CPSC Coding Project (due December 17)

matLearn

For the coding project, as a class we are going to develop a new Matlab toolbox for supervised learning, called *matLearn*. This toolbox will make a wide range of methods available, for addressing a variety of different supervised learning task. The main documentation for the toolbox will be a sequence of demos that compares/contrasts the different models in different situations. Your coding project is to implement at least one model and make at least one demo comparing this model to other models.

1 Structure of the Package

The matLearn functions will use the interface

```
[model] = matLearn_task_name(X,y,options)
```

In the above, you will replace 'task' with one of the tasks that *matLearn* will address (such as 'regression') and you will replace 'name' with the name of the model (such as 'logistic').

The variable X is our usual N by d design matrix and y is our usual n by 1 target vector. The variable *options* is a struct that specifies any non-default options. For example, if you are doing multi-class classification with logistic regression you could specify the number of classes using something like

```
options.nClasses = 5;
model = matLearn_classification_logistic(X,Y,options);
```

Calling the function *myProcessOptions* within the function lets you specify the possible arguments as well as their default values. Note that some models (like boosting) will actually take other models as an input. This will let users mix and match things where possible.

The returned variable *model* is also a struct. It should store the variables that are needed to make a prediction after the model has been trained, and it should also provide a function *predict* that makes predictions using the model,

```
yhat = model.predict(model,Xhat)
```

Thus, the format is a generalization of what we used in Assignment 6.

The demos will consist of scripts that are called directly, such as

```
matLearn_example_regression
```

Each demo should compare two or models on a supervised learning task, and somehow show a visualization of their performance. For example, in the assignments we've used regression problems in one variable and classification problems in two variables to illustrate various methods. You could alternately make demos that compare different methods in terms of test error or interpretability.

2 Examples

Several models/demos are included in *matLearn.zip*, you can run use these to get a rough idea of the expected code structure.

3 Notation

The above framework means it is easy to add new models, but we also want to be able to modify and understand existing models in the package. To achieve this goal, we are going to define standard notation for different commonly-occuring variables. Below is the current list, and if you have extra suggestions please post them to Piazza and I will update the list:

- X: Matrix where each row is a training example and each column is a feature.
- y: Vector containing the target variable for each training example.
- w: Weight vector for linear models.
- nTrain: Number of training examples (number of rows in X and Y).
- nTest: Number of testing examples (number of rows in $\hat{X}$ in the *predict*).
- nFeatures: Number of features (number of columns in X).
- nClasses: Number of classes (for multi-class classification).
- nMixtures: Number of mixtures (for mixture models).
- i: index of training example (use $i1$ and $i2$ if you need two indices).
- j: index of feature (use $j1$ and $j2$ if you need two indices).
- c: index of class label (use $c1$ and $c2$ if you need two indices).
- λ: strength of regularization parameter (use λ_1 and λ_2 if there are two regularizers).
- $nModels$: number of sub-models (for ensemble methods).
- z: Weight for each training sample.

In the function header, define all of the potential options fields for the model, as well as what they mean. Also put the year and your name. Further, please comment what each part of the code is doing, and try to make the code as readable as possible.

If your model has sub-routines, you should define them within the *.m* file containing the model, as opposed to external files. If people find they are using common sub-routines, we will define external files and update the common *matLearn.zip* file.

4 Optimization

Several models require solving generic numerical optimization problems. You can implement specialized solvers as options to solve these problems, but for solving generic problems the following solvers are included:

- *minFunc*: Finds a local minimizer of a differentiable function. This is basically a more advanced version of the *findMin* function from the assignments.
- *L1General*: Finds a local minimizer of a differentiable function plus ℓ_1-regularization.

- *minConf*: A set of functions that use projection/proximal methods to solve problems with simple constraints or simple regularizers.

5 Marking Scheme

The marking scheme is as follows:

- Notation and readability (**2 points**): You will get two points if you follow the standard notation, the function header follows the guidelines, you document all parts of the code, and write simple/readable code.
- Correctness and runnability (**3 points**): You will get three points if your implementation of the model is correct and if your demo runs after adding your files to a fresh download of *matLearn.zip*. You will lose 1 point each time I need to contact you because your code doesn't run.
- Value-added (**2 points**): You must have at least two aspects of your model or demo that make it more interesting than simply translating the model from a textbook or Wikipedia. There is a lot of flexibility in what the actual aspects could be, but a list of suggestions is given in Section 7.
- Collaboration (**1 point**): To get the final point, you must get in touch with one or more others member of the class who are working on a related model, and make a second demo that compares/contrasts the different models. Alternately, if it makes sense you could write one model function that combines two or more models where an extra field in *options* allows you to switch between the models. Some suggestions for this part are given in Section 8.

Note that course auditors only need to get 4 points to pass the course, so they do not necessarily have to do the value-added or collaboration parts (e.g., a correct and documented textbook implementation that runs is sufficient, but you are welcome to go further than that).

6 List of Models

Here is a list of the four tasks we will focus on this semester, as well as associated methods. We will keep a master list of the model each person has chose to do on Piazza (first come, first served), and you will have to choose a model to implement among those that have not yet been chosen. You can also implement a model that is not on the list, subject to approval by me. Items in blue are still available since the last time this document was updated (but check Piazza for the master list).

6.1 Task: regression

These are models that use x_i predict a single target label $y_i \in \mathbb{R}$:

- *matLearn_regression_stump.m*: Regression based on choosing the best variable $(x_i)_j$.
- *matLearn_regression_NB.m*: Regression based on independently minimizing the squared loss for each variable, $\sum_{i=1}^N \sum_{j=1}^d (w_j x_{ij} - y_i)^2$. This is like a naive Bayes version of least squares.
- *matLearn_regression_L2.m*: Regression based on minimizing the squared loss, $\sum_{i=1}^N (w^T x_i - y_i)^2$.
- *matLearn_regression_L1.m*: Regression based on minimizing the absolute loss, $\sum_{i=1}^N |w^T x_i - y_i|$.
- *matLearn_regression_Huber.m*: Regression based on minimizing the Huber loss.
- *matLearn_regression_student.m*: Regression based on minimizing a student 't' loss.

- *matLearn_regression_totalL2.m*: Regression based on the total least squares loss.
- *matLearn_regression_SVR.m*: Support vector regression using the ϵ-insensitive loss.
- *matLearn_regression_ARD.m*: Squared error with ℓ_2-regularization fit using type II maximum likelihood (automatic relevance determination).
- *matLearn_regression_MLP.m*: Squared error within a multi-layer perceptron.
- *matLearn_regression_KNN.m*: Interpolation based k-nearest neighbours.
- *matLearn_regression_NW.m*: Global interpolation using kernel function.

There are models that take a model as a sub-routine:

- *matLearn_regression_tree.m*: Regression tree.
- *matLearn_regression_GAM.m*: regression using generalized additive model.
- *matLearn_regression_local.m*: Local regression (fitting a model based on closest points).
- *matLearn_regression_bagging.m*: Regression based on the average prediction among models fit to bootstrap samples.
- *matLearn_regression_basis.m*: Regression under a change of basis.
- *matLearn_regression_CV.m*: Regression where cross-validation is used to choose a hyper-parameter.
- *matLearn_regression_CV2.m*: Regression where cross-validation is used to choose two hyper-parameters (e.g., λ_1 and λ_2 with elastic net regularization).

6.2 Task: classification2

These are models that use x_i predict a single binary target label $y_i \in \{-1, 1\}$:

- *matLearn_classification2_stump.m*: Classification based on a decision stump.
- *matLearn_classification2_perceptron.m*: Classification using the perceptron algorithm.
- *matLearn_classification2_logistic.m*: Classification using logistic regression.
- *matLearn_classification2_probit.m*: Classification using probit regression.
- *matLearn_classification2_SVM.m*: Classification using support vector machine
- *matLearn_classification2_SSVM.m*: Classification using smooth SVM (squared hinge loss).
- *matLearn_classification2_HSVM.m*: Classification using Huberized SVM.
- *matLearn_classification2_extreme.m*: Classification using a GLM with the extreme-value link.
- *matLearn_classification2_Cauchit.m*: Classification using a GLM with the Cauchit link.
- *matLearn_classification2_MLP.m: Classification using a multi-layer perceptron with binary loss (like cross-entropy).*
- *matLearn_classification2_mixtureLogistic.m: Classification using a mixture of logistic regression models.*

There are models that take a model as a sub-routine:

- *matLearn_classification2_regression.m*: Classification by using a regression model.
- *matLearn_classification2_tree.m*: Classification based on a decision tree.

- *matLearn_classification2_classification.m*: Using a multi-class classifier and treating the special case of binary labels.
- *matLearn_classification2_bagging.m*: Classification based on the average prediction among models fit to bootstrap samples.
- *matLearn_classification2_boosting.m*: Classification based on boosting a binary classifier.
- *matLearn_classification2_randomForest.m*: Classification based on a random forest.
- *matLearn_classification2_GAM.m*: classification using generalized additive model.
- *matLearn_classification2_basis.m: Classification under a change of basis.*
- *matLearn_classification2_CV.m*: Classification where cross-validation is used to choose a hyper-parameter.
- *matLearn_classification2_CV2.m*: Classification where cross-validation is used to choose two hyper-parameters (e.g., λ_1 and λ_2 with elastic net regularization).

6.3 Task: classification

These are models that use x_i predict a single target label $y_i \in \{1, 2, \ldots, K\}$:

- *matLearn_classification_KNN.m*: Classification using k-nearest neighbours.
- *matLearn_classification_stump.m*: Classification based on a decision stump.
- *matLearn_classification_logistic.m*: Classification using multinomial logistic regression.
- *matLearn_classification_SVM.m: Classification using multi-class support vector machine.*
- *matLearn_classification_MLP.m*: Classification using a multi-layer perceptron with a multi-class loss (like softmax).

Below are generative approaches to classification:

- *matLearn_classification_generativeNB.m*: Naive Bayes.
- *matLearn_classification_generativeGDA.m*: Gaussian discriminant analysis.
- *matLearn_classification_generativeLaplace.m*: Generative classifier where you fit a Laplace distribution to each classifier.
- *matLearn_classification_generativeStudent.m*: Generative classifier where you fit a multivariate t distribution to each classifier.
- *matLearn_classification_generativeMixtureGaussian.m*: Generative classifier where you fit a mixture of Gaussian distribution to each classifier.
- *matLearn_classification_generativeMixtureLaplace.m*: Generative classifier where you fit a mixture of Laplace distribution to each classifier.
- *matLearn_classification_generativeMixtureStudent.m: Generative classifier where you fit a mixture of Laplace distribution to each classifier.*
- *matLearn_classification_generativeKDE.m*: Generative classifier where you use a kernel density estimator (AKA Parzen window).

There are models that take a model as a sub-routine:

- *matLearn_classification_tree.m*: Classification based on a decision tree.

- *matLearn_classification_regression.m*: Classification by using a multiple regression model and a '1 of K' of the class labels.
- *matLearn_classification_1vA.m*: Training and combining binary 'one vs. all' classifiers.
- *matLearn_classification_1v1.m*: Training and combining binary 'one vs. one' classifiers.
- *matLearn_classification_ECOC.m*: Training and combining binary classifiers using error-correcting output codes.
- *matLearn_classification_GAM.m*: classification using generalized additive model.
- *matLearn_classification_bagging.m*: Classification based on the average prediction among models fit to bootstrap samples.
- *matLearn_classification_boosting.m*: Classification based on boosting a classifier.
- *matLearn_classification_basis.m: Classification under a change of basis.*
- *matLearn_classification_CV.m*: Classification where cross-validation is used to choose a hyper-parameter.
- *matLearn_classification_CV2.m*: Classification where cross-validation is used to choose two hyper-parameters (e.g., λ_1 and λ_2 with elastic net regularization).

6.4 Task: ordinal

This is the same task as *classification*, but these models assume that the classes are ordered:

- *matLearn_ordinal_logistic.m*: Ordinal logistic regression.
- *matLearn_ordinal_probit.m*: Ordinal probit regression.
- *matLearn_ordinal_extreme.m*: Ordinal extreme-value regression.
- *matLearn_ordinal_regression.m*: Ordinal regression by fitting a basic regression model.

7 Value-Added

Here are some suggestions for the *value-added* part of your project (you may have to do some research):

- *Weighting*: Add the ability to specify an *options.weights*, giving the weight for each training example.
- *MAP*: Add the ability to use a regularizer, such as ℓ_2-regularization or ℓ_1-regularization (for linear models) or a beta prior (for models like naive Bayes).
- *Splitting Criteria and pruning trees*: For decision trees, add several options for the splitting criterion (classification error, infomax, Gini index, etc.) and add a pruning strategy.
- *Scalability*: Make an implementation that scales to larger data sets (e.g. allow conjugate gradient if you are doing least squares, or allow diagonal covariance if you are fitting a Gaussian, or coordinate descent or stochastic gradient methods for numerical optimization).
- *Variants*: Add some common variants on the basic method (e.g., logitBoost in addition to AdaBoost).
- *Probabilistic Output*: Implement an extra function in the model that makes probabilistic predictions over the classes instead of a hard assignment.
- *Dual optimization*: Add the ability to optimize the model using dual coordinate ascent.

- *Use sparsity*: Make the code take advantage of sparsity present in the design matrix X.
- *Bayesian*: Allow the ability to use the posterior mean, or median, give the full posterior predictive distribution, or use type II maximum likelihood.
- *Smarter searches*: In methods that use a naive search, like cross-validation, add the ability to prune the search or add the ability to optimize the parameter (e.g., for regularization parameters).
- *Kernelize*: Add the ability to use kernels with the method.

Finally, instead of having the value-added component related to the model, you could also produce an interesting demo. This could be based on an easily-interpretable dataset or some visualizing interesting property.

8 Possible Collaborations

There are two possible collaboration types: *refactoring* or *comparison*. The *refactoring* type involves combining two or more of the above models into a single function, where you can switch between different models using specific choices of the *options* structure. The *comparison* type involves comparing two or more of the above models. There is no limit to the size of the collaboration groups, but they should be sensical collaborations (e.g., it wouldn't make sense to have refactor decision trees and SVMs into one function).

For *refactoring* collaborations, some examples of logical refactoring might be (there are many other possible combinations):

- Combine decision stumps and decision tree to use one common model function.
- Combine logistic regression and probit regression to use one common model function.
- Combine the bagging models for the different tasks.
- Combine the cross-validation models fothe different tasks.
- Write a generic generative classifier that can takes a generative model as an input option.

For *comparison* collaborations, some examples of logical comparisons that would make good demos include (there are many other possible combinations):

- Comparing linear regression or classification methods using different loss functions.
- Comparing decision stumps to decision trees, or comparing these before/after boosting.
- Comparing MLPs trained with squared error to training with the softmax error.
- Generative classifiers using different types of generative distributions.
- Comparing test performance with and without cross-validation.
- Comparing linear models (like logistic regression) to non-linear models (like GAMs, mixtures, or MLPs) or to non-parametric models (like KNN).

9 Deliverables

By the end of December 17, you need to e-mail me a .zip file containing:

1. Your model function (and external functions that it calls).
2. Your individual demo script.

3. A text file containing a short description of your value-added contributions, and your collaboration contribution.

If you did a *comparison* collaboration, you also submit the comparison demo script (if it is different than your individual demo script). If you did a *refactoring* collaboration, you have the option of submitting one zip file on behalf of the collaboration group.