Machine Learning	**Homework 3**
Comp540 Spring 2019	**due 15 February 2019 at 8 pm**

This homework is due February 15 at 8 pm on Canvas. The code base `hw3.zip` *for the assignment is an attachment to* Assignment 3 *on Canvas. You will add your code at the indicated spots in the files there. Place your answers to Problems 1, 2, 3, 4, 5 (typeset) in a file called* `writeup.pdf`*. Please submit the following items as separate attachments before the due date and time:*

- *Your writeup pdf, named* `writeup.pdf`*.*
- *Your edited .py files – submit each one separately.*
- *Your jupyter notebooks, saved in HTML format. If there are multiple notebooks, submit each one separately.*

You will solve this assignment in a group of two – only one submission per group, please.

We take the Rice Honor Code very seriously in this class. Please read the academic integrity section of the course policies at ***https://canvas.rice.edu/courses/20448/pages/comp-540-statistical-machine-learning-course-policies****, and enter the statement that you have followed the honor code described there in the submission box for your homework. Homeworks without the honor pledge will not be graded.*

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.
- (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 Reject option in classifiers (10 points)

In many classification problems one has the option either of assigning x to class j or, if you are too uncertain, of choosing the reject option. If the cost for rejects is less than the cost of falsely classifying the object, it may be the optimal action. Let α_i mean you choose action i, for $i = 1, \ldots, C + 1$, where C is the number of classes and $C + 1$ is the reject action. Let $y = j$ be the true (but unknown) label of x. Define the loss function as follows

$$L(\alpha_i \mid y = j) \;=\; \begin{cases} 0 & \text{if } i = j \text{ and } i, j \in \{1, \ldots, C\} \\ \lambda_r & \text{if } i = C + 1 \\ \lambda_s & \text{otherwise} \end{cases}$$

In other words, you incur 0 loss if you correctly classify, you incur λ_r loss (cost) if you choose the reject option, and you incur λ_s loss (cost) if you make a misclassification error.

- (5 points) Show that the minimum risk is obtained if we decide $y = j$ if $p(y = j \mid x) \geq p(y = k \mid x)$ for all k (i.e., j is the most probable class) and if $p(y = j \mid x) \geq 1 - \frac{\lambda_r}{\lambda_s}$, otherwise we decide to reject.

- (5 points) Describe qualitatively what happens as λ_r/λ_s is increased from 0 to 1 (i.e., the relative cost of rejection increases).

4 Kernelizing k-nearest neighbors (5 points)

The nearest neighbor classifier assigns a new vector $x \in \Re^d$ to the same class as that of the nearest neighbor $x^{(i)}$ in the training set $\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid x^{(i)} \in Re^d, y^{(i)} \in \{-1, 1\}\}$ where the distance between two points in $\Re^d$ is the Euclidean distance. Re-express this classification rule in terms of dot products and then make use of the kernel trick to formulate the k-nearest neighbor algorithm for a general non-linear kernel function K satisfying the Mercer condition.

5 Constructing kernels (10 points)

Show that given valid kernels $k_1(x, x^{'})$ and $k_2(x, x^{'})$ defined over $x, x^{'} \in \Re^d$, the following new kernels will also be valid.

- (2 points) $k(x, x^{'}) = ck_1(x, x^{'})$ where $c > 0$.
- (4 points) $k(x, x^{'}) = f(x)k_1(x, x^{'})f(x^{'})$ where $f(x)$ is any function on $x \in \Re^d$.
- (4 points) $k(x, x^{'}) = k_1(x, x^{'}) + k_2(x, x^{'})$

6 One_vs_all logistic regression (15 points)

In this problem, you will implement a one-vs-all (OVA) logistic classifier and apply it to a version of the CIFAR-10 object recognition dataset. 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` folder inside the `hw3` folder. 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 50,000 labeled images for training and 10,000 labeled images for testing. The function further partitions the 50,000 training images into a train set and a validation set for selection of hyper parameters. We have provided a function to read this data in `data_utils.py`. Each image is a 32×32 array of RGB triples. It is preprocessed by subtracting the mean image from all images. We flatten each image into a 1-dimensional array of size 3072 (i.e., $32 \times 32 \times 3$). 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.

Name	Edit?	Read?	Description
softmax_cifar.ipynb	Yes	Yes	Python notebook to run softmax regression on CIFAR-10 dataset
ova_cifar.ipynb	Yes	Yes	Python notebook to run OVA logistic regression on CIFAR-10 dataset
utils.py	No	Yes	contains functions for loading data, and for visualizing CIFAR-10 data.
one_vs_all.py	Yes	Yes	Class and methods defining the one_vs_all classifier
softmax.py	Yes	Yes	Class and methods defining the softmax classifier
linear_classifier.py	Yes	Yes	Class and methods for defining mini-batch gradient descent
gradient_check.py	No	Yes	functions for checking the analytical gradient numerically
datasets	No	Yes	directory containing shell script to download CIFAR-10 data
hw3.pdf	No	Yes	this document

Table 1: Contents of `hw3`

Problem 6.1. Implementing a one-vs-all classifier for the CIFAR-10 dataset (10 points)

In this part of the exercise, you will implement one-vs-all classification by training multiple regularized logistic regression classifiers, one for each of the ten classes in our dataset. You should now complete the code in `one_vs_all.py` to train one classifier for each class. In particular, your code should return all the classifier parameters in a matrix $\Theta \in \Re^{3073 \times K}$, where each column of Θ corresponds to the learned logistic regression parameters for one class. You can do this with a for-loop from 0 to $K-1$, training each classifier independently. When training the classifier for class $k \in \{0, \ldots, K-1\}$, you should build a new label for each example x as follows: label x as 1 if x belomgs to class k and zero otherwise. You can use sklearn's logistic regression function to learn each classifier.

Problem 6.2: Predicting with a one-vs-all classifier (5 points)

After training your one-vs-all classifier, you can now use it to predict the labels of the set aside test images. For each image in the test set, you should compute the probability that it belongs to each class using the trained logistic regression classifiers. Your one-vs-all prediction function will pick the class for which the corresponding logistic regression classifier outputs the highest probability and return the class label as the prediction for that input example. You should now complete code in the predict method `one_vs_all.py`. Once you

are done, our notebook script `ova_cifar.ipynb` will make predictions on a set aside test set and print out the confusion matrix. The next cell in the notebook will visualize the learned coefficients for each of the 10 one-vs-rest classifiers.

Comparing your OVA classifier with sklearn's classifier

The last two cells in the `ova_cifar.ipynb` notebook setup sklearn's OVA classifier and visualize the coefficients learned by that classifer. Compare what you obtain with your version of the OVA classifier against sklearn's implementation.

7 Softmax regression (45 points)

In this problem, you will implement a softmax classifier and apply it to a version of the CIFAR-10 object recognition dataset. You will need to compare the performance of this classifier with the OVA classifier and comment on which approach is better for the CIFAR-10 dataset and why. There are 10 extra credit points in this problem for experimenting with hyperparameters and optimization method.

Problem 7.1: 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 $k = 1, \ldots, K$.

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

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.

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} I\{y^{(i)} = k\} log \frac{exp({\theta^{(k)}}^T x^{(i)})}{\sum_{j=1}^{K} exp({\theta^{(j)}}^T x^{(i)})} + \frac{\lambda}{2m} \sum_{j=0}^{d} \sum_{k=1}^{K} {\theta_j^{(k)}}^2$$

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, a cell in the notebook `softmax_cifar.ipynb` 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?).

Problem 7.2: 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

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

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

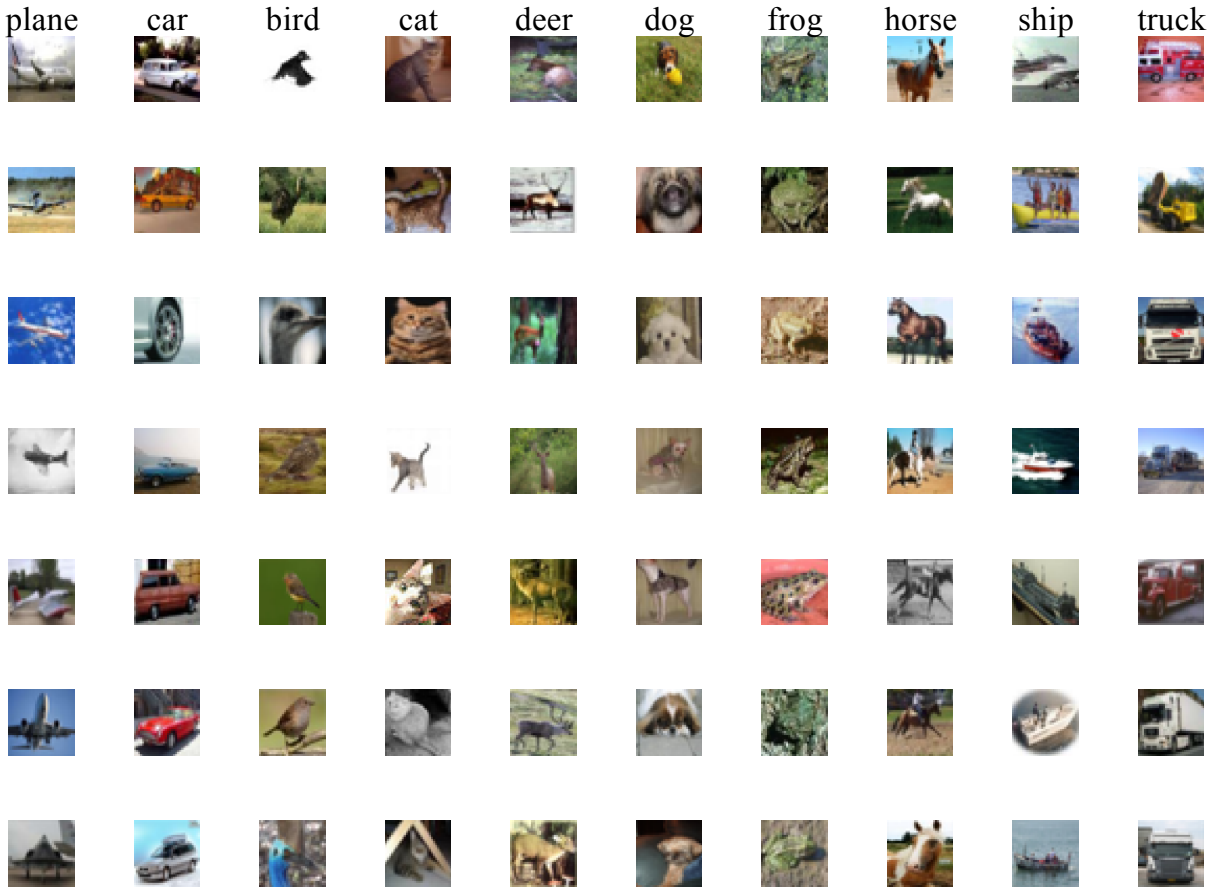

Figure 1: A sample of the CIFAR-10 dataset.

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. A cell in the `softmax_cifar.ipynb` notebook 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.

Problem 7.3: Implementing the loss function for softmax regression (vectorized version) (10 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 7.4: 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, a cell in `softmax_cifar.ipynb` 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 7.5: 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 7.6: 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_cifar.ipynb` to find the best combination of these two hyper parameters. Fill in the code TODO in the marked cell in `softmax_cifar.ipynb`.

Problem 7.7: Training a softmax classifier with the best hyperparameters (5 points)

Once you find the best values of λ and learning rate, insert code in the notebook `softmax_cifar.ipynb` 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_cifar.ipynb` constructs this plot and displays it inline. 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 7.8: 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?

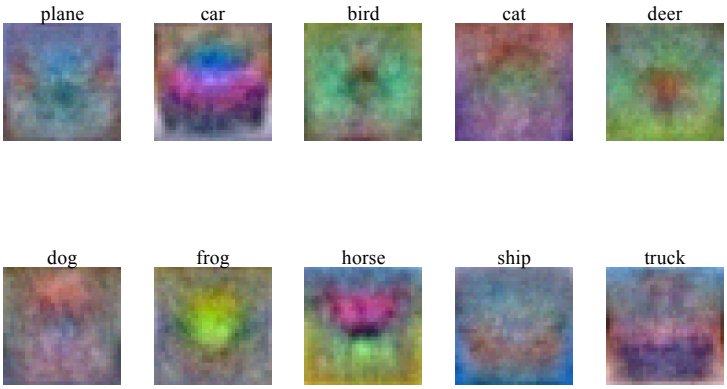

Figure 2: Visualization of θ for the CIFAR-10 dataset. The color scale is from red (low values) to blue (high values).

Problem 7.9. Comparing OVA binary logistic regression with softmax regression (5 points)

Compare the performance results from your OVA and softmax regression classifier. Provide a table with classification performance of each classifier on each CIFAR-10 category. Explain performance differences, if any, or explain why their performance is similar, if it is. Which approach would you recommend for the CIFAR-10 classification problem? Place answers to these questions in `writeup.pdf`.