Syntactic and Functional Variability of a Million Code Submissions in a Machine Learning MOOC

Jonathan Huang      Chris Piech
Andy Nguyen         Leonidas Guibas
A variety of assessments

How can we efficiently grade 10,000 students?

Below is a poem by Emily Dickinson, known by its first line, "I taste a liquor never brewed."

I taste a liquor never brewed —
From Tankards scooped in Pearl —
Not all the Vats upon the Rhine
Yield such an Alcohol!

Inebriate of Air — am I —
And Debaucheet of Dew —
Reeling — thro endless summer days —
From inns of Molten Blue —

When "Landlords" turn the drunken Bee
Out of the Foxglove's door —
When Butterflies — renounce their "drams" —
I shall but drink the more!

Till Seraphs swing their snowy Hats —
And Saints — to windows run —
To see the little Tippler
Leaning against the — Sun —

In your short essay, do a “close reading” of this poem. Use as a model the close readings done in the several filmed discussions of other poems by Dickinson.

You may, for example, discuss at least briefly every line of the poem. Or you may choose what you consider to be key lines (or metaphors or terms) and explain each of them fully.
Complex and informative feedback in MOOCs

- **Short Response**
  - Easy to automate
  - Requires crowdsourcing (for now anyway)
  - Limited ability to ask expressive questions or require creativity

- **Long Response**
  - Can assign complex assignments and provide complex feedback

- Multiple choice
- Coding assignments
- Proofs
- Essay questions
Binary feedback for coding questions

Linear Regression submission (Homework 1) for Coursera’s ML class

function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
%GRADIENTDESCENT Performs gradient descent to learn theta
% theta = GRADIENTDESCENT(X, y, theta, alpha, num_iters) updates theta by
% taking num_iters gradient steps with learning rate alpha

m = length(y); % number of training examples
J_history = zeros(num_iters, 1);

for iter = 1:num_iters
    theta = theta - alpha*1/m*(X'*((X*theta)-y));
    J_history(iter) = computeCost(X, y, theta);
end

Test Inputs

What would a human grader (TA) do?

Correct / Incorrect?
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
    m = length(y);
    J_history = zeros(num_iters, 1);
    for iter = 1:num_iters
        hypo = X*theta;
        newMat = hypo - y;
        trans1 = (X(:,1));
        trans1 = trans1';
        newMat1 = trans1 * newMat;
        temp1 = sum(newMat1);
        temp1 = (temp1 * alpha) / m;
        A = [temp1];
        theta(1) = theta(1) - A;
        trans2 = (X(:,2))';
        newMat2 = trans2*newMat;
        temp2 = sum(newMat2);
        temp2 = (temp2 * alpha) / m;
        B = [temp2];
        theta(2)= theta(2) - B;
        J_history(iter) = computeCost(X, y, theta);
    end
    theta(1) = theta(1);
    theta(2)= theta(2);
end
ML-class by the numbers

- Registered Users: 120,000
- Users who submitted code: 25,839
- Users who submitted to every homework assignment: 10,405

Number of unique users submitting to each problem:

- Optional problems: 120,000
- 42 coding problems (ordered chronologically): 25,839

Topics: Linear regression, Logistic Regression, SVMs, Neural Nets, PCA, Nearest Neighbor...

How will Andrew give feedback to 25,000 students (~40,000 code submissions)?
Gradient descent for linear regression

~40,000 submissions
Unit test output classes

Student #1: [0.1, 0.3, 0]

Student #2: [0.1, 0.4, 0]

Student #3: [0.1, 0.3, 0]

Student #4: NaN

Student #5: NaN

Student #6: [0.1, 0.4, 0]

...
Output based feedback

Output class #1

[0.1, 0.3, 0]

Instructor Feedback:
Great job! Keep up the good work!

Output class #2

[0.1, 0.4, 0]

Instructor Feedback:
Did you remember to preprocess the data before running the SVD?

Output class #3

[NaN]

Instructor Feedback:
You might have accidentally divided by zero when you tried to normalize

How many output classes must Andrew Ng annotate?
Functional Variability

Problems with over 1000 ways to be wrong!

Regularized Logistic regression (i.e., poor Andrew…)

Of course, not all output classes are born equal…
How many students are covered?

If Andrew gives feedback on 11 output classes, he “covers” 75% of the users.

Return a 5x5 identity matrix

Linear Regression (Gradient Descent)

Regularized Logistic Regression
Beyond output classes

```matlab
function [J, grad] = lrCostFunction (theta, X, y, lambda)
    m = length (y);
    J = 0;
    grad = zeros (size (theta));
    for i = 1:m
        J = J + (-y (i) * log (sigmoid (X (i, :) * theta)))
            + (1 - y (i)) * log (1 - sigmoid (X (i, :) * theta));
    endfor;
    J = J / m;
    for j = 2:size (theta)
        J = J + (lambda * (theta (j) ^ 2) / (2 * m));
    endfor;
    for j = 1:size (theta)
        for i = 1:m
            grad (j) = grad (j) + (sigmoid (X (i, :) * theta) - y (i)) * X (i, j);
        endfor;
        grad (j) = grad (j) / m;
    endfor;
    for j = 2:size (theta)
        grad (j) = grad (j) + (lambda * theta (j)) / m;
    endfor;
    grad = grad (:);
endfunction
```

Output based feedback ignores:
- vectorization,
- multiple approaches with the same result,
- coding style, ...

Two “correct” implementations of logistic regression
function A = warmUpExercise()
% hellooo!
A = [];
A = eye(5);
endfunction

Abstract syntax tree

ASTs ignore:
- Whitespace
- Comments
- Differences in variable names
  ...
Syntactic Variability

and again: not all ASTs are born equal…
How many students are covered?

Idea: If Alice and Bob have highly similar submissions (albeit not identical), the same feedback can often apply to both students!!

If Andrew gives feedback on 20 ASTs, he “covers” ~30% of the users.
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)

m = length(y);
J_history = zeros(num_iters, 1);

for iter = 1:num_iters
    h=X*theta;
    theta=theta-alpha*(1/m).*(X*(h-y));
    J_history(iter) = computeCost(X, y, theta);
end

function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)

m = length(y);
J_history = zeros(num_iters, 1);

for iter = 1:num_iters
    temp1 = (alpha/m) * ( ( (theta' * X')' - y)' * X(:,1) ) ;
    temp2 = (alpha/m) * ( ( (theta' * X')' - y)' * X(:,2) ) ;
    theta(1) = theta(1) – temp1;
    theta(2) = theta(2) – temp2;
    fprintf('Theta: %f, %f\n', theta(1), theta(2))
    J_history(iter) = computeCost(X, y, theta);
end
Tree matching

function A = warmUpExercise()
    A = eye(5);
endfunction

function A = warmUpExercise()
    A = [];  
    A = eye(5);
endfunction

FUNCTION (warmUpExercise)
    STATEMENT_LIST
        ASSIGN
            TARGET
                IDENT (A)
            SOURCE
                CONST ([[]])
        ASSIGN
            TARGET
                IDENT (A)
            SOURCE
                IDX_EXPR
                    IDENT (eye)
                    ARG_LIST
                        CONST (5)
                NO_OP (endfunction)
Matched ASTs for two submitted octave implementations

Gradient descent for one-dimensional linear regression

<table>
<thead>
<tr>
<th>Purple</th>
<th>tokens common to both implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red/Blue</td>
<td>tokens distinct to one of the implementations</td>
</tr>
</tbody>
</table>
Code webs

Graphs of code submissions with nodes connected to nearest neighbors

• Codewebs can act as a backbone for:
  – Clustering
  – Outlier identification
  – Automatic suggestions for alternative (better) implementations
  – Propagating complex and informative feedback/annotations
Labeled code webs

Gradient descent for linear regression
~40,000 submissions
Interplay between functional and syntactic similarity

Log distribution for pairs that **agree** on unit tests

Log distribution for pairs that **disagree** on unit tests

Interplay between functional and syntactic similarity
Finding Prototypical Submissions

(we use affinity propagation [Frey, Dueck, 2007])
function [theta, J_history] = gradientDescent(X,y,theta,alpha,num_iters)

m = length(y);
J_history = zeros(num_iters, 1);

#1
for iter = 1:num_iters
    theta = theta-alpha*1/m*(X'*((X*theta)-y));
    J_history(iter) = computeCost(X, y, theta);
end

m = length(y);
J_history = zeros(num_iters, 1);

#2
for iter = 1:num_iters,
    delta = X' *((X * theta) - y);
    theta = theta - (alpha / m * delta);
    J_history(iter) = computeCost(X, y, theta);
end

m = length(y);
J_history = zeros(num_iters, 1);

#3
for iter = 1:num_iters
    t0 = theta(1,1) - ((alpha / m) * sum (((X * theta) - y)));
    t1 = theta(2,1) - ((alpha / m) * sum (((X * theta) - y).* X (:,2)));
    theta = [t0; t1];
    J_history(iter) = computeCost(X, y, theta);
end

Most common implementation, vectorized, one-line update

Vectorized, with temporary variable

Unvectorized, synchronous update
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
    m = length(y);
    J_history = zeros(num_iters, 1);
    for iter = 1:num_iters
        hypo = X*theta;
        newMat = hypo - y ;
        trans1 = (X(:,1)) ;
        newMat1 = trans1 * newMat;
        temp1 = sum(newMat1);
        temp1 = (temp1 *alpha)/m;
        A = [temp1];
        theta(1) = theta(1) - A;
        trans2 = (X(:,2))' ;
        newMat2 = trans2*newMat;
        temp2 = sum(newMat2);
        temp2 = (temp2 *alpha)/m;
        B = [temp2];
        theta(2)= theta(2) - B;
        J_history(iter) = computeCost(X, y, theta);
    end
    theta(1) = theta(1) ;
    theta(2)= theta(2);
end
Finding Outlier Submissions

function [C, sigma] = dataset3Params(X, y, Xval, yval)

i = 1; j = 1;
global gl_sigma = d(j);
gl_sigma = d(j);
function f = f(x1, x2)
    global gl_sigma;
    f = gaussianKernel(x1, x2, gl_sigma);
end
model = svmTrain(X, y, d(i), @f);
pred = svmPredict(model, Xval);
ans = mean(double(pred ~= yval));
if (i == 1 && j == 1) || ans < res
    res = ans;
    C = d(i);
    sigma = d(j);
end

i = 1; j = 2;
global gl_sigma = d(j);
gl_sigma = d(j);
function f = f(x1, x2)
    global gl_sigma;
    f = gaussianKernel(x1, x2, gl_sigma);
end
model = svmTrain(X, y, d(i), @f);
pred = svmPredict(model, Xval);
ans = mean(double(pred ~= yval));
if (i == 1 && j == 1) || ans < res
    res = ans;
    C = d(i);
    sigma = d(j);
end

i = 1; j = 3;
global gl_sigma = d(j);
gl_sigma = d(j);
function f = f(x1, x2)
    global gl_sigma;
    f = gaussianKernel(x1, x2, gl_sigma);
end
model = svmTrain(X, y, d(i), @f);
pred = svmPredict(model, Xval);
ans = mean(double(pred ~= yval));
if (i == 1 && j == 1) || ans < res
    res = ans;
    C = d(i);
    sigma = d(j);
end

i = 1; j = 4;
global gl_sigma = d(j);
gl_sigma = d(j);
function f = f(x1, x2)
    global gl_sigma;
    f = gaussianKernel(x1, x2, gl_sigma);
end
model = svmTrain(X, y, d(i), @f);
pred = svmPredict(model, Xval);
ans = mean(double(pred ~= yval));
if (i == 1 && j == 1) || ans < res
    res = ans;
    C = d(i);
    sigma = d(j);
end

i = 1; j = 5;
global gl_sigma = d(j);
gl_sigma = d(j);
function f = f(x1, x2)
    global gl_sigma;
    f = gaussianKernel(x1, x2, gl_sigma);
end
model = svmTrain(X, y, d(i), @f);
pred = svmPredict(model, Xval);
ans = mean(double(pred ~= yval));
if (i == 1 && j == 1) || ans < res
    res = ans;
    C = d(i);
    sigma = d(j);
end

i = 1; j = 6;
global gl_sigma = d(j);
gl_sigma = d(j);
function f = f(x1, x2)
    global gl_sigma;
    f = gaussianKernel(x1, x2, gl_sigma);
end
model = svmTrain(X, y, d(i), @f);
pred = svmPredict(model, Xval);
ans = mean(double(pred ~= yval));
if (i == 1 && j == 1) || ans < res
    res = ans;
    C = d(i);
    sigma = d(j);
end

i = 1; j = 7;
global gl_sigma = d(j);
gl_sigma = d(j);
function f = f(x1, x2)
    global gl_sigma;
    f = gaussianKernel(x1, x2, gl_sigma);
end
model = svmTrain(X, y, d(i), @f);
pred = svmPredict(model, Xval);
ans = mean(double(pred ~= yval));
if (i == 1 && j == 1) || ans < res
    res = ans;
    C = d(i);
    sigma = d(j);
end

i = 1; j = 8;
global gl_sigma = d(j);
gl_sigma = d(j);
function f = f(x1, x2)
    global gl_sigma;
    f = gaussianKernel(x1, x2, gl_sigma);
end
model = svmTrain(X, y, d(i), @f);
pred = svmPredict(model, Xval);
ans = mean(double(pred ~= yval));
if (i == 1 && j == 1) || ans < res
    res = ans;
    C = d(i);
    sigma = d(j);
end
Code Webs

Paste your solution to Homework 1.3 below, and we will provide you with feedback. This feedback is not a submission to coursera. It is 100% anonymous and will not impact your grade in any way. Learn More.

pasta your entire function here

Get Feedback