CS74/174, Winter 2014, Problem Set # 2

February 4, 2014

Due: February 18, 2014 @ 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 MATLAB and apply them to the enclosed data sets. The questions requiring a written answer are denoted with **W**, while the MATLAB programming questions are marked with **M**.

You must provide your written answers in paper form and leave them in the course mailbox located near the lobby of the Sudikoff building (the mailbox is tagged with the label CS74/174). Since each problem will be graded by a different person, we ask that you provide a separate submission for each individual problem. Please write your name, date and time of submission on the front page of each problem submission. A penalty of 5 points will be applied if you fail to obey these instructions.

Instead, your MATLAB code must be submitted via Blackboard. We have provided MATLAB 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 also some commands to help you get started with the implementation. Because we will be grading your code using automatic scripts, 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 (data, .fig files automatically generated by the provided scripts, and function files) in a single folder (without subfolders). Zip up the entire folder, and upload the compressed file to Blackboard.

1. [**15 points**] Consider a binary classification problem where the input features are real-valued, i.e., $x_j \in \mathbb{R}$ for $j = 1, \dots, n$. We want to learn a Naive Bayes classifier from a training set $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ via maximum likelihood estimation (as usual the examples are assumed to be I.I.D. within each class, and independent across classes). Recall that Naive Bayes makes the assumption that the features are conditionally independent given the class. We model the class conditional density of each feature x_j given the class as a separate

Gaussian distribution. Thus, our generative model is:

$$p(y=1) \quad = \quad \phi_y \tag{1}$$

$$p(x|y=0) \quad = \quad \prod_{j=1}^{n} \mathcal{N}(x_j; \mu_{j|0}, \sigma_{j|0}^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi\sigma_{j|0}^2}} \exp\left\{-\frac{1}{2\sigma_{j|0}^2}(x_j - \mu_{j|0})^2\right\} \tag{2}$$

$$p(x|y=1) \quad = \quad \prod_{j=1}^{n} \mathcal{N}(x_j; \mu_{j|1}, \sigma_{j|1}^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi\sigma_{j|1}^2}} \exp\left\{-\frac{1}{2\sigma_{j|1}^2}(x_j - \mu_{j|1})^2\right\} \tag{3}$$

(a) [**W, 5 points**] Write down the expression for the log-likelihood function.

(b) [**W, 10 points**] Compute the maximum likelihood estimates for parameters $\phi_y, \mu_{j|c}, \sigma_{j|c}^2$ where $c \in \{0, 1\}$. Show your derivation but omit redundant calculations (e.g., show the derivations for $\mu_{j|0}$ and $\sigma_{j|0}^2$ but omit those for $\mu_{j|1}$ and $\sigma_{j|1}^2$, since they are analogous).

2. [**25 points**] For this problem you will need to implement logistic regression in MATLAB. You will evaluate your algorithm on the Parkinsons dataset contained in *parkinsons.mat*, which we obtained from the UCI Machine Learning Repository (the data is formatted as in the previous homework). Look at *parkinsons.names* for a description of the features. The objective is to recognize healthy people from those with Parkinson's disease using a series of biomedical voice measurements.

(a) [**M, 16 points**] Implement a gradient ascent algorithm maximizing the logistic regression log-likelihood function (you are not allowed to use MATLAB built-in functionalities for gradient ascent optimization). The method requires selecting a value for the learning rate α: for now, use a fixed small value (e.g., $\alpha = 10^{-6}$). Your task is to fill the body of the following functions that we have provided to you:

- [**M**, 2 points] `q2_initialize.m` initializes the weights for maximum log likelihood training. The function provides two options: *random* and *heuristic* initialization. We have implemented for you the *random* option. Your task is to code the *heuristic* option that we have discussed in class (i.e., the one involving the solution of a simple least squares system).
- [**M**, 1 point] `q2_predict.m` computes the label predictions and the class posterior probabilities for the input examples in matrix X using the parameter vector θ.
- [**M**, 4 points] `q2_loglik.m` computes the log likelihood of the training data given the model θ. [Hint: you can call `q2_predict.m` inside this function].
- [**M**, 3 points] `q2_gradient.m` computes the gradient of the log likelihood at the current θ. [Hint: use `q2_predict.m` inside this function].
- [**M**, 5 points] `q2_train.m` trains logistic regression with gradient ascent using a fixed step size α. [Hint: you should call `q2_loglik.m` and `q2_gradient.m` inside this function].
- [**M**, 1 point] `q2_error.m` calculate the misclassification rate by comparing the predicted labels to the true labels Y. The misclassification rate is given by the number of misclassified examples over the total number of examples.

After implementing the above functions, you should be able to run `q2a.m`. This script will train logistic regression via gradient ascent using a small fixed step size $\alpha = 10^{-6}$. It will report both the training and the test error, the number of iterations needed to reach convergence, and will plot a curve of the log likelihood as a function of the number of iterations. [Hint: if you have implemented the functions correctly, the log likelihood curve should be monotonically increasing].

(b) **[M, 5 points]** Implement the function `q2_train_line_search.m` which trains logistic regression using line search to refine the step size α adaptively at each iteration (see lecture slides for the pseudocode). The line search method requires an initial value α_0. As discussed in class this should be chosen fairly large, e.g., $\alpha_0 = 10^{-4}$. Again, you are not allowed to use MATLAB built-in functionalities for gradient ascent optimization. After finishing this function, you can run the `q2b.m` to test your gradient ascent algorithm with line search. The script will report, for both the version using the fixed α (as coded for the previous question) and the one using line search, the following information: the training and the test errors, the number of iterations needed to reach convergence, and the log likelihood as a function of the number of iterations. Based on these results, answer the questions below:

(c) **[W, 2 points]** Compare the training and test errors of the two variants of gradient ascent. Are the errors different in the two cases? Explain why or why not.

(d) **[W, 2 points]** Now compare the two log likelihood curves. Does the method using line search converge faster or slower than the version using a fixed step size? Explain the result.

3. *Problem for graduate credit only: if you are enrolled in COSC74 (instead of COSC174) you do not need to solve this problem.*
[10 points] This problem requires you to implement the k-Nearest Neighbors classifier (kNN) based on Euclidean distance and evaluate it on the Parkinsons dataset *parkinsons.mat* of the previous problem.

(a) **[M, 5 points]** Implement the following functions:

- [**M**, 4 points] `q3_predict.m` predicts the class label of an input test example using kNN.
- [**M**, 1 point] `q3_test_error.m` computes the misclassification rates of kNN on a test set for different choices of hyperparameters k. [Hint: reuse the function `q2_error.m` that you wrote for the previous question].

You now should be able to run the script `q3a.m` to plot the test errors for different parameters $k \in \{1, 3, 5, 7, 9, 11, 13\}$.

(b) **[M, 1 point]** Implement `q3_cross_validation_error.m` to evaluate the cross validation errors for different parameters k. Note that the cross validation error for a classification problem is defined as the average misclassification rate over the N folds. After you complete this function, run the script `q3b.m` to plot the 5-fold cross validation errors for different parameters $k \in \{1, 3, 5, 7, 9, 11, 13\}$.

(c) **[M, 2 points]** Implement `q3_leave_one_out_error.m` to evaluate the validation errors of *leave-one-out* for different parameters k. After you complete this function, run the script `q3c.m` to plot the leave-one-out errors for different parameters $k \in \{1, 3, 5, 7, 9, 11, 13\}$.

Now, looking at the curves of the cross-validation error, the leave-one-out error and the test error, answer the following questions:

(d) **[W, 1 point]** Is the shape of leave-out-out validation different from the curve generated using 5-fold cross-validation? Why?

(e) **[W, 1 point]** Which of the two validation strategies (5-fold cross-validation or leave-one-out) produces the error curve most similar to the test error curve? Why?

4. **[M, 50 points]** In this problem you will have a chance to develop your own spam filter. You will actually implement several versions, corresponding to the following classifiers: Naïve Bayes, Decision Tree, and Random Forest. The data set that you will be using ("spam-data.mat") was derived from spam and non-spam emails collected a few years ago at Hewlett Packard Research Labs to test different spam detection algorithms. Each email is represented as a vector of 48 binary features indicating whether or not particular words occur in the email (the complete list of words can be found in the file "spambase_names.txt"). The label y takes on two values, one corresponding to "ham" (i.e., valid email) and the other to "spam". It should be easy for you to figure out whether "spam" is denoted by 0 or 1 (hint: look at the frequency of occurrence of certain cue words in examples of class 0 and class 1). There are 1500 training examples and 3101 test examples.

(a) **[M, 12 points]** Implement the Maximum Likelihood Naïve Bayes classifier with Laplacian smoothing illustrated in class and train it on the training set. Report the misclassification rate (i.e., the fraction of examples misclassified) obtained on the training set as well as on the test set. In order to do this, you need to implement the following functions:

- [**M**, 8 points] `q4_nb_train.m` trains Naïve Bayes given a training set. The function should learn the class prior and the class conditional probabilities for every feature.
- [**M**, 4 points] `q4_nb_predict.m` predicts the class labels for a given set of examples using a trained model. **Note:** Consider these tips to implement this function: first, because $p(x)$ is a constant for a given x, instead of computing exactly $p(y = 0|x)$ and $p(y = 1|x)$, you just need to find $\arg\max_y p(x|y)p(y)$; second, to avoid the numerical issues arising from the product of many small numbers in $p(x|y)$, use the logarithm function, which being a monotonically increasing function, will not change the result of the $\arg\max$ operator.

After you are done coding the two functions above, run the script `q4a.m` to compute the training and the test errors of your Naïve Bayes classifier.

(b) **[M, 3 points]** Using the learned Naive Bayes model, compute $p(y = 0|x_j = 1)$ for all features $j = 1, \ldots, 48$. Based on these probabilities, list the 6 words that are most indicative of a message being "spam", and the 6 words most indicative of a message being "ham" (the words associated to the binary features can be found in file "spambase_names.txt"). To accomplish this task, you must simply implement the function `q4_top_words.m`. Then run the script `q4b.m` to output the top 6 most indicative words for both classes.

(c) **[M, 23 points]** Write code to learn a spam filtering decision tree with binary decision tests of the form ($x_j = 1$?) at each node. You will use the entropy as a measure of impurity. Choose at each node the test yielding the maximum decrease of impurity. Stop splitting at a node if the number of training examples assigned to its decision region is less than $C \cdot m$, where m is the training set size and $C \in \{0.005, 0.01, 0.05, 0.1\}$. Plot the training error and the test error as functions of C in a single figure.

In order to accomplish this task, you need to implement the following functions:

- [**M**, 2 points] `q4_entropy.m` computes the entropy for a given set of class labels. [Hint: you must pay special care to the case of zero probabilities].
- [**M**, 2 points] `q4_info_gain.m` computes the information gain for a given subset (X, Y) of the training set using a specific feature as split test. You should call `q4_entropy.m` inside this function. [Hint: in your calculation of the information gain, omit the term corresponding to the entropy of the subset (X, Y) (i.e., the term denoted with $i(\tau)$ in the slides); the entropy of (X, Y) is a constant and it will have no impact on the choice of split as determined by comparing the information gains of different feature splits].
- [**M**, 1 point] `q4_leaf_info.m` computes the majority label and the posterior at the leaf node given the labels of the training examples falling in this node.
- [**M**, 4 points] `q4_split.m` finds the feature for splitting current training data that maximizes the information gain. You should use `q4_info_gain.m` in this function.
- [**M**, 8 points] `q4_dt_train_recursive.m` trains the decision tree recursively. The main function for learning a decision tree is `q4_dt_train.m` which is provided for you and requires no change. **Please read carefully the comments in `q4_dt_train.m`: they describe the data structure to represent the decision tree.** The function `q4_dt_train_recursive.m` should call your function `q4_leaf_info.m` when you reach a leaf node and it should use `q4_split.m` to find the best split at a non-leaf node. Use recursive calls to `q4_dt_train_recursive.m` to perform the learning on the children of the current node.
- [**M**, 6 points] `q4_dt_predict.m` predicts the labels of a set of examples using a given learned tree.

You can now run the script `q4c.m` to plot the training error and the test error of a decision tree learned for different choices of parameter $C \in \{0.005, 0.01, 0.05, 0.1\}$.

(d) **[M, 12 points]** Implement a random forest consisting of 11 independently learned random decision trees. At each node, consider a subset of 3 features randomly selected from all features yielding non-degenerate partitions (a partition is degenerate if one of the regions of the partition contains 0 training examples). Obviously, use all the features producing non-degenerate partitions if these are fewer than 3. Use the same impurity measure and the stopping criterion as above, but now with $C = 0.01$. Predict the class label using both the majority class scheme and the strategy based on the largest average posterior. Report the errors on the training set and the test set.

To accomplish these tasks, you are asked to implement the following functions:

- [**M**, 4 points] `q4_rf_split.m` this functions first identifies the feasible set of features (e.g. features that produce two non-empty branches). If the feasible set is empty, it should return 0. Otherwise, the function should first randomly choose a subset of (up to) three features from this feasible set, then return the best split among these features.

- [**M**, 4 points] `q4_rf_train_recursive.m` trains the random tree recursively in the random forest manner. The main function for training the random forest is `q4_rf_train.m` which we have provided and should be left unchanged.
- [**M**, 4 points] `q4_rf_predict.m` predicts the labels for a set of test examples X given a learned random forest *using the majority class scheme.* This function also computes the *average* class posteriors over all trees for the test examples.

You now should be able to run the script `q4c.m` to compute the training error and the test error for both prediction strategies, the majority class scheme and the largest average posterior.