# CSEP 546 Data Mining/Machine Learning, Winter 2014: Homework 1

Due: Monday, January $20^{th}$, beginning of class

## 1 Simpson's Paradox [14 points]

Imagine you and your friend, playing the slot machine in a casino. Having played on two separate machines for a while, you decide to switch machines between yourselves to measure for differences in "luck". The wins/losses of you and your friend for each machine are tabulated below.

| Machine 1 | Wins | Losses |
|---|---|---|
| You | 40 | 60 |
| Friend | 30 | 70 |

| Machine 2 | Wins | Losses |
|---|---|---|
| You | 210 | 830 |
| Friend | 14 | 70 |

Assuming that the outcome of playing the slot machine is independent of its history and that of the other machine, answer the following questions.

1. *(3 points)* Estimate the winning probability of you and your friend for each of the machines. Compare your winning probability with your friend's on different machines. Who is more likely to win ?
2. *(3 points)* Suppose both of you didn't keep track of the wins/losses for each machine, but only of the total in the casino. Estimate the overall winning probability of you and your friend in the casino (assume that there are only two slot machines in the casino). Who is more likely to win ?
3. *(5 points)* Compare your conclusions from (1) and (2). Can you explain this result both intuitively and theoretically (Hint: write down the relationship between the probabilities in (1) and (2)) ?
4. *(3 points)* When will the conclusions of (1) and (2) be the same ?

## 2 MLE [36 points]

This question uses a probability distribution called the Poisson distribution. A discrete random variable $X$ follows a Poisson distribution with parameter $\lambda$ if

$$X \sim \texttt{Pois}(\lambda) : \Longleftrightarrow \Pr(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}, k \in \{0, 1, 2, \ldots, \infty\}.$$

You are Bill's officemate in Seattle. Being a graduate student, Bill comes into the office every day, and for the sake of this experiment, you make it to the office everyday before him. On rainy days, Bill has to get an umbrella to keep from getting wet on his way from the parking lot. Suppose that the probability of Bill getting an umbrella on a rainy day is $\theta$.

Also suppose that the number of rainy days in a month, is distributed according to some Poisson distribution. The number of rainy days in different months are mutually independent. In order to account for the seasons, suppose that this distribution is given by $\texttt{Pois}(\lambda_1)$ for the months October-April, and by $\texttt{Pois}(\lambda_2)$

for the months May-September. Note that according to this model, there is a finite (but tiny) probability of there being more than 31 days of rain in any given month (yes, Seattle is that rainy!). Assume, if you must, for the sake of simplicity that, every month has countably infinite number of days :) [1].

Let $R_i$ be the random variable corresponding to the number of rainy days in month $i$, and let $D_i$ be the corresponding number of days on which Bill is drenched by the rain. For convenience, let $i = 1, \ldots, 7$ index Oct-April and $i = 8, \ldots, 12$ index May-Sep. Your observations of $\mathbf{R} = (R_i)_{i=1}^{12}, \mathbf{D} = (D_i)_{i=1}^{12}$ over the course of a year are given below [2][3]

| Month | Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Rainy Days ($R$) | 13 | 9 | 11 | 8 | 6 | 4 | 2 | 3 | 7 | 12 | 18 | 14 |
| Drenched Days ($D$) | 6 | 5 | 4 | 3 | 3 | 2 | 0 | 1 | 4 | 9 | 11 | 8 |

Assuming the above model, answer the following questions. Unless otherwise indicated, assume that the values of $\lambda_1, \lambda_2, \theta$ are given.

1. *(3 points)* What is the probability of there being 13 days of rain in the month of January ?
2. *(4 points)* What is the log-likelihood of any given observation $\mathbf{r} = (r_i)_{i=1}^{12}$ over the course of a year ?
3. *(6 points)* Give an expression for the Maximum Likelihood Estimate of $\lambda_1$ and $\lambda_2$ given a realization of $\mathbf{R}$.
4. *(4 points)* Compute the MLE of $\lambda_1$ and $\lambda_2$, for the given observation of $\mathbf{R}$ in the table.
5. *(4 points)* Given that there were $r_i$ rainy days in month $i$, what is probability of Bill having been drenched for $d_i$ days in the same month ? Does your answer depend on $\lambda_1, \lambda_2$ ? Explain.
6. *(8 points)* Give an expression for the joint log-likelihood of the realization $\mathbf{r} = (r_i)_{i=1}^{12}, \mathbf{d} = (d_i)_{i=1}^{12}$.
7. *(5 points)* Give an expression for the Maximum Likelihood Estimate of $\lambda_1, \lambda_2, \theta$, given a realization of $\mathbf{R}$ and $\mathbf{D}$.
8. *(2 points)* Compute the MLE of $\lambda_1, \lambda_2, \theta$, for the given observations of $\mathbf{R}, \mathbf{D}$ in the table.
9. *(Extra credit: 5 points)* Suppose you know that the number of rainy days in a month $R \sim \texttt{Pois}(\lambda)$, with $D$ depending on $R$ as before. What is probability that Bill is drenched for $d$ days in a month ? Note that you're not given the number of rainy days $r$ in the month. (Hint: $\sum_{n=0}^{\infty} \frac{x^n}{n!} = \mathtt{e}^x$).

# 3 Programming Question [50 points]

## 3.1 Implement coordinate descent to solve the Lasso

The Lasso is the problem of solving

$$\arg\min_{\mathbf{w}, w_0} \|\mathbf{X}\mathbf{w} + w_0 - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_1$$

Here $\mathbf{X}$ is an $N \times d$ matrix of data, $\mathbf{y}$ is an $N \times 1$ vector of response variables, $\mathbf{w}$ is a $d$ dimensional weight vector, $w_0$ is a scalar offset term, and $\lambda$ is a regularization tuning parameter.

For the programming part of this homework, you are to implement the coordinate descent method to solve the Lasso. Your solver should

- Include an offset term $w_0$ that is not regularized

[1] On a more serious note, you should realize that models are always approximate. For $\lambda = 15$ (the mean of $\texttt{Pois}(\lambda)$ is $\lambda$) , the probability of there being more than 31 rainy days is $\sim 10^{-4}$. The model, therefore, is more than adequate for the given data. Using this model would be silly if the data indicated that most days of the month were rainy.

[2] Variables in lower case denote realizations, whereas those in upper case correspond to random variables.

[3] Indexed vector notation: $(a_i)_{i=1}^n = (a_1\ a_2\ \ldots\ a_n)^T$.

- Take optional initial conditions for $\mathbf{w}$ and $w_0$
- Be able to handle both dense and sparse $\mathbf{X}$ matrices
- Avoid unnecessary computation

You may use any language for your implementation, but we recommend Python. Python is a very useful language, and you should find that Python achieves reasonable enough performance for this problem. You may use common computing packages (such as NumPy or SciPy), but please, do not use an existing Lasso solver.

For a description of the algorithm, please refer to Murphy page 441. Note that we recommend you initialize the algorithm slightly differently (see the fourth hint below).

Before you get started, here are some hints that you may find helpful:

- With the exception of computing objective values or initial conditions, the only matrix operations required are adding vectors, multiplying a vector by a scalar, and computing the dot product between two vectors. If you find you are doing many large matrix operations, there is likely a more efficient implementation.
- To ensure that a solution $(\hat{\mathbf{w}}, \hat{w}_0)$ is correct, you can compute the vector

$$2\mathbf{X}^T(\mathbf{X}\hat{\mathbf{w}} + \hat{w}_0 - \mathbf{y})$$

  This is a $d$-dimensional vector that should take the value $-\lambda \text{sign}(\hat{\mathbf{w}}_j)$ at $j$ for each $\hat{\mathbf{w}}_j$ that is nonzero. For the zero indices of $\hat{\mathbf{w}}$, this vector should take values lesser in magnitude than $\lambda$. (This is similar to setting the gradient to zero, though more complicated because the objective function is not differentiable.) Another simple check is to ensure the objective value nonincreasing with each step.
- It is up to you to decide on a suitable stopping condition. A common criteria is to stop when no element of $\mathbf{w}$ changes by more than a value $\delta$ during an iteration. If you need your algorithm to run faster, an easy place to start is to loosen this condition.
- For several problems, you will need to solve the Lasso on the same dataset for many values of $\lambda$. This is called a regularization path. One way to do this efficiently is to start at a large $\lambda$, and then for each consecutive solution, initialize the algorithm with the previous solution, decreasing $\lambda$ by a constant ratio until finished.
- The smallest value of $\lambda$ for which the solution $\hat{\mathbf{w}}$ is entirely zero is given by

$$\lambda_{max} = 2 \left\| \mathbf{X}^T (y - \bar{y}) \right\|_\infty$$

  This is helpful for choosing the first $\lambda$ in a regularization path.

Finally here are some pointers toward useful parts of Python:

- `numpy`, `scipy.sparse`, and `matplotlib` are useful computation packages.
- For storing sparse matrices, the `scipy.sparse.csc_matrix` (compressed sparse column) format is fast for column operations.
- `scipy.io.mmread` reads sparse matrices in Matrix Market Format.
- `numpy.random.randn` is nice for generating random Gaussian arrays.
- `numpy.linalg.lstsq` works for solving unregularized least squares.
- If you're new to Python but experienced with Matlab, consider reading NumPy for Matlab Users at `http://wiki.scipy.org/NumPy_for_Matlab_Users`.

### 3.2 Try out your work on synthetic data

We will now try out your solver with some synthetic data. A benefit of the Lasso is that if we believe many features are irrelevant for predicting $\mathbf{y}$, the Lasso can be used to enforce a sparse solution, effectively differentiating between the relevant and irrelevant features.

Let's see if it actually works. Suppose that $\mathbf{x} \in \mathbb{R}^d, y \in \mathbb{R}, k < d$, and pairs of data $(\mathbf{x}_i, y_i)$ are generated independently according to the model

$$y_i = w_0^* + w_1^* x_{i,1} + w_2^* x_{i,2} + \ldots w_k^* x_{i,k} + \epsilon_i$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is some Gaussian noise. Note that since $k < d$, the features $k+1$ through $d$ are unnecessary (and potentially even harmful) for predicting $y$.

With this model in mind, you are now tasked with the following:

1. *(7 points)* Let $N = 50, d = 75, k = 5$, and $\sigma = 1$. Generate some data by drawing each element of $\mathbf{X} \in \mathbb{R}^{N \times d}$ from a standard normal distribution $\mathcal{N}(0, 1)$. Let $w_0^* = 0$ and create a $\mathbf{w}^*$ by setting the first $k$ elements to $\pm 10$ (choose any sign pattern) and the remaining elements to 0. Finally, generate a Gaussian noise vector $\epsilon$ with variance $\sigma^2$ and form $\mathbf{y} = \mathbf{X}\mathbf{w}^* + w_0^* + \epsilon$. You may use the following code to generate data.

```
##########################################################################
from scipy import *

def generate_data(N, d, k, sigma, seed=12231):
    """
    (y, X, w_ori, eps) = generate_data(N, d, k, sigma, seed=12231)

    Generate data for linear regression.
    y = (X * w_ori[1:] + w_ori[0]) + eps

    where
    X_{ij} ~ N(0, 1)
    w_ori[0] = 0, w_ori[1:d+1] = +/- 1
    eps ~ N(0, sigma^2)
    """
    random.seed(seed)

    X = randn(N, d)

    wg = zeros(1 + d)
    wg[1:k + 1] = 10 * sign(randn(k))
    eps = randn(N) * sigma
    y = X.dot(wg[1:]) + wg[0] + eps

    return (y, X, wg, eps)
##########################################################################
```

   With your synthetic data, solve multiple Lasso problems on a regularization path, starting at $\lambda_{max}$, then decreasing $\lambda$ by a constant ratio until few features are chosen correctly. Compare the sparsity pattern of your Lasso solution $\hat{\mathbf{w}}$ to that of the true model parameters $\mathbf{w}^*$. Record values for precision (number of correct nonzeros in $\hat{\mathbf{w}}$/total number of nonzeros in $\hat{\mathbf{w}}$) and recall (number of correct nonzeros in $\hat{\mathbf{w}}$/k).

   How well are you able to discover the true nonzeros? Comment on how $\lambda$ affects these results and include plots of precision and recall vs. $\lambda$.

2. *(6 points)* Change $\sigma$ to 10, regenerate the data, and solve the Lasso problem using a value of $\lambda$ that worked well for the case when $\sigma = 1$. Run the code a few times. What happens? How are precision and recall affected? How might you change $\lambda$ in order to achieve better precision or recall?

3. *(7 points)* Set $\sigma$ back to 1, and solve the Lasso for the following 6 sets of values:

$$(N = 50, d = 75) \quad (N = 100, d = 75)$$
$$(N = 50, d = 150) \quad (N = 100, d = 150)$$
$$(N = 50, d = 5000) \quad (N = 100, d = 5000)$$

Use a regularization path, and briefly discuss the precision and recall results for each problem. Based on your results, what might you guess the sample complexity is for recovering the sparsity pattern using the Lasso? Possible answers include $N = \mathcal{O}(d^2)$, $N = \mathcal{O}(d)$, or $N = \mathcal{O}(\log(d))$. If needed, try additional values for $N$ and $d$ or repeat the experiment multiple times.

(What we're getting at is if $d$ increases substantially, how does $N$ need to increase in order to still achieve good performance? Why might this be significant?)

### 3.3 Become a data scientist at Yelp

We'll now put the Lasso to work on some real data. Recently Yelp held a recruiting competition on the analytics website Kaggle. Check it out at `http://www.kaggle.com/c/yelp-recruiting`. (As a side note, browsing other competitions on the site may also give you some ideas for class projects.)

For this competition, the task is to predict the number of useful upvotes a particular review will receive. For extra fun, we will add the additional task of predicting the review's number of stars based on the review's text alone.

For many Kaggle competitions (and machine learning methods in general), one of the most important requirements for doing well is the ability to discover great features. We can use our Lasso solver for this as follows. First, generate a large amount of features from the data, even if many of them are likely unnecessary. Afterward, use the Lasso to reduce the number of features to a more reasonable amount.

Yelp provides a variety of data, such as the review's text, date, and restaurant, as well as data pertaining to each business, user, and check-ins. This information has already been preprocessed for you into the following files on the course website:

| | |
|---|---|
| `upvote_data.csv` | Data matrix for predicting number of useful votes |
| `upvote_labels.txt` | List of useful vote counts for each review |
| `upvote_features.txt` | Names of each feature for interpreting results |
| `star_data.mtx` | Data matrix for predicting number of stars |
| `star_labels.txt` | List of number of stars given by each review |
| `star_features.txt` | Names of each feature |

For each task, data files contain data matrices, while labels are stored in separate text files. The first data matrix is stored in CSV format, each row corresponding to one review. The second data matrix is stored in Matrix Market Format, a format for sparse matrices. Meta information for each feature is provided in the final text files, one feature per line. For the upvote task, these are functions of various data attributes. For the stars task, these are strings of one, two, or three words (n-grams). The feature values correspond roughly to how often each word appears in the review. All columns have also been normalized.

To get you started, the following code should load the data:

```
##############################################################################
import numpy as np
import scipy.io as io
import scipy.sparse as sparse

# Load a text file of integers:
y = np.loadtxt("upvote_labels.txt", dtype=np.int)

# Load a text file of strings:
featureNames = open("upvote_features.txt").read().splitlines()

# Load a csv of floats:
A = np.genfromtxt("upvote_data.csv", delimiter=",")

# Load a matrix market matrix, convert it to csc format:
```

```
B = io.mmread("star_data.mtx").tocsc()
##########################################################################
```

For this part of the problem, you have the following tasks:

1. *(6 points)* Solve lasso to predict the number of useful votes a Yelp review will receive. Use the first 4000 samples for training, the next 1000 samples for validation, and the remaining samples for testing.

   Starting at $\lambda_{max}$, run Lasso on the training set, decreasing $\lambda$ using previous solutions as initial conditions to each problem. Stop when you have considered enough $\lambda$'s that, based on validation error, you can choose a good solution with confidence (for instance, when validation error begins increasing or stops decreasing significant). At each solution, record the root-mean-squared-error (RMSE) on training and validation data. In addition, record the number of nonzeros in each solution.

   Plot the RMSE values together on a plot against $\lambda$. Separately plot the number of nonzeros as well.

2. *(3 points)* Find the $\lambda$ that achieves best validation performance, and test your model on the remaining set of test data. What RMSE value do you obtain?

3. *(3 points)* Inspect your solution and take a look at the 10 features with weights largest in magnitude. List the names of these features and their weights, and comment on if the weights generally make sense intuitively.

4. *(6 points)* A significant issue with the Lasso is that in order to shrink a large amount of features to zero, the $\ell_1$-regularization introduces a large amount of bias to the nonzero weights. As a result, we can often achieve better results by debiasing the estimate (see Murphy page 439). This is done by first solving the Lasso to choose a sparsity pattern. Afterward, we solve a least squares problem on only the nonzero features and use the resulting weights in our final parameter vector.

   Run Lasso on a regularization path. At each value of $\lambda$, debias the weight vector learned by Lasso. Record RMSE values on validation data both before and after debiasing.

   Plot RMSE values both with and without debiasing together vs. $\lambda$. Does performance change with debiasing? How does the value of $\lambda$ affect this outcome?

5. *(12 points)* Repeat parts 1, 2, and 3 using the data matrix and labels for predicting the score of a review. To avoid using too much memory, your algorithm should keep the matrix in a sparse format. Use the first 30,000 examples for training and divide the remaining samples between validation and testing.

# 4 Linear Regression and LOOCV [Extra credit: 10 points]

In class you learned about using cross validation as a way to estimate the true error of a learning algorithm. A solution that provides an almost unbiased estimate of this true error is *Leave-One-Out Cross Validation* (LOOCV), but it can take a really long time to compute the LOOCV error. In this problem you will derive an algorithm for efficiently computing the LOOCV error for linear regression using the *Hat Matrix*. [4] (This is the *cool trick* alluded to in the slides!)

Assume that there are $n$ given training examples, $(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n)$, where each input data point $X_i$, has $d$ real valued features. The goal of regression is to learn to predict $y_i$ from $X_i$. The *linear* regression model assumes that the output $y$ is a *linear* combination of the input features plus Gaussian noise with weights given by $w$.

We can write this in matrix form by stacking the data points as the rows of a matrix $X$ so that $x_i^{(j)}$ is the $i$-th feature of the $j$-th data point. Then writing $Y$, $w$ and $\epsilon$ as column vectors, we can write the matrix form of the linear regression model as:

$$Y = Xw + \epsilon$$

where:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \ldots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \ldots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(3)} & x_2^{(3)} & \ldots & x_d^{(3)} \end{bmatrix}, w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}, \quad \text{and } \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

Assume that $\epsilon_i$ is normally distributed with variance $\sigma^2$. We saw in class that the maximum likelihood estimate of the model parameters $w$ (which also happens to minimize the sum of squared prediction errors) is given by the *Normal equation*:

$$\hat{w} = (X^T X)^{-1} X^T Y$$

[4] Unfortunately, such an efficient algorithm may not be easily found for other learning methods.

Define $\hat{Y}$ to be the vector of predictions using $\hat{w}$ if we were to plug in the original training set $X$:

$$\begin{aligned}\hat{Y} &= X\hat{w} \\ &= X(X^TX)^{-1}X^TY \\ &= HY\end{aligned}$$

where we define $H = X(X^TX)^{-1}X^T$ ($H$ is often called the *Hat Matrix*).

As mentioned above, $\hat{w}$, also minimizes the sum of squared errors:

$$\text{SSE} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Now recall that the Leave-One-Out Cross Validation score is defined to be:

$$\text{LOOCV} = \sum_{i=1}^{n}(y_i - \hat{y}_i^{(-i)})^2$$

where $\hat{Y}^{(-i)}$ is the estimator of $Y$ after removing the $i$-th observation (i.e., it minimizes $\sum_{j\neq i}(y_j - \hat{y}_j^{(-i)})^2$).

1. *(1 points)* What is the time complexity of computing the LOOCV score naively? (The naive algorithm is to loop through each point, performing a regression on the $n-1$ remaining points at each iteration.)
   *Hint:* The complexity of matrix inversion is $O(k^3)$ for a $k \times k$ matrix [5].
2. *(1 point)* Write $\hat{y}_i$ in terms of $H$ and $Y$.
3. *(3 points)* Show that $\hat{Y}^{(-i)}$ is also the estimator which minimizes SSE for $Z$ where

$$Z_j = \begin{cases} y_j, & j \neq i \\ \hat{y}_i^{(-i)}, & j = i \end{cases}$$

4. *(1 point)* Write $\hat{y}_i^{(-i)}$ in terms of $H$ and $Z$. By definition, $\hat{y}_i^{(-i)} = Z_i$, but give an answer that is analogous to *2.*
5. *(2 points)* Show that $\hat{y}_i - \hat{y}_i^{(-i)} = H_{ii}y_i - H_{ii}\hat{y}_i^{(-i)}$, where $H_{ii}$ denotes the $i$-th element along the diagonal of $H$.
6. *(2 points)* Show that

$$LOOCV = \sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - H_{ii}}\right)^2$$

   What is the algorithmic complexity of computing the LOOCV score using this formula?
   Note: We see from this formula that the diagonal elements of $H$ somehow indicate the impact that each particular observation has on the result of the regression.

[5] There are faster algorithms out there but for simplicity we'll assume that we are using the naive $O(k^3)$ algorithm.