# COMPSCI 671D Homework 1

January 2019

## 1 Perceptron I

a ) Consider a dataset $\mathcal{D} = (X, Y)$ where $X$ is an $n \times p$ feature matrix (that is, with $n$ $p$-dimensional samples) and $Y$ is an $n$-dimensional binary output vector with dimensions corresponding to the $n$ samples. Let $\mathcal{D}^{'}$ be a permutation of the samples of $\mathcal{D}$. Suppose we run the perceptron algorithm on $\mathcal{D}$ and $\mathcal{D}'$ respectively by looking at the samples in a sequential order, do we necessarily get the same hyperplane? Do they necessarily converge in the same amount of time? (if yes, prove the claim, and if no, give an example).

b ) Show that $\mathcal{D} = ([X_1, X_2], Y)$ given by Table 1 (where $X_1 \in \{0, 1\}$, $X_2 \in \{0, 1\}$, and $Y = X_1$ XOR $X_2$) is not linearly separable. In fact, the XOR function can be generalized to more inputs, and this is called the parity function. The parity function is true if the number of ones in a bit vector is odd, and false otherwise. Prove that the parity function for length $n$ inputs is also not linearly separable for any $p \geq 2$ dimensions. (Again create a dataset that has one data point on each possible point in the space.)

c ) One way to deal with functions that aren't linearly separable is to augment your feature space so that the target function is linearly separable in the augmented space. Demonstrate (by providing a set of weights for a linear model) that XOR is linearly separable in the feature space with three features $X_1$, $X_2$, ($X_1$ AND $X_2$).

d ) (*Optional*, **Extra Credit**) Suppose we have $n$ points in $\mathbb{R}^p$. Consider all possible assignments of these $n$ points into two classes. We have $2^n$ such assignments. How many of these partitions yield linearly separable classes, i.e. where the two classes can be perfectly separated by an $(p-1)$ dimensional hyperplane? (The only thing we assume about the points is that they are in general position, which means that any subset containing $n$ points where $n < p$ is linearly independent.) [Hint: You might find it useful to perform recursion on the number of points. Think about whether there is a separating hyperplane that would intersect the new point or not]

Table 1: Dataset $\mathcal{D}$ showing values for $X_1$, $X_2$ and corresponding $Y$ for each sample

| $X_1$ | $X_2$ | $Y$ |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

## 2 Perceptron II

In this question we will explore the application of the perceptron algorithm to real-world problems. The basic idea of this section is to consider various hypothetical questions and use them to better understand the behavior of the algorithm.

For this question, we will try to classify the breast cancer data (available on the UCI ML repository) into two classes: no-recurrence-events and recurrence-events. Note that the data has 9 features in total. We have created a cleaned up version of the online dataset in a file titled "breast-cancer-cleaned.csv" which is available with the homework zip file. (Do not use the online version.)

For this problem, we will only consider two of these features: "tumor-size" and "age." Download the data-set from the homework folder, load and shuffle the data; finally, divide your now-shuffled data into a training set consisting of 70% of the data and a test set consisting of the remaining 30%.

a ) Examine the the projection of the training data in the "tumor-size"-"age" feature space. Are our two classes of interest linearly separable? (In other words, if we consider only these two features from the data, can we construct a separating hyperplane between samples with no-recurrence-events and those with recurrence-events?) Support your answer with evidence. Plot the data on a 2-D scatter plot ("tumor-size" versus "age").

b ) Implement the perceptron algorithm (in your favorite programming language).

c ) Train your perceptron model using the training data using above mentioned features ("tumor-size" and "age") and outcome class (no-recurrence-events, recurrence-events) corresponding to each sample. Plot the learned decision boundary on the scatter plot you plotted in the previous part.

d ) Now shuffle the training dataset and repeat the training procedure in exactly the same way. Plot the second decision boundary on the same plot. Are both the decision boundaries same? Explain your observation.

e ) Plot the training and test accuracy at each update for both the cases (before shuffling and after shuffling) and compare their empirical rates of convergence. Are they the same? Explain your results.

# 3 ROC, AUC and Cross-Validation

Download the forest cover dataset from UCI machine learning repository. The challenge is to predict forest cover type. As per the data description provided by UCI ML repository:

> The challenge is to predict forest cover type from cartographic variables only (no remotely sensed data). The actual forest cover type for a given observation (30 x 30 meter cell) was determined from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent variables were derived from data originally obtained from US Geological Survey (USGS) and USFS data. Data is in raw form (not scaled) and contains binary (0 or 1) columns of data for qualitative independent variables (wilderness areas and soil types). This study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. These areas represent forests with minimal human-caused disturbances, so that existing forest cover types are more a result of ecological processes rather than forest management practices. Some background information for these four wilderness areas: Neota (area 2) probably has the highest mean elevational value of the 4 wilderness areas. Rawah (area 1) and Comanche Peak (area 3) would have a lower mean elevational value, while Cache la Poudre (area 4) would have the lowest mean elevational value. As for primary major tree species in these areas, Neota would have spruce/fir (type 1), while Rawah and Comanche Peak would probably have lodgepole pine (type 2) as their primary species, followed by spruce/fir and aspen (type 5). Cache la Poudre would tend to have Ponderosa pine (type 3), Douglas-fir (type 6), and cottonwood/willow (type 4). The Rawah and Comanche Peak areas would tend to be more typical of the overall dataset than either the Neota or Cache la Poudre, due to their assortment of tree species and range of predictive variable values (elevation, etc.) Cache la Poudre would probably be more unique than the others, due to its relatively low elevation range and species composition.

We will perform the experiment with an aim to understand the usage of ROC, AUC and CV better. In particular, our goal is to observe whether nested cross-validation actually succeeds in choosing the best parameter value. (Remember that there's no guarantee that it will!) For this problem, you should use one of the many machine learning toolkits for the languages we use in this course. For example, scikit-learn is the most widely used Python library for ML; alternatively, many of the most popular ML algorithms are implemented in the Machine Learning Toolbox for Matlab. You are allowed to use any package of your choice as long as it implements a version of **Regularized Logistic Regression** with a regularization parameter which we will call $K$. This is the classification algorithm we will use throughout this question.

a ) Set up 10-fold CV (the version that is just for evaluation). Run vanilla logistic regression (that is, with no regularization). Show the ROC curves for all 10 test folds on the same plot (they will overlap a lot).

b ) Fix one of the folds as a test set for this next experiment (1/10th of the data, which is not used for training at all). There are many variations of nested CV, but we will choose one particular version. We set up nested CV (the version that tunes the regularization parameter) by doing the following: choose 12 possible values for the regularization parameter (maybe $K$=1, .1, .01, etc). Pick a value of $K$, train with 8 of the (training) folds, and rotate which is the remaining validation fold (this operation gets repeated 9 times, with each of the training folds as an internal validation fold). Report the mean validation accuracy for each $K$ across the 9 experiments. Use this strategy to determine the best value for $K$ and report which $K$ you chose. Now you have completed one "fold" of nested cross validation. Let's see if it actually worked in the next parts of the problem. If it worked, you chose the best $K$.

c ) Create models for *each* of the 12 different values of $K$ on the full training set (9/10ths of the data), evaluating accuracy for each of these 12 models on the test set. Did the $K$ you chose using your limited version of nested CV perform the best?

d ) In the previous part, we asked you to create models for each of the 12 $K$'s on the training set. Visually display the distribution of performance of the 12 models, where performance is measured according to (a) test accuracy (b) AUC. You can use box plots (the box plot input would be 12 values, one for each model) or violin plots if you would like to (those are both Cynthia's favorite) or another method. From these plots, how substantial of an effect does it appear tuning parameters such as this one can have? (Note that in general, your results will depend on your choices of the $K$ values as well as the dataset you are working on and the algorithm you use.)

e ) Using the same set of 12 models trained in part b): generate the ROC curve on the test set for each model. Show all of these on the same plot. This will provide you a visual description of how sensitive the result is to the choice of $K$ in a more detailed sense than the single-values from part c) because the ROC curves display the full ranking of predictions on the data. After this experiment, would you be willing to run an algorithm without tuning the parameter values? Why or why not?