MIDS-W261-2015-HWK-Week01-Cordell

January 18, 2016

1 MIDS W261 Spring 2016 Homework Week 1

Ron Cordell W261-4 ron.cordell@ischool.berkeley.edu January 19, 2016

1.1 HW1.0.0.

Define big data. Provide an example of a big data problem in your domain of expertise. "Big Data" is any analysis or problem that involves working with data at volumes, velocities and varieties that are beyond the boundaries of "traditional" vertically scaled solutions and systems. Big Data problems are such that they require tools and techniques using parallel processing, sampling of high speed data streams and/or combination of varieties of disparate data and formats (to name a few) to provide solutions.

An example of a big data problem in the world of health care is in the area of population health management. Regional medical systems such as Sutter Health, John Muir, and even the VA consist of a huge number of disparate data systems all producing event data as patients, care providers, medications, procedures and more move through the dance of episodic medical care. A regional medical group or system can be a nucleus of hospitals, perhaps acquired or merged over time or as a public institution, with satellite medical offices and federated health care providers like pediatricians or Ob/Gyns, pharmacies, etc. A simple episodic example is the patient that comes into a doctor's office for an acute medical issue. The visit spawns a number of events in the medical records system of the doctor's office - patient demographic data, doctor's notes, pharmacy orders, doctor's orders for labs, etc. If the patient is sent to a lab to have a blood test, the lab medical systems generate more events: more patient demographic data, the lab order, eventually the intermediate lab results, the final lab results. The patient may then receive a prescription, again generating events for yet more demographic data, presctription data that may be coded in several different medication coding systems, etc. And this is a simple case. Consider what happens when the patient is referred to a hospital or surgery: multiple care providers generating orders, labs, results, medication, procedures, equipment, insurance data... We talk about digital exhaust often in the sense of social media or leaving footprints in our day to day lives but the amount of data generated from a single patient episode is much much larger.

Now take this event data, this patient data exhaust, multiply it by the number of health care providers, the number of health care institutions, the number of patients on a daily basis, and the sheer variety of disparate systems involved. I haven't even really pointed out the insurance data, the medical supply chain data, the pharmaceutical supply chain data... When a medical community wants to understand patient populations they need data to provide insight, but the data is in the silos of systems that don't talk to each other and don't generally have readily common data formats. There are three major medication coding systems in the US alone, and none of them line up one for one - they all contain different sets of concepts and translations between them can be imperfect. There is some level of common health care data information interchange, but every system that implements it uses a unique dialect that is then customized upon installation. A single patient is replicated in all these disparate systems with different identifiers, mistyped or incorrect demographic data, or just empty data because people were in a hurry - all making it very challenging to match that patient across all those systems and across separate episodes.

A medical community looking into patient population health may see obvious things such as regional and cultural influences on health, chronic disease by demographic, or the population density of common allergies but they have the potential to see so much more. For example, the propensity to diagnose a particular condition or procedure, the use of a particular drug, the observed outcomes of a procedure, how medications

move through distribution channels to the patients and more. Looking at the data across regional health care systems can provide an almost unimaginable wealth of information about effective treatments, resource allocations of medications and supplies, etc.

1.2 HW1.0.1.

In 500 words (English or pseudo code or a combination) describe how to estimate the bias, the variance, the irreducible error for a test dataset T when using polynomial regression models of degree 1, 2, 3, 4, 5 are considered. How would you select a model?

1. For each degree d of polynomial
2. Create bootstrap samples B of the test data set T
3. For each boostrap sample B apply Polynomial learning algorithm of degree d to obtain hypothesis h_B
4. Compute the predicted value $h_B(x_T)$ for each data point in T that is not in B
5. Compute the test error $y_T - h_B(x_T)$
6. Compute the variance $h_B(x_T) - E[h_B(x_T)]$ where $E[h_B(x_T)]$ is the average model prediction
7. Compute the bias $E[h_B(x_T)]$ - y_T

Choose the model that minimizes test error and bias and variance as compared across model complexity given by polynomial degree d

1.3 HW 1.1

Read through the provided control script (pNaiveBayes.sh) and all of its comments. When you are comfortable with their purpose and function, respond to the remaining homework questions below.

```
In [1]: print 'done'
```

```
done
```

1.4 HW 1.2

Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will determine the number of occurrences of a single, user-specified word. Examine the word "assistance" and report your results.

1.5 Map

```
In [59]: %%writefile mapper.py
         #!/usr/bin/python
         ## mapper.py
         ## Author: Ron Cordell
         ## Description: mapper code for HW1.2
         ## Given a file and list of words, read lines and count occurrences of words

         import sys
         import re

         ## Lines in the file have 4 fields:
         ## ID \t SPAM \t SUBJECT \t CONTENT \n
         WORD_RE = re.compile(r"[\w']+")

         filename = sys.argv[1]

         ## Words in the word list are space delimited
```

```
        wordlist = sys.argv[2].lower().split(' ')
        counts = {}

        with open (filename, "rU") as myfile:
            for line in myfile.readlines():
                fields = line.split('\t')
                if len(fields) > 3:
                    # we are interested in subject and body but need to be resilient for missing fields
                    text = '{0} {1}'.format(fields[2],fields[3])
                elif len(fields) > 2:
                    text = fields[2]
                # extract only words from the combined subject and body text
                for word in WORD_RE.findall(text):
                    if word.lower() in wordlist:
                        try:
                            counts[word.lower()] += 1
                        except:
                            counts[word.lower()] = 1

        for word in counts:
            sys.stdout.write('{0}\t{1}\n'.format(word, counts[word]))
```

```
Overwriting mapper.py
```

```
In [26]: # Set the execution permissions of the Python script
         !chmod a+x mapper.py
```

1.6 Reduce

```
In [56]: %%writefile reducer.py
         #!/usr/bin/python
         ## reducer.py
         ## Author: Ron Cordell
         ## Description: reducer code for HW1.2
         ## given a list of intermediate word count files, combine into a single word count list
         import sys
         counts = {}

         for intermediate_file in sys.argv:
             with open(intermediate_file, 'rU') as infile:
                 # intermediate files are word <tab> count per line
                 for line in infile.readlines():
                     word_count = line.split('\t')
                     if len(word_count) == 2:
                         try:
                             counts[word_count[0]] += int(word_count[1])
                         except KeyError:
                             counts[word_count[0]] = int(word_count[1])
         for word in counts:
             sys.stdout.write('{0}\t{1}\n'.format(word, counts[word]))
```

```
Overwriting reducer.py
```

```
In [39]: # Set the execution permissions of the Python script
         !chmod a+x reducer.py
```

1.7 Run the file

```
In [35]: # Set the execution permissions of the pNaiveBayes.sh bash shell script
         !chmod a+x pNaiveBayes.sh

In [63]: !./pNaiveBayes.sh 4 "assistance"
```

1.8 Results

```
In [64]: # dump the reducer output to check out results
         !cat enronemail_1h.txt.output

assistance        10
```

1.9 HW 1.3 - 1.5

Evolve a mapper and reducer that first takes a single word, then several words, then all words to classify spam or ham

1.10 Map

The mapper.py code has evolved over the 3 iterations of single to all word execution. The final result handles all three cases.

This mapper works by taking each record (a single email) and extracting the id, subject and body. The subject and body are concatenated and then tokens matched against a regular expression that screens for words, numbers and apostrophes. The result is a key,value pair output for each word in a record. The key consists of (email id, label, vocabulary word flag, word) and the value is the count. An example key for a non-spam email might be:

```
(0007.2001-02-09.kitchen, 0, 1, assistance) = 1
```

In the case that all tokens are used then all keys will have the vocabulary word flag = 1.

The key value pairs are output to STDOUT

```
In [223]: %%writefile mapper.py
          #!/usr/bin/python
          ## mapper.py
          ## Author: Ron Cordell
          ## Description: mapper code for HW1.4
          ## Given a file and list of words, read lines and count occurrences of words
          ## Output a key, value => (id, class, token, term_flag) = count

          import sys
          import re

          ## Lines in the file have 4 fields:
          ## ID \t SPAM \t SUBJECT \t CONTENT \n
          WORD_RE = re.compile(r"[\w']+")

          filename = sys.argv[1]

          ## Words in the word list are space delimited
          wordlist = sys.argv[2].lower().split(' ')

          all_words = False
          if wordlist[0] == '*':
```

```
            all_words = True

        counts = {}

        with open (filename, "rU") as myfile:
            for line in myfile.readlines():
                fields = line.split('\t')
                if len(fields) > 3:
                    # we are interested in subject and body but need to be resilient for missing fiel
                    text = '{0} {1}'.format(fields[2],fields[3])
                elif len(fields) > 2:
                    text = fields[2]
                # extract only words from the combined subject and body text
                for word in WORD_RE.findall(text):
                    term = '0'
                    if all_words:
                        term = '1'
                    elif word.lower() in wordlist:
                        term = '1'
                    try:
                        counts[(fields[0],fields[1],word.lower(), term)] += 1
                    except:
                        counts[(fields[0],fields[1],word.lower(), term)] = 1
                # ensure that all words in the list are represented even if they don't occur
                if not all_words:
                    for word in wordlist:
                        try:
                            if counts[(fields[0],fields[1], word, '1')] > 0:
                                pass
                        except KeyError:
                            counts[(fields[0], fields[1], word, '1')] = 0

        for key in counts:
            sys.stdout.write('{0}\t{1}\t{2}\t{3}\t{4}\n'.format(key[0], key[1], key[2], key[3], counts
```

```
Overwriting mapper.py
```

```
In [149]: # Set the execution permissions of the Python script
          !chmod a+x mapper.py
```

1.11 Reduce

The reducer.py code implements the consolidation of the key, value pairs emitted by the mapper into a single dictionary and subsequent helper data structures.

`counts` is a dictionary of key, value pairs where the key is a `tuple (id, label, vocab_flag, word)`

`spam_doc_ids` and `ham_doc_ids` are lists of the email ids for each label `spam` or `ham`

`term_counts` is a dictionary of vocabulary terms generated based on the `counts` key tuple `vocab_flag` value. If the value is `1` it indicates that the word is a vocabulary word and will be added to the `term_counts` dictionary. The count value is accumulated depending on whether the term occurs in a `spam` document or a `ham` document. Probabilities of `P(term|ham)` and `P(term|spam)` are also calculated in this dictionary.

Laplace smoothing is used in the calculation of the `P(term|label)` for all terms.

```
In [229]: %%writefile reducer.py
          #!/usr/bin/python
          ## reducer.py
```

```
## Author: Ron Cordell
## Description: reducer code for HW1.4
## given a list of intermediate word count files, compute NB classification
import sys
counts = {}
term_counts = {}
spam_doc_ids = []
ham_doc_ids = []
spam_doc_word_count = 0.0
ham_doc_word_count = 0.0
spam_term_count = 0.0
ham_term_count = 0.0
terms = 0.0

for intermediate_file in sys.argv:
    with open(intermediate_file, 'rU') as infile:
        # intermediate files are id <tab> class <tab> word <tab> term_flag <tab> count per li
        for line in infile.readlines():
            word_count = line.split('\t')
            if len(word_count) == 5:
                # some things to make this more readable
                mail_id = word_count[0]
                label = word_count[1]
                word = word_count[2]
                if word_count[3] == '0':
                    vocab_word = False
                else:
                    vocab_word = True
                count = float(word_count[4].strip())

                if (mail_id, label, vocab_word, word) in counts:
                    counts[(mail_id, label, vocab_word, word)] += count
                else:
                    counts[(mail_id, label, vocab_word, word)] = count

spam = { k:v for k,v in counts.items() if k[1] == '1' }
for k in spam:
    spam_doc_word_count += spam[k]
    if k[0] not in spam_doc_ids:
        spam_doc_ids.append(k[0])
    if k[2]:
        if k[3] in term_counts:
            term_counts[k[3]]['spam_count'] += spam[k]
        else:
            term_counts[k[3]] = {'spam_count' : spam[k],
                                 'ham_count' : 0.0,
                                  'prob_ham'  : 0.0,
                                  'prob_spam' : 0.0}

ham = { k:v for k,v in counts.items() if k[1] == '0' }
for k in ham:
    ham_doc_word_count += ham[k]
    if k[0] not in ham_doc_ids:
        ham_doc_ids.append(k[0])
```

```
    if k[2]:
        if k[3] in term_counts:
            term_counts[k[3]]['ham_count'] += ham[k]
        else:
            term_counts[k[3]] = {'ham_count' : ham[k],
                                 'spam_count' : 0.0,
                                  'prob_ham'  : 0.0,
                                  'prob_spam' : 0.0}

# now we should have consolidated the intermediate counts and we can compute the rest

# count the number of terms
term_count = len(term_counts.keys()) * 1.0

prior = (len(spam_doc_ids)*1.0)/(1.0*(len(spam_doc_ids) + len(ham_doc_ids)))

# calculate the P(term|class) for each term
for term in term_counts:
    term_counts[term]['prob_ham'] = (term_counts[term]['ham_count'] + 1.0)/(ham_doc_word_coun
    term_counts[term]['prob_spam'] = (term_counts[term]['spam_count'] + 1.0)/(spam_doc_word_c

# for debugging
if False:
    for k in counts:
        print '{0} {1} {2} {3} {4}'.format(k[0],k[1],k[2],k[3],counts[k])
    for t in term_counts:
        print '{0} {1} {2} {3} {4}'.format(t, term_counts[t]['spam_count'],
                                           term_counts[t]['ham_count'],
                                           term_counts[t]['prob_spam'],
                                           term_counts[t]['prob_ham'])
    print len(spam_doc_ids)
    print len(ham_doc_ids)
    print spam_doc_word_count
    print ham_doc_word_count
    print term_count
    print prior

right = 0.0
for email in spam_doc_ids + ham_doc_ids:
    if email in ham_doc_ids:
        true_class = '0'
    else:
        true_class = '1'
    docs = {k:v for k,v in counts.items() if k[0]==email}
    label = '0'
    prob_spam = 0.0
    prob_ham  = 0.0
    for term in term_counts:
        # does this email contain any of the vocabulary terms
        ham = { k:v for k,v in docs.items() if k[3] == term }
        for h in ham:
            if ham[h] > 0.0:
                if prob_spam > 0.0:
                    prob_spam = prob_spam * term_counts[term]['prob_spam']**ham[h]
```

```
                            prob_ham = prob_ham * term_counts[term]['prob_ham']**ham[h]
                        else:
                            prob_spam = prior * term_counts[term]['prob_spam']**ham[h]
                            prob_ham = (1.0 - prior) * term_counts[term]['prob_ham']**ham[h]
                if prob_spam > 0.0:
                    if prob_spam > prob_ham:
                        label = '1'
                sys.stdout.write('{0}\t{1}\t{2}\n'.format(email, true_class, label))
                if true_class == label:
                    right += 1.0
            sys.stdout.write('Score: {0}/{1}'.format(right, len(spam_doc_ids) + len(ham_doc_ids)))
```

```
Overwriting reducer.py
```

```
In [219]: # Set the execution permissions of the Python script
          !chmod a+x reducer.py
```

1.12 Run the file

```
In [156]: # Set the execution permissions of the pNaiveBayes.sh bash shell script
          !chmod a+x pNaiveBayes.sh
```

1.12.1 HW 1.3: Classify using the single word "assistance" and report results

```
In [228]: !./pNaiveBayes.sh 4 "assistance"
```

The output of the classifier using the single word "assistance" and using Laplace smoothing yields a classification success rate of $60/100 = 60\%$.

1.12.2 HW 1.4: Classify using the single word "assistance", "viagra" and "enlargementWithATypo" and report results

These instructions weren't entirely clear as to what was meant by "enlargementWithATypo". I couldn't find the occurance of any form or portions of the word "enlargement" in the corpus. I decided to go with "enlargement" for this excercise.

```
In [227]: !./pNaiveBayes.sh 4 "assistance viagra enlargement"
```

The output of the classifier improved slightly to $62/100 = 62\%$

1.12.3 HW 1.5: Classify using all terms and report results

```
In [231]: !./pNaiveBayes.sh 4 "*"
```

The performance of the classifier improved to $99/100 = 99\%$

1.13 HW 1.6

- Run the Multinomial Naive Bayes algorithm (using default settings) from SciKit-Learn over the same training data used in HW1.5 and report the Training error (please note some data preparation might be needed to get the Multinomial Naive Bayes algorithm from SkiKit-Learn to run over this dataset)
- Run the Bernoulli Naive Bayes algorithm from SciKit-Learn (using default settings) over the same training data used in HW1.5 and report the Training error
- Please prepare a table to present your results
- Explain/justify any differences in terms of training error rates over the dataset in HW1.5 between your Multinomial Naive Bayes implementation (in Map Reduce) versus the Multinomial Naive Bayes implementation in SciKit-Learn (Hint: smoothing, which we will discuss in next lecture)

- Discuss the performance differences in terms of training error rates over the dataset in HW1.5 between the Multinomial Naive Bayes implementation in SciKit-Learn with the Bernoulli Naive Bayes implementation in SciKit-Learn

Let DF represent the training set in the following: Err(Model, DF) = |{(X, c(X)) ∈ DF : c(X) != Model(x)}| / |DF|

Where || denotes set cardinality; c(X) denotes the class of the tuple X in DF; and Model(X) denotes the class inferred by the Model "Model"

```
In [241]: from sklearn.feature_extraction.text import TfidfVectorizer
          from sklearn.naive_bayes import MultinomialNB, BernoulliNB

In [245]: X_train = []
          Y_train = []

          # replicate our mapper code here where we take the subject and body together
          # except now we grab the label field as well to use for the Y values
          with open('enronemail_1h.txt', 'rU') as infile:
              for line in infile.readlines():
                  fields = line.split('\t')
                  if len(fields) > 3:
                      # we are interested in subject and body but need to be resilient for missing fiel
                      text = '{0} {1}'.format(fields[2],fields[3])
                  elif len(fields) > 2:
                      text = fields[2]
                  # extract only words from the combined subject and body text
                  X_train.append(text)
                  Y_train.append(fields[1])

          # Use the TfidVectorizer to create the feature vectors
          # We should override the tokenizer regular expression to make it the same as what we used
          # in our poor man's mapper code
          vectorizer = TfidfVectorizer(token_pattern = "[\w']+")
          vf = vectorizer.fit(X_train,Y_train)

          clf = MultinomialNB()
          clf.fit(vf.fit_transform(X_train), Y_train)
          print 1.0 - clf.score(vf.fit_transform(X_train), Y_train)
```

```
0.0
```

```
In [247]: # The change in the tfidf vectorizer to use the same token regex as our mapper results in
          # a 2% increase of training error, from 16% to 18%
          clf = BernoulliNB()
          clf.fit(vf.fit_transform(X_train), Y_train)
          print 1.0 - clf.score(vf.fit_transform(X_train), Y_train)
```

```
0.18
```

1.14 Results

The following table summarizes our results of the various classification method training error:

Classification Methodology	Training error
map/reduce single word `assignment`	40%

Classification Methodology	Training error
map/reduce multiple word `assignment`, `vallium`, `enlargement`	38%
map/reduce all words	1%
scikit-learn MultinomialNB	0%
scikit-learn BernoulliNB	18%

In [ ]: