CMSC 25025 / STAT 37601

Machine Learning and Large Scale Data Analysis

Assignment 3 Out: Tuesday, April 18, 2017
Due: Thursday, April 27, 2017 at 1:30 pm.

1. *Classification* (10 points)

   1.1 Suppose that $\mathbb{P}(Y = 1) = \mathbb{P}(Y = 0) = \frac{1}{2}$ and $X \,|\, Y = 0 \sim N(0, 1)$ and $X \,|\, Y = 1 \sim \frac{1}{2}N(-5, 1) + \frac{1}{2}N(5, 1)$.
   (a) Find expressions for the Bayes classifier and the Bayes risk.
   (b) What linear classifier minimizes the risk and what is its risk?

   1.2 Now suppose that $\mathbb{P}(Y = 1) = \mathbb{P}(Y = -1) = \frac{1}{2}$ and $X \,|\, Y = -1 \sim \text{Uniform}(-10, 5)$ and $X \,|\, Y = 1 \sim \text{Uniform}(-5, 10)$.
   (a) Find an expression for the Bayes classifier and find an expression for the Bayes risk.
   (b) Consider the linear classifier $h_\alpha(x) = \text{sign}(x - \alpha)$ where $\alpha \in \mathbb{R}$. What linear classifier minimizes the risk and what is its risk?
   (c) Compute the hinge risk $R_\phi(\beta) = \mathbb{E}(1 - Y\beta X)_+$, where $(\cdot)_+$ denotes positive part.

2. *Logistic regression* (10 points)

   (a) Show that if Newton's method is applied to the logistic regression log-likelihood, it leads to the reweighted least squares algorithm.
   (b) Show that if the data are perfectly linearly separable, the conditional maximum likelihood estimator for the logistic regression model does not exist. Comment on the behavior of the iteratively reweighted least squares algorithm.
   (c) Give a detailed derivation of the Newton algorithm for ridge logistic regression, using a penalty $\lambda\|\beta\|^2$. Compare it with the iteratively reweighted least squares algorithm and comment on the differences between the two.

LSDA

3. *Sparse coding of natural images and digits* (40 points)

   In this problem you will implement the sparse coding procedure first proposed by Olshausen and Field, as a possible computational mechanism underlying the evolution of neural representations in the visual cortex of mammals.[1] You will use the actual images used in this

[1] B. Olshausen and D. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature* 381, 607–609, 1996.

landmark paper. As discussed in class, sparse coding is an important conceptual component of deep learning, but one that can be understood from first (statistical) principles.

This problem has two parts. In the first part you will apply sparse coding to learn a codebook for small image patches, training over the ten images from the Olshausen and Field paper. In the second part, you will learn a codebook for the MNIST images of handwritten digits.

Let $\{y^{(i)}\}_{i=1,\ldots,N}$ be the data to be represented with respect to some learned basis, where each instance $y^{(i)} \in \mathbb{R}^n$ is an n-dimensional vector. The optimization problem is

$$\begin{aligned} \min_{\beta, X} \quad & \sum_{i=1}^{N} \left\{ \frac{1}{2n} \left\| y^{(i)} - X\beta^{(i)} \right\|_2^2 + \lambda \cdot \text{SparsityPenalty}(\beta^{(i)}) \right\} \\ \text{such that} \quad & \|X_j\|_2 \leq 1 \end{aligned}$$

Here X is an $n \times p$ matrix with columns X_j; these are the "dictionary" entries or basis vectors to be learned. It is not required that the basis vectors are orthogonal. The penalty on the coefficients $\beta^{(i)}$ encourages sparsity, so that each data vector $y^{(i)}$ is represented by only a small number of dictionary elements. Sparsity is important here. Intuitively, a feature is estimated on only those examples where it is important. This allows the features to specialize, and to capture salient properties of the data.

This optimization problem is not jointly convex in $\beta^{(i)}$ and X. However, for fixed X, each weight vector $\beta^{(i)}$ is obtained as penalized least squares regression.

For this problem you will implement a stochastic gradient procedure to learn the dictionary (or codebook). The algorithm proceeds as follows. First, initialize the codebook X randomly. Then iterate over the following steps.

(a) Sample a collection $\{y^{(i)}\}_{i=1}^{N}$ of N data points from some population. (This is the stochastic part of SGD in this case.)

(b) For each data point $y^{(i)}$, carry out a sparse regression onto the dictionary X, getting coefficients $\beta^{(i)}$. You can use the lasso (if you know what that is) or you can greedily add in one column at a time, until you get a total of (say) 10 selected codewords.[2]

(c) After fitting the coefficients over the randomly selected patches, the dictionary is updated according to

$$\begin{aligned} X_j &\longleftarrow X_j - \eta g_j \\ X_j &\longleftarrow \frac{X_j}{\|X_j\|} \quad (\text{if } \|X_j\| > 1) \end{aligned}$$

where the (stochastic) gradient is

$$g_j = -\sum_{i \in \text{batch}} \beta_j^{(i)} \Big(y^{(i)} - X\beta^{(i)} \Big).$$

[2] In each step, you add in the column (codeword) j that maximizes $\left| X_j^T r \right|$, where r is the current residual.

(You should verify all of these formulas!)

Part 1: Sparse coding of natural images

To run the sparse coding algorithm over the images, we have provided a function that selects random patches. This can be run on the Olshausen-Field images using the following code

```
import scipy.io
data =
scipy.io.loadmat('/project/cmsc25025/sparsecoding/IMAGES_RAW.mat')
images = data['IMAGESr']
# images is a 3D array of size [512,512,10]
# where the 10 images are of size 512 x 512

import matplotlib.pyplot as plt
plt.imshow(images[:,:,0], cmap='gray')
plt.axis('off')
plt.show()
```

This will show you the first image:

The following function may be useful for sampling patches (or you can write your own).

```
import random
import numpy
def sample_random_square_patches(image, num, width):
    patches =
    numpy.zeros([width,width,num]);
```

```
for k in range(num):
    i, j = random.sample(range(image.shape[0]-width),2)
    patches[:,:,k] = image[i:i+width,j:j+width]
return patches
```

Run this sparse coding scheme over the images. Monitor the convergence of the algorithm by displaying the codebook after several iterations of SGD. How long does it take to converge? Experiment with a step size η for your algorithm. Display the codebook after initialization, after convergence, and at several intermediate stages. Comment on your results. Are they consistent with the results presented in class? Show reconstructions of patches or entire images using the sparse representation.

Part 2. Sparse coding of MNIST

Now apply sparse coding to learn a dictionary for the MNIST digits. In this case you should use the entire image of a digit as a data point—don't select a subpatch of these images. As above, run SGD by sampling a batch of images. Initialize the codebook randomly and then SGD until the codebook has stabilized. Display the codebook, and give your best interpretation of what the codewords represent in the data. Compare the results to what you get with PCA. Give a discussion of your findings.

LSDA

4. *Stochastic gradient descent on beer reviews* (40 points)

In this problem, you will work with 2,924,163 beer reviews from `https://www.ratebeer.com`. One entry contains the text of the consumer's review together with ratings of *appearance, aroma, palate, style, taste*, and an *overall* score from 0 to 20. The data also include the name, id and brewer of the beer. You will give reviews binary label: 1 for a *positive* review with overall score at least 14, and 0 for a *negative* review.

Part 1: Data inspection.

To warm up, check the mean, median and standard deviation of the overall ratings for each beer and brewer. Do you think people have similar taste?

Part 2: Sentiment analysis.

Your first task to classify the sentiment of reviews from the text. Text can have neutral, mixed, sarcastic, or otherwise ambiguous sentiment. *Sentiment analysis*[3] can be challenging even for people.

Here is one example review that you will be working on:

```
I got this one in Indianapolis. The body was dark brown. The aroma
```

[3] `http://en.wikipedia.org/wiki/Sentiment_analysis`

```
+----------+-----+-------+--------------------+------+-------+------+--------------------+---------+-----+-----+
|appearance|aroma|beer_id|           beer_name|brewer|overall|palate|              review|review_id|style|taste|
+----------+-----+-------+--------------------+------+-------+------+--------------------+---------+-----+-----+
|       4.0|  6.0|  45842|John Harvards Sim...|  3084|   13.0|   3.0|On tap at the Spr...|        0|   17|  6.0|
|       4.0|  6.0|  45842|John Harvards Sim...|  3084|   13.0|   4.0|On tap at the Joh...|        1|   17|  7.0|
|       4.0|  5.0|  95213|John Harvards Cri...|  3084|   14.0|   3.0|UPDATED: FEB 19, ...|        2|   33|  6.0|
|       2.0|  4.0|  65957|John Harvards Fan...|  3084|    8.0|   2.0|On tap the Spring...|        3|   33|  4.0|
|       5.0|  8.0|  41336|John Harvards Van...|  3084|   16.0|   4.0|Springfield, PA l...|        5|   58|  7.0|
|       4.0|  5.0|  80424|John Harvards Ame...|  3084|   12.0|   3.0|On tap at the Spr...|        6|   73|  6.0|
|       2.0|  6.0|  51269|John Harvards Gra...|  3084|   14.0|   3.0|Sampled @ the Spr...|        7|   12|  7.0|
|       4.0|  8.0|  51269|John Harvards Gra...|  3084|   16.0|   3.0|Springfield... Po...|        8|   12|  7.0|
|       3.0|  8.0|  51269|John Harvards Gra...|  3084|   17.0|   4.0|UPDATED: FEB 19, ...|        9|   12|  8.0|
|       4.0|  4.0|  73033|John Harvards Bel...|  3084|   11.0|   2.0|UPDATED: FEB 19, ...|       10|   62|  5.0|
|       3.0|  5.0|  56415|John Harvards Cas...|  3084|   14.0|   3.0|UPDATED: FEB 19, ...|       11|   24|  7.0|
|       3.0|  6.0|  68538|John Harvards Yin...|  3084|   12.0|   3.0|On tap at Springf...|       13|   50|  6.0|
|       3.0|  5.0|  68538|John Harvards Yin...|  3084|    9.0|   2.0|On tap at the Spr...|       14|   50|  5.0|
|       4.0|  8.0|  21566|Barley Island Bar...|  1786|   15.0|   4.0|Handbottled from ...|       15|   66|  8.0|
|       4.0|  8.0|  21566|Barley Island Bar...|  1786|   15.0|   4.0|On tap at the Gre...|       16|   66|  8.0|
|       3.0|  7.0|  21566|Barley Island Bar...|  1786|   14.0|   3.0|UPDATED: JUL 7, 2...|       17|   66|  6.0|
|       4.0|  5.0|  21566|Barley Island Bar...|  1786|   13.0|   4.0|On cask at BI - A...|       18|   66|  4.0|
|       3.0|  7.0|  21566|Barley Island Bar...|  1786|   13.0|   3.0|Name: BA Count Ho...|       20|   66|  7.0|
|       3.0|  7.0|  21566|Barley Island Bar...|  1786|   14.0|   3.0|Aroma is sweet bo...|       21|   66|  7.0|
|       3.0|  6.0|  21566|Barley Island Bar...|  1786|   13.0|   3.0|Cask at GTMW.  Po...|       22|   66|  7.0|
+----------+-----+-------+--------------------+------+-------+------+--------------------+---------+-----+-----+
```

```
df = spark.read.json('/project/cmsc25025/beer_review/labeled.json')
df.show()
```

```
was caramel and chocolate with some pancakes. The taste was pancakes
with coffee and chocolate. Well made, but a little boring.
```

Other reviews seem easier to classify according to positive or negative sentiment:

```
I was surprised by this one. A really nice local sour beer.
Pours a great looking brown with a slight red hue. Not much head,
but a good amount of lacing. Aroma is nutty, with cherries and a
tartness to it. Very earthy. Flavor is nice and sour, lots of red
fruits and nuts, with just a hint of yeast. The wood really comes
into play here, and works nicely with the other elements. This one
is tasty for sure.
```

After loading the labeled data, proceed as follows.

(1) *Generating features.* You need to represent text reviews in terms of a vector of features (covariates). One simple but effective representation is to use membership in a fixed vocabulary. Suppose the vocabulary contains p words. For a given review, you normalize the text, and separate it into space-delimited tokens. For each of the tokens, if it is in the dictionary you have a one for the corresponding word in the feature vector, and you ignore it otherwise.

For example, suppose the review is

```
"I like this beer!  Thanks for sharing!  Yay!"
```

and after normalization you process this into the list

```
["i", "like", "this", "beer", "thanks", "for", "sharing", "yay"].
```

If "i", "this", "for" and "yay" are not in your dictionary, but the other words are, you have four active features

```
["like", "beer", "thanks", "sharing"].
```

So, this review would be a p-dimensional vector where four features are set to 1, and the rest are set to 0. You should drop a review if none of its words is in the dictionary.

We have processed a few vocabularies for you. The file `vocab_50.json` contains `word:id` pairs for 30,009 words. Those words appear at least 50 times across all the reviews. Similarly for `vocab_{10,20,30}.json`. The most common 50 words are thrown out. You are more than welcome to use your own vocabulary. To load our vocabulary, run

```
with open('/project/cmsc25025/beer_review/vocab_50.json', 'r') as f:
    vocab = json.load(f)
```

Note that most of your feature vectors are going to be very sparse, as most reviews have very few words. In order to speed up the computation, using a sparse vector representation for high dimensional data is crucial. We recommend `scipy.sparse.coo_matrix`[4] since it's easy to construct from scratch.

This is just an example of a simple representation. Feel free to use it. You can use other text features that you think may be more effective. Give a detailed description of your feature representation.

(2) *Stochastic gradient descent.* Your next job is to train an ℓ_2-regularized logistic regression classifier using stochastic gradient descent. Recall the SGD framework that was covered in class: the reviews are coming in one by one and processed on the fly. As you process a review, update your model using the (stochastic) gradient computed on that review. We will use a variant algorithm called *mini-batch* SGD:

i. Initialize the model with $\beta = 0$ (uniform).

ii. Randomly split the training data into mini-batches. Make one pass of the data, processing one mini-batch in every iteration. This is called one training epoch.

iii. Repeat the last step a few times.

Remember the role that the learning rate plays in SGD. Tiny learning rates lead to slow convergence, while large rates can impede convergence—it might make the objective value oscillate. Typical learning rates are of the form

$$c_1/t^{c_2},$$

where c_1, c_2 are positive constants and t is the iteration. You could try different values of c_1, c_2. You can also try adaptive learning rates[5] using `Adagrad` or `Adam`.

[4] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html#scipy.sparse.coo_matrix

[5] http://sebastianruder.com/optimizing-gradient-descent/index.html

To select the regularization parameter λ, you should hold out a subset of the training data as a validation set (see the next question for the proportion). Select the λ that yields the smallest validation error.

Implement stochastic gradient descent, plot the error rate and negative log-likelihood for both training and validation set as a function of iterations.

Part 3: Scores versus text.

Quantitatively measure our life is trendy and believed to be informative. Despite text reviews, the users also scored *appearance, aroma, palate, style, taste* of a beer. In this problem, you will check whether those scores could reflect people's opinion better than text.

Train another logistic regression model using those features. You should use the same SGD algorithm as before. Compare the model using score features with that using review text. In order to do this, split your data into training, validation and testing sets with proportion 0.7:0.15:0.15. Use the validation set to tune the regularization parameters, and retrain the model on the union of training and validation set. Finally, compute the prediction error on the testing set.

Which model predicts better? Is the representation you constructed for text more powerful, or are the scores? Why? Comment on your findings and discuss your thinking.

Part 4 (Competition): build your own linear classifier.

In this part, you are free to use any raw or engineered feature, except `beer id`, `beer name`, `brewer name` and `review id`. Don't use them to build features, either.

You are welcome to use your favorite *linear* classifier other than logistic regression. You can use machine learning libraries of `pyspark` or any other packages like `sklearn`. Please be careful to check the documentation to see whether the package you use does a nonlinear transformation implicitly. Only linear classifiers are allowed.

Describe your approach and predict on the 437,815 reviews in the unlabeled dataset `/project/cmsc25025/beer_review/unlabeled.json`.

Form a dictionary of `review_id:prediction` pairs in json, and dump it to a file:

```
with open('assn3_prob4_predict.json', 'w') as f:
    json.dump(your_prediction, f)
```

Submit this file. We will rank the classifiers and give an award to the best team.

Pass in your code and documentation as an IPython notebook `assn3_prob4.ipynb`, and your prediction as `assn3_prob4_predict.json`.

`chisubmit` instructions

Three `ASSIGNMENT_ID`s will be created in the `chisubmit` system: `a3` (Problems 1 & 2, pencil & paper), `a3sparsecoding` (Problem 3) and `a3beerreviews` (Problem 4)

Please name your submissions

- `assn3_theory.pdf` for `a3`
- `assn3_prob3.ipynb` for `a3sparsecoding`
- `assn3_prob4.ipynb` and `assn3_prob4_predict.json` for `a3beerreviews`.

If you work on LSDA problems with a partner, you need to register `a3sparsecoding` and `a3beerreviews` using the `chisubmit --partner` option. Please see the details at `http://chi.cs.uchicago.edu/chisubmit/students.html`.