MIDS-W261-2016-HWK-Week01-Chan

January 18, 2016

MIDS W261: Machine Learning at Scale

Konniam Chan
Time of submission: 8:00pm PST
W261-3 Spring 2016
Week 1: Homework
January 18, 2016

HW1.0.0. Define big data. Provide an example of a big data problem in your domain of expertise.

Big data refers to data sets so large that traditional computing frameworks are inadequate. One example of big data in the field of financial markets, where prices fluctuate on the micro-second level. The volume and velocity of thousands of securities make short-term trading suitable only to companies with big data infrastructure.

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?

Irreducible error: This represents the noise (squared) in the data set. If we have multiple values (k of them) of y around the same x, we can estimate this quantity by taking the variance of y at x_i, then average for all x_i:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\frac{(y_{ik}-\bar{y}_i)^2}{K-1}$$

Bias: Construct B bootstrap replicates of training set T, by drawing from T with replacement, and of length N (total number of data points). For each B, fit a model and make a predict on the out-of-bag points. Now, for each x_i, there is a set of $g_{i1}...g_{ik}$ predictions, with an average prediction $\bar{g}_i$. The bias-squared can be calculated as:

$$\frac{1}{N}\sum_{i=1}^{N}(\bar{g}_i-y_i)^2$$

Variance: The variance can be calculated as:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\frac{(g_{ik}-\bar{g}_i)^2}{K-1}$$

Model selection:
For each polynomial model, calculate the expected out-of-sample error by summing the above 3 terms, i.e. $var + bias^2 + noise^2$. Pick the model with the lowest expected error.

HW1.1. 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.

A simple cell in the notebook with a print statmement with a "done" string will suffice here. (dont forget to include the Question Number and the quesition in the cell as a multiline comment!)

```
In [137]: print "done"
```

```
done
```

HW1.2. 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.

To do so, make sure that

- mapper.py counts all occurrences of a single word, and
- reducer.py collates the counts of the single word.

```
In [1]: # (Optional) Convert data file from Windows to Unix line breaks
        # Run this on the command line: vi -c "%s/^M/\r/g | wq" "enronemail_1h.txt"
        # with ^M = CTRL-V CTRL-M
```

```
In [140]: %%writefile mapper.py
          #!/usr/bin/env python
          ## mapper.py
          ## Author: Konniam Chan
          ## Description: mapper code for HW1.2
          import sys
          import re
          count = 0
          # Command line inputs
          filename = sys.argv[1]
          findword = sys.argv[2]
          findword_regex = re.compile(findword, re.IGNORECASE)
          # Add all occurrences of a single match
          with open (filename, "r") as myfile:
              for line in myfile.readlines():
                  count += len(findword_regex.findall(line))
          print count
```

```
Overwriting mapper.py
```

```
In [141]: %%writefile reducer.py
          #!/usr/bin/env python
          ## reducer.py
          ## Author: Konniam Chan
          ## Description: reducer code for HW1.2
          import sys
          sum = 0
          # Iterate through files with intermediate counts
          for filename in sys.argv[1:]:
              with open(filename, "r") as myfile:
                  for line in myfile.readlines():
                      sum += int(line)
          print sum
```

```
Overwriting reducer.py
```

```
In [142]: !chmod a+x mapper.py; chmod a+x reducer.py
```

```
In [143]: # Test mapper and reducer with 5 processes and a single word "assistance"
          !./pNaiveBayes.sh 5 "assistance"
          !cat enronemail_1h.txt.output
```

```
10
```

HW1.3. Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by a single, user-specified word using the multinomial Naive Bayes Formulation. Examine the word "assistance" and report your results.

To do so, make sure that

- mapper.py and
- reducer.py

that performs a single word Naive Bayes classification. For multinomial Naive Bayes, the Pr(X="assistance"|Y=SPAM) is calculated as follows:

the number of times "assistance" occurs in SPAM labeled documents / the number of words in documents labeled SPAM

NOTE if "assistance" occurs 5 times in all of the documents Labeled SPAM, and the length in terms of the number of words in all documents labeld as SPAM (when concatenated) is 1,000. Then Pr(X="assistance"|Y=SPAM) = 5/1000. Note this is a multinomial estimated of the class conditional for a Naive Bayes Classifier. No smoothing is needed in this HW.

```
In [2]: %%writefile mapper.py
        #!/usr/bin/env python
        ## mapper.py
        ## Author: Konniam Chan
        ## Description: mapper code for HW1.3
        import sys
        import re
        # Keep track of number of spam documents
        label_map = {"1":"spam", "0":"ham"}
        docs = {"spam":0, "ham":0}
        # Count of terms
        count_terms = {"spam":0, "ham":0}
        count_all = {"spam":0, "ham":0}
        # Assume one input word
        filename = sys.argv[1]
        findword = sys.argv[2]
        findword_regex = re.compile(findword, re.IGNORECASE)
        # Iterate over documents
        with open (filename, "r") as myfile:
            for line in myfile.readlines():
                # Split fields and count spam/ham docs
                if len(line.strip().split("\t")) == 4:
                    doc_id, label, subject, body = line.strip().split("\t")
                # Some lines have missing subjects
                else:
                    subject = "na"
                    doc_id, label, body = line.strip().split("\t")
                label = label_map[label]
```

```
            docs[label] += 1
            # Strip quotes and concatenate subject and body
            subject = subject.strip('"')
            body = body.strip('"')
            content = (subject + " " + body)
            # Count total terms and matched terms
            count_terms[label] += len(findword_regex.findall(content))
            count_all[label] += len(content.split())

    print '\t'.join(map(str, [docs["spam"], docs["ham"], count_all["spam"], count_all["ham"]]))
    print '\t'.join(map(str, [findword, count_terms["spam"], count_terms["ham"]]))
```

Overwriting mapper.py

In [3]:
```
%%writefile reducer.py
#!/usr/bin/env python
## reducer.py
## Author: Konniam Chan
## Description: reducer code for HW1.3
from __future__ import division
from collections import defaultdict
import sys

docs = defaultdict(int)
count_terms = {"spam": defaultdict(int), "ham": defaultdict(int)}
count_all = defaultdict(int)

# Iterate through files with intermediate counts
for filename in sys.argv[1:]:
    with open(filename, "r") as myfile:
        lines = myfile.readlines()
        spam_counts, word_counts = lines[0], lines[1:]
        # Sum total spam and ham docs, total term frequencies
        spam, ham, count_all_spam, count_all_ham = map(int, spam_counts.strip().split('\t'))
        docs["spam"] += spam
        docs["ham"] += ham
        count_all["spam"] += count_all_spam
        count_all["ham"] += count_all_ham
        # Sum individual term counts, assuming one word input
        for line in word_counts:
            findword, count_terms_spam, count_terms_ham = line.strip().split('\t')
            count_terms["spam"][findword] += int(count_terms_spam)
            count_terms["ham"][findword] += int(count_terms_ham)

# Establish priors
priors = {}
priors["spam"] = count_all["spam"] / (count_all["spam"] + count_all["ham"])
priors["ham"] = count_all["ham"] / (count_all["spam"] + count_all["ham"])

# NB classification
NB_probs = priors.copy()
for term in count_terms["spam"]:
    # Skip terms that don't exist in the training set
    if count_terms["spam"][term] == 0 and count_terms["ham"][term] == 0:
        continue
```

```
        NB_probs["spam"] *= count_terms["spam"][term] / count_all["spam"]
        NB_probs["ham"] *= count_terms["ham"][term] / count_all["ham"]
    predicted_class = "spam" if NB_probs["spam"] > NB_probs["ham"] else "ham"

    print ("The predicted class is {}.".format(predicted_class))
    print ("Estimated NB probabilities for spam: {:.4f} and ham: {:.4f}."
           .format(NB_probs["spam"]/(NB_probs["spam"]+NB_probs["ham"]),
                   NB_probs["ham"]/(NB_probs["spam"]+NB_probs["ham"])))
```

```
Overwriting reducer.py
```

```
In [4]: # Test NB classifier with 5 processes and a single word "assistance"
        !./pNaiveBayes.sh 5 "assistance"
        !cat enronemail_1h.txt.output
```

```
The predicted class is spam.
Estimated NB probabilities for spam: 0.8000 and ham: 0.2000.
```

HW1.4. Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by a list of one or more user-specified words. Examine the words "assistance", "valium", and "enlargementWithATypo" and report your results.

To do so, make sure that

- mapper.py counts all occurrences of a list of words, and
- reducer.py

performs the multiple-word multinomial Naive Bayes classification via the chosen list. No smoothing is needed in this HW.

```
In [5]: %%writefile mapper.py
        #!/usr/bin/env python
        ## mapper.py
        ## Author: Konniam Chan
        ## Description: mapper code for HW1.4
        import sys
        import re
        from collections import defaultdict
        # Keep track of number of spam documents
        label_map = {"1":"spam", "0":"ham"}
        docs = {"spam":0, "ham":0}
        # Count of terms
        count_terms = {"spam": defaultdict(int), "ham": defaultdict(int)}
        count_all = {"spam":0, "ham":0}
        # Assume multiple input words
        filename = sys.argv[1]
        findwords = sys.argv[2].strip().split()
        findwords_regex = {term: re.compile(term, re.IGNORECASE) for term in findwords}
        # Iterate over documents
        with open (filename, "r") as myfile:
            for line in myfile.readlines():
                # Split fields and count spam/ham docs
                if len(line.strip().split("\t")) == 4:
                    doc_id, label, subject, body = line.strip().split("\t")
                # Some lines have missing subjects
                else:
```

```
                subject = "na"
                doc_id, label, body = line.strip().split("\t")
            label = label_map[label]
            docs[label] += 1
            # Strip quotes and concatenate subject and body
            subject = subject.strip('"')
            body = body.strip('"')
            content = (subject + " " + body)
            # Count total terms and matched terms
            count_all[label] += len(content.split())
            for term, regex in findwords_regex.items():
                count_terms[label][term] += len(regex.findall(content))

    print '\t'.join(map(str, [docs["spam"], docs["ham"], count_all["spam"], count_all["ham"]]))
    for term in findwords:
        print '\t'.join(map(str, [term, count_terms["spam"][term], count_terms["ham"][term]]))
```

```
Overwriting mapper.py
```

In [6]:
```
%%writefile reducer.py
#!/usr/bin/env python
## reducer.py
## Author: Konniam Chan
## Description: reducer code for HW1.4
from __future__ import division
from collections import defaultdict
import sys

docs = defaultdict(int)
count_terms = {"spam": defaultdict(int), "ham": defaultdict(int)}
count_all = defaultdict(int)

# Iterate through files with intermediate counts
for filename in sys.argv[1:]:
    with open(filename, "r") as myfile:
        lines = myfile.readlines()
        spam_counts, word_counts = lines[0], lines[1:]
        # Sum total spam and ham docs, total term frequencies
        spam, ham, count_all_spam, count_all_ham = map(int, spam_counts.strip().split('\t'))
        docs["spam"] += spam
        docs["ham"] += ham
        count_all["spam"] += count_all_spam
        count_all["ham"] += count_all_ham
        # Iterate through all terms
        for line in word_counts:
            term, count_terms_spam, count_terms_ham = line.strip().split('\t')
            count_terms["spam"][term] += int(count_terms_spam)
            count_terms["ham"][term] += int(count_terms_ham)

# Establish priors
priors = {}
priors["spam"] = count_all["spam"] / (count_all["spam"] + count_all["ham"])
priors["ham"] = count_all["ham"] / (count_all["spam"] + count_all["ham"])

# NB classification
```

```
NB_probs = priors.copy()
for term in count_terms["spam"]:
    # Skip terms that don't exist in the training set
    if count_terms["spam"][term] == 0 and count_terms["ham"][term] == 0:
        continue
    NB_probs["spam"] *= count_terms["spam"][term] / count_all["spam"]
    NB_probs["ham"] *= count_terms["ham"][term] / count_all["ham"]
predicted_class = "spam" if NB_probs["spam"] > NB_probs["ham"] else "ham"

print ("The predicted class is {}.".format(predicted_class))
print ("Estimated NB probabilities for spam: {:.4f} and ham: {:.4f}."
       .format(NB_probs["spam"]/(NB_probs["spam"]+NB_probs["ham"]),
               NB_probs["ham"]/(NB_probs["spam"]+NB_probs["ham"])))
```

```
Overwriting reducer.py
```

In [7]:
```
# Test NB classifier with 5 processes and a single word "assistance"
!./pNaiveBayes.sh 5 "assistance"
!cat enronemail_1h.txt.output
```

```
The predicted class is spam.
Estimated NB probabilities for spam: 0.8000 and ham: 0.2000.
```

In [8]:
```
# Test NB classifier with 5 processes and a single word "valium"
!./pNaiveBayes.sh 5 "valium"
!cat enronemail_1h.txt.output
```

```
The predicted class is spam.
Estimated NB probabilities for spam: 1.0000 and ham: 0.0000.
```

In [9]:
```
# Test NB classifier with 5 processes and a single word "enlargementWithATypo"
!./pNaiveBayes.sh 5 "enlargementWithATypo"
!cat enronemail_1h.txt.output
```

```
The predicted class is spam.
Estimated NB probabilities for spam: 0.5768 and ham: 0.4232.
```

For words that aren't in the training data, the terms are simply skipped in the calculation of probabilities. The prediction would therefore be based on the priors only.

In [10]:
```
# Test NB classifier with 5 processes and multiple words "make a lot of money"
!./pNaiveBayes.sh 5 "make a lot of money"
!cat enronemail_1h.txt.output
```

```
The predicted class is spam.
Estimated NB probabilities for spam: 0.9926 and ham: 0.0074.
```

In [11]:
```
# Test NB classifier with 5 processes and multiple words "pay stub",
# that results in a ham prediction
!./pNaiveBayes.sh 5 "pay stub"
!cat enronemail_1h.txt.output
```

```
The predicted class is ham.
Estimated NB probabilities for spam: 0.4444 and ham: 0.5556.
```