Comp 6321 - Machine Learning - Assignment 3

Federico O'Reilly Regueiro

November 10^{th}, 2016

Question 1: Midterm preparation question

Propose an adequate learning algorithm for each instance.

1.a 1000 samples, 6-dimensional continuous space, classify ∼100 examples.

This could be a candidate for knn, despite the dimensionality approaching higher orders. We can still somewhat escape the curse of dimensionality, since for non-parametric methods, in order to have an optimal error rate e we require at least n samples where $n \sim \left(\frac{c}{e}\right)^{\frac{d+4}{4}}$, c is the number of classes, and d is the number of dimenions[1]. Having 1000 samples in a 6-dimensional space would still allow for an error $e = 0.0632$ in the best case[2]. An appropriate value for k would need to be found via cross-validation.

1.b Clasifier for children in special-ed, justified to the board before it's implemented.

One of the easiest classification algorithms to explain in layman's terms is decision trees; since the method should be justified to the board, this would probably be an adequate choice. Furthermore, given that the stakes for such a classification are very high, an ensemble approach such as random forests / bagging could increase the classifier's performance by diminishing the tree's inherent variance and tendency to overfit.

1.c Binary classification, train with very large data-set of products / customer preferences. Input - 1 million bits - other clients' preferences. Frequent updates.

A recommender system of this nature could use naive bayes in a similar way to the document classification example presented in class. In this case, the features of each product are the clients who have shown interest in the product. The training would rely on the trends in clients' preferences accross all products. NB could work well given the size of the dataset and the need for frequent updates. However, a drawback to this approach is that the recommender assumes feature-independence while relying on the underlying relation between customer preferences, which in turn implies feature-dependency[3]. A similar problem arises in document classification, where the presence of words in a document cannot really be considered independent, yet NB still performs well for said task.

[1] See the non-parametric methods pdf used in class.

[2] Said otherwise, a possible success rate of 93.68%

[3] Eg, If customer A and customer B have both shown interest in a large set of products, they are likely to show similar preferences for new products.

1.d 40 attributes, discrete and continuous, some have noise; only about 50 labeled observations.

With few examples and a fair amount of features, the curse of dimensionality haunts this classifiation. The presence of noise and the need for some sort of reduction in dimensionality might be well served by logistic regression with L1 regularization extensible to a kernel approach if the classes do not seem linearly separable. K-fold cross-validation, with a relatively small k given the dataset size, would be necessary in order to find the appropriate rate for regularization.

Question 2: Properties of entropy

2.a Compute the following for (X, Y): $p(0,0) = 1/3, p(0,1) = 1/3, p(1,0) = 0, p(1,1) = 1/3$.

i $H[x] = -\sum_x p(x)log_2(p(x)) = -\frac{1}{3}log_2\left(\frac{1}{3}\right) - \frac{2}{3}log_2\left(\frac{2}{3}\right) = .9182$

ii $H[y] = -\sum_y p(y)log_2(p(y)) = -\frac{1}{3}log_2\left(\frac{1}{3}\right) - \frac{2}{3}log_2\left(\frac{2}{3}\right) = .9182$

iii $H[y|x] = -\sum_x p(x)H[Y|X=x] = -\frac{2}{3}\left(\frac{1}{2}log_2\left(\frac{1}{2}\right) + \frac{1}{2}log_2\left(\frac{1}{2}\right)\right) = \frac{2}{3}$

iv $H[x|y] = -\sum_y p(x)H[X|Y=y] = -\frac{2}{3}\left(\frac{1}{2}log_2\left(\frac{1}{2}\right) + \frac{1}{2}log_2\left(\frac{1}{2}\right)\right) = \frac{2}{3}$

v $H[x,y] = -\sum_x\sum_y p(x,y)log_2(p(x,y)) = 3\left(-\frac{1}{3}log_2\left(\frac{1}{3}\right)\right) = 1.5849$

vi $I[x,y] = \sum_x\sum_y p(x,y)log_2\left(\frac{p(x,y)}{p(x)p(y)}\right) = H[x] - H[x|y] = 0.2516$

2.b Prove maximum entropy in a discrete distribution happens in U

We wish to find:

$$\arg\max_{p_n}\sum_{n=1}^{N}p_n log(p_n)$$

With constraints:

$$1 - \sum_{n=1}^{N}p_n = 0$$

$$p_i \geq 0,\ \forall i \in \{1, 2, \dots, N\}$$

We use Lagrange for maximization with constraints with a lagrangian multiplier only for the first constraint[4]:

$$\mathcal{L}(p_1, p_2, \dots, p_n, \lambda) = \sum_{n=1}^{N}p_n log(p_n) - \lambda(1 - \sum_{n=1}^{N}p_n)$$

[4] The second series of constraints are satisfied by the solution using only $1 - \sum_n p_n = 0$.

And by setting the gradient of the Lagrangian function to 0, $\nabla_{p_1,p_2,\ldots p_N,\lambda}\mathcal{L}(p_1, p_2, \ldots, p_n, \lambda = 0,$, we are left with a system of equations:

$$\begin{aligned}
\frac{\partial_{\mathcal{L}}}{\partial_{p_1}} \sum_{n=1}^{N} p_n log(p_n) - \lambda(1 - \sum_{n=1}^{N} p_n) = 0 \\
\frac{\partial_{\mathcal{L}}}{\partial_{p_2}} \sum_{n=1}^{N} p_n log(p_n) - \lambda(1 - \sum_{n=1}^{N} p_n) = 0 \\
\vdots \\
\frac{\partial_{\mathcal{L}}}{\partial_{p_N}} \sum_{n=1}^{N} p_n log(p_n) - \lambda(1 - \sum_{n=1}^{N} p_n) = 0 \\
\frac{\partial_{\mathcal{L}}}{\partial_{\lambda}} \sum_{n=1}^{N} p_n log(p_n) - \lambda(1 - \sum_{n=1}^{N} p_n) = 0
\end{aligned}$$

Which in turn yields:

$$\begin{aligned}
log(p_1) + 1 - \lambda p_1 = 0 \\
log(p_2) + 1 - \lambda p_2 = 0 \\
\vdots \\
log(p_N) + 1 - \lambda p_N = 0 \\
1 - \sum_{n=1}^{N} p_n = 0 \qquad (1)
\end{aligned}$$

From which:

$$\lambda = \frac{\log(p_1) + 1}{p_1} = \frac{\log(p_2) + 1}{p_2} = \ldots \frac{\log(p_N) + 1}{p_N} \qquad (2)$$

it is clear from equations 1 and 2 that $p_1 = p_2 = \ldots p_N = \frac{1}{N}$, which is precisely a discrete uniform distribution.

2.c Show that T_1 wins

The notes show two possible tests for a decision tree. T1, where the left child has $[20+, 10-]$ posible outcomes in its sub-trees and the right node has $[10+, 0-]$. T2, on the other hand, yields: $left = [15+, 7-]; right = [15+, 3-]$.

The best choice should yield the maximum mutual information or information gain $I[p, T_n]$,$n \in \{1, 2\}$. So for T_1:

$$\begin{aligned}
H[p] &= -\frac{1}{4} log_2\left(\frac{1}{4}\right) - \frac{3}{4} log_2\left(\frac{3}{4}\right) = 0.8112 \\
H[p|T_1 = t] &= -\frac{2}{3} log_2\left(\frac{2}{3}\right) - \frac{1}{3} log_2\left(\frac{1}{3}\right) = 0.9182 \\
H[p|T_1 = f] &= 0 \\
H[p|T_1] &= p(T_1 = t)H[p|T_1 = t] + p(T_1 = f)H[p|T_1 = f] \\
&= 0.6887 \\
I[p, T_1] &= H[p] - H[p|T_1] = 0.1225
\end{aligned}$$

Whereas for T_2 we have:

$$\begin{aligned}
H[p|T_2 = t] =& -\frac{15}{22}log_2\left(\frac{15}{22}\right) - \frac{7}{22}log_2\left(\frac{7}{22}\right) = 0.9024\\
H[p|T_2 = f] =& -\frac{15}{18}log_2\left(\frac{15}{18}\right) - \frac{3}{18}log_2\left(\frac{3}{18}\right) = 0.65002\\
H[p|T_2] =& p(T_2 = t)H[p|T_2 = t] + p(T_2 = f)H[p|T_2 = f]\\
=& \frac{22}{40}0.9024 + \frac{18}{40}0.65002 = 0.7888\\
I[p, T_2] =& H[p] - H[p|T_2] = 0.02245
\end{aligned}$$

From which we can see that we gain much more information from knowing the result of T_1 than by knowing the result of T_2.

Question 3: Kernels

Suppose $k_1(\boldsymbol{x}, \boldsymbol{z})$ and $k_2(\boldsymbol{x}, \boldsymbol{z})$ are valid kernels over $\mathbb{R}^n \times \mathbb{R}^n$. Prove or disprove that the following are valid kernels.

Use Mercer's theorem regarding the Gram matrix[5] or the fact that a kernel can be expressed as $k(x, z) = \phi(\boldsymbol{x})^T\phi(\boldsymbol{z})$.

preliminaries

From Mercer, we know for each $k_1(\boldsymbol{x}, \boldsymbol{z})$ and $k_2(\boldsymbol{x}, \boldsymbol{z})$ we have corresponding kernel matrices $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ which are symmetric and positive semi-definite.

For both $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$:

Symmetry:

$$\boldsymbol{M}_i = \boldsymbol{M}_i^T \tag{3}$$

Positive semidefiniteness:

$$\boldsymbol{x}^T\boldsymbol{M}_i\boldsymbol{x} \geq 0 \tag{4}$$

$$|\boldsymbol{M}_i| \geq 0 \tag{5}$$

3.a $k(\boldsymbol{x}, \boldsymbol{z}) = ak_1(\boldsymbol{x}, \boldsymbol{z}) + bk_2(\boldsymbol{x}, \boldsymbol{z}), a, b > 0;\ a, b \in \mathbb{R}$

Firstly, we establish that if $k(\boldsymbol{x}, \boldsymbol{z})$ is a valid kernel, then $ak(\boldsymbol{x}, \boldsymbol{z})$ is also a valid kernel $\forall a > 0;\ a \in \mathbb{R}$:

We know that for a square matrix $\boldsymbol{A}$ of size $n \times n$, $|a\boldsymbol{A}| = a^n\,|A|$. And, since $a \geq 0$, we know that $a^n \geq 0$. Thus equation 5 holds for both of our summands. Additionally, since the scalar multiplication of a symmetric matrix yields another symmetric matrix, both summands are are symmetric and therefore valid kernels.

Now, let us say:

$$ak_1(\boldsymbol{x}, \boldsymbol{z}) = k_1'(\boldsymbol{x}, \boldsymbol{z})$$

and

$$bk_2(\boldsymbol{x}, \boldsymbol{z}) = k_2'(\boldsymbol{x}, \boldsymbol{z})$$

are both valid kernels with kernel matrices $\boldsymbol{M}_1'$ and $\boldsymbol{M}_2'$. The addition of two symmetric matrices yields a symmetric matrix, so we need to check for positive semi-definiteness.

Since both $\boldsymbol{M}_1'$ and $\boldsymbol{M}_2'$ are symmetric we can write:

[5] Equivalently known as the kernel matrix.

$$M_1' = U^T \Lambda_U U$$
$$M_2' = V^T \Lambda_V V$$

and using equation 4:

$$(x^T U^T \Lambda_U U x + x^T V^T \Lambda_V V x) \geq 0$$
$$x^T (U^T \Lambda_U U + V^T \Lambda_V V) x \geq 0$$
$$x^T (M_1' + M_2') x \geq 0$$

Which proves that $k(\boldsymbol{x}, \boldsymbol{z}) = ak_1(\boldsymbol{x}, \boldsymbol{z}) + bk_2(\boldsymbol{x}, \boldsymbol{z}), a, b > 0; a, b \in \mathbb{R}$ is a valid kernel.

3.b $k(\boldsymbol{x}, \boldsymbol{z}) = ak_1(\boldsymbol{x}, \boldsymbol{z}) - bk_2(\boldsymbol{x}, \boldsymbol{z}), a, b > 0; a, b \in \mathbb{R}$

Suppose:

$$a = 1, b = 1, M_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, M_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

Both$\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ symetric, positive semi-definite matrices. Yet $\boldsymbol{M'} = a\boldsymbol{M}_1 - b\boldsymbol{M}_2$ would yield:

$$M_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The eigenvalues of which are $\lambda_1 = -1, \lambda_2 = 1$, making $\boldsymbol{M'}$ a non-positive semi-definite matrix and thus $k(\boldsymbol{x}, \boldsymbol{z})$ is not a valid kernel.

3.c $k(\boldsymbol{x}, \boldsymbol{z}) = k_1(\boldsymbol{x}, \boldsymbol{z})k_2(\boldsymbol{x}, \boldsymbol{z})$

The kernel matrix $\boldsymbol{M'}$ of the product of two matrices $k_1(\boldsymbol{x}, \boldsymbol{z}), k_2(\boldsymbol{x}, \boldsymbol{z})$ is equivalent to the element-wise multiplication of the respective two kernel matrices $\boldsymbol{M'} = \boldsymbol{M}_1 \odot \boldsymbol{M}_2$. This is also known as the Hadamard product or the Schur product. The Schur product theorem states that said product of two positive semi-definite matrices is also positive semi-definite. It is trivial to show that symmetry is preserved under such conditions. Thus $k(\boldsymbol{x}, \boldsymbol{z}) = k_1(\boldsymbol{x}, \boldsymbol{z})k_2(\boldsymbol{x}, \boldsymbol{z})$ is a valid kernel.

3.d $k(\boldsymbol{x}, \boldsymbol{z}) = f(\boldsymbol{x})f(\boldsymbol{z}), where\ f : \mathbb{R}^n \to \mathbb{R}$

Here we rely on the fact that a kernel can be expressed as $k(x, z) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{z})$ where $\phi(\boldsymbol{x})$ maps $\boldsymbol{x}$ onto an n-dimensional space.

It is trivial to see that if $n = 1$ and $\phi = f$, $f(\boldsymbol{x})f(\boldsymbol{z})$ constitutes a valid kernel sinc it can be expressed as $k(x, z) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{z})$.

3.e $k(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x})p(\boldsymbol{z}), where\ p\ pdf$.

The same rationale as question 3.d applies here, $k(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x})p(\boldsymbol{z})$ is a valid kernel.

Question 4: Nearest neighbour vs decision trees, do boundaries coincide?

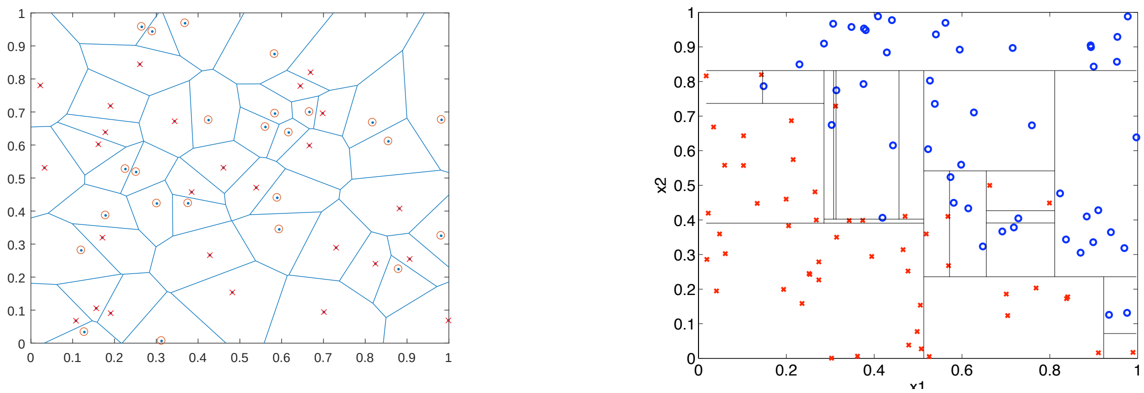
Boundaries do not necessarily coincide for these two classification strategies; moreover, in typical usage, they would tend to be non-coincidental but in some rare or contrived cases[6] the boundaries might equate.

[6] Eg A dataset consisting of two points; the usage of decision functions of an arbitrary number of features, etc.

Decision tree boundaries are typically composed of hyper-planes that are orthogonal to the features f_d chosen for each decision; boundaries pass through the midpoint between points neighboring on a projection along the axis of f_d. Thus each segment of a decision-tree boundary will generally have one out of n directions for an n-dimensional space.

Conversely, boundaries for nearest-neibours form a Voronoi tessellation, where each boundary segment corresponds to a hyper-plane running orthogonal to the line between a given point and its nearest neighbors while passing through the midpoint of such a line (thus the ensemble of said hyperplanes has a wide gammut of directions within the space).

For an example, see figures 1a and 1b.



(a) Nearest-neighbour

(b) decision tree

Figure 1: A Voronoi tessellation has boundary segments in many different directions, perpendicular to the lines between any two nearest-neighbors whereas decision-tree boundary segments are typically perpendicular to any one of a given set of features or feature combinations

Question 5: Bayes rate

For the following univariate case where $P(\omega_i) = \frac{1}{c}$ and

$$P(x|\omega_i) = \begin{cases} 1 & 0 \leq x \leq \frac{cr}{c-1} \\ 1 & i \leq x \leq i + 1 - \frac{cr}{c-1} \\ 0 & otherwise \end{cases}$$

5.a Show that $P^* = r$

The minimal multi-class classification error rate P^* is given by:

$$P^* = 1 - \int \arg\max_i P(\omega_i|x)P(x)dx$$

We use Bayes to obtain $P(x)$:

$$P(x) = P(x|\omega_i)P(\omega_i) = \begin{cases} \frac{1}{c} & 0 \leq x \leq \frac{cr}{c-1} \\ \frac{1}{c} & i \leq x \leq i + 1 - \frac{cr}{c-1} \\ 0 & otherwise \end{cases}$$

We also use Bayes to obtain $P(\omega_i|x)$:

$$\begin{aligned}
P(\omega_i|x) =& \frac{P(x|\omega_i)P(x)}{P(\omega_i)} \\
=& \frac{P(x|\omega_i)P(x|\omega_i)P(\omega_i)}{P(\omega_i)} \\
=& P^2(x|\omega_i)
\end{aligned}$$

$$P(\omega_i|x) = \begin{cases} 1 & 0 \leq x \leq \frac{cr}{c-1} \\ 1 & i \leq x \leq i+1-\frac{cr}{c-1} \\ 0 & otherwise \end{cases}$$

Given the class density and probability, we can see that for any region with overlapping densities, the choice of any i will maximize. Additionally, we see that the constraints imposed by existing densities demand that $0 \leq r \leq \frac{c-1}{c}$. This in turn implies that densities overlap only in $[0, \frac{cr}{c-1}]$ thus:

$$\begin{aligned}
P^* =& 1 - \int P(\omega_1|x)P(x)dx \\
=& 1 - \frac{1}{c}\int_0^{\frac{cr}{c-1}} dx - \sum_{i=1}^{c} \frac{1}{c}\int_i^{i+1-\frac{cr}{c-1}} dx \\
=& 1 - \frac{1}{c}\frac{cr}{c-1} - 1 - \frac{cr}{c-1} \\
=& \frac{cr-r}{c-1} \\
=& r
\end{aligned}$$

5.b Show the nearest-neighbor rate $P = P^*$

$$\begin{aligned}
LNN =& \int \left[1 - \sum_{i=1}^{c} P^2(\omega_i|x)\right] p(x)dx \\
=& \int \left[1 - \sum_{i=1}^{c} \left(\frac{P(x|\omega_i)P(\omega_i)}{p(x)}\right)^2\right] p(x)dx \\
=& \int p(x) - \sum_{i=1}^{c} \frac{P(x|\omega_i)^2 P(\omega_i)^2}{p(x)} dx \\
=& \int p(x) - \sum_{i=1}^{c} \frac{P(x|\omega_i)P(\omega_i)(P(x|\omega_i)P(\omega_i))}{p(x)} dx \\
=& \int p(x) - \sum_{i=1}^{c} \frac{P(x|\omega_i)P(\omega_i)p(x)}{p(x)} dx \\
=& \int p(x)dx - \frac{1}{c}\int_0^{\frac{cr}{c-1}} dx - \sum_{i=1}^{c} \frac{1}{c}\int_i^{i+1-\frac{cr}{c-1}} dx \\
=& 1 - \frac{1}{c}\frac{cr}{c-1} - 1 - \frac{cr}{c-1} \\
=& \frac{cr-r}{c-1} \\
=& r
\end{aligned}$$

Question 6: Implementation

In the interest of comparing methods, I have chosen to do both adaboost and knn. Both implementations rely on a single driver script, (please see attached `A3_q6_driver.m`) which we include also at the end of this sub-section for the reader's convenience. In order to run the code created for this assignment, one must call `A3_q6_driver.m` from within Matlab while including the function files that were created for each method in the working directory or executable path. Adaboost relies on `ada_boost.m`, `stump.m` and `calculate_error.m`, while KNN simply relies on `knn.m`. The function files are included as attachments and shall be detailed and printed in sub-sections 6.a and 6.b for the reader's convenience.

The driver script deals with loading the data creating the partitions for the k-fold CV, instantiating maximum number of iterations or maximum number of neighbors for each one of the methods and plotting results. Each classifier, relies on function scripts written for the classifier.

`A3_q6_driver.m` (omitting header and final print instructions):

```matlab
X = load('wpbcx.dat');
y = load('wpbcy.dat');

num_folds = 10;

folds_info = cvpartition(length(y), 'kfold', num_folds);
folds_idx = randperm(length(y));

iters = 50;
max_k = 50;

errs_ada = zeros(iters, num_folds, 2);
errs_knn = zeros(max_k, num_folds, 2);

for fold = 1:num_folds
    disp(sprintf('Performing %d-fold CV, fold: %d', num_folds, fold));
    idxs_prev = 1:sum(folds_info.TestSize(1:(fold-1)));
    if ~isempty(idxs_prev)
        offset = idxs_prev(end);
    else
        offset = 0;
    end
    idxs_xcl = (1:folds_info.TestSize(fold))+offset;
    idx_after_skip = length(y)-(sum(folds_info.TestSize((fold+1):end))-1);
    idxs_next = idx_after_skip:length(y);
    X_train = X(folds_idx([idxs_prev, idxs_next]), :);
    X_test = X(folds_idx(idxs_xcl), :);
    y_train = y(folds_idx([idxs_prev, idxs_next]));
    y_test = y(folds_idx(idxs_xcl));

    h = zeros(iters, 3);
    alphas = zeros(iters,1);

    m = length(y_train);
    W = ones(m, 1)/m;
    %%% Adaboost loop %%%
    for i = 1:iters
        % keep the user informed on longer runs
        if mod(i, 100) == 0
            disp(sprintf('\tworking on iter %d', i));
```

```
45        end
46        [h(i,:), alphas(i), W] = ada_boost(X_train, y_train, W);
47        errs_ada(i, fold, 1) = calculate_error(X_train, y_train, ...
48                                       h(1:i, :), alphas(1:i));
49        errs_ada(i, fold, 2) = calculate_error(X_test, y_test, ...
50                                       h(1:i, :), alphas(1:i));
51    end
52    %%% KNN loop %%%
53    for l = 1:m
54        % remove point i from the training set
55        X_train_temp = X_train;
56        y_train_temp = y_train;
57        X_train_temp(l,:) = [];
58        y_train_temp(l) = [];
59        yi_hats = knn(X_train_temp, y_train_temp, X_train(l, :),max_k);
60        for k = 1:max_k
61            if yi_hats(k) ~= y_train(l)
62                errs_knn(k, fold, 1) = errs_knn(k, fold, 1) + 1/m;
63            end
64        end
65    end
66    n = length(y_test);
67    for l = 1:n
68        yi_hats = knn(X_train, y_train, X_test(l, :),max_k);
69        for k = 1:max_k
70            if yi_hats(k) ~= y_train(l)
71                errs_knn(k, fold, 2) = errs_knn(k, fold, 2) + 1/n;
72            end
73        end
74    end
75 end
```

6.a Adaboost

The implementation relies on a driver script (common to both adaboost and knn) and two function files `ada_boost.m`, `stump.m` and `calculate_error.m`.

The driver calls `ada_boost.m` for as many iterations as indicated, while retaining a copy of the weak classifiers h, the classifier weights α and the observation weights w after each iteration. Each time it runs, `ada_boost.m` calls `stump.m` with the appropriate observation-weights and the later returns a weak classifier h, for which `ada_boost.m` calculates a weight α. After each iteration i of training with the training set, the driver script uses the accrued h_i, α_i pairs to call `calculate_error.m` both with the training and testing data in order to store training and testing errors (see lines 46 to 50 in the driver script).

Note that most of the effort required to get adaboost working was put into, not so much `ada_boost.m` but into `stump.m`, which

`ada_boost.m` (omitted header)

```
function [h, alpha, W] = ada_boost(X, y, W)
%ADA_BOOST takes an observation matrix X (each observation as a row), labels
%y and a vector W of weights for each observation. It returns a series
%h [thr,dim,polarity] tuples per stump, their associated weights and a a
%two-column error matrix.
%   for the error matrix, row is associated with each iteration and training
%   and test errors populate the first and second columns respectively.
```

```matlab

    [ Threshold, Dim, polarity, err ] = stump(X, y, W);
    h = [Threshold, Dim, polarity];

    y_hat = X(:,h(2)) > h(1);
    if h(3) == -1
        y_hat = ~y_hat;
    end

    alpha = 0.5 * log((1-err)/err);
    % a vector containing 1 for correctly classified entries given h(round)
    % and -1 for misclassified entries.
    classifiedRightOrNot = ((2*y)-1).*((2*y_hat)-1);
    W = W .* exp(-alpha*(classifiedRightOrNot));
    W = W./sum(W);
end
```

stump.m (omitted header)

```matlab
function [ Threshold, Dim, polarity, err ] = stump( X, y, W )
%STUMP Returns the best decision stump as a threshold-dimension pair for
%observation contained in the X matrix given class label y and weights W
%vectors
%   X should contain row entries for observations, where columns are
%   features. y is a vector containing binary classifications and W are the
%   weights associated with each observation in x (eg weights assigned by a
%   boosting algorithm. The threshold is the point at which a decision
%   boundary perpendicular to the selected dimension Dim should pass.
    [m,d] = size(X);
    % Normalize W, in case it hasn't been done
    W = W/sum(W);
    Obs = zeros(m,d);
    IDXs = zeros(m,d);
    % Order each feature, keep a matrix of row indices per feature and then
    % keep row classes and errors.
    [Obs, IDXs] = sort(X);
    Ys = (y(IDXs)*2)-1;
    E = NaN*(ones(m,d));
    % we can only have a split between neighbours of different classes, don't
    % bother checking between neighbours of the same class
    diff_neighbors = [2*ones(1,d);0.5*abs(diff(Ys))];

    for n = 1:d
        for i = 1:m
            if diff_neighbors(i,n) == 1
                C_hat = X(:,n) >= Obs(i,n);
                err = sum((C_hat ~= y).*W);
                E(i,n) = err;
            end
        end
    end

    % get the error farthest from 0.5, the most informative split
    [mx,idx] = max(abs(0.5-E));

    % we need to know along which direction it happened
```

```
    [~, Dim] = max(mx);

    % recover the true index
    idx = idx(Dim);

    err = E(idx,Dim);

    % polarity 1 means that positive class observations are contained at
    % higher values for that feature, polarity -1 means positive observations
    % live at lower values than the threshold
    if err > 0.5
       polarity = -1;
       err = 1 - err;
    else
       polarity = 1;
    end

    % Now calculate where the split happens, the midpoint between two adjacent
    % points on that feature dimension, repeat the first point so that decision
    % at the lower bound is just the smallest value of the feature space
    cut_point = Obs(idx,Dim);
    feature_vals = unique(Obs(:,Dim));
    the_real_idx = cut_point == feature_vals;
    feature_vals = [(feature_vals(1) - feature_vals(2)); feature_vals];
    midpoints = feature_vals + 0.5*([diff(feature_vals);0]);
    Threshold = midpoints(the_real_idx);
end
```

`calculate_error.m` (omitted header)

```
function [ err, y_hat ] = calculate_error( X, y, h, alphas )
%CALCULATE_ERROR calculates the error given a dataset X, its target labels
%y and the ensemble adaboost predictor h, alpha.
%   X are the observations given in row form, y the labels given as a
%   column vector, h tha matrix of [threshold, dim, polarity] weak
%   classifiers and alphas is the column vector of weights associated with
%   each weak classifier tuple.

y_hat = zeros(length(y), 1);

m = length(y);
for i = 1:m
    for k = 1:length(alphas)
        threshold = h(k, 1);
        dim = h(k, 2);
        pol = h(k, 3);
        contrib_k = (2*(pol*X(i, dim) >= pol*threshold)-1)*alphas(k);
        y_hat(i) = y_hat(i) + contrib_k;
    end
    y_hat(i) = y_hat(i) >= 0;
end

err = sum((y_hat > 0) ~= y)/m;
end
```

Surprisingly, adaboost's did not improve significantly (it actually degraded) for the test set through-out iterations. Errors on the training set, however, decreased exponentially as expected. Performance considerations, however, are fully addressed in subsection 6.c.

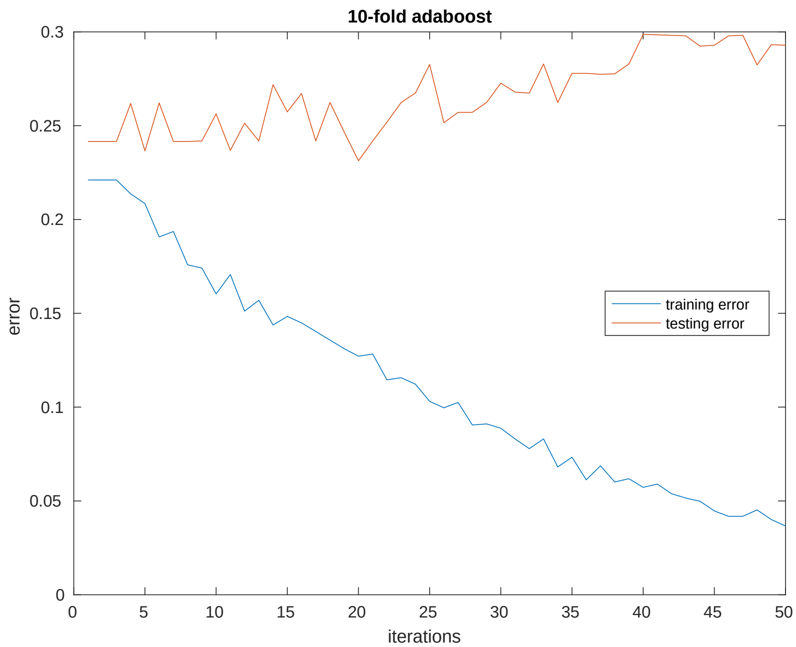


Figure 2: Training and testing errors for a 10-fold adaboost classification of the Wisconsin dataset used for Assignment 2

6.b KNN

KNN was significantly simpler to implement. The difficulty lying more in establishing what it means to have training and testing on a method that does not really *learn* the data. We chose to use the training set as neighbours to both the training and testing instances, making sure to remove the point we sought to classify from the training set during training instances. We refer the reader to the driver script, lines 54 through 73.

knn.m (omitted header)

```
function [yi_hats, IDXs] = knn(X, y, xi, max_k)
%KNN returns the class estimates for xi, given xi's 1:max_k nearest neighbours.
%It also returns the indexes of said neighbours while requiring a
%row-entry matrix of observations X and a corresponding column of labels y.
    m = length(y);
    dist = zeros(m);
    for i = 1:m
        dist(i) = norm(xi - X(i,:));
    end
    [~, idx] = sort(dist);
    for k = 1:max_k
        IDXs = idx(1:k);
        yi_hats(k) = round(sum(y(IDXs))/k);
    end
end
```

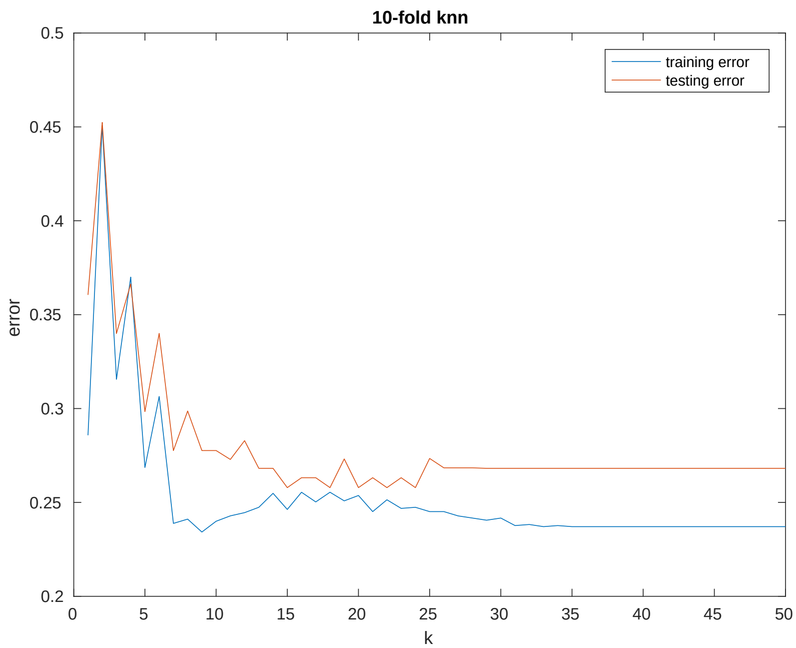


Figure 3: Training and testing errors for a 10-fold KNN of the Wisconsin dataset used for Assignment 2

6.c Results

Despite significant differences in performances for training data and errors against the probed hyper-parameters, both methods yielded comparable best rates for testing data.

Moreover, a comparison between classifiers and simply using class priors as a means of classifying from `A3_q6_driver.m` :

```
% how well did we do?
disp('Here''s how well we did:')
disp(sprintf('best prediction given on test data by adaboost:\n\t%d',...
            min(mean(errs_ada(:,:,2), 2)) ));
disp(sprintf('best prediction given on test data by knn:\n\t%d',...
            min(mean(errs_knn(:,:,2), 2)) ));
disp(sprintf('empirical ratio of class 1 to class 0:\n\t%d',...
            sum(y)/length(y) ));
```

yields the following output.

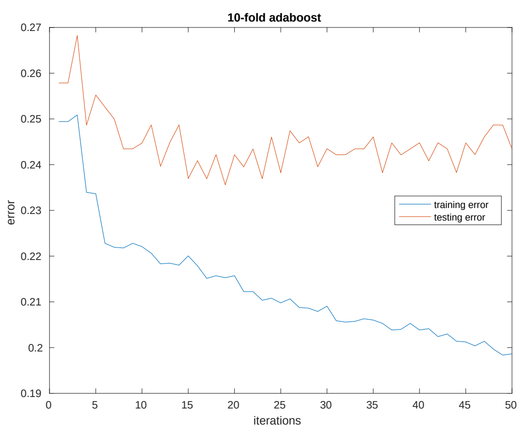
```
best prediction given on test data by adaboost:
min(mean(errs_ada(:,:,2), 2))
        2.313158e-01
best prediction given on test data by knn:
min(mean(errs_knn(:,:,2), 2))
        2.578947e-01
empirical ratio of class 1 to class 0:
sum(y)/length(y)
        2.371134e-01
```

I refer the reader to the analysis I performed on the same data for assignment 2, where the significant class-overlap across most dimensions yielded classification slightly more effective than using empirical means.

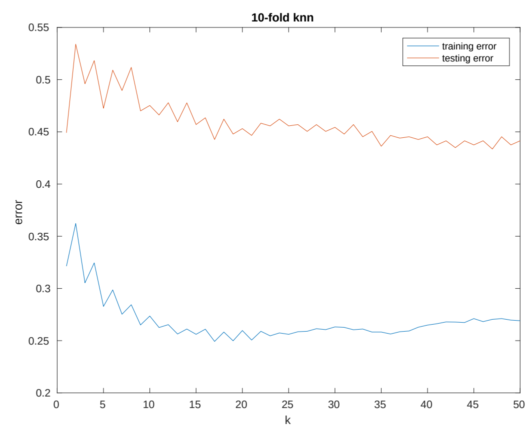
Curious to see the contribution of this particular dataset to such results, I downloaded another dataset with less class overlap from UCI: (https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/) which is included as an attachment. For this dataset, the performance of the classifiers is quite different from what we observe in the previous example. See figures 4a and 4b.

We note that the reader need only substitute lines 5 and 6 from the driver script by:

```
pima = load('pima-indians-diabetes.data');
X = pima(:,1:end-1);
y = pima(:,end);
```



(a) adaboost - pima dataset



(b) knn - pima dataset

Figure 4: Adaboost and KNN behave quite differently with this dataset than they do with the wisconsin dataset.

And the output of the script shows that Adaboost performs significantly better with this dataset however KNN performs significantly worse.

```
Here's how well we did:
best prediction given on test data by adaboost:
        2.356118e-01
best prediction given on test data by knn:
        4.335441e-01
empirical ratio of class 1 to class 0:
        3.489583e-01
```

This serves as a rather informal observation regarding the suitability of different methods for different kinds of problems and different datasets.