Machine Learning	**Homework 2**
Comp540 Spring 2016	**due 22 February 2016 at 8 pm**

This homework is due February 22 at 8 pm on Owlspace. The code base `hw3.zip` *for the assignment is an attachment to* Assignment 3 *on Owlspace. You will add your code at the indicated spots in the files there. Place your answers to Problems 1 and 2 (typeset) in a file called* `writeup.pdf` *and add it to the zip archive. Upload the entire archive back to Owlspace before the due date and time.*

1 MAP and MLE parameter estimation (10 points)

Consider a data set $\mathcal{D} = \{x^{(i)} | 1 \leq i \leq m\}$ where $x^{(i)}$ is drawn from a Bernoulli distribution with parameter θ. The elements of the data set are the results of the flips of a coin where $x^{(i)} = 1$ represents *heads* and $x^{(i)} = 0$ represents *tails*. We will estimate the parameter θ which is the probability of the coin coming up heads using the data set $\mathcal{D}$.

- (5 points) Use the method of maximum likelihood estimation to derive an estimate for θ from the coin flip results in $\mathcal{D}$.
- (5 points) Assume a beta prior distribution on θ with hyperparameters a and b. The beta distribution is chosen because it has the same form as the likelihood function for $\mathcal{D}$ derived under the Bernoulli model (such a prior is called a conjugate prior).

  $$Beta(\theta|a, b) \propto \theta^{a-1}(1-\theta)^{b-1}$$

  Derive the MAP estimate for θ using $\mathcal{D}$ and this prior distribution. Show that under a uniform prior (Beta distribution with $a = b = 1$), the MAP and MLE estimates of θ are equal.

2 Logistic regression and Gaussian Naive Bayes (15 points)

Consider a binary classification problem with dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)}) | 1 \leq i \leq m; x^{(i)} \in \Re^d, y^{(i)} \in \{0, 1\}\}$. You will derive a connection between logistic regression and Gaussian Naive Bayes for this classification problem. For logistic regression, we use the sigmoid function $g(\theta^T x)$, where $\theta \in \Re^{d+1}$ and we augment x with a 1 in front to account for the intercept term θ_0. For the Gaussian Naive Bayes model, assume that the y's are drawn from a Bernoulli distribution with parameter γ, and that each x_j from class 1 is drawn from a univariate Gaussian distribution with mean μ_j^1 and variance ${\sigma_j}^2$, and each x_j from class 0 is drawn from a univariate Gaussian distribution with mean μ_j^0 and variance ${\sigma_j}^2$. Note that the variance is the same for both classes, just the means are different.

- (2 points) For logistic regression, what is the posterior probability for each class, i.e., $P(y = 1|x)$ and $P(y = 0|x)$? Write the expression in terms of the parameter θ and the sigmoid function.

Name	Edit?	Read?	Description
softmax_hw.py	Yes	Yes	Python script to run softmax regression on CIFAR-10 dataset
utils.py	Yes	Yes	contains functions for loading data, and for visualizing data.
softmax.py	Yes	Yes	Class and methods defining the softmax classifier
gradient_check.py	No	Yes	functions for checking the analytical gradient numerically
softmax_music.py	Yes	Yes	Python script for comparing softmax regression and OVA regression for genre classification
datasets	No	Yes	directory containing shell script to download CIFAR-10 data
hw3.pdf	No	Yes	this document

Table 1: Contents of `hw3`

- (5 points) Derive the posterior probabilities for each class, $P(y = 1|x)$ and $P(y = 0|x)$, for the Gaussian Naive Bayes model, using Bayes rule, the (Gaussian) distribution on the x_j's, $j = 1, \ldots, d$, and the Naive Bayes assumption.
- (8 points) Assuming that class 1 and class 0 are equally likely (uniform class priors), simplify the expression for $P(y = 1|x)$ for Gaussian Naive Bayes. Show that with appropriate parameterization, $P(y = 1|x)$ for Gaussian Naive Bayes with uniform priors is equivalent to $P(y = 1|x)$ for logistic regression.

3 Softmax regression and OVA logistic regression (45 points)

In this assignment, you will first implement multi-class logistic regression, or softmax regression, and apply it to a version of the CIFAR-10 object recognition dataset. Then, you will compare the performance of softmax regression with one-vs-all binary logistic regression on the problem of music genre classification. The material for this problem is in `hw3.zip` and Table 1 contains the contents of the folder created by unzipping it.

Download the data

Open up a terminal window and navigate to the `datasets` directory of `hw3`. Run the `get_datasets.sh` script. On my Mac, I just type in `./get_datasets.sh` at the shell prompt. A new folder called `cifar_10_batches_py` will be created and it will contain 60,000 labeled images for training and 10,000 labeled images for testing. We have provided a function to read this data in `utils.py` – this function also partitions the training data into a training

set of 49,000 images and a validation set of 1,000 images used for selecting hyperparameters for softmax regression. Each image is a 32×32 array of RGB triples. It is preprocessed by subtracting the mean image from all images, and flatted into a 1-dimensional array of size 3072. Then a 1 is appended to the front of that vector to handle the intercept term. So the training set is a `numpy` matrix of size 49000×3073, the validation set is a matrix of size 1000×3073 and the set-aside test set is of size 10000×3073.

Problem 3.1: Implementing the loss function for softmax regression (naive version) (5 points)

Softmax regression generalizes logistic regression to classification problems where the class label y can take on more than two possible values. This is useful for such problems as music genre classification and object recognition, where the goal is to distinguish between more than two different music genres or more than two different object categories. Softmax regression is a supervised learning algorithm, but we will later be using it in conjunction with deep learning and unsupervised feature learning methods. Recall that we are given a data set

$$\mathcal{D} = \{(x^{(i)}, y^{(i)}) | 1 \leq i \leq m; x^{(i)} \in \Re^{d+1}; {x_0}^{(i)} = 1, y^{(i)} \in \{1, \ldots, K\}\}, K > 2$$

Our probabilistic model $h_\theta(x)$ is defined as

$$h_\theta(x) \quad = \quad \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \ldots \\ P(y=K|x;\theta) \end{bmatrix}$$

where

$$P(y=k|x;\theta) = \frac{exp({\theta^{(k)}}^T x)}{\sum_{j=1}^{K} exp({\theta^{(j)}}^T x)}$$

The parameter θ is a $(d+1) \times K$ matrix, where each column represents the parameter vector for class $k = 1, \ldots, K$.

$$\theta = \begin{bmatrix} | & | & \ldots & | \\ | & | & \ldots & | \\ \theta^{(1)} & \theta^{(2)} & \ldots & \theta^{(K)} \\ | & | & \ldots & | \\ | & | & \ldots & | \end{bmatrix}$$

Hint: numerical stability issues can come up in the computation of $P(y = k|x;\theta)$. Consider K=3, and $\theta^T x = [123, 456, 789]$. To compute $P(y = k|x;\theta)$ from these scores, we need to calculate $exp(123)$, $exp(456)$ and $exp(789)$, and sum them. These are very large numbers. However, we can get the same probabilities by subtracting the maximum (789)

from every element in $\theta^T x$. Then we have the vector $[-666, -333, 0]$, and we can calculate $exp(-666)$, $exp(-333)$ and $exp(0)$, sum them (call the sum S) and then calculate $exp(-666)/S$, $exp(-333/S)$ and $exp(0)/S$.

The cost function $J(\theta)$ for softmax regression is derived from the negative log likelihood of the data $\mathcal{D}$, assuming that $P(y|x;\theta) = h_\theta(x)$ as defined above.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} 1\{y^{(i)} = k\} log \frac{exp(\theta^{(k)^T} x)}{\sum_{j=1}^{K} exp(\theta^{(j)^T} x)} + \lambda \sum_{j=1}^{K}\sum_{k=1}^{K} {\theta_{jk}}^2$$

where $I\{c\}$ is the indicator function which evaluates to 1 when c is a true statement and to 0 otherwise. The second term is a regularization term, where λ is the regularization strength. While it is customary to exclude the bias term in L2 regularization, we include it here because it does not make a huge difference in the final result. You can check this for yourself on the CIFAR-10 dataset. You should implement this loss function using `for` loops for the summations in the function `softmax_loss_naive` in `softmax.py`. Once you have the loss function implemented, the script `softmax_hw.py` will run your loss function for a randomly initialized θ matrix with 49000 training images and labels with λ set to 0. You should expect to see a value of about $-log_e(0.1)$ (Why?). It also saves a visualization of a small sample of the training data in `cifar10_samples.pdf`.

Problem 3.2: Implementing the gradient of loss function for softmax regression (naive version) (5 points)

The derivative of the loss function $J(\theta)$ with respect to the $\theta^{(k)}$ is

$$\nabla_{\theta^{(k)}} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[x^{(i)}(1\{y^{(i)} = k\} - P(y^{(i)} = k|x^{(i)};\theta))]$$

Implement the analytical derivative computation in `softmax_loss_naive` in `softmax.py`.

Checking your gradient implementation with numerical gradients

In the previous problem you have implemented the *analytical* gradient computed using calculus. We check your implementation of the gradient using the method of finite differences. The functions in `gradient_check.py` compute the numerical gradient of a function f as follows:

$$\frac{\partial f(x)}{\partial x} = \frac{f(x+h) - f(x-h)}{2h}$$

for a very small h. The script `softmax_hw.py` will check your gradient against the numerically approximated gradient – you should expect to see differences between the two gradients of the order of 10^{-7} or less.

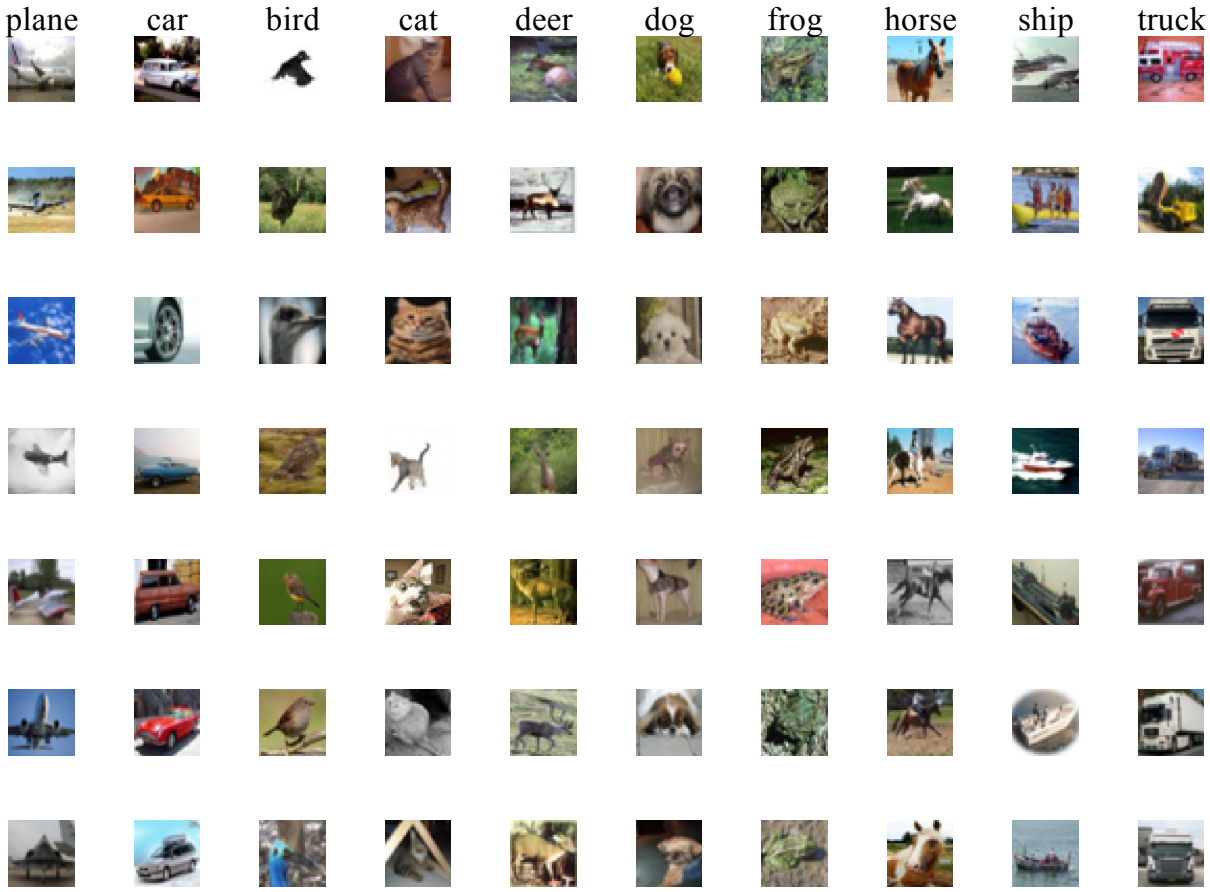

Figure 1: A sample of the CIFAR-10 dataset.

Problem 3.3: Implementing the loss function for softmax regression (vectorized version) (5 points)

Now complete the function `softmax_loss_vectorized` in `softmax.py` to implement the loss function $J(\theta)$ without using any `for` loops. Re-express the computation in terms of matrix operations on X, y and θ.

Problem 3.4: Implementing the gradient of loss for softmax regression (vectorized version) (5 points)

Now vectorize the gradient computation in `softmax_loss_vectorized` in `softmax.py`. Once you complete this, `softmax_hw.py` will run and time your naive and vectorized implementations – you should expect to see at least one order of magnitude difference in run time between the two implementations.

Problem 3.5: Implementing mini-batch gradient descent (5 points)

In large-scale applications, the training data can have millions of examples. Hence, it seems wasteful to compute the loss function over the entire training set in order to perform only a single parameter update. A very common approach to addressing this challenge is to compute the gradient over batches of the training data. For example, a typical batch contains 256 examples from a training set of over 1.2 million. This batch is then used to perform a parameter update:

$$\theta^{(k)} \rightarrow \theta^{(k)} - \alpha \nabla_{\theta^{(k)}} J(\theta)$$

where α is the step size or learning rate for gradient descent.

Implement mini-batch gradient descent in the method `train` in `softmax.py` using the description provided in the documentation of the method. You can set the `verbose` argument of `train` to be `True` and observe how the loss function varies with iteration number.

Problem 3.6: Using a validation set to select regularization lambda and learning rate for gradient descent (5 points)

There are many hyper parameters to be selected for mini batch gradient descent – the batch size, the number of iterations, and the learning rate. For the loss function, we also need to select λ, the regularization strength. In this exercise, we have pre-selected a batch size of 400 and an iteration count of 4000. Now, use the validation set provided to sweep the learning rate and the λ parameter space, using the suggested values in `softmax_hw.py` to find the best combination of these two hyper parameters. Fill in the code to do this in `softmax_hw.py` at the indicated spot.

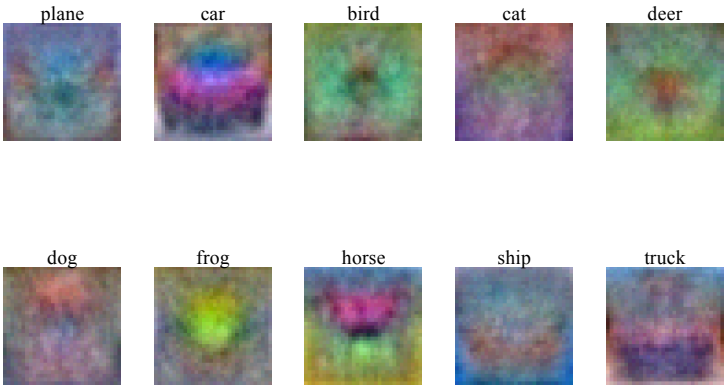

Figure 2: Visualization of θ for the CIFAR-10 dataset. The color scale is from red (low values) to blue (high values).

Problem 3.7: Training a softmax classifier with the best hyperparameters (5 points)

Once you find the best values of λ and learning rate, insert code in `softmax_hw.py` to train a softmax classifier on the training data with the best hyper parameters and save this classifier in the variable `best_softmax`. The classifier will be evaluated on the set aside test set and you should expect to see overall accuracy of over 35%.

Visualizing the learned parameter matrix

We can remove the bias term from the θ matrix and reshape each column of the matrix which is a parameter vector of size 3072 back into an array of size $32 \times 32 \times 3$ and visualize the results as an image. `softmax_hw.py` constructs this plot and saves it in PDF form as `cifar_theta.pdf`. You should see a plot like the one shown in Figure 2. Compute the confusion matrix (you can use the `sklearn` function) on the test set for your predictor and interpret the visualized coefficients in the light of the errors made by the classifier.

Extra credit: Problem 3.8: Experimenting with other hyper parameters and optimization method (10 points)

We chose a batch size of 400 and 4000 iterations for our previous experiments. Explore larger and smaller batch sizes, choosing an appropriate number of iterations (by specifying a tolerance on differences in values of the loss function or its gradient in successive iterations) with the validation set. Produce plots that show the variation of test set accuracy as a function of batch size/number of iterations. You will have to determine the right settings

for regularization strength λ and learning rate for each batch size/ number of iterations combination. What is the best batch size/number of iterations/learning rate/regularization strength combination for this problem? What is the best test set accuracy that can be achieved by this combination? Now, use `fmin` as in Homework 2 instead of using batch gradient descent as the optimization engine. You only need to find a good setting for λ using the validation set. What is the best achievable accuracy with `fmin`?

Problem 3.8: Comparing OVA binary logistic regression with softmax regression on music genre classification (10 points)

You have already written an OVA classifier for the genre classification problem in Homework 2. Here, you will fill in `softmax_music.py` to run softmax regression on the music genre classification problem. In particular, you must read in all the data, divide the data into a training, validation and test set (80/20 split on train/test, and split the training data into training/validation sets in 90/10 ratio), select the appropriate hyperparameters for softmax regression using the validation set, and obtain classification accuracy on the set aside test set for the softmax classifier trained with the best hyper parameters. Compare these results with the results from your OVA classifier. Provide a table comparing the per genre classification performance of each classifier. Explain performance differences, if any, or explain why their performance is similar, if it is. Which approach would you recommend for the music genre classification problem?

What to turn in

Please zip up all the files in the archive (including files that you did not modify) and submit it as `hw3_netid`.zip on Owlspace before the deadline, where `netid` is to be replaced with your netid. Include a PDF file in the archive that presents your plots and discussion from the programming component of the assignment. Also include typeset solutions to the written problems 1 and 2 of this assignment in your writeup. Only one submission per group of two, please.