on arbitrary dimensions.

**Logistic regression model**

Logistic regression model, also known as differentiable perceptron, which is as follows:

$$\boldsymbol{f(\vec{w}^T\vec{f}) = \text{sigmoid}\,(\vec{w}^T\vec{f}) = \frac{1}{1+e^{-\vec{w}^T\vec{f}}}}$$

**Note**:

a) This logistic regression model is different from the one in the lecture slide. You should implement this one in this task, **NOT** the one in the slide.

b) The derivative of sigmoid function is $f'(x) = f(x) \times (1 - f(x))$

- **Features:** The coordinates of every points. Denote the number of points as *N*, the dimension of coordinates as *P*, the feature matrix should be *P*N*.
- **Training:** Achieve the training process of Logistic Regression model. Recall the loss function of logistic regression in lecture slide, which is as follows: (Note: a better mesurement would be logistic loss which is not required in this MP. If you are interested, see Logistic regression here.)

$$L(y_1, \dots, y_n, \vec{f}_1, \dots, \vec{f}_n) = \sum_{i=1}^{n} \left(y_i - \text{sigmoid}\,(\vec{w}^T\vec{f}_i)\right)^2$$

- **Testing:** The code provided has already achieve the testing process for you. You do **NOT** need to achieve this. But do **Not** forget to report the test results on your report.
- **Evaluation:** We repeated the process of training and testing for many times, and take the average training error and testing error as our evaluation of our model. This is also achieved in the skeleton code for you.

## Extra Credit Suggestion

Implement the naive Bayes algorithm over a bigram model as opposed to the unigram model. Bigram model is defined as follows:

$$P(w_1 \dots w_n) = P(w_1)P(w_2 \mid w_1) \dots P(w_n \mid w_{n-1})$$

Then combine the bigram model and the unigram model into a mixture model defined with parameter λ:

$$(1-\lambda)P(Y)\prod_{i=1}^{n} P(w_i \mid Y) + \lambda P(Y)\prod_{i=1}^{m} P(b_i \mid Y)$$

Did the bigram model help improve accuracy? Find the best parameter λ that gives the highest classification accuracy. Report the optimal parameter λ and report your results (Accuracy number) on the bigram model and optimal mixture model, and answer the following questions:

1. Running naive Bayes on the bigram model relaxes the naive assumption of the model a bit. However, is this always a good thing? Why or why not?
2. What would happen if we did an *N*-gram model where *N* was a really large number?

## Provided Code Skeleton

We have provided ( zip file) all the code to get you started on your MP.

For part 1, you are provided the following. The doc strings in the python files explain the purpose of each function.

- image_main.py- This is the main file which loads the dataset and calls your Naive Bayes algorithms.
- naive_bayes.py- This is the only file that needs to be modified.
- x_train.npy, y_train.npy, x_test.npy and y_test.npy- These files contain the training and testing examples.

For part 2, you are provided the following. The doc strings in the python files explain the purpose of each function

- text_main.py- This is the main file which loads the dataset and calls your Naive Bayes Algorithm.
- TextClassifier.py- This is the only file that needs to be modified.
- train_text.csv- This file contains the training examples.
- dev_text.csv- This file contains the development examples for testing your model.
- stop_words.csv- This file contains the stop words which are required for preprocessing the dataset.

For part 3, you are provided the following. The doc strings in the python files explain the purpose of each function

- linear_classifier_main.py- This is the main file which loads the dataset and calls your Perceptron and Logistic Regression Algorithm.
- logistic.py- This is the only file that needs to be modified to achieve your Logistic Regression Algorithm.
- mkdata.py - This is the file to make synthetic data for your algorithm.
- plotdata.py - This file is to plot the experiment result of your perceptron model.
- plotdata_log_reg.py- This file is to plot the experiment result of your Logistic Regression model.

## Deliverables

This MP will be submitted via blackboard.

Please upload **only the following files** to blackboard.

1. naive_bayes.py - your solution python file to part 1
2. TextClassifier.py - your solution python file to part 2
3. logistic.py - your solution python file to part 3
4. report.pdf - your project report in pdf format

## Report Checklist

Your report should briefly describe your implemented solution and fully answer the questions for every part of the assignment. Your description should focus on the most "interesting" aspects of your solution, i.e., any non-obvious implementation choices and parameter settings, and what you have found to be especially important for getting good performance. Feel free to include pseudocode or figures if they are needed to clarify your approach. Your report should be self-contained and it should (ideally) make it possible for us to understand your solution without having to run your source code.
Kindly structure the report as follows:

1. **Title Page:**
   List of all team members, course number and section for which each member is registered, date on which the report was written
2. **Section I:**
   Image Classification. Report average classification rate, the classification rate for each class and the confusion matrix. For each class, show the test examples from that class that have the highest and lowest posterior probabilities according to your classifier. Show the ten visualization plots both feature likelihoods.
3. **Section II:**
   Text Classification. Report all your results, confusion matrix, recall, precision, F1 score for all the 14 classes. Include the top feature words for each of the classes. Also, report the change in accuracy results when the class prior changes to uniform distribution and when its removed. Provide the reasoning for these observations
4. **Section III:**
   Linear Classifier. Report all your average error rate of training and test set for your Logistic Regression model. Show your visual result of the models.
5. **Extra Credit:**
   If you have done any work which you think should get extra credit, describe it here

6. **Statement of Contribution:**
   Specify which team-member performed which task. You are encouraged to make this a many-to-many mapping, if applicable. e.g., You can say that "Rahul and Jason both implemented the BFS function, their results were compared for debugging and Rahul's code was submitted. Jason and Mark both implemented the DFS function, Mark's code never ran successfully, so Jason's code was submitted. Section I of the report was written by all 3 team members. Section II by Mark and Jason, Section III by Rahul and Jason."… and so on.

Only attach files that are the required deliverables in blackboard. Your report must be a formatted pdf document. Pictures and example outputs should be incorporated into the document. Exception: items which are very large or unsuitable for inclusion in a pdf document (e.g. videos or animated gifs) may be put on the web and a URL included in your report. **You can write your report either in English or Chinese.**

**Extra credit:**

We reserve the right to give **bonus points** for any advanced exploration or especially challenging or creative solutions that you implement. This includes, but is not restricted to, the extra credit suggestion given above.