Your Name
email@fas.harvard.edu
CS281-F17

Assignment #1 v 1.1

Due: 11:59pm September 22, 2017

Collaborators: John Doe, Fred Doe

Problem 1 (A Classic on the Gaussian Algebra, 10pts)

Let X and Y be independent univariate Gaussian random variables. In the previous problem set, you likely used the closure property that $Z = X + Y$ is also a Gaussian random variable. Here you'll prove this fact.

(a) Suppose X and Y have mean 0 and variances σ_X^2 and σ_Y^2 respectively. Write the pdf of $X + Y$ as an integral.

(b) Evaluate the integral from the previous part to find a closed-form expression for the pdf of $X+Y$, then argue that this expression implies that $X + Y$ is also Gaussian with mean 0 and variance $\sigma_X^2 + \sigma_Y^2$. Hint: what is the integral, over the entire real line, of

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right),$$

i.e., the pdf of a univariate Gaussian random variable?

(c) Extend the above result to the case in which X and Y may have arbitrary means.

(d) Univariate Gaussians are supported on the entire real line. Sometimes this is undesirable because we are modeling a quantity with positive support. A common way to transform a Gaussian to solve this problem is to exponentiate it. Suppose X is a univariate Gaussian with mean μ and variance σ^2. What is the pdf of e^X?

Problem 2 (Regression, 13pts)

Suppose that $X \in \mathbb{R}^{n \times m}$ with $n \geq m$ and $Y \in \mathbb{R}^n$, and that $Y \sim \mathcal{N}(Xw, \sigma^2 I)$. You learned in class that the maximum likelihood estimate $\hat{w}$ of w is given by

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

(a) Why do we need to assume that $n \geq m$?

(b) Define $H = X(X^T X)^{-1} X^T$, so that the "fitted" values $\hat{Y} = X\hat{w}$ satisfy $\hat{Y} = HY$. Show that H is an orthogonal projection matrix that projects onto the column space of X, so that the fitted y-values are a projection of Y onto the column space of X.

(c) What are the expectation and covariance matrix of $\hat{w}$?

(d) Compute the gradient with respect to w of the log likelihood implied by the model above, assuming we have observed Y and X.

(e) Suppose we place a normal prior on w. That is, we assume that $w \sim \mathcal{N}(0, \tau^2 I)$. Show that the MAP estimate of w given Y in this context is

$$\hat{w}_{MAP} = (X^T X + \lambda I)^{-1} X^T Y$$

where $\lambda = \sigma^2/\tau^2$. (You may employ standard conjugacy results about Gaussians without proof in your solution.)

[Estimating w in this way is called *ridge regression* because the matrix λI looks like a "ridge". Ridge regression is a common form of *regularization* that is used to avoid the overfitting (resp. underdetermination) that happens when the sample size is close to (resp. higher than) the output dimension in linear regression.]

(f) Do we need $n \geq m$ to do ridge regression? Why or why not?

(g) Show that ridge regression is equivalent to adding m additional rows to X where the j-th additional row has its j-th entry equal to $\sqrt{\lambda}$ and all other entries equal to zero, adding m corresponding additional entries to Y that are all 0, and then computing the maximum likelihood estimate of w using the modified X and Y.

Problem 3 (The Dirichlet and Multinomial Distributions, 12pts)

The Dirichlet distribution over K categories is a generalization of the beta distribution. It has a shape parameter $\alpha \in \mathbb{R}^K$ with non-negative entries and is supported over the set of K-dimensional positive vectors whose components sum to 1. Its density is given by

$$f(\theta_{1:K}|\alpha_{1:K}) = \frac{\Gamma\left(\sum_k \alpha_k\right)}{\prod_k \Gamma(\alpha_k)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1}$$

(Notice that when $K = 2$, this reduces to the density of a beta distribution.) For the rest of this problem, assume a fixed $K \geq 2$.

(a) Suppose θ is Dirichlet-distributed with shape parameter α. Without proof, state the value of $E(\theta)$. Your answer should be a vector defined in terms of either α or K or potentially both.

(b) Suppose that $\theta \sim \text{Dir}(\alpha)$ and that $X \sim \text{Cat}(\theta)$, where *Cat* is a Categorical distribution. That is, suppose we first sample a K-dimensional vector θ with entries in $(0, 1)$ from a Dirichlet distribution and then roll a K-sided die such that the probability of rolling the number k is θ_k. Prove that the posterior $p(\theta|X)$ also follows a Dirichlet distribution. What is its shape parameter?

(c) Now suppose that $\theta \sim \text{Dir}(\alpha)$ and that $X^{(1)}, X^{(2)}, \ldots \overset{iid}{\sim} \text{Cat}(\theta)$. Show that the posterior predictive after $n - 1$ observations is given by,

$$P(X^{(n)} = k|X^{(1)}, \ldots, X^{(n-1)}) = \frac{\alpha_k^{(n)}}{\sum_k \alpha_k^{(n)}}$$

where for all k, $\alpha_k^{(n)} = \alpha_k + \sum_{i=1}^{n-1} \mathbf{1}\{X^{(i)} = k\}$. (Bonus points if your solution does not involve any integrals.)

(d) Consider the random vector $Z_k = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{X^{(i)} = k\}$ for all k. What is the mean of this vector? What is the distribution of the vector? (If you're not sure how to rigorously talk about convergence of random variables, give an informal argument. Hint: what would you say if θ were fixed?) What is the marginal distribution of a single class $p(Z_k)$?

(e) Suppose we have K distinct colors and an urn with α_k balls of color k. At each time step, we choose a ball uniformly at random from the urn and then add into the urn an additional new ball of the same color as the chosen ball. (So if at the first time step we choose a ball of color 1, we'll end up with $\alpha_1 + 1$ balls of color 1 and α_k balls of color k for all $k > 1$ at the start of the second time step.) Let $\rho_k^{(n)}$ be the fraction of all the balls that are of color k at time n. What is the distribution of $\lim_{n\to\infty} \rho_k^{(n)}$? Prove your answer.

Physicochemical Properties of Protein Tertiary Structure

In the following problems we will code two different approaches for solving linear regression problems and compare how they scale as a function of the dimensionality of the data. We will also investigate the effects of linear and non-linear features in the predictions made by linear models.

We will be working with the regression data set Protein Tertiary Structure: `https://archive.ics.uci.edu/ml/machine-learning-databases/00265/` (download CASP.csv). This data set contains information about predicted conformations for 45730 proteins. In the data, the target variable y is the root-mean-square deviation (RMSD) of the predicted conformations with respect to the true properly folded form of the protein. The RMSD is the measure of the average distance between the atoms (usually the backbone atoms) of superimposed proteins. The features $\mathbf{x}$ are physico-chemical properties of the proteins in their true folded form. After downloading the file CASP.csv we can load the data into python using

```
>>> import numpy as np
>>> data = np.loadtxt("CASP.csv", delimiter = ",", skiprows = 1)
```

We can then obtain the vector of target variables and the feature matrix using

```
>>> y = data[:, 0]
>>> X = data[:, 1:]
```

We can then split the original data into a training set with 90% of the data entries in the file CASP.csv and a test set with the remaining 10% of the entries. Normally, the splitting of the data is done at random, but here **we ask you to put into the training set the first 90% of the elements from the file CASP.csv** so that we can verify that the values that you will be reporting are correct. (This should not cause problems, because the rows of the file are in a random order.)

We then ask that you **normalize** the features so that they have zero mean and unit standard deviation in the training set. This is a standard step before the application of many machine learning methods. After these steps are done, we can concatenate a **bias feature** (one feature which always takes value 1) to the observations in the normalized training and test sets.

We are now ready to apply our machine learning methods to the normalized training set and evaluate their performance on the normalized test set. In the following problems, you will be asked to report some numbers and produce some figures. Include these numbers and figures in your assignment report. **The numbers should be reported with up to 8 decimals**.

Problem 4 (7pts)

Assume that the targets y are obtained as a function of the normalized features $\mathbf{x}$ according to a Bayesian linear model with additive Gaussian noise with variance $\sigma^2 = 1.0$ and a Gaussian prior on the regression coefficients $\mathbf{w}$ with *precision* matrix $\Sigma^{-1} = \tau^{-2}\mathbf{I}$ where $\tau^{-2} = 10$. Code a routine using the **QR decomposition** (see Section 7.5.2 in Murphy's book) that finds the Maximum a Posteriori (MAP) value $\hat{\mathbf{w}}$ for $\mathbf{w}$ given the normalized training data

- Report the value of $\hat{\mathbf{w}}$ obtained.
- Report the root mean squared error (RMSE) of $\hat{\mathbf{w}}$ in the normalized test set.

Problem 5 (14pts)

L-BFGS is an iterative method for solving general nonlinear optimization problems. For this problem you will use this method as a black box that returns the MAP solution by sequentially evaluating the objective function and its gradient for different input values. The goal of this problem is to use a built-in implementation of the L-BFGS algorithm to find a point estimate that maximizes our posterior of interest. Generally L-BFGS requires your black box to provide two values: the current objective and the gradient of the objective with respect to any parameters of interest. To use the optimizer, you need to first write two functions: (1) to compute the loss, or the *negative* log-posterior and (2) to compute the gradient of the loss with respect to the weights w.

As a preliminary to coming work in the class, we will use the L-BFGS implemented in PyTorch. [Warning: For this assignment we are using a small corner of the PyTorch world. Do not feel like you need to learn everything about this library.]

There are three parts to using this optimizer:

1. Create a vector of weights in NumPy, wrap in a pytorch `Tensor` and `Variable`, and pass to the optimizer.

```
from torch import Tensor
from torch.autograd import Variable

# Construct a PyTorch variable array (called tensors).
weights = Variable(Tensor(size))

# Initialize an optimizer of the weights
optimizer = torch.optim.LBFGS([weights])

...
```

2. Write a python function that uses the current weights to compute the log-posterior **and** sets weights.grad to be the gradient of the log-posterior with respect to the current weights.

```
def black_box():
    # Access the value of the variable as a numpy array.
    weights_data = weights.data.numpy()

    ...

    # Set the gradient of the variable.
    weights.grad = Tensor({numpy})

    return {objective}
```

3. Repeatedly call `optimizer.step(black_box)` to optimize.

[If you are feeling adventurous, you might find it useful to venture into the land of autograd and check your computation with PyTorch's `torch.autograd.gradcheck.get_numerical_jacobian`.]

- After running for 100 iterations, report the value of $\hat{\mathbf{w}}$ obtained.
- Report the RMSE of the predictions made with $\hat{\mathbf{w}}$ in the normalized test set.

Problem 6 (14pts)

Linear regression can be extended to model non-linear relationships by replacing the original features $\mathbf{x}$ with some non-linear functions of the original features $\boldsymbol{\phi}(\mathbf{x})$. We can automatically generate one such non-linear function by sampling a random weight vector $\mathbf{a} \sim \mathcal{N}(0, \mathbf{I})$ and a corresponding random bias $b \sim \mathrm{U}[0, 2\pi]$ and then making $\phi(\mathbf{x}) = \cos(\mathbf{a}^{\mathrm{T}}\mathbf{x} + b)$. By repeating this process d times we can generate d non-linear functions that, when applied to the original features, produce a non-linear mapping of the data into a new d dimensional space. We can encode these d functions into a matrix $\mathbf{A}$ with d rows, each one with the weights for each function, and a d-dimensional vector $\mathbf{b}$ with the biases for each function. The new mapped features are then obtained as $\boldsymbol{\phi}(\mathbf{x}) = \cos(\mathbf{A}\mathbf{x} + \mathbf{b})$, where cos applied to a vector returns another vector whose elements are the result of applying cos to the individual elements of the original vector.

Generate 4 sets of non-linear functions, each one with $d = 100, 200, 400, 600$ functions, respectively, and use them to map the features in the original normalized training and test sets into 4 new feature spaces, each one of dimensionality given by the value of d. After this, for each value of d, find the MAP solution $\hat{\mathbf{w}}$ for $\mathbf{w}$ using the corresponding new training set and the method from problem 4. Use the same values for σ^2 and τ^{-2} as before. You are also asked to record the time taken by the method QR to obtain a value for $\hat{\mathbf{w}}$. In python you can compute the time taken by a routine using the time package:

```
>>> import time
>>> time_start = time.time()
>>> routine_to_call()
>>> running_time = time.time() - time_start
```

Next, compute the RMSE of the resulting predictor in the normalized test set. Repeat this process with the method from problem 5 (L-BFGS).

- Report the test RMSE obtained by each method for each value of d.

You are asked to generate a plot with the results obtained by each method (QR and L-BFGS) for each value of d. In this plot the x axis should represent the time taken by each method to run and the y axis should be the RMSE of the resulting predictor in the normalized test set. The plot should contain 4 points in red, representing the results obtained by the method QR for each value of d, and 4 points in blue, representing the results obtained by the method L-BFGS for each value of d. Answer the following questions:

- Do the non-linear transformations help to reduce the prediction error? Why?
- What method (QR or L-BFGS) is faster? Why?
- (Extra Problem, Not Graded) Instead of using random $\mathbf{A}$, what if we treat $\mathbf{A}$ as another parameter for L-BFGS to optimize? You can do this by wrapping it as a variable and passing to the constructor. Compute its gradient as well in *black_box* either analytically or by using PyTorch *autograd*.