Homework 1

201900800302 赵淇涛

1. The Federalist Papers

Overview

Among 85 federalist papers, most of them were written by **Hamilton** and **Madison**. However, authors of 12 papers are not exactly known (by Hamilton or Madison). We need to use **MLE** or **MAP** to estimate the author for each of those 12 papers.

Since we have no prior knowledge for those 12 papers, I adopted the **MLE** algorithm insted of **MAP** to estimate the authors. Specifically, I classified 12 papers by investigating whether or not some words appear in each article, since the usage of words could reflect the writing style of author. I first sorted out Hamilton and Madison's articles. For each of two authors, I combined individual articles into a big text set and counted the numter of occurance of all words that appear in this text set. Then I constructed a vocabulary that contains words that appear for not less than 5 times and calculated the frequency of occurance over articles of each author. These frequencies can be used as features to analyze any given article. According to **MLE**, the probability of each event in a random test can be approximated by their frequencies in a data set. Here, I did so for the words. Provided a paper of unknown author, for each word in the vocabulary, I assigned 1 to a list element corresponding to this word if the word is included in the article, and 0 otherwise. Combine both the list that holds information of existance of some words in the article and the estimated word usage probability distribution for two authors so that probabilities that the article belongs to each author could be calculated.

Code and process

1. Download the plain text (assignment1/text/download.py)

```
import urllib.request
import re
from bs4 import BeautifulSoup

def getHtml(url):
    page = urllib.request.urlopen(url)
    html = page.read().decode('utf-8')
    return html

def getWord(html):
    bs = BeautifulSoup(html, "html.parser")
    namelist = bs.findAll("p")
    return namelist

def download(url, filename):
  html = getHtml(url)
  namelist = getWord(html)
  File = open(filename, 'w')
  for name in namelist:
      File.write(name.get_text())

  File.close()

hamilton = [1, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 21, 22, 23, 24, 25, 26, 27, 28, 29,
      30, 31, 32, 33, 34, 35, 36, 59, 60, 61, 65, 66, 67, 68, 69, 70, 71, 72, 73,
      74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85]

madison = [10, 14, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 58]

unknown = [49, 50, 51, 52, 53, 54, 55, 56, 57, 62, 63]

for index in hamilton:
# for index in madison:
# for index in unknown:
  print(index)
  if index < 10:
    download("https://avalon.law.yale.edu/18th_century/fed0%d.asp" % index, '%d.txt' %
index)
  else:
    download("https://avalon.law.yale.edu/18th_century/fed%d.asp" % index, '%d.txt' %
index)
```

2. Class *Book* reads and analyses the text (assignment1/MLE/text_reader.py)

```
class Book():
    """Define a class reading and analysing the text"""
    def __init__(self, n=None):
        self.num_words = 0
        self.num_count = n
        # A list (['word', ...]) updates each time Book read()
        self.text = []
        # A dict ({'word': num_occur, ...}) updates each time Book countWords()
        self.stat = {}

    def read(self, route):
        """Output processed text"""
        text = open(route).read()
        # Change all characters into lowercase
        text = text.lower()
        # Replace symbols with spaces
        for i in '!"#$%&()*+,-./:;<=>?@[\\]^_‘{|}~':
            text = text.replace(i, " ")
        text = text.split()
        self.num_words += len(text)
        self.text.extend(text)
    def countWords(self):
        """Count words in the text"""
        self.stat = {}
        counts = {}
        for word in self.text:
            counts[word] = counts.get(word, 0) + 1
        items = list(counts.items())
        # Sort by the second item in each element of the list
        items.sort(key=lambda x:x[1], reverse=True)
        # for i in range(len(items)):
        if self.num_count is None:
            self.num_count = len(items)
        for i in range(self.num_count):
            word, count = items[i]
            self.stat[word] = count

    def show(self, n):
        """Show the vocabulary"""
        items = list(self.stat.items())
        items.sort(key=lambda x:x[1], reverse=True)
        # for i in range(len(items)):
        for i in range(n):
            word, count = items[i]
            print("{0:<10}{1:>5}    {2:>.6f}".format(word, count,
(count/self.num_words)))
        print("Total number:", self.num_words)
```

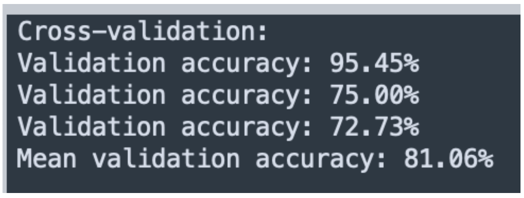
3. MLE with cross-validation and test (assignment1/MLE/MLE.py)

```
from text_reader import Book
import numpy as np


def stat(author, indices, verbose=True):
  book = Book()
  book.author = author
  for i in indices:
    book.read(r"/Users/zhaoqitao/Desktop/assignment1/text/%s/%d.txt" % (author, i))
    book.countWords()
  if verbose:
    print(author + ":")
    book.show(10)
  return(book)

def MLE(X_train, y_train, X_val, y_val, margin, mode, indices=None, verbose=True):
  num_train, D = X_train.shape
  num_val, D = X_val.shape
  predicted_label = np.ndarray((num_val, ), dtype='S32')
  indices_H = np.where(y_train==b'hamilton')[0]
  indices_M = np.where(y_train==b'madison')[0]

  probs_integrated_H = np.minimum(np.maximum(X_train[indices_H, :].mean(axis=0),
margin), 1-margin)
  probs_integrated_M = np.minimum(np.maximum(X_train[indices_M, :].mean(axis=0),
margin), 1-margin)

  for i in range(num_val):
    mask_0 = X_val[i, ] == 0
    mask_1 = X_val[i, ] == 1
    log_prob_H = (np.log(probs_integrated_H[mask_1]) * X_val[i, mask_1]).sum()
    log_prob_H += (np.log(1-probs_integrated_H[mask_0]) * (1-X_val[i, mask_0])).sum()
    log_prob_M = (np.log(probs_integrated_M[mask_1]) * X_val[i, mask_1]).sum()
    log_prob_M += (np.log(1-probs_integrated_M[mask_0]) * (1-X_val[i, mask_0])).sum()
    if log_prob_H > log_prob_M:
      predicted_label[i] = 'hamilton'
    elif log_prob_H < log_prob_M:
      predicted_label[i] = 'madison'
    else:
      print("Value error!")
  if mode == 'val':
    accuracy = np.sum(predicted_label==y_val) / num_val
    if verbose:
      print("Validation accuracy: {:.2f}%".format(accuracy*100))
    return accuracy
  else:
    print("Predict with MLE:")
    for i in range(len(indices)):
      print("The no.%d article is most likely written by %s" % (indices[i],
predicted_label[i].decode('utf-8').capitalize()))

def crossValidation(X, y, num_folds, margin, k=None):
  print("Cross-validation:")
  X_train_folds = np.array_split(X, num_folds)
  y_train_folds = np.array_split(y, num_folds)
  accuracies = []

  for i in range(num_folds):
    X, y = X_train_folds[:], y_train_folds[:]
    X.pop(i)
    y.pop(i)
    X_train = X_train_folds[i]
    y_train = y_train_folds[i]
    X_val = np.concatenate(X, axis=0)
    y_val = np.concatenate(y, axis=0)
    accuracies.append(MLE(X_train, y_train, X_val, y_val, margin, 'val', verbose=True))

  print("Mean validation accuracy: {:.2f}% \n".format(sum(accuracies)/num_folds*100))
  return sum(accuracies)/num_folds


hamilton = [1, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 21, 22, 23, 24, 25, 26, 27, 28, 29,
      30, 31, 32, 33, 34, 35, 36, 59, 60, 61, 65, 66, 67, 68, 69, 70, 71, 72, 73,
      74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85]
madison = [10, 14, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 58]
unknown = [49, 50, 51, 52, 53, 54, 55, 56, 57, 62, 63]

#################### Extract data ####################

H = stat('hamilton', hamilton, False)
M = stat('madison', madison, False)

vocabulary = []
for k, v in H.stat.items():
  if v >= 5 and k not in vocabulary:
    vocabulary.append(k)
for k, v in M.stat.items():
  if v >= 5 and k not in vocabulary:
    vocabulary.append(k)

num_books = len(hamilton) + len(madison)
len_vocabulary = len(vocabulary)
data = np.ndarray((num_books, len_vocabulary), dtype=np.float32)
label = np.ndarray((num_books, ), dtype='S32')
X_test = np.ndarray((len(unknown), len_vocabulary), dtype=np.float32)

for i in range(len(hamilton)):
  temp = stat('hamilton', [hamilton[i]], False)
  num_occur = np.zeros((len_vocabulary, ))
  for j in range(len(vocabulary)):
    num_occur[j] = vocabulary[j] in temp.stat
  data[i] = num_occur
  label[i] = temp.author

for i in range(len(madison)):
  temp = stat('madison', [madison[i]], False)
  num_occur = np.zeros((len_vocabulary, ))
  for j in range(len(vocabulary)):
    num_occur[j] = vocabulary[j] in temp.stat
  data[i+len(hamilton)] = num_occur
  label[i+len(hamilton)] = temp.author

for i in range(len(unknown)):
  temp = stat('unknown', [unknown[i]], False)
  num_occur = np.zeros((len_vocabulary, ))
  for j in range(len(vocabulary)):
        num_occur[j] = vocabulary[j] in temp.stat
    X_test[i] = num_occur

np.random.seed(2021)

# Randomize data and labels
random_list = np.arange(num_books)
np.random.shuffle(random_list)
data = data[random_list, :].astype(np.int32)
label = label[random_list]

# Cross-validation
# margin = 10 ** np.random.uniform(-3, -1)
margin = 0.05
crossValidation(data, label, 3, margin, 'MLE')

# Predict
MLE(data, label, X_test, None, margin, 'test', unknown)
```

Note that a variable 'margin' is used to avoid log(0), which is a hyperparameter.

Result and discussion

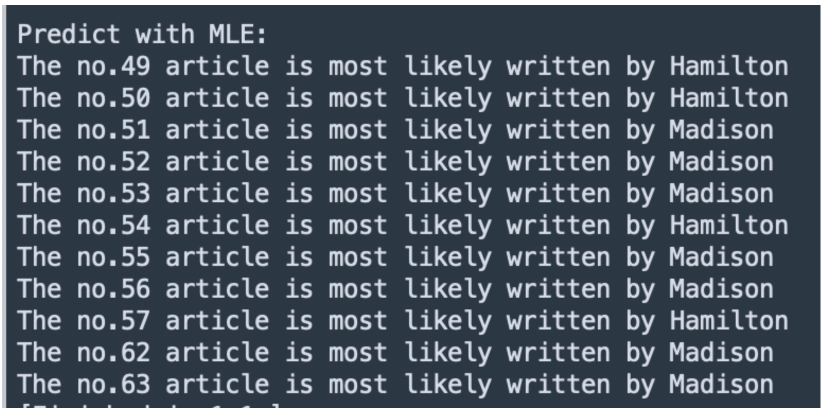
1. Cross-validation



```
Cross-validation:
Validation accuracy: 95.45%
Validation accuracy: 75.00%
Validation accuracy: 72.73%
Mean validation accuracy: 81.06%
```

I shuffled the whole data set and split it into three parts to apply cross-validation. A larger number of folds would lead to a worse performance since our data set is quite biased. Note that the first validation accuracy achieved 95.45% whereas the average accuracy is relatively low (81.06%). This is because the arrangement of the train set in first validation process is more reasonable.

2. Prediction



```
Predict with MLE:
The no.49 article is most likely written by Hamilton
The no.50 article is most likely written by Hamilton
The no.51 article is most likely written by Madison
The no.52 article is most likely written by Madison
The no.53 article is most likely written by Madison
The no.54 article is most likely written by Hamilton
The no.55 article is most likely written by Madison
The no.56 article is most likely written by Madison
The no.57 article is most likely written by Hamilton
The no.62 article is most likely written by Madison
The no.63 article is most likely written by Madison
```

My prediction is that most of thses papers of unknown author were written by Madison (7 of 11). This time all articles of known author were treated as train data.

2. KNN on Fisher Iris Data

Overview

Here we are required to implement the **k-nearest neighbors** algorithm given Fisher iris data. Note that there are 150 data points in this data set and each of them is labeled with class (one of three), holding four attributes.

To conveniently handle these data, I converted the raw data from a txt file into two numpy arrays. The first one is of size[150, 4], carrying four attributes while the second one is of size[150,], providing the correct classes for each data point.

I adopted the L2 distance to measure the similarity between data points, and tried several ks to investigate the classifying capability of the model. I first shuffled the whole data set since it was originally neatly arranged by class and split it into 5 folds. Then, combine the first 4 folds into the train set such that the last fold was the validation set. For each k from [1, 5, 10, ..., 150], I implemented both training and validation process. The training process may sound misleading. That is to classify train set given the train set itself, which aimed at checking the correctness of code implementation (the train accuracy must achieve 100% when k=1). The validation process showed the performance of the k-nn classifier for each k selected.

Code (assignment1/knn/knn.py) and process

1. Creat data set

```
# Extract data set
data = np.ndarray((150, 4), dtype=float)
label = np.ndarray((150, ), dtype='S32')
file = open('./iris.txt')
org = file.readlines()
for i, line in enumerate(org):
  data[i, :4] = np.array(line[:-1].split(',')[:4], dtype=float)
  label[i] = line[:-1].split(',')[4]

# Split the data set into train_set and val_set
np.random.seed(231)
random_list = np.arange(150)
np.random.shuffle(random_list)
data = data[random_list, :]
label = label[random_list]
X_train = data[:120, :]
y_train = label[:120]
X_val = data[120:, :]
y_val = label[120:]
```

2. K-nn function

```
def knn(X_train, y_train, X_val, y_val, k, set_mode):
  num_val, D = X_val.shape
  predicted_label = np.ndarray((num_val, ), dtype='S32')
  for i in range(num_val):
    L2 = np.sqrt(np.sum((X_train - X_val[i, :]) ** 2, axis=1))
    predicted_label[i] = mode(y_train[np.argsort(L2, axis=0)][:k])[0][0]
    predicted_label[i] = predicted_label[i].decode('utf-8')
  accuracy = np.sum(predicted_label==y_val) / num_val
  print("k = %d, %s accuracy: %.2f" % (k, set_mode, accuracy))
  return accuracy
```

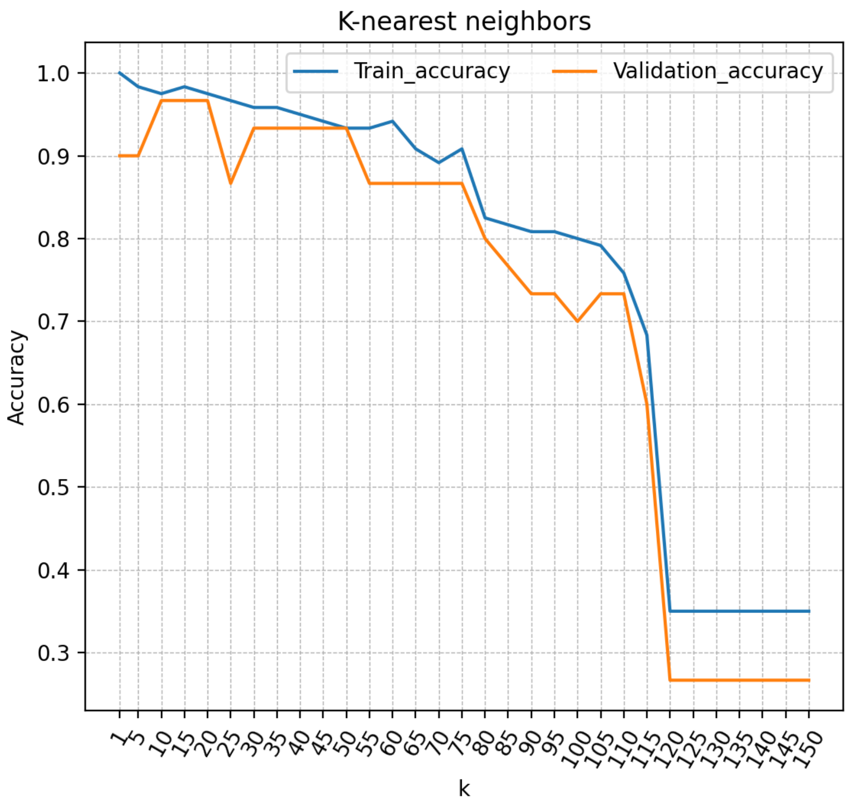
3. Train and validation

```
# Train & validation
x = np.linspace(1, 150, 31).astype(int)
train_accuracy = []
val_accuracy = []
for k in x:
  train_accuracy.append(knn(X_train, y_train, X_train, y_train, k, 'train'))
  val_accuracy.append(knn(X_train, y_train, X_val, y_val, k, 'validation'))
```

4. Plot the result

```
# Plot the result
plt.plot(x, train_accuracy, label="Train_accuracy")
plt.plot(x, val_accuracy, label="Validation_accuracy")
plt.title('K-nearest neighbors')
plt.xticks(x, rotation=60)
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend(loc="best", ncol=4)
plt.grid(linestyle='--', linewidth=0.5)
plt.show()
```

5. Result and discussion



On the whole, both train accuracy and validation accuracy decrease as k increases. As illustrated, validation accuracy is maximized (97%) when k=10, 15, 20. Intuitively, the model cannot gather enough information from the given data set with a small k, whereas the model can be confused by noises as k goes extreme. Hence, a appropriate k should not be too small or too big, which can be determined by looping over many different k values.