CS452, Spring 2018, Problem Set # 1
Ashesi University

January 25, 2018

Due: February 13th 2018, 11:59pm

This problem set requires you to provide written answers to a few technical questions. In addition, you will need to implement some learning algorithms in Python and apply them to the enclosed data sets. The questions requiring a written answer are denoted with **W**, while the Python programming questions are marked with **P**.

Both the written answers as well as the code must be submitted electronically via Courseware.

For the written answers, you must provide a single PDF containing all your answers. Upload the PDF file through the appropriate link in Courseware. The PDF file can be generated using editing software (e.g., Latex) or can be a scanned copy of your hand-written answers. If you opt for scanned copies, please make sure to use a scanner, *not* your phone camera. Familiarize yourself with the process of scanning documents a few days before the due date. Late submissions due to scanning problems will be treated and penalized as any other form of late submission, as per the rules stated in the Syllabus. Also, make sure your scanned submission is readable. If we cannot read it clearly, you will receive 0 points for that problem.

Your Python code must also be submitted via Courseware. We have provided Python function files with an empty body for all the functions that you need to implement. Each file specifies in detail the input and the output arguments of the function. A few of the functions include some commands to help you get started with the implementation. The function stubs include also *import* commands for all the libraries that are needed for your implementation (e.g., numpy, math). Do not add any other import command, i.e., do not use any other libraries except those already specified in the function stubs. If in doubt, please ask. We have also provided scripts that check the sizes of the inputs and outputs of your functions. Do not modify these scripts. In order to allow us to test your code, it is imperative that you do not rename the files or modify the syntax of the functions. Make sure to include all the files necessary to run your code. **If we are unable to run one of your functions, you will receive 0 points for that question.** Please place all your files (unmodified scripts and checking functions, data, .png files automatically generated by the provided scripts, and your modified function files) in a single folder (**without subfolders**). Zip up the entire folder, and upload the compressed zip file to Courseware. Note that we have provided **two** separate links for your homework submission: one for your written answers, one for your code.

1. **[W, 10 points]** Consider one-dimensional random variables x and y. Show that if x and y are independent, their covariance is zero, i.e., $\mathbf{cov}(x, y) = 0$.

2. **[W, 12 points]** Suppose we have three boxes: r (red), g (green), and b (blue). Box r contains 3 apples, 4 oranges, and 3 limes; box g contains 3 apples, 3 oranges, and 4 limes; box b contains 1 apple, 2 oranges, and 0 limes. If a box is selected at random with probabilities $p(r) = 0.2, p(g) = 0.5, p(b) = 0.3$, and a piece of fruit is picked from the box (with equal probability of choosing any of the items in the box), then what is the probability of selecting an apple? If we observe that the selected fruit is an orange, what is the probability that it came from the green box?

3. **[30 points]** Let's flip a coin. Let $c \in \{0, 1\}$ be a random variable indicating the result of the flip (1 for heads, 0 for tails). The probability that the coin lands heads on any trial is given by a parameter μ. Note that we can write the distribution of c as: $P(c\ ; \mu) = \mu^c(1-\mu)^{1-c}$. We flip the coin m times, and denote the result of the i-th flip by variable $c^{(i)}$. We assume that the coin flips are *independent*. We observe heads H times.

   (a) [**W**, 3 points] Write the likelihood function, i.e., the probability of the data $\mathcal{D} = \{c^{(1)}, ..., c^{(m)}\}$ given the model described above. Keep in mind that the likelihood is the probability of the observed training set $\mathcal{D}$ (rather than all possible training sets containing H heads in m examples). Thus your likelihood function should *not* include a combinatorial term counting the number of different ways H heads could occur in m trials.

   (b) [**W**, 9 points] Derive the parameter μ using Maximum Likelihood estimation (hint: maximize the log likelihood).

   (c) [**P**, 2 point] Code the Python function `q3_likelihood.py` to compute the likelihood given scalar input arguments H, m and an input vector μ (the function should return a vector of likelihood values as big as μ). You can now inspect how the likelihood function differs for the three cases $\{m = 1, H = 1\}$, $\{m = 100, H = 100\}$, and $\{m = 100, H = 80\}$ by executing the script `q3c.py`, which calls your function `q3_likelihood.py`. To execute the script type "`run q3c`" in your Python shell.

   Let us now assume that we have good reasons to believe that the coin is not counterfeit. However, we are not certain. We model this prior knowledge in the form of a prior distribution over μ:

$$p(\mu; a) = \frac{1}{Z}\mu^{a-1}(1-\mu)^{a-1} \tag{1}$$

   where a is a parameter governing the prior distribution and Z is a normalization constant (so that $\int_0^1 p(\mu; a)d\mu = 1$).

   (d) [**P**, 2 point] Implement the prior function by coding the file `q3_prior.py`. Then, execute the script `q3d.py`, which plots the prior for $a = 2$ ($Z = 1/6$), and $a = 10$ ($Z = 1/923780$).

   (e) [**W**, 9 points] Assuming the prior $p(\mu; a)$ in Eq. 1, derive the analytical expression of parameter μ using Maximum A Posteriori (MAP) estimation (hint: maximize the log posterior and drop the term related to the evidence, i.e., the denominator in Bayes' rule).

(f) [**W**, 3 points] By looking at the MAP estimate derived in the previous point, can you provide an interpretation of parameter a in terms of training examples?

(g) [**P**, 2 points] Implement the posterior by coding the file `q3_posterior.py`, and plot the posterior for the same coin flipping results considered above (i.e., $\{m = 1, H = 1\}$, $\{m = 100, H = 100\}$, and $\{m = 100, H = 80\}$), by running the script `q3g.py`. Do not include the evidence term (i.e., the denominator) in the posterior calculation. There is no way to estimate the evidence and in any case it is just a constant. Make sure to plot the posterior, not the log posterior.

4. [**48 points**] Write Python code implementing a regression algorithm for multi-dimensional inputs x and 1D outputs y. Your software must learn the regression hypothesis by minimizing the regularized least-square objective

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left[y^{(i)} - \theta^\top b(x^{(i)})\right]^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2 \tag{2}$$

where $\theta = [\theta_0, \theta_1, ..., \theta_d]^\top$ and $b(x)$ is a vector that encodes either of the following two distinct choices of features:

- $b^l(x) = [1, x_1, ..., x_d]^\top$ (where, d is the number of input entries in vector x and x_j denotes the j-th element of vector x), or

- $b^q(x) = [1, x_1, ..., x_d, x_1^2, x_1x_2, ..., x_1x_d, x_2^2, x_2x_3, ..., x_2x_d, ..., x_d^2]^\top$ (i.e., a quadratic function of the elements of x).

Please note that the regularization term in Eq.2 does not include the bias term θ_0 (i.e., the parameter θ_0 is *not* encouraged to stay close to zero). The justification for this is that θ_0 is not associated to a real feature and thus is not affected by data noise as much as the others, so it is beneficial to let it vary without penalty. The closed-form solution optimizing this variant of the least-squares regularized objective is obtained by solving the following system of linear equations:

$$(B^\top B + \lambda U)\theta = B^\top y \tag{3}$$

where U is a diagonal matrix having value 0 in position $(1, 1)$ and value 1 in the other diagonal entries, $B = \left[b(x^{(1)}), ..., b(x^{(m)})\right]^\top$, and $y = \left[y^{(1)}, ..., y^{(m)}\right]^\top$.

You will use these models to predict the MPG (miles per gallon) of a car from several attributes of the vehicle. The data set is contained in the enclosed file *autompg.mat* [1]. You can load the data in Python by typing:

```
import scipy.io as spio
S = spio.loadmat('autompg.mat', squeeze_me=True)
Xtrain = S['trainsetX']
Ytrain = S['trainsetY']
```

[1]These examples were derived from the Auto-Mpg data set available at the UC Irvine Machine Learning Data Repository.

```
Xtest = S['testsetX']
Ytest = S['testsetY']
```

This sequence of commands will load four variables in the workspace: *Xtrain*, *Ytrain*, *Xtest*, and *Ytest*. Each row in these numpy.ndarray matrices corresponds to one example: *Xtrain[i,:]* contains the input vector of training example i, and *Ytrain[i]* the corresponding output value. Look at *autompg.names* for a description of the features. You should train your algorithm on the training set {*Xtrain*, *Ytrain*}, and evaluate its performance on the test set {*Xtest*, *Ytest*}. For all experiments, you should measure the performance as mean squared error on the test set, i.e., evaluate it by computing $\frac{1}{M}\sum_{i=1}^{M}\left[\hat{y}^{(i)} - f_\theta(\hat{x}^{(i)})\right]^2$ where $(\hat{x}^{(i)}, \hat{y}^{(i)})_{i=1,..,M}$ are the M test examples, and f_θ is the function learned on the training set.

(a) [**P**, 20 points] Implement the Python functions described as follows:
- [7 points] `q4_features.py` computes the linear and quadratic features $b^l(x)$ and $b^q(x)$.
- [3 point] `q4_mse.py` calculates the mean squared error.
- [8 points] `q4_train.py` learns model parameters θ given a training set of examples by solving the system in Eq. 3 (use Python numpy function `numpy.linalg.solve` to solve the system).
- [2 points] `q4_predict.py` performs prediction by evaluating a given learned model for the input examples.

(b) [**P**, 10 points] The performance of the algorithm will depend on the choice of the parameter λ. Implement the function `q4_cross_validation_error.py` to perform N-fold cross-validation. Given a training set of examples and a finite set of values for hyperparameter λ, this function should return the cross-validation score, i.e., the average of the mean squared errors over the validation sets. Once you have coded this function, you can run the script `q4b.py` to plot the 10-fold cross-validation scores for $\lambda \in \{10^{-5}, 10^{-3}, 10^{-1}, 10, 10^3, 10^5, 10^7\}$ with both the linear model and the quadratic model.

(c) [**W**, 5 points] Are there any values of λ producing underfitting? If yes, which values?

(d) [**W**, 5 points] Are there any values of λ producing overfitting? If yes, which values?

(e) [**W**, 3 points] Which of the two versions of feature vector $b(x)$ produces more overfitting, $b^l(x)$ or $b^q(x)$? Can you explain why?

(f) [**P**, 4 points] Code the function `q4_test_error.py` which trains a model on the training set and returns the test error. Then, execute the script `q4f.py`, which plots the *test set* mean squared error for the same set of values of λ as before.

(g) [**W**, 1 point] Is the cross-validation score a good predictor of performance on the test set? Please comment on why or why not.