CSE517A – Entry Exam

M. Neumann

due: 14 Jan 2020 at 10am

- This entry test is a **take-home exam** to assess the knowledge on the **prerequisites** for CSE517A – especially materials covered in CSE417T. You have to work on this test on *your own* without the help of any third party. No group work! Please, understand that we cannot offer TA or instructor support on the tested materials as they are *prerequisites* and thus not part of CSE517A.
- This entry test is *only* for students that did **NOT** take CSE417T and are **conditionally enrolled** in or **waitlisted** for CSE517A.[1]
- To be enrolled in or stay waitlisted for CSE517A, you will need to pass this test. **A passing grade is any score** $\geq 70\%$**.** If you do not pass this test or fail to submit it on time, we will **unenroll** you from the course or **remove** you from the waitlist.
- **Academic integrity**: Everything that you turn in for this test must be your *own* work. If you willfully misrepresent **someone else's work** as your own, you are guilty of cheating. Cheating, in any form, will not be tolerated and will be reported to the **School of Engineering academic integrity officer**, who will handle the case. If the committee finds that a violation of the academic integrity policy as stated here and in the CSE517A course syllabus has occurred, this violation will then be noted in the student's permanent record at Washington University. You can find more details about academic integrity and the process on the CSE FAQ (`https://www.cse.wustl.edu/%7Ecytron/AdvisingFAQ/#quicklinkacademicintegrity`).
- **Submission instructions**:
  - Include your **name**, **wustlkey**, and (6-digit) **student ID** on *every* page.
  - We recommend that you type all solutions in LaTeX and compile a PDF. If you do not type your submission and your hand writing is not readable, we **cannot give you credit**.
  - **Start every problem on a new page!**
  - Keep your written answers brief and to the point. Include all derivations and define all used variables. Incorrect or rambling statements can hurt your score on a question.
  - Submit your **printed written solutions** at the beginning of the lecture on TUE Jan 14 2020 at 10am. **This is a hard deadline – no submission via email.**
  - Email the solutions to all **implementation** problems keeping the provided folder structure intact to our head TA (email address will be announced in the first lecture) before the end of the day of the first lecture.

[1] We will not grade this test for students that took CSE417T/fulfill the prerequisites.

Part I: Machine Learning Foundations

Problem 1 (*10 points*) Probability

Problem 1.7 in Learning from Data (LFD) by Abu-Mostafa, Magdon-Ismail, and Lin (CSE417T course book)

Problem 2 (*10 points*) Thinking about Errors

(a) (*5 pts*) Suppose you have the following kind of learning algorithm: a linear separator that returns an (augmented) weight vector $\mathbf{w}$, that, when applied to an (augmented) example $\mathbf{x}$, returns some $\mathbf{w}^T\mathbf{x} = s \in \mathbb{R}$, and the hypothesis for classification purposes is defined as $\text{sign}(s)$. Let $e_{\text{class}}(s, y)$ be the classification error on a point with true label y, $e_{\text{sq}}(s, y) = (y - s)^2$ be the squared error measure between the actual real number s and the true label y. Show that $e_{\text{class}}(s, y) \leq e_{\text{sq}}(s, y)$.

(b) (*5 pts*) People sometimes directly use linear regression even when y is binary. Point out (1) a possible technical problem with the predictions of this model, and (2) a possible problem in training the model (hint: for part (2), consider what happens when $|\mathbf{w}^T\mathbf{x}|$ is very high on a particular point).

Problem 3 (*5 points*) Regularization

Alice is trying to learn a classifier for a specific problem. She tries two different learning methods, A and B, and two different levels of regularization for each, Level 1, which is weaker regularization, and Level 2, which is stronger regularization. She has a lot of data, so she splits it into training and validation sets that are large enough to give good estimates, and then finds the training and validation errors for all four models. She then sends this information to Bob. Unfortunately, it gets corrupted on the way, and some of the error numbers get erased (those that remain are correct). Bob receives the following information:

Method	Training Error	Validation Error
A1	9%	
A2		
B1		7%
B2	10%	

Unfortunately, Bob has no way of getting touch with Alice and has to simply use this data to decide which method to use on the test data. If you were Bob, which method would you choose, and why?

Problem 4 (*15 points*) Generalization and VC-Dimension

Problem 2.13 in Learning from Data (LFD) by Abu-Mostafa, Magdon-Ismail, and Lin (CSE417T course book)

Part II: Perceptron and k-Nearest Neighbor

Problem 5 (*15 points*) Consider the following experiment on **perceptron learning** for random training sets of dimension 10:

- Generate an 11-dimensional weight vector $\mathbf{w}^*$, where the first dimension is 0 and the other 10 dimensions are sampled independently at random from the uniform (0, 1) distribution (the first dimension will serve as the threshold and we set it to 0 for convenience).
- Generate a random training set with 100 examples, where each dimension of each training example is sampled independently at random from the uniform (-1, 1) distribution. The examples are all classified in accordance with $\mathbf{w}^*$.
- Run the perceptron learning algorithm, starting with a zero weight vector, on the training set you just generated. Keep track of the number of iterations it takes to learn a hypothesis that correctly separates the training data.

(a) (*5 pts*) Write code in Matlab to perform the above experiment and then repeat it 1000 times (note that you're generating a new $\mathbf{w}^*$ and a new training set each time). You may complete the two stub files (`perceptron_experiment.m` and `perceptron_learn.m`) for this purpose. The files have comments that explain their inputs and outputs.

Email your final versions of all code files as a single compressed folder using **your wustlkey** as foldername to our head TA (the email address will be provided in the first lecture) keeping the original folder structure intact. For this problem the subfolder `code_perceptron` needs to include all your modified **stub functions**.

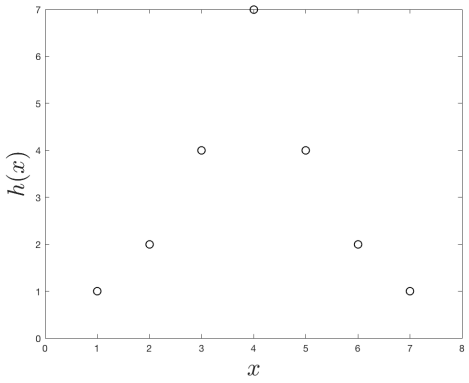
Once you have your code working:

(b) (*5 pts*) Plot a histogram of the number of iterations the algorithm takes to learn a linear separator; you should submit this with your writeup.

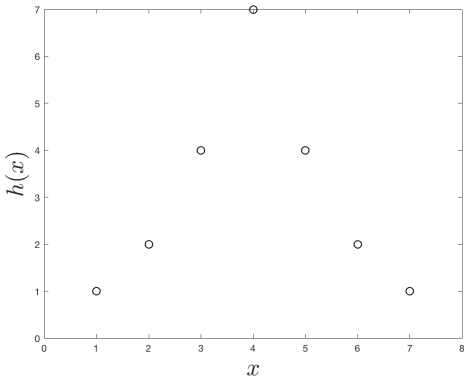
(c) (*5 pts*) How does the number of iterations compare with the bound on the number of iterations derived in part (e) of the previous problem? Note that this bound will be different for each instantiation of $\mathbf{w}^*$ and training set so in order to answer this question, you will need to analyze the distribution of differences between the bound and the number of iterations. Plot and submit a histogram of the log of this difference. Discuss your interpretation of these results.

Problem 6 (*15 points*) Consider k-**NN for regression** on the 1-dimensional input data plotted below.

(a) (*8 pts*) Draw the predictive model for $k = 2$ using the average label (left panel) and the weighted average label (right panel) where the weight is the inverse proportion to the distance from the test point.



2-NN regression



weighted 2-NN regression

(b) (*4 pts*) Provide the parametric function for the unweighted 2-NN model for the example data above.

(c) (*3 pts*) Why is this still a *non-parametric* model?

Part III: Decision Trees, Bagging, and Boosting

Problem 7 (*20 points*) Consider the following set of training examples (PLAYSPORTS is the predictive variable):

OUTLOOK	TEMPERATURE	WIND	PLAYSPORTS
Sunny	Hot	Strong	-
Sunny	Mild	Strong	-
Sunny	Hot	Weak	-
Overcast	Hot	Strong	-
Overcast	Hot	Strong	-
Overcast	Hot	Strong	-
Overcast	Mild	Strong	+
Overcast	Mild	Strong	+
Overcast	Hot	Weak	+

(a) (*4 pts*) What is the termination criteria for a vanilla decision tree algorithm for binary/categorical class labels? Build a tree that splits on OUTLOOK (feature 1) on the root node, then on TEMPERATURE (feature 2) on the next level (all nodes in that level), WIND (feature 3) on the next level, OUTLOOK (feature 1) on the next level etc. until the termination criteria is reached. Draw the tree below.

(b) (*10 pts*)Compare the information gain for *the performed first split* (x_i = OUTLOOK) with the other two possible first splits (x_i = TEMPERATURE and x_i = WIND). Was the first split optimal wrt. the information gain measure? Include all relevant information gain values and their computation steps in your answer.

(c) (*6 pts*) Assume you have the following validation set.

OUTLOOK	TEMPERATURE	WIND	PLAYSPORTS
Sunny	Hot	Weak	-
Overcast	Mild	Strong	+
Overcast	Hot	Weak	-

How can you use this to prune your decision tree? Briefly describe the validation set pruning method and prune the tree obtained in part (a). Draw the pruned tree. What is the validation error before pruning? What is the validation error after pruning?

Problem 8 (*10 points*) Recall the following properties of **AdaBoost**:

$$\alpha = \frac{1}{2}\log\left(\frac{1-\epsilon}{\epsilon}\right)$$

$$Z = 2\sqrt{\epsilon(1-\epsilon)} \quad \text{(normalization constant)}$$

(a) (*4 pts*) Let the error of the first weak learner be $\epsilon > 0$. State the weight updates in AdaBoost for correctly classified inputs and misclassified inputs (without normalization) in terms of e^{α}.

(b) (*6 pts*) Show that after the weight updates and normalizing all weights, the re-weighted error of the first weak learner is exactly $\epsilon^* = 0.5$.

Problem 9 (*50 points*) Implementation: Bagged and Boosted Decision Trees

In this problem you will implement a decision tree algorithm and then use it for bagging and boosting. Download the stub files from the course webpage. You should have the following files:

Stubs:

`id3tree.m`	Returns a decision tree using the minimum entropy splitting rule (implemented for you in `entropysplit.m`)
`evaltree.m`	Evaluates a decision tree on a test data.
`prunetree.m`	Prunes a tree using a validation set.
`forest.m`	Builds a forest of id3trees.
`evalforest.m`	Learns and evaluates a forest.
`boosttree.m`	Applies adaboost to your id3tree.
`evalboost.m`	Learns and evaluates a boosted classifier.

Already implemented:

`entropysplit.m`	This function implements minimum entropy splitting using information gain/entropy.
`cart_tictoc.m`	This function computes train and test accuracy and runtime for all your algorithms (uses `analyze.m`).
`cart_visual.m`	This function visualizes trees (uses `vistree.m`).

Datasets: (download from Piazza resources)

`iris.mat`	Subset of the Iris flower dataset (`https://en.wikipedia.org/wiki/Iris_flower_data_set`)
`heart.mat`	Another dataset for binary classification.

Important: This problem will be automatically tested and graded by our autograder. Do not change the signatures (input, output data structures) of any function! Do not add any additional files or folders.

As MATLAB does not support pointers and is really only good with matrices, we will represent a tree through a **matrix** T. A tree T with q nodes must be of dimensions $6 \times q$, where each column represents a different node in the tree. The very first column is the root node and each column represents the following information:

(1) prediction at this node

(2) index of feature to cut

(3) cutoff value c ($<= c$: left, and $> c$: right)

(4) index of left subtree (0 = leaf)

(5) index of right subtree (0 = leaf)

(6) parent (0 = root)

Please take a minute and familiarize yourself with the implementation of `entropysplit.m`. This will make the following implementation tasks easier. `entropysplit.m` searches through all features and returns the best split (feature, cut-threshold combination) based on information gain (in this case entropy). You don't need to change this anything in this file.

(a) Implement the function `id3tree.m` which returns a decision tree based on the minimum entropy splitting rule. The function takes training data, test data, a maximum depth, and the weigh of each training example (used in entropy splitting). You can visualize your tree with the command `cart_visual`. Maximum depth and weight are optional arguments. If they are not provided you should make maximum depth infinity and equally weight each example. (Hint: To speed up your code make sure you initialize the matrix T with the command `T = zeros(6, q)` with some estimate of q.) You should use the function `entropysplit.m`

(b) Implement the function `evaltree.m`, which evaluates a decision tree on a given test data set. You can test the accuracy of your implementation with `cart_tictoc`.

(c) Decision trees (without depth limit) are **high variance** classifiers. Consequently, overfitting is a serious problem. One cure around it is to prune your tree to stop making "silly" splits that overfit to the training set. You prune a tree bottom up, by removing leaves that do not reduce the validation error (**reduced error pruning**). Implement the function `prunetree.m`, which returns a decision tree pruned for a validation data set. The accuracy on the validation set should be the same or higher after pruning.

(d) A more effective way to prevent overfitting is to use **bagging**. Implement the function `forest.m`, which builds a *forest* (**not** a random forest) of ID3 decision trees. Each tree should be built using training data drawn by randomly sampling n examples from the training data with replacement, you can use MATLAB's `randsample` function to do this. (Do **not** randomly sample features.)

(e) Implement the function `evalforest.m`, which evaluates a forest on a test data set. Remember that random forests make predictions by majority vote.

(f) Another option to improve your decision trees is to build trees of small depth (e.g. only depth=3 or depth=4). These do not have high variance, but instead suffer from **high bias**. You can reduce the bias of a classifier with **boosting**. Implement the function `boosttree.m`, which applies ADABOOST on your `id3tree` functions.

(g) Implement the function `evalboost.m`, which evaluates an ensemble of boosted decision trees.

Once, you have implemented all functions run `cart_tictoc.m` on both datasets to compute training and test error as well as runtimes for all four algorithms. Play around with the input parameters of your forest and Adaboost methods.

Email your implementations as a compressed folder using **your wustlkey** as foldername to our head TA (the email address will be provided in the first lecture). The subfolder `code_cart` needs to include all your modified **stub functions** – stub files **only**, do not include the data or any already implemented files.

Part IV: Support Vector Machines and Kernels

Consider the *hard margin* SVM formulation without *slack variables* for all problems.

Problem 10 (*25 points*) The support vector machine (SVM) solution to the linear classification problem $h(\mathbf{x}) = sign(\mathbf{w}^\top \mathbf{x} + b)$ on $D = \{(\mathbf{x}_i, y_i)\}_{i=1\ldots n}$ is the separating hyperplane that maximizes the margin, where $\mathbf{x}, \mathbf{x}_i, \mathbf{w} \in \mathbb{R}^d$ and $b, y_i \in \mathbb{R}$.
Important: Clearly define all newly introduced variables and the dimensions of vectors and matrices.

(a) (*3 pts*) Derive the formula for the margin $\gamma(\mathbf{w}, b)$ of a separating hyperplane $\mathcal{H}$ as the distance between the hyperplane and the nearest training point.

(b) (*5 pts*) Now, we want to maximize this margin and ensure that $\mathcal{H}$ is indeed separating the training data. Write down the optimization problem that yields the maximum margin classifier and derive the **(primal) SVM optimization problem**. Show your derivations and briefly explain every step.

(c) (*2 pts*) How many parameters/variables does the primal optimization problem have? How many constraints?

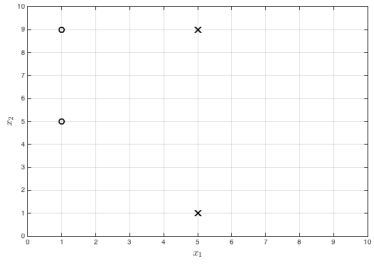
(d) (*2 pts*) When is the primal optimization as solution to the SVM classifier disadvantageous? Discuss two situations.

(e) (*8 pts*) Carefully derive the **dual SVM optimization problem**. Include all steps of your derivation with brief verbose explanations.

(f) (*2 pts*) How many parameters/variables does the dual optimization problem have? How many constraints?

(g) (*3 pts*) State the dual form of the SVM classifier used for predictions and explain why only support vectors contribute to the prediction.

Problem 11 (*10 points*) Assume you train a hard margin SVM on the following data set.



(a) (*2 pts*) Draw in the decision boundary.

(b) (*2 pts*) Add two $\times$ data points such that the problem still yields a feasible solution, i.e., the SVM must still find a valid solution after each point is added. The first data point (mark as big $\times$) should not affect the decision boundary. The second data point (mark as x with circle $\otimes$) should affect the decision boundary (do **not** add the new decision boundary!).

(c) (*6 pts*) Explain why the leave-one-out cross validation error of a binary SVM (without slack variables) trained on n data points is bounded by the number of support vectors m:

$$\varepsilon_{\text{LOO}} \leq \frac{m}{n}$$

Problem 12 (*15 points*) Kernel SVM

(a) (*5 pts*) In your own words carefully explain what the *kernel trick* is and why and when it is useful.

(b) (*5 pts*) Kernelize the SVM classifier. Clearly state the **kernel SVM optimization problem** and the formula of the SVM classifier used for predictions. Include all derivations.
Important: Clearly define all newly introduced variables and their dimensions.

(c) (*5 pts*) If we are to store each real number as a floating point number, how many floating point numbers would be required to store the trained model in its *primal/non-kernelized* version and in the *kernelized version* of SVM?