Machine Learning 10-701 Jan 20, 2015
http://alex.smola.org/teaching/10-701-15/ due on Feb 2, 2015
Carnegie Mellon University

Homework 2

START HERE: Instructions

- The homework is due at 11:59pm on Feb 2, 2015. Anything that is received after that time will be considered as late submission.
- Answers to everything but the coding question will be submitted electronically (e.g. as a PDF or handwritten and scanned). We'll give exact instructions on the email list soon; in the meantime, make sure you prepare the answers to each question separately.
- Please follow the instruction for code submission in problem correctly.
- The handout for question can be found on Autolab or here.
- Collaboration on solving the homework is allowed (after you have thought about the problems on your own). However, when you do collaborate, you should list your collaborators! You might also have gotten some inspiration from resources (books or online etc...). This might be OK only after you have tried to solve the problem, and couldn't. In such a case, you should cite your resources.
- If you do collaborate with someone or use a book or website, you are expected to write up your solution independently. That is, close the book and all of your notes before starting to write up your solution.

This is a programming assignment. You are asked to implement several algorithms in Octave. Octave is a free scientific programming language, with syntax (almost) identical to that of Matlab. Installation instructions can be found on the Octave website as linked above. If you've never used Octave or Matlab before, you may wish to check out this tutorial or this one.

For each question you will be given a single function signature, and asked to write a single Octave function which satisfies the signature. Please do *not* change the names of the functions and their variables. Do *not* modify the structure of the directories.

The problems are automatically graded using the CMU Autolab system. The code which you write will be executed remotely against a suite of tests, and the results used to automatically assign you a grade. In order for your code to execute correctly on our servers you should avoid using libraries beyond the *basic* octave libraries.

The Autolab interface for this course can be found at https://autolab.cs.cmu.edu/. You can sign in using your andrew credentials. You should make sure to edit your account information and choose a nickname/handle. This handle will be used to display your results for the challenge question on the class leaderboard.

We have provided you with a single folder containing each of the functions you need to complete. *Do not modify the structure of this directory or rename these files*. Complete each of these functions, then compress this directory as a tar file and submit to autolab online. You may submit as many times as you like (up to the due date).

Please *only* submit a tarball called "code.tar", which is compressed from "code" folder. Do not submit any other files such as data files.

If you have a question, please post it on Piazza https://piazza.com/class/i4ivtbjbrt219e. If you discover a bug or want to talk other technical issues, please send an email to Jin Sun jins@andrew.cmu.edu.

Notations

- **XTrain**: $\mathcal{R}^{nTrain \times f}$ is a matrix of training data, where each row is a training instance with f features.
- **XTest**: $\mathcal{R}^{nTest \times f}$ is a matrix of test data, where each row is a test instance with f features.
- **yTrain**: $\mathcal{R}^{nTrain \times 1}$ is a vector of labels (for classification) or real numbers (for regression).

Machine Learning 10-701 Jan 20, 2015
http://alex.smola.org/teaching/10-701-15/ due on Feb 2, 2015
Carnegie Mellon University

Homework 2

Datasets (provided in handout folder)

- **Iris dataset**: A dataset for plants from UCI, used for classification.
- **WBC dataset**: Wisconsin Breast Cancer Database from UCI, used for classification.
- **Dataset for leaderboard**: Only the training set is provided to you. Your implementation will be tested on the hidden test set.

1 Gaussian Naive Bayes [30 pts]

In this question you will implement the Gaussian Naive Bayes Classification algorithm. As a reminder, in the Naive Bayes algorithm we calculate $p(k|f) \sim p(f|k)p(k) = p(k)\prod_i p(f_i|k)$. In Gaussian Naive Bayes we learn a one-dimensional Gaussian for each feature in each class, i.e. $p(f_i|k) = N(f_i; \mu_{i,k}, \sigma^2_{i,k})$, where $\mu_{i,k}$ is the mean of feature f_i for those instances in class k, and $\sigma^2_{i,k}$ is the variance of feature f_i for instances in class k.

1. **Prior [5 pts]**
   Complete the function $prior(yTrain)$ which returns a $nClass \times 1$ vector y, where y_k is the prior probability of class k. ($nClass$ denotes the number of classes)

2. **Likelihood [15 pts]**
   Complete the function $likelihood(XTrain, yTrain)$ which returns $[mu, sigma]$. mu is a $f \times nClass$ matrix where $mu_{i,k}$ is the conditional mean of feature i given class k. $sigma$ is an $f \times nClass$ matrix where $sigma_{i,k}$ is the conditional standard deviation of feature i given class k.

3. **Naive Bayes Classifier [10 pts]**
   Complete the function $naiveBayesClassify(XTrain, yTrain, XTest)$ which returns a $nTest \times 1$ vector y of predicted class values, where y_j is the predicted class for the jth row of $XTest$.

2 Linear Regression [30 pts]

In this question you will implement Regularized Linear Regression with gradient descent. The padding process (pad 1 for each data entry) is done to the data matrix before running the program, so you don't need to worry about it in your implementation. The objective function for a Ridge Regression (l_2 regularized regression) is as follows:

$$\hat{w} = argmin_w \frac{1}{2}\|y - \hat{y}\|_2^2 + \frac{\lambda}{2}\|w\|_2^2 = argmin_w \frac{1}{2}\|y - Xw\|_2^2 + \frac{\lambda}{2}\|w\|_2^2$$

1. **Compute Weights [25 pts]**
   Complete the function $compute\_weights(XTrain, yTrain, init\_w, nIter, lr, lambda)$ which returns w. w is a $f \times 1$ vector learned with *batched* gradient descent. Please do not implement stochastic gradient descent or mini-batch gradient descent. Also some reference may divide the gradient by the total number of data points. Please do not it in this assignment. $init_w$ is a $f \times 1$ vector as the initialization for the weight vector. $nIter$ is the number of iterations. In each iteration, you need to compute the gradient for all training data points and only update the weight vector after the iteration. lr is the learning rate. $lambda$ is the coefficient of the penalty term.

2. **Linear Regression [5 pts]**
   Complete the function $linearRegression(XTest, w)$ which returns y. y is a $nTest \times 1$ vector of predicted values. y_j is the predicted value for the jth row of $XTest$.

3 k Nearest Neighbor [35+5 pts]

In this question you will implement the k Nearest Neighbor Classification algorithm.

1. **Compute Distance [20 pts]**
   Complete the function $compute\_distance(XTrain, XTest, d)$ which returns a $nTrain \times nTest$ matrix D, where $D_{i,j}$ is the distance between ith row of $XTrain$ and jth row of $XTest$. d denotes the distance metric. If d is "euclidean", D should return the euclidean distance between data points. Implementing euclidean distance is mandatory. You need to implement other distance metric for the leaderboard problem.

2. **knnClassify [10 pts]**
   Complete the function $knnClassify(XTrain, yTrain, XTest, k, d)$ which returns a $nTest \times 1$ vector y of predicted class values, where y_j is the predicted class for the jth row of $XTest$.

3. **knn [5+5 pts]**
   This is the wrapper function for knn algorithm. You need to play with different values of k and implement different distance metrics to get the best performance. Submit your knn.m with the best k and best distance metric your get. You also need to implement the distance metric in compute_distance.m. There is a class leaderboard for this question on the autolab website, where students are identified by their chosen nickname/handle. 5 points will be given for all students above the baseline accuracy (35%). Bonus points will be awarded to students who have top 10 classification accuracy on the class leaderboard.

4 Hints for the leaderboard problem

4.1 Hints for validation

While constructing a classifer or regressor, it is extremely important to know whether the model is overfitting or underfitting to the data. Usually people partition their dataset into three parts: training set, validation set, and test set. Training the model on the training set (e.g. doing gradient descent for regression), tune the parameters on the validation set (e.g. finding out best coefficient for the penalty term in regression, finding out best k for knn), and test the accuracy on the test test. We will cover more details in model selection topic later in this course. You should first read some online articles (e.g. wikipedia) about k-fold cross validation and leave-one-out cross validation, and do cross validation with your model. Here I list some useful information for handling overfitting and underfitting.

4.2 Hints for distance metrics

Searching on google for "distance metrics" will return a dizzying array of different approaches, each best suited to slightly different data. There are the basic distance metrics L1, L2, hamming etc. There are psuedometrics such as KL divergence. There are approaches which attempt to align the data such as dynamic time warping. There are approaches which first transform the domain (Laplace Transform, Fourier Transform, Cosine Transform) then apply a basic distance metric. I would strongly suggest you begin by plotting the

Homework 2

	Overfit	Underfit
Performance	Training accuracy much higher than validation accuracy	Both accuracies are low
Data	Need more data	If two accuracies are close, no need for extra data
Model	Use a simpler model	Use a more complicated model
Features	Reduce number of features	Increase number of features
Regularization	Increase regularization	Reduce regularization

data (using `plot(XTrain(1,:))`) to get a feel for what it looks like, then consider what sort of data it might be, and what distance metrics suit this type of data. Furthermore this is real data, and real data is dirty! You may want to consider normalizing the data prior to computing distances via centering etc.