CS 246: Problem Set 4

Tony Hyun Kim

March 7, 2012

1 Strategies for high-frequency trading

1.1 Baseline: Mean reversion

Fig. 1 shows the trajectory of stock #679 between 10 AM and 3 PM on Jan 5, 2006. The bold blue trace shows the result of removing bad trades, *i.e.* all ticks where the trade price is not between the bid and ask prices.

The percent accuracy, utilizing the $+1$ and -1 scoring for correct and incorrect predictions respectively, is about 10%. I suppose this is better than random guessing. The Matlab script is attached.

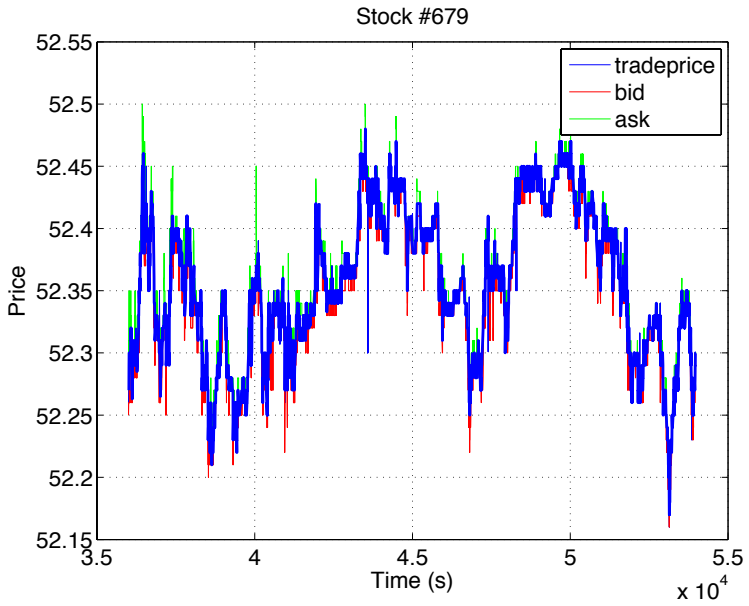

Figure 1: Trajectory of stock #679. The bold blue trace shows only "good" trades, *i.e.* ticks where the trade price is between the bid and ask prices.

Stock	Coeff	StdErr	tStat	pVal
679 (Small)	-9.50×10^{-5}	8.68×10^{-6}	-10.95	6.00×10^{-27}
679 (Big)	-1.25×10^{-4}	5.19×10^{-5}	-2.41	1.61×10^{-2}
980 (Small)	-3.35×10^{-4}	1.54×10^{-5}	-21.79	7.28×10^{-92}
980 (Big)	-4.16×10^{-4}	3.10×10^{-5}	-13.44	5.18×10^{-39}
17948 (Small)	-1.30×10^{-4}	1.02×10^{-4}	-1.28	0.20
17948 (Big)	-2.81×10^{-4}	1.90×10^{-4}	-1.48	0.14
27969 (Small)	-2.52×10^{-4}	1.80×10^{-5}	-13.98	6.70×10^{-42}
27969 (Big)	-3.53×10^{-4}	1.32×10^{-4}	-2.68	7.49×10^{-3}

Table 1: Regression statistics when taking into account whether the last move is "big" or "small" (with respect to the rolling high-low). For both big and small moves, I see a negative correlation between the previous and next move, which leads to a trade strategy that is identical to mean reversion.

1.2 Improvement 1

1.2.1 Rolling 5 minute high-low

A simple dynamic programming algorithm that uses 1-minute buckets would be as follows. For the current minute, utilize a priority queue to determine the high and the low within that minute. When the minute is up, store the final high and low to a shift register of length four. We then compute the "five-minute rolling" statistics using the shift register and the instantaneous priority queue.

The Matlab script is attached.

1.2.2 Regression for next-tick's percentage change

I have tried the model that takes into account small and large moves on stocks 679, 980, 17948 and 27969. Statistics are given in Table 1. For all stocks, I find that the coefficient for both small and large moves is negative.

1.2.3 Primitive algorithmic trading

As shown in Table 1, I find that the previous price move is negatively correlated with future moves, whether or not the prior move was big or small compared to the rolling window. Hence, the trading strategy will be identical to mean reversion of Section 1.1, and I hard-coded for mean reversion. I applied this trading strategy to the stock trajectories during 3-4 PM, and found the following profits for each of the stocks:

- Stock #679: \$1.1050
- Stock #980: \$1.8100
- Stock #17948: \$0.0900
- Stock #27969: \$1.2090

I suppose, technically, we made some money.

1.3 Hello world SVM

- We use $C \cdot \sum_i e_i$ in the objective in order to penalize the use of slack variables e_i. If we were to set $C = 0$, then the slack variables can take any value and our model will basically ignore the training set.
- One obtains the dual form of the SVM by formulating the optimization problem by introducing Lagrange multipliers α_i (one for each training example)

$$\mathcal{L} = \frac{1}{2} w^T w + C \sum_i e_i + \sum_i \alpha_i g_i$$

where the first two terms correspond to the original cost function, and

$$g_i = -\left[y^{(i)} \left(\langle w, x^{(i)} \rangle + b \right) - 1 + e_i \right] \leq 0$$

represent the margin and slack constraints. The dual function of the SVM optimization problem is obtained by defining $W(\alpha) = \min_{w,b} \mathcal{L}$. The dual optimization task is then to maximize over α

$$W(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

subject to the constraints $0 \leq \alpha_i \leq C$ and $\sum_i \alpha_i y^{(i)} = 0$. Under further conditions (known as KKT conditions), the dual solution may be used to obtain the solutions to the original ("primal") cost function. Note that I consulted my machine learning course notes (CS 229) for this section.

1.4 Improvement 2: SVM-based trading

Here is my toy implementation:

- At any time index i, the feature vector is the last K price moves, *i.e.*

$$X(i) = \left[\begin{array}{cccc} p_i - p_{i-1} & p_{i-1} - p_{i-2} & \cdots & p_{i-(K-1)} - p_{i-K} \end{array} \right)]$$

I selected $K = 3$ admittedly without too much thought. I also tried a few values of the regularization parameter C.

- On the training dataset, I computed the margin $|w^T X + b|$ for each example, and computed the 33%, 50%, and 75% quantiles. This is to give me a sense of the typical "confidences" observed in the training portion of the stream.
- During the trading period, I streamed the past K price differentials as was done in my training model, and computed the margin $w^T X + b$ with respect to the separating hyperplane. Depending on the magnitude of the margin with respect to the previously-computed quantiles at 33%, 50% and 75%, I traded 1, 2 or 5 units in the proper direction (*i.e.* sell when the model predicted the price to fall, and buy when the price was predicted to rise).

The Matlab script is attached.

```
%-------------------------------------------------------------
% Tony Hyun Kim
% CS 246, PS 4, Problem 1(a)
% Load trades
%-------------------------------------------------------------
clear all; close all;

%part a)
%plot all the trades
tradeFolder='Trades/DATA/';
quoteFolder='Quotes/DATA/';
daten=05;

files = dir(strcat(tradeFolder,'679.asc'));
for i = 1:size(files,1)
    f = files(i);
    name = strcat(tradeFolder, f.name );
    fid = fopen(name);
    C = textscan(fid, '%s %s %f %d  %*[^\n]', 'delimiter', ',', ...
            'EmptyValue', -Inf);
    fclose(fid);
    time=char(C{2});
    date=char(C{1});
    dateidx=(str2num(date(:,4:5))==daten); %#ok<*ST2NM>

    hour=str2num(time(dateidx,1:2));
    minute=str2num(time(dateidx,4:5));
    sec=str2num(time(dateidx,7:end));
    tsecs=hour*60*60+minute*60+sec;

    price=C{3}(dateidx);

    name = strcat(quoteFolder, f.name );
    fid = fopen(name);
    C = textscan(fid, '%s %s %s %f %f %d %d  %*[^\n]',...
       'delimiter', ',','EmptyValue', -Inf);
    fclose(fid);

    date=char(C{1});
    dateidx=(str2num(date(:,4:5))==daten);

    time=char(C{2});
    exch=char(C{3});
    idx=(exch=='N') & dateidx;

    hour=str2num(time(idx,1:2));
    minute=str2num(time(idx,4:5));
    sec=str2num(time(idx,7:end));
    qsecs=hour*60*60+minute*60+sec;
    bid=C{4}(idx);
    ask=C{5}(idx);

    figure;
    starttime=10*60*60; %10am
    pidx=(tsecs>starttime) & (tsecs<(starttime+6*60*60)); %plus 6h
    qidx=(qsecs>starttime) & (qsecs<(starttime+6*60*60)); %plus 6h

    plot(tsecs(pidx),price(pidx),'b',qsecs(qidx),bid(qidx),'r',...
    qsecs(qidx),ask(qidx),'g');
    hold on;
    legend('tradeprice','bid','ask')
end

close all;
q1a_removebad;
```



```
%-------------------------------------------------------------
% Tony Hyun Kim
% CS 246, PS 4, Problem 1(a)
% Remove bad trades
%-------------------------------------------------------------
t_in = tsecs(pidx);
p_in = price(pidx);

q_in = qsecs(qidx);
b_in = bid(qidx);
a_in = ask(qidx);

[q_in ind] = unique(q_in);
b_in = b_in(ind);
a_in = a_in(ind);

plot(t_in,p_in); hold on;
plot(q_in,b_in,'r');

t = [];
p = [];

for i = 1:length(t_in)
    % Current value
    t0 = t_in(i);
    p0 = p_in(i);

    % Interpolate bid and ask
    bi = interp1(q_in,b_in,t_in(i));
    ai = interp1(q_in,a_in,t_in(i));

    if((bi<=p0)&&(p0<=ai)) % Definition of good trade
        t = [t; t0];
        p = [p; p0];
    end
end

grid on;
set(gca,'FontSize',16);
plot(t,p,'Linewidth',2);
xlabel('Time (s)');
ylabel('Price');
% title('Stock #679');
```



```
%-------------------------------------------------------------
% Tony Hyun Kim
% CS 246, PS 4, Problem 1(b)
% Stream through price and compute rolling statistics
%   and make trades
%-------------------------------------------------------------

% Previously, we computed the trade time and price
%   in the vectors t and p respectively

% Training time is 10 AM to 3 PM, i.e. 5 total hours. Note
%   that t(1) has been set in 'load_file.m' to be 10 AM.
t_tr = t(1) + 5*60*60;
N = find(t_tr<t,1); % Training examples

X = zeros(N,2); % Format: [Small Big] with sign for up/down
Y = zeros(N,1);

% We shall stream through the N data points, and implement
%   the shift register and rolling queue
shift  = zeros(2,4); % shift(1/2,:) = min/max
pqueue = zeros(2,1); % pqueue(1/2)  = min/max

minute_prev = -1;
for i = 2:(N-1)
    % Data structure for cached rolling min-high
    %-------------------------------------------------
    minute = floor(t(i)/60);
    if(minute==minute_prev) % Still in the same minute
        % Update priority queue
        if(p(i)>pqueue(2))  % Current price larger than max
            pqueue(2) = p(i);
        elseif(p(i)<pqueue(1)) % Smaller than min
            pqueue(1) = p(i);
        end
    else % Minute is up
        % Load into priority queue (at front)
        shift = [pqueue shift(:,1:(end-1))];

        % Reset priority queue with new value
        pqueue = p(i)*[1 1]';
    end
    minute_prev = minute;

    % Compute rolling high-low
    %-------------------------------------------------
    high = max([shift(2,:) pqueue(2)]);
    low  = min([shift(1,:) pqueue(1)]);
    highlow = high-low;

    % Classify ticks among
    %   {up-small, down-small, up-big, down-big}
    %-------------------------------------------------
    thresh   = 0.5*highlow;
    prevmove = p(i)-p(i-1);
    if(abs(prevmove)>thresh) % Big move
        X(i,2) = sign(prevmove);
    else % Small move
        X(i,1) = sign(prevmove);
    end
    Y(i) = (p(i+1)-p(i))/p(i); % Percentage change of next move
end

% Statistics of regression
%-------------------------------------------------

% Remove sparse entries in X/Y (i.e. no price move recorded)
moveIndices = (Y~=0);
```



```
X = X(moveIndices,:);
Y = Y(moveIndices,:);

whichstats = {'tstat'};
stats = regstats(Y, X, [1 0; 0 1], whichstats);
ts = stats.tstat;
CoeffTable = dataset({ts.beta,'Coef'},{ts.se,'StdErr'}, ...
        {ts.t,'tStat'},{ts.pval,'pVal'})

% Now we trade (by mean reversion)
%-------------------------------------------------
profit = 0;
for i = N:(length(t)-1)
    prevMove = p(i)-p(i-1);
    if(prevMove>0) % Previous move went up,
                   %    predict it will go down,
                   %    so SELL
        profit = profit + (p(i)-p(i+1));
    elseif(prevMove<0) % Previous move went down,
                       %    predict it will go up,
                       %    so BUY
        profit = profit + (p(i+1)-p(i));
    end
end
disp(profit)
```

```
%-------------------------------------------------------------
% Tony Hyun Kim
% CS 246, PS 4, Problem 1(c)
% Hello SVM
%-------------------------------------------------------------
clear all; close all;

n = 1000;
d = 3;
X = randn(n,d);
randnoise = randn(n,1);
Y = ((X(:,1)+X(:,2)+randnoise)>0)*2-1;

% Training set is first 30
n = 30;
Yt = Y(1:n,1);
Xt = X(1:n,:);

C = norm(mean(abs(Xt)));
cvx_begin % Classical SVM
    variables w(d) e(n) b
    dual variable alpha
    minimize( 0.5*w'*w + C*sum(e)) % norm(w) takes an extra sqrt
    subject to
        Yt.*(Xt*w+b)-1+e > 0 : alpha ;
        e>0; % slack
cvx_end
```



```
%-------------------------------------------------------------
% Tony Hyun Kim
% CS 246, PS 4, Problem 1(d)
% SVM model
%-------------------------------------------------------------

% Previously, we computed the trade time and price
%   in the vectors t and p respectively

% Training time is 10 AM to 3 PM, i.e. 5 total hours. Note
%   that t(1) has been set in 'load_file.m' to be 10 AM.
t_tr = t(1) + 5*60*60;
N = find(t_tr<t,1); % Training examples

% As for our data features, we will look at the last K
%   K price differentials
K = 3;

X = zeros(N,K);
Y = zeros(N,1);

shift = zeros(1,K); % Shift register for price differential
for i = 2:N
    dp = p(i)-p(i-1); % Newest observed move
    shift = [dp shift(1:(end-1))]; % Insert into shift reg
    X(i,:) = shift; % Current state of shift register are the features
    Y(i-1) = sign(dp);
end

% Remove sparse entries in X/Y (i.e. no price move recorded)
%-------------------------------------------------------------
moveIndices = (Y~=0);
X = X(moveIndices,:);
Y = Y(moveIndices,:);
N = length(Y);

% Let's try SVM regression
%-------------------------------------------------------------
C = 1e4*norm(mean(abs(X)));
cvx_begin
    variables w(K) e(N) b
    dual variable alpha
    minimize(0.5*w'*w + C*sum(e))
    subject to
        Y.*(X*w+b)-1+e > 0 : alpha;
        e>0;
cvx_end

% Apply resulting model to training set
%-------------------------------------------------------------
Ysvm = X*w+b;
% Ysvm = sign(Ysvm);

% Consider the distribution of our "confidences" in the training set
qu = quantile(abs(Ysvm),[0.75 0.50 0.33]);

% Now we trade (by mean reversion)
%-------------------------------------------------
profit = 0;
remTrades = 10;
shift = zeros(1,K);
for i = N:(length(t)-1)
    if(remTrades==0) % No more trades to perform
        break;
    end

    % Load in new prices
    dp = p(i)-p(i-1);
```



```
    shift = [dp shift(1:(end-1))];

    pred_svm = shift*w+b;
    conf = abs(pred_svm); % Our confidence in the answer (i.e. the margin)

    if(conf>qu(1)) % High confidence.
        units  = min(5,remTrades);
        if(pred_svm>0) % Expect price to go up. Buy.
            profit = profit + units*(p(i+1)-p(i));
        else % Expect price to go down. Sell.
            profit = profit + units*(p(i)-p(i+1));
        end
        remTrades = remTrades - units;
    elseif(conf>qu(2)) % Middle confidence.
        units  = min(2,remTrades);
        if(pred_svm>0) % Expect price to go up. Buy.
            profit = profit + units*(p(i+1)-p(i));
        else % Expect price to go down. Sell.
            profit = profit + units*(p(i)-p(i+1));
        end
        remTrades = remTrades - units;
    elseif(conf>qu(3)) % Low confidence.
        units  = min(1,remTrades);
        if(pred_svm>0) % Expect price to go up. Buy.
            profit = profit + units*(p(i+1)-p(i));
        else % Expect price to go down. Sell.
            profit = profit + units*(p(i)-p(i+1));
        end
        remTrades = remTrades - units;
    end
end
```

2 Decision tree learning

2.1 Reduction of impurity

Our toy decision tree seeks to predict whether a person enjoys beer.

2.1.1 Necessary and sufficient condition for an attribute to be "useful," $G > 0$

We have a set D that is split by a binary attribute into disjoint sets D_k, where $k = 0, 1$ labels the value taken by the attribute. Furthermore, D_0 has u positive and v negative examples, and D_1 has x positive and y negative examples. It follows that the set D has $x + u$ positive and $y + v$ negative examples.

We then compute the impurities of sets D, D_0 and D_1,

$$\begin{aligned} I(D) &= (x+y+u+v)\cdot\left[1-\left(\frac{x+u}{x+y+u+v}\right)^2-\left(\frac{y+v}{x+y+u+v}\right)^2\right] \\ I(D_0) &= (u+v)\cdot\left[1-\left(\frac{u}{u+v}\right)^2-\left(\frac{v}{u+v}\right)^2\right] \\ I(D_1) &= (x+y)\cdot\left[1-\left(\frac{x}{x+y}\right)^2-\left(\frac{y}{x+y}\right)^2\right] \end{aligned}$$

and hence the reduction in impurity, defined by $G = \max\left[I(D) - I(D_0) - I(D_1)\right]$, is

$$G = \frac{u^2}{u+v} + \frac{v^2}{u+v} + \frac{x^2}{x+y} + \frac{y^2}{x+y} - \frac{(x+u)^2}{x+y+u+v} - \frac{(y+v)^2}{x+y+u+v}.$$

Finally, we apply the condition of "usefulness," *i.e.* $G > 0$, which yields after simple rearrangement

$$\frac{u^2}{u+v} + \frac{v^2}{u+v} + \frac{x^2}{x+y} + \frac{y^2}{x+y} > \frac{(x+u)^2}{x+y+u+v} + \frac{(y+v)^2}{x+y+u+v}.$$

Since all of our manipulations are reversible, the above condition is sufficient and necessary for the reduction in impurity to be positive. Note that we assume $x + y > 0$ and $u + v > 0$; if these assumptions are false, the attribute is useless for tree construction.

2.1.2 Wine, Running and Pizza

I have transcribed the given {wine, running, pizza}×beer information into the previous notation of x, y, u and v in Table 2. Given the reductions G of each attribute, I would choose **pizza** as the root attribute for the decision tree.

Attribute	x	y	u	v	G_{attr}
Wine	30	20	30	20	0.0000
Running	20	10	40	30	0.3810
Pizza	50	30	10	10	0.5000

Table 2: Potential indicators for whether someone enjoys beer. There are 100 people overall in the sample set, of which 60 people like beer and 40 do not.

2.2 Prevention of overfitting

2.2.1 A ridiculous example

The given scenario is an extreme. If we have a dataset that contains all possible assignments to 100 binary attributes, and we were to construct a complete binary decision tree, we will have a full binary tree of depth 100, with 2^{100} leaves that correspond exactly to each of the examples. This is the "truest" definition of overfitting, since we are explicitly setting the labels (as seen in the training set) for all possible values of the attributes.

Since we are told that the target label is equal to the first of these attributes a_1 for 99% of these examples, a more sensible decision tree is one that looks at only a_1 and predicts the label accordingly.

2.2.2 Pruning the tree

Sequence of trees

Fig. 2 shows the sequence of trees that I generated manually using the α metric.

Test error

With respect to the given four-element test set, the trees $T_0, \cdots, T_4$ yield a corresponding error sequence $2, 2, 1, 0, 2$. The tree T_3 has the best generalization error.

2.3 Categorical attributes

2.3.1 Partitioning the attributes

Partitioning d possible values into two classes (*i.e.* the left and right branches) is identical to the task of assigning 0 or 1 to each value. Hence, the number of possible partitions is 2^d. (Note that, if we do not distinguish between "left" and "right" classes, the number of possibilities is 2^{d-1}.)

2.3.2 Number of partitionings that need to be considered is only $O(d)$ if the attribute categories are ordered in increasing $P(Y = 1|X = x_i)$

Expressions for the impurity

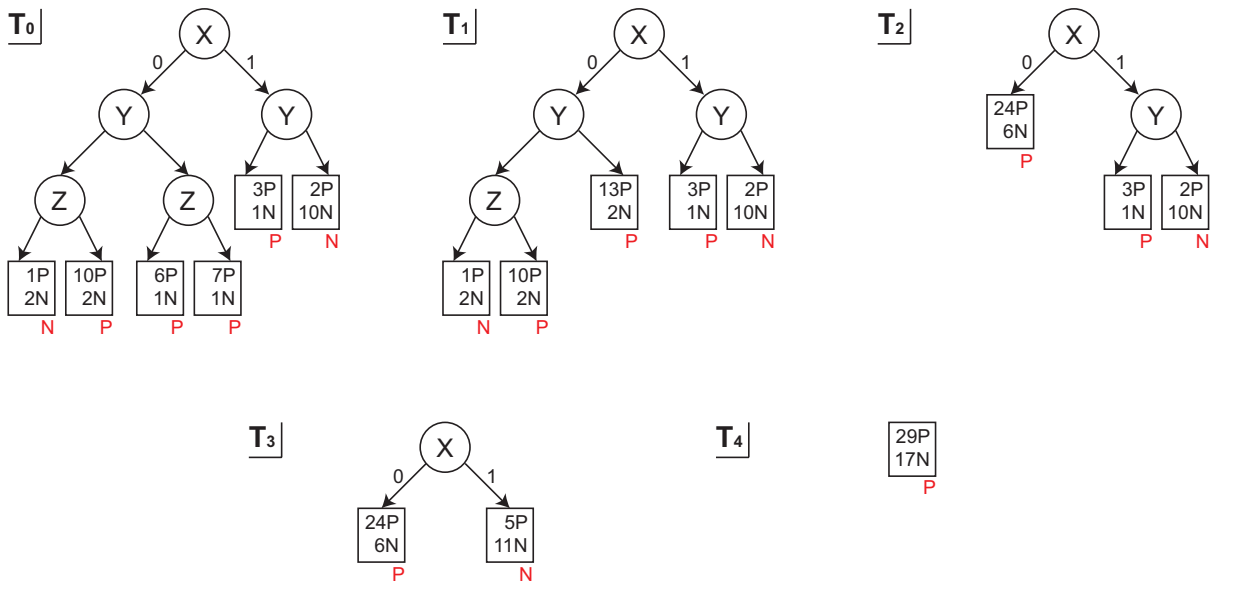

Figure 2: Sequence of pruned trees. Nodes are indicated as circles, and leaves as squares. The red label below each leaf indicates the predicted class of each leaf. Nodes (possibly sub-trees) are removed in order of apparent error rate per pruned node α (computed with respect to the training set).

We wish to show that

$$I(D_L) = |D| \cdot \sum_{i \in L} P(X = x_i) \cdot \left[2P(Y = 1|X \in L) - 2P^2(Y = 1|X \in L)\right]. \tag{1}$$

We begin with the definition of the impurity for a binary target variable Y:

$$I(D_L) = |D_L| \cdot (1 - p^2_{Y=1|L} - p^2_{Y=0|L}).$$

Now, it is clear that $|D_L| = |D| \cdot \sum_{i \in L} P(X = x_i)$ and that $p_{Y=0|L} = 1 - p_{Y=1|L}$. Making these substitutions in the above expression, we find

$$\begin{aligned} I(D_L) &= |D| \cdot \sum_{i \in L} P(X = x_i) \cdot \left[1 - p^2_{Y=1|L} - \left(1 - p_{Y=1|L},\right)^2\right] \\ &= |D| \cdot \sum_{i \in L} P(X = x_i) \cdot \left[2p_{Y=1|L} - 2p^2_{Y=1|L}\right]. \end{aligned}$$

Partial derivatives of the combined impurity

As per the assignment description, we will assume the following result.

Let the variable a_i indicate whether $x_i \in L$ (by $a_i = 1$) or $x_i \in R$ (by $a_i = 0$). The partial derivative of the combined impurity $I(D_L) + I(D_R)$ evaluated at $a_i = 0$ exists, and is of the form

$$P(X = x_i) \cdot [AP(Y = 1|X = x_i) + B] \tag{2}$$

for some constants A, B that does not depend on i. Furthermore, the partial derivative maintains the same sign over $[0, 1]$.

Only "contiguous" binary splits need to be considered

Let us begin by interpreting how to "use" the partial derivative of the impurity with respect to a_i. First, recall that a_i takes only binary values. We have two cases:

- Suppose $\frac{\partial}{\partial a_i}\left[I(D_L) + I(D_R)\right]_{a_i=0} > 0$. Since the derivative maintains the same sign over $[0, 1]$, it follows that $\left[I(D_L) + I(D_R)\right]_{a_i=1} > \left[I(D_L) + I(D_R)\right]_{a_i=0}$. Since we seek to minimize the combined impurity, in this case we would then assign $a_i = 0$.
- Suppose $\frac{\partial}{\partial a_i}\left[I(D_L) + I(D_R)\right]_{a_i=0} < 0$. By a similar reasoning, we would then assign $a_i = 1$.

Hence, it is seen that the sign of the partial derivative (with respect to a_i) determines whether we assign $a_i = 0$ or $a_i = 1$. Now, returning to the result of Eq. 2, we see that the sign of the partial derivative is controlled by the linear function $AP(Y = 1|X = x_i) + B$. Since the categorical values are ordered with respect to $P(Y = 1|X = x_i)$, they fall – in the same order – on the line defined by (A, B). Let the index l indicate the value for which the derivative changes sign, *i.e.*

$$AP(Y = 1|X = x_l) + B \neq AP(Y = 1|X = x_{l+1}) + B.$$

It then follows that we would assign all x_i for $i \leq l$ *or* all x_i for $i > l$ to L. (The choice between the two forms depend on the signs of A and B.)

Splitting colors

When we sort the given colors in increasing order of $P(Y = 1|\,\text{color})$, we obtain {green, red, blue, black}. I then performed the calculation of G, and find {green}, {red, blue, black} to be the best split.

3 Clustering data streams

3.1 A simple proof

I believe that the inequality holds for any real number a and b, not just non-negative numbers. We proceed by expanding the LHS:

$$\begin{aligned}(a+b)^2 &\leq 2a^2 + 2b^2\\ a^2 + 2ab + b^2 &\leq 2a^2 + 2b^2\\ 0 &\leq a^2 - 2ab + b^2\\ 0 &\leq (a-b)^2\end{aligned}$$

which is true for any real number a and b.

3.2 Prove $\text{cost}(S, T) \leq 2 \cdot \text{cost}_w(\hat{S}, T) + 2\sum_{i=1}^{l} \text{cost}(S_i, T_i)$

As always, we begin with definition

$$\text{cost}(S, T) = \sum_{x \in S} d(x, T)^2 = \sum_{i=1}^{l} \sum_{x \in S_i} d(x, T)^2 = \sum_{i=1}^{l} \sum_{x \in S_i} \left[\min_{z \in T} d(x, z)\right]^2 \tag{3}$$

where we have used the fact that $S = S_1 \cup S_2 \cup \cdots S_l$.

Now recall the triangle inequality, *i.e.* for any x, y, z we have

$$d(x, z) \leq d(x, y) + d(y, z)$$

from which it follows that (for fixed x, y)

$$\min_{z \in T} d(x, z) \leq \min_{z \in T} [d(x, y) + d(y, z)] = d(x, y) + \min_{z \in T} d(y, z).$$

We apply the above result in Eq. 3. Furthermore, for every $x \in S_i$, we let $y = t_{ij}$ where j is the index for which $x \in S_{ij}$. In other words, y is the centroid that $x \in S_i$ is assigned to in the i-th iteration of `ALG`. We then obtain:

$$\begin{aligned} \text{cost}(S, T) &= \sum_{i=1}^{l} \sum_{x \in S_i} \left[\min_{z \in T} d(x, z) \right]^2 \\ &\leq \sum_{i=1}^{l} \sum_{x \in S_i} \left[d(x, y) + \min_{z \in T} d(y, z) \right]^2 \end{aligned}$$

to which we apply the result of Section 3.1

$$\text{cost}(S, T) \leq 2 \sum_{i=1}^{l} \sum_{x \in S_i} d(x, y)^2 + 2 \sum_{i=1}^{l} \sum_{x \in S_i} \left[\min_{z \in T} d(y, z) \right]^2.$$

Consider the first term. We have defined y to be the cluster vector to which each $x \in S_i$ is assigned. It thus follows that $\sum_{x \in S_i} d(x, y)^2 = \sum_{x \in S_i} d(x, T_i)^2 = \text{cost}(S_i, T_i)$.

Now consider the second term. We note that y takes the values in $\hat{S} = \{t_{ij}\}$, and the number of times that y takes a particular outcome t_{ij} is proportional to the number of times $x \in S_i$ is assigned to cluster center t_{ij}. We conclude that $\sum_{i=1}^{l} \sum_{x \in S_i} d(y, T)^2 = \sum_{y \in \hat{S}} |S_{ij}| \cdot d(y, T)^2 = \text{cost}_w(\hat{S}, T)$.

Putting these two results together, we conclude

$$\text{cost}(S, T) \leq 2 \sum_{i=1}^{l} \text{cost}(S_i, T_i) + 2 \cdot \text{cost}_w(\hat{S}, T). \tag{4}$$

3.3 Prove $\sum_{i=1}^{l} \text{cost}(S_i, T_i) \leq \alpha \cdot \text{cost}(S, T^*)$

Consider each term $\text{cost}(S_i, T_i)$ in the summation. The subroutine `ALG` guarantees that

$$\text{cost}(S_i, T_i) \leq \alpha \cdot \text{cost}(S_i, T_i^*) \leq \alpha \cdot \text{cost}(S_i, T^*).$$

In the first inequality, T_i^* is defined to be the globally optimum assignment of the cluster centers given S_i. The inequality follows the assumption that the routine `ALG` returns a set T_i that is α-approximate of T_i^*. The second inequality follows since T_i^* is the optimal clustering set for S_i, *i.e.* it must necessarily have a cost that is lower than any other candidate T' including T^*.

Thus we may write

$$\sum_{i=1}^{l} \text{cost}(S_i, T_i) \leq \alpha \sum_{i=1}^{l} \text{cost}(S_i, T^*) = \alpha \cdot \text{cost}(S, T^*).$$

The final equality uses the fact that $S = \cup_{i=1}^{l} S_i$ to collect the multiple summations over S_i over a single sum over S (embedded in the definition of the cost function).

3.4 Prove $\text{cost}_w(\hat{S}, T) \leq \alpha \cdot \text{cost}_w(\hat{S}, T^*)$

The proof is identical to the previous part. Let $\hat{T}^*$ denote the optimal solution of the clustering problem for dataset $\hat{S}$. Then, the α-approximate property of `ALG` yields

$$\text{cost}_w(\hat{S}, T) \leq \alpha \cdot \text{cost}_w(\hat{S}, \hat{T}^*) \leq \alpha \cdot \text{cost}_w(\hat{S}, T^*)$$

where the last inequality follows from the fact that T^* must necessarily have a larger cost with respect to $\hat{S}$ (with weight w) than $\hat{T}^*$ since the latter is defined to be the global optimum.

3.5 Prove $\text{cost}_w(\hat{S}, T^*) \leq 2 \sum_{i=1}^{l} \text{cost}(S_i, T_i) + 2 \cdot \text{cost}(S, T^*)$

This proof will be similar to that in Section 3.2. As before, our basic technique is to use the triangle inequality of d, and the results of Section 3.1.

First, we begin by carefully writing out the definition of $\text{cost}_w(\hat{S}, T^*)$

$$\text{cost}_w(\hat{S}, T^*) = \sum_{t \in \hat{S}} w(t) \cdot d(t, T^*)^2 = \sum_{i=1}^{l} \sum_{t_{ij} \in T_i} w(t_{ij}) \cdot d(t_{ij}, T^*)^2 = \sum_{i=1}^{l} \sum_{t_{ij} \in T_i} |S_{ij}| \cdot d(t_{ij}, T^*)^2. \quad (5)$$

Here, we have utilized the fact that $\hat{S} = \cup_{i=1}^{l} T_i$ and that $w(t_{ij}) = |S_{ij}|$.

We now consider the "innermost" term $|S_{ij}| \cdot d(t_{ij}, T^*)^2$. We have

$$\begin{aligned} |S_{ij}| \cdot d(t_{ij}, T^*)^2 &= \sum_{k=1}^{|S_{ij}|} d(t_{ij}, T^*)^2 \\ &\leq \sum_{x \in S_{ij}} \left[d(t_{ij}, x) + d(x, T^*) \right]^2 \\ &\leq 2 \sum_{x \in S_{ij}} d(t_{ij}, x)^2 + 2 \sum_{x \in S_{ij}} d(x, T^*)^2 \end{aligned}$$

In the first equality, the RHS consists of $|S_{ij}|$ identical terms. In the second line, we apply the triangle inequality with different intermediate point x for each term. The third line uses the results of Section 3.1.

Inserting the above inequality into Eq. 5 we find:

$$\text{cost}_w(\hat{S}, T^*) \leq 2 \sum_{i=1}^{l} \sum_{t_{ij} \in T_i} \sum_{x \in S_{ij}} d(t_{ij}, x)^2 + 2 \sum_{i=1}^{l} \sum_{t_{ij} \in T_i} \sum_{x \in S_{ij}} d(x, T^*)^2$$

Finally, it is clear that $\sum_{t_i j \in T_i} \sum_{x \in S_{ij}} d(t_{ij}, x)^2$ is simply the (unweighted) cost of $S_i = \cup_j S_{ij}$ with respect to T_i. Likewise, the summation $\sum_{i=1}^{l} \sum_{t_{ij} \in T_i} \sum_{x \in S_{ij}} d(x, T^*)^2$ simply iterates over all elements $x \in S$. We therefore conclude

$$\text{cost}_w(\hat{S}, T^*) \leq 2 \sum_{i=1}^{l} \text{cost}(S_i, T_i) + 2 \cdot \text{cost}(S, T^*).$$

Note that the above result can also be written using Section 3.3

$$\text{cost}_w(\hat{S}, T^*) \leq 2\alpha \cdot \text{cost}(S, T^*) + 2 \cdot \text{cost}(S, T^*). \tag{6}$$

3.6 Conclude that $\text{cost}(S, T) \leq (4\alpha^2 + 6\alpha) \cdot \text{cost}(S, T^*)$

Here, the task is to simply cascade all of our previous results. We begin with Eq. 4, and apply the results of Sections 3.2–3.5 sequentially

$$\begin{aligned}
\text{cost}(S, T) &\leq 2 \cdot \text{cost}_w(\hat{S}, T) + 2 \sum_{i=1}^{l} \text{cost}(S_i, T_i) \\
&\leq 2 \cdot \text{cost}_w(\hat{S}, T) + 2\alpha \cdot \text{cost}(S, T^*) \\
&\leq 2\alpha \cdot \text{cost}_w(\hat{S}, T^*) + 2\alpha \cdot \text{cost}(S, T^*) \\
&\leq 2\alpha \cdot [2\alpha \cdot \text{cost}(S, T^*) + 2 \cdot \text{cost}(S, T^*)] + 2\alpha \cdot \text{cost}(S, T^*) \\
&\leq \left(4\alpha^2 + 6\alpha\right) \cdot \text{cost}(S, T^*).
\end{aligned}$$

So, `ALGSTR`, which is based on α-approximate `ALG` applied to the "segmented" datasets S_i, is $(4\alpha^2 + 6\alpha)$-approximate for the overall problem on $S = \cup_i S_i$.

3.7 Memory requirement

Suppose $|S| = n$ and we run `ALGSTR` with k-centroid centers. Let S be partitioned into l parts $S_1, \cdots, S_l$. We will run `ALG` on each of the partitions at a time. We must also accumulate the results T_i for each partition. Since $|T_i| \propto k$, the basic memory requirement is

$$\text{Required memory} \propto n/l + k \cdot l.$$

Let $l = \sqrt{n/k}$. In then follows that the memory usage of `ALGSTR` is $O(\sqrt{nk})$.

4 Data streams

4.1 Prove that $\tilde{F}[i] \geq F[i]$ for any $i = 1, 2, \cdots, n$

Let the true count of item i be c, *i.e.* $F[i] = c$. For every hash function h_j, item i will be hashed into the bucket $h_j(i)$ precisely c times. Now, it may be the case that the hash h_j maps other indices to $h_j(i)$. Hence, we conclude

$$c_{j,h_j(i)} \geq c$$

for every j. It follows immediately that

$$\tilde{F}[i] = \min_j \left\{ c_{j,h_j(i)} \right\} \geq c = F[i].$$

4.2 Prove that $E[c_{j,h_j(i)}] \leq F[i] + \frac{\epsilon}{e}(t - F[i])$ for all i, j

Consider j to be fixed. First, we note that the expectation is over the distribution of possible hash functions that can be assigned to h_j.

Now, for any hash function h_j, the algorithm will map the $F[i]$ occurrences of the item i into bucket $h_j(i)$. Given the distribution over the hash function h_j, all other items $i' \neq i$ are equally likely to be mapped to any of the buckets $1, 2, \cdots, \lceil e/\epsilon \rceil$. Thus, we obtain the following expectation

$$\begin{aligned} E[c_{j,h_j(i)}] &= F[i] + \frac{1}{\lceil e/\epsilon \rceil} \sum_{i' \neq i} F[i'] = F[i] + \frac{1}{\lceil e/\epsilon \rceil} (t - F[i]) \\ &\leq F[i] + \frac{1}{e/\epsilon} (t - F[i]) = F[i] + \frac{\epsilon}{e}(t - F[i]) \end{aligned}$$

where, in the last line, we have utilized the fact that $\lceil e/\epsilon \rceil \geq e/\epsilon$.

4.3 Prove that $Pr\left(\tilde{F}[i] \leq F[i] + \epsilon t\right) \geq 1 - \delta$

We consider the probability of the complement event. We make use of the fact that for the minimum of a set to exceed some threshold value, all elements in the set must exceed that threshold.

$$\begin{aligned} Pr\left(\tilde{F}[i] > F[i] + \epsilon t\right) &= Pr\left(\min_j \left\{ c_{j,h_j(i)} \right\} > F[i] + \epsilon t\right) \\ &= \prod_j Pr\left(c_{j,h_j(i)} > F[i] + \epsilon t\right) = \prod_j Pr\left(c_{j,h_j(i)} - F[i] > \epsilon t\right) \\ &\leq \left[Pr\left(c_{j,h_j(i)} - F[i] > \epsilon t\right) \right]^{\log 1/\delta}. \end{aligned}$$

In the second line, we use the fact that the hash functions h_j are i.i.d. In the last line, we obtain the inequality since $\log 1/\delta \leq \lceil \log 1/\delta \rceil$.

We then apply the Markov inequality, incorporating the results of Section 4.2, yielding

$$Pr\left(c_{j,h_j(i)} - F[i] > \epsilon t\right) \leq \frac{E(c_{j,h_j(i)} - F[i])}{\epsilon t} \leq \frac{(\epsilon/e)(t - F[i])}{\epsilon t} = \frac{1}{e} \cdot \frac{t - F[i]}{t}.$$

Making use of the above result, we find

$$Pr\left(\tilde{F}[i] > F[i] + \epsilon t\right) \leq \left[\frac{1}{e} \cdot \frac{t - F[i]}{t}\right]^{\log 1/\delta} = \delta \cdot \left(\frac{t - F[i]}{t}\right)^{\log 1/\delta} \leq \delta.$$

Note that the final inequality follows since $\frac{t - F[i]}{t} \leq 1$.

Taking the complement, we thus conclude

$$Pr\left(\tilde{F}[i] \leq F[i] + \epsilon t\right) \geq 1 - \delta.$$

4.4 Application to the dense subgraph search

The algorithm for dense subgraph search in the previous problem set required us to count the number of edges connected to each node (*i.e.* the degree of the node). With n nodes in the graph, my implementation had used an array of n counters. We can clearly use the current algorithm to approximate the counts in $O(\log n)$ space.

The approximate counting routine is particularly suitable to the previous dense subgraph search algorithm. As noted in Section 4.3, the approximation is better for items i where $F[i]$ is not very small compared to t. In the dense subgraph search algorithm, we are seeking nodes that possess large degrees. So, the approximation should work well in this application.