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

Jonathan Huang
Chris Piech
Andy Nguyen
Leonidas Guibas

LYTICS
moocshop
AIED 2013 MEMPHIS
LyticsLab@Stanford

A variety of assessments

How can we efficiently grade 10,000 students?

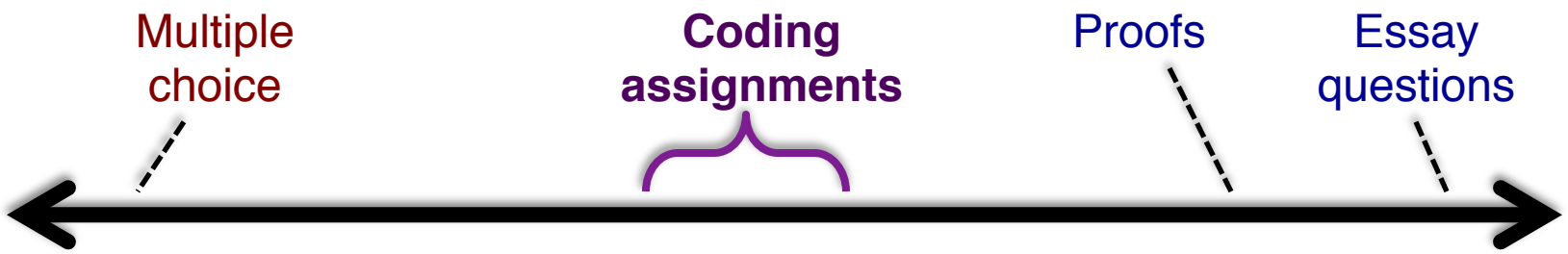
Below is a poem by Emily Dickinson, known by its first line, "I taste a liquor never brewed."

> 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 Debauchee 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 ... **Long Response**

- Easy to automate
- Limited ability to ask expressive questions or require creativity
- Requires crowdsourcing *(for now anyway)*
- Can assign complex assignments and provide complex feedback

Binary feedback for coding questions

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

Test Inputs

```
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 Outputs

What would a human grader (TA) do?

Correct / Incorrect ?

Complex, Informative Feedback

```
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);
```

Please come up with better variable names!

Better: theta = theta-(alpha/m)

X'(X*theta-y)

Why??

Correctness	Good
Efficiency	Good
Style	Poor
Elegance	Poor

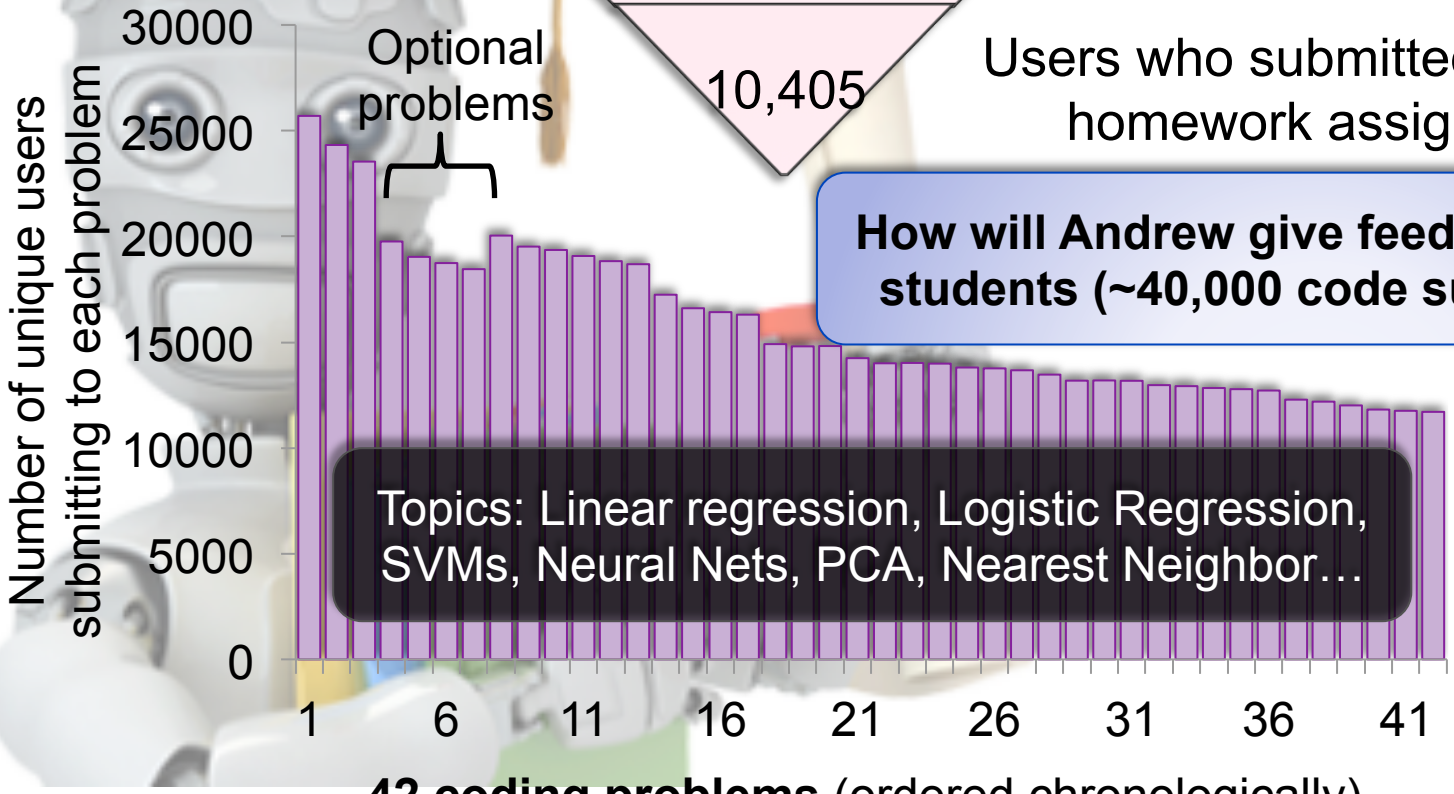
Taught by Andrew Ng

ML-class by the numbers

120,000 Registered Users

25,839 Users who submitted code

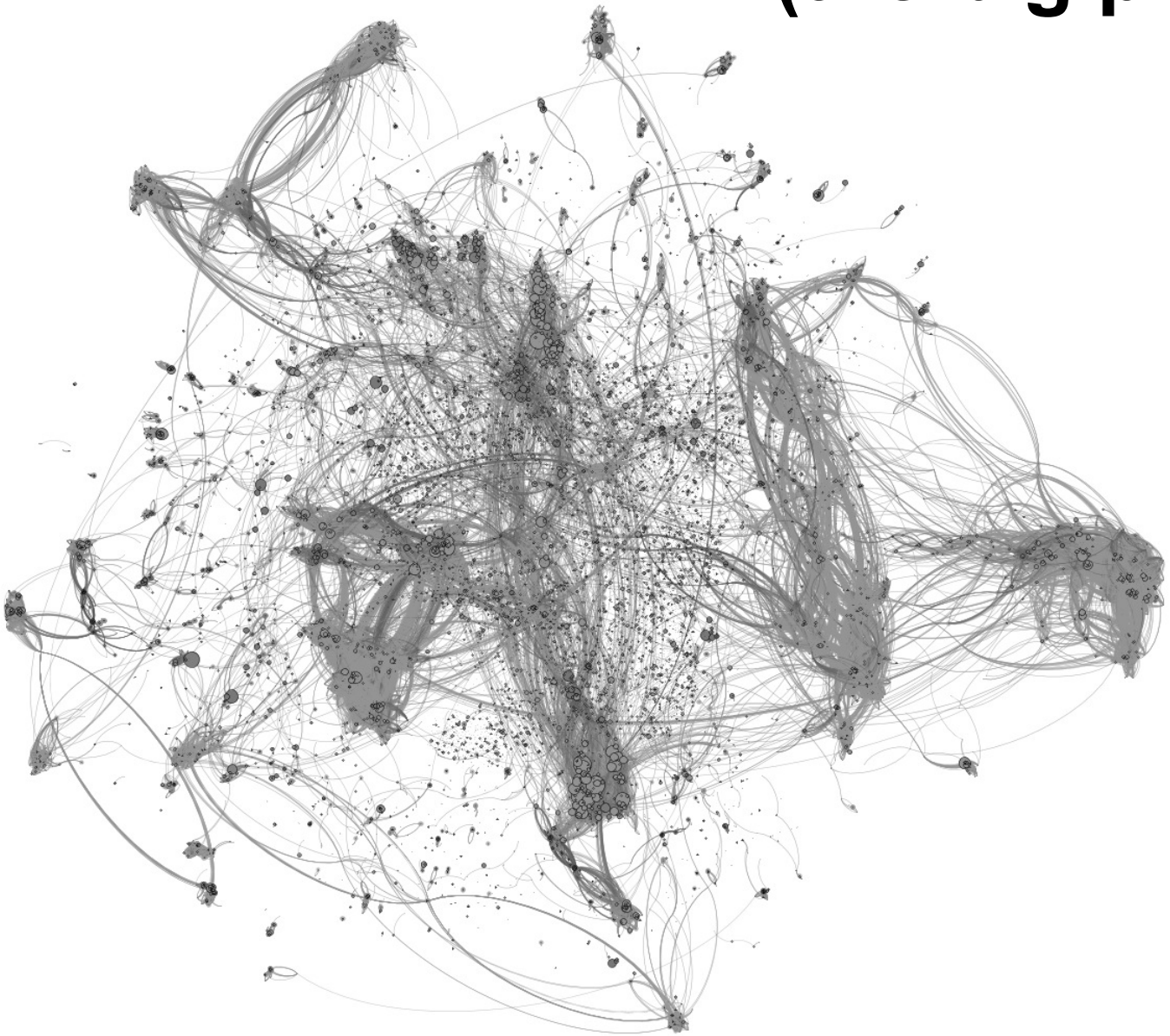
10,405 Users who submitted to every homework assignment



How will Andrew give feedback to 25,000 students (~40,000 code submissions)?

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

(the big picture)

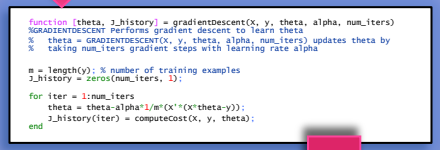


Gradient descent for linear regression
~40,000 submissions

Unit test output classes

Unit test inputs

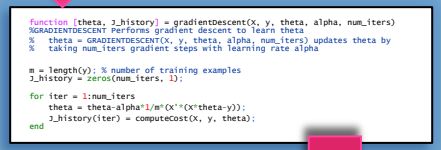
Student #1



[.1, .3, 0]

Unit test inputs

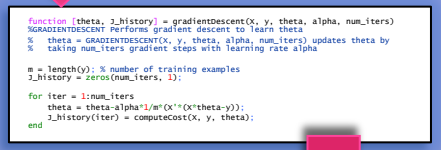
Student #2



[.1, .4, 0]

Unit test inputs

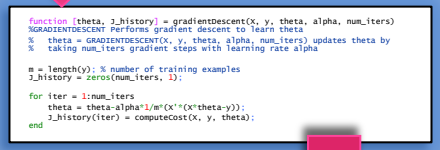
Student #3



[.1, .3, 0]

Unit test inputs

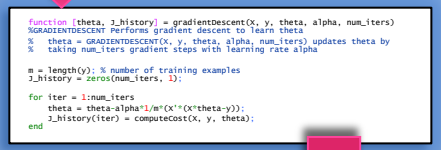
Student #4



[NaN]

Unit test inputs

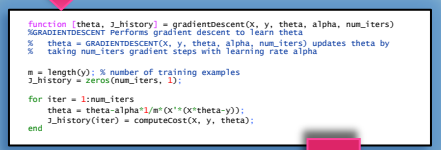
Student #5



[NaN]

Unit test inputs

Student #6

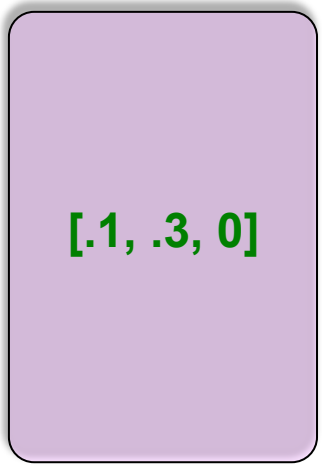


[.1, .4, 0]

. . .

Output based feedback

Output class #1

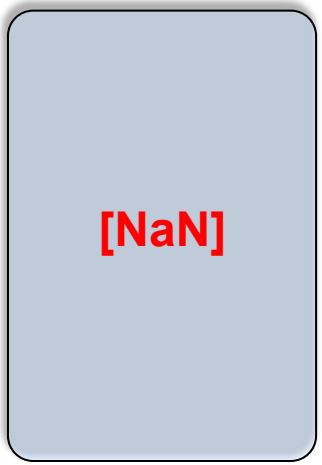


Instructor Feedback:
Great job! Keep up the good work!

Output class #2

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

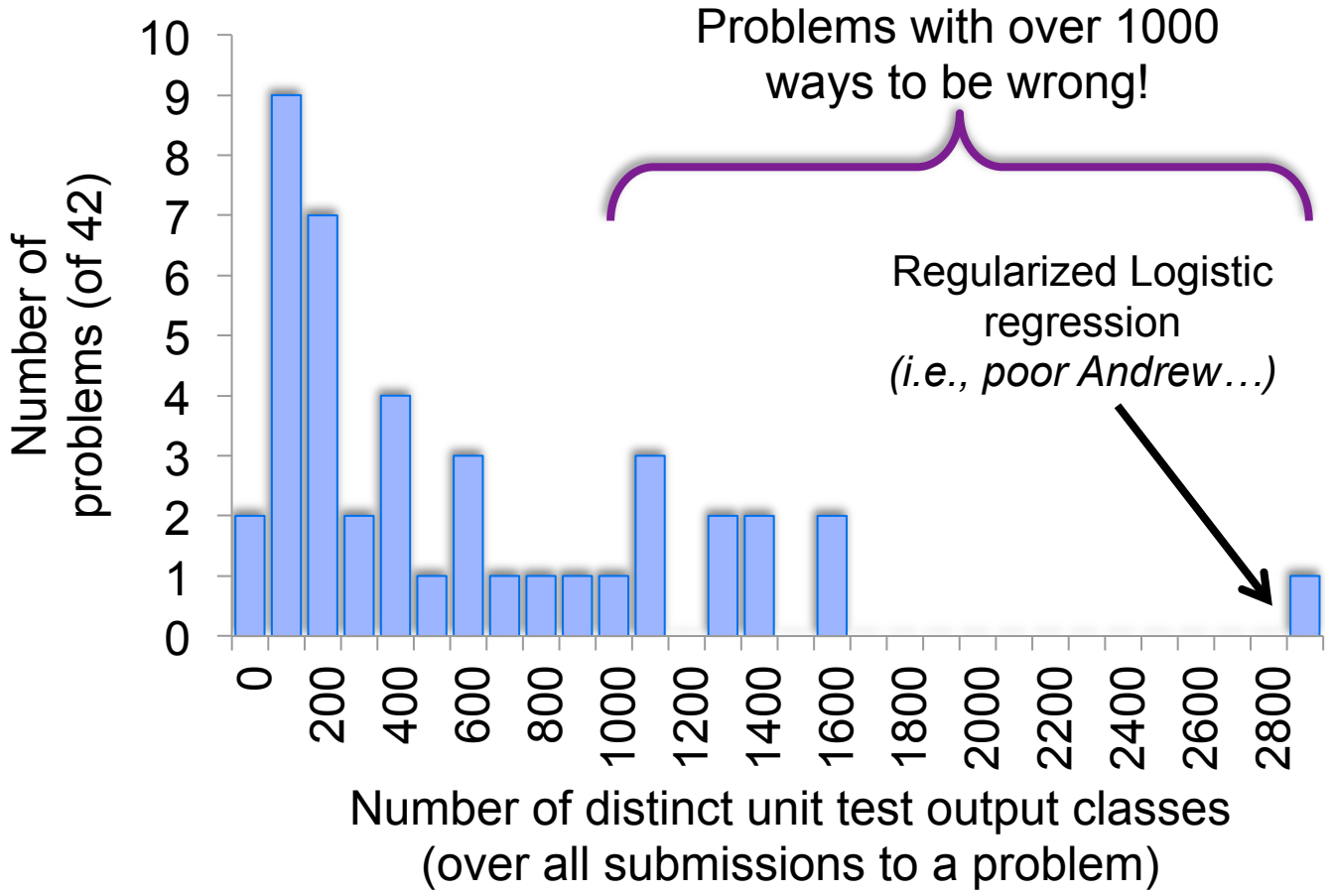
Output class #3



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

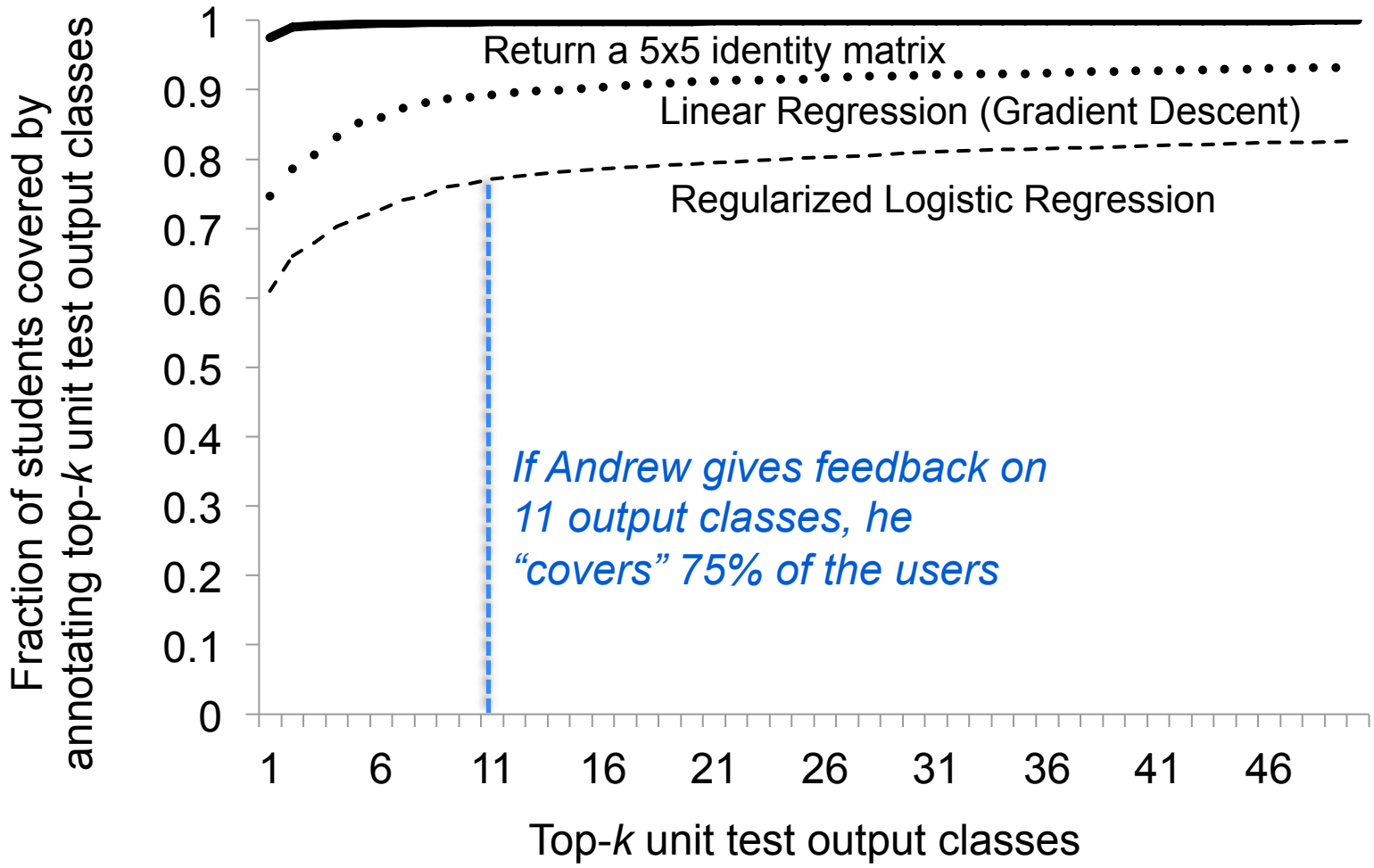
How many output classes must Andrew Ng annotate?

Functional Variability



Of course, *not all output classes are born equal…*

How many students are covered?



Beyond output classes

```
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, :) * th
    endfor;
    grad (j) = grad (j) / m;
  endfor;
  for j = 2:size (theta)
    grad (j) = grad (j) + (la
  endfor;
  grad = grad (:);
```

Output based feedback ignores:

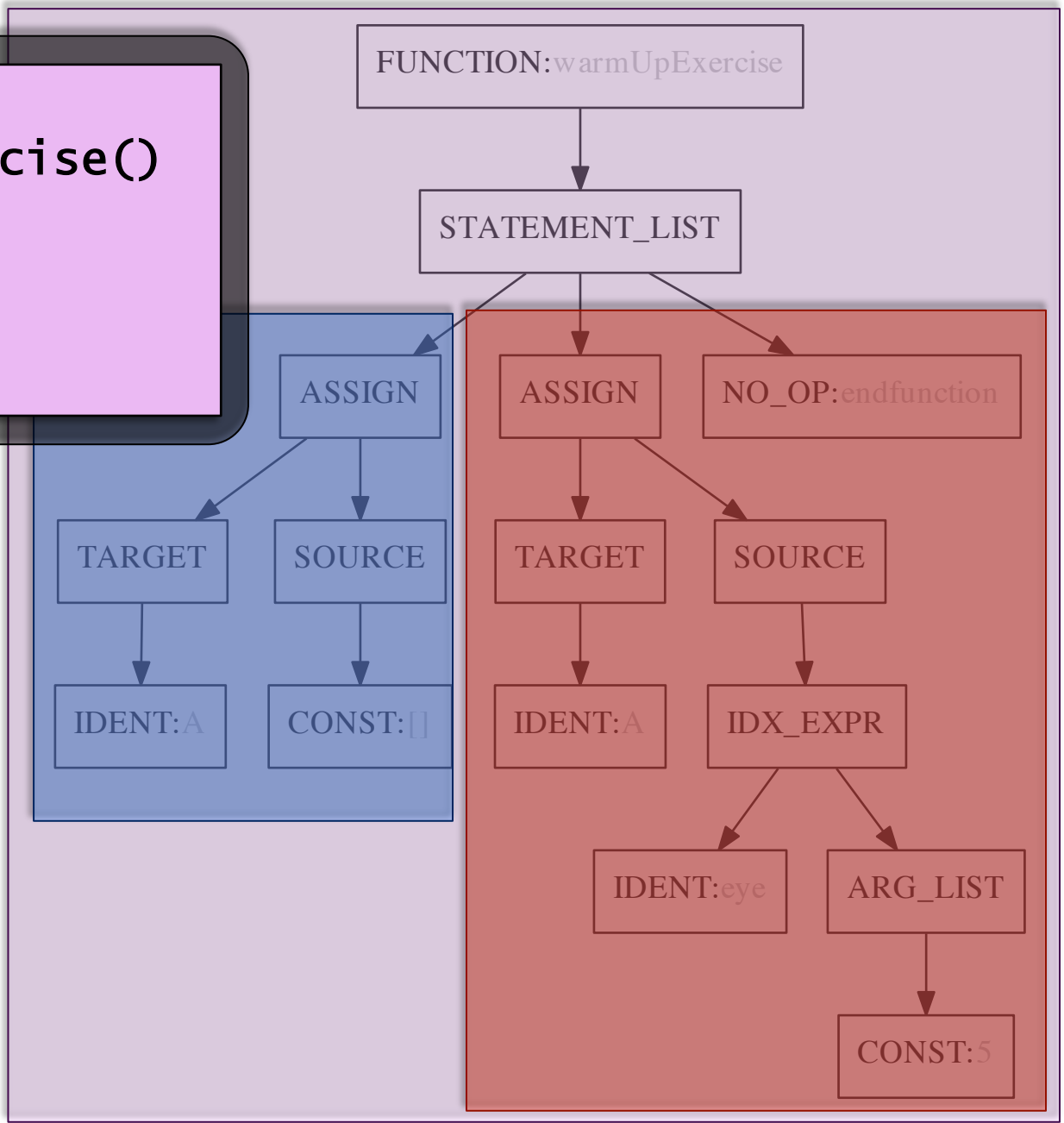
- vectorization,
- multiple approaches with the same result,
- coding style, …

```
function [J, grad] = lrCostFunction (theta, X, y, lambda)
  m = length (y);
  J = 0;
  grad = zeros (size (theta));
  n = size (X, 2);
  h = sigmoid (X * theta);
  J = ((-y)' * log (h) - (1 - y)' * log (1 - h)) / m;
  theta1 = [0; theta(2:size (theta), :)];
  soma = sum (theta1' * theta1);
  p = (lambda * soma) / (2 * m);
  J = J + p;
  grad = (X' * (h - y) + lambda * theta1) / m;
```

Two “correct” implementations of logistic regression

From function… to syntax!

```
function A =
         warmUpExercise()
  % hellooooo!
  A = [];
  A = eye(5);
endfunction
```

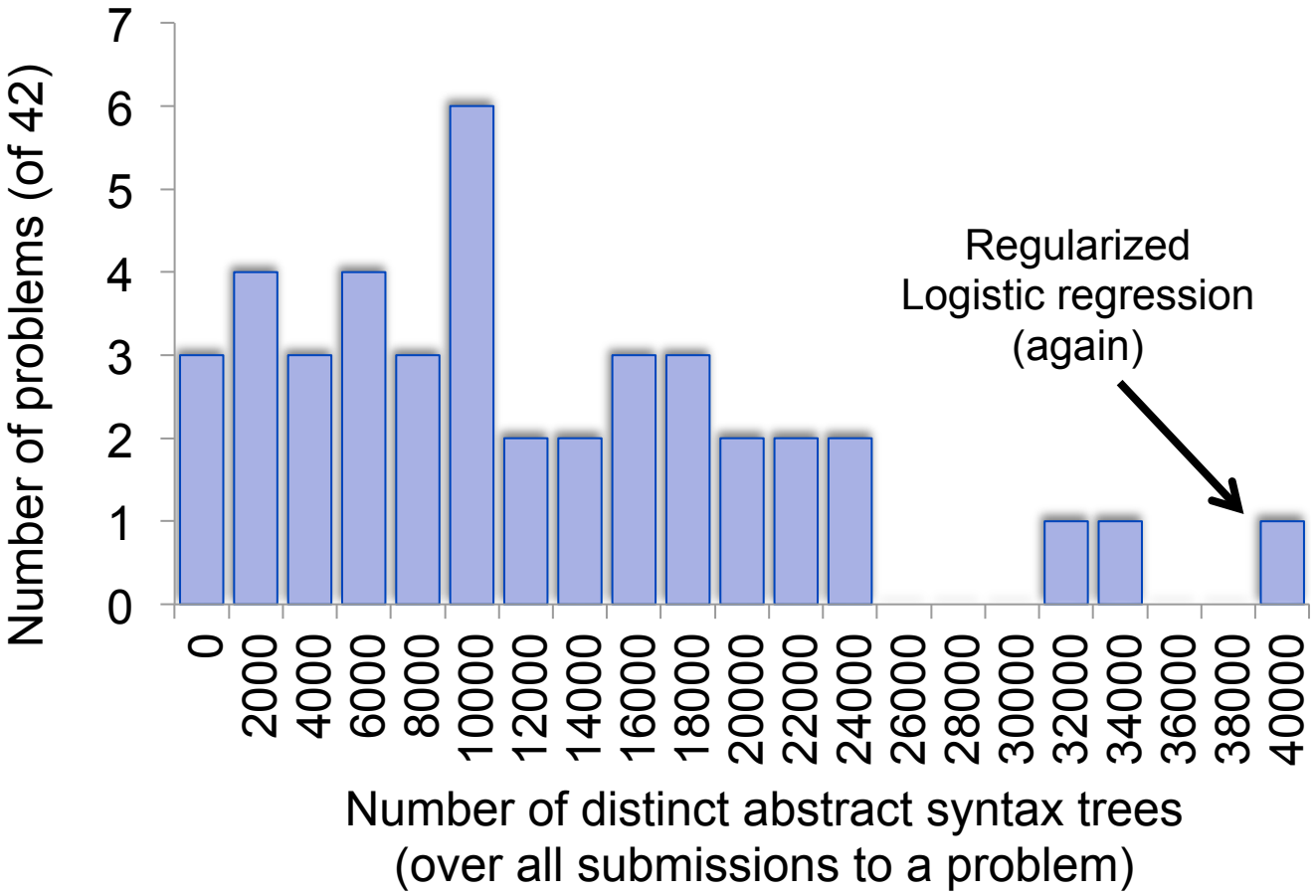


Abstract syntax tree

ASTs ignore:

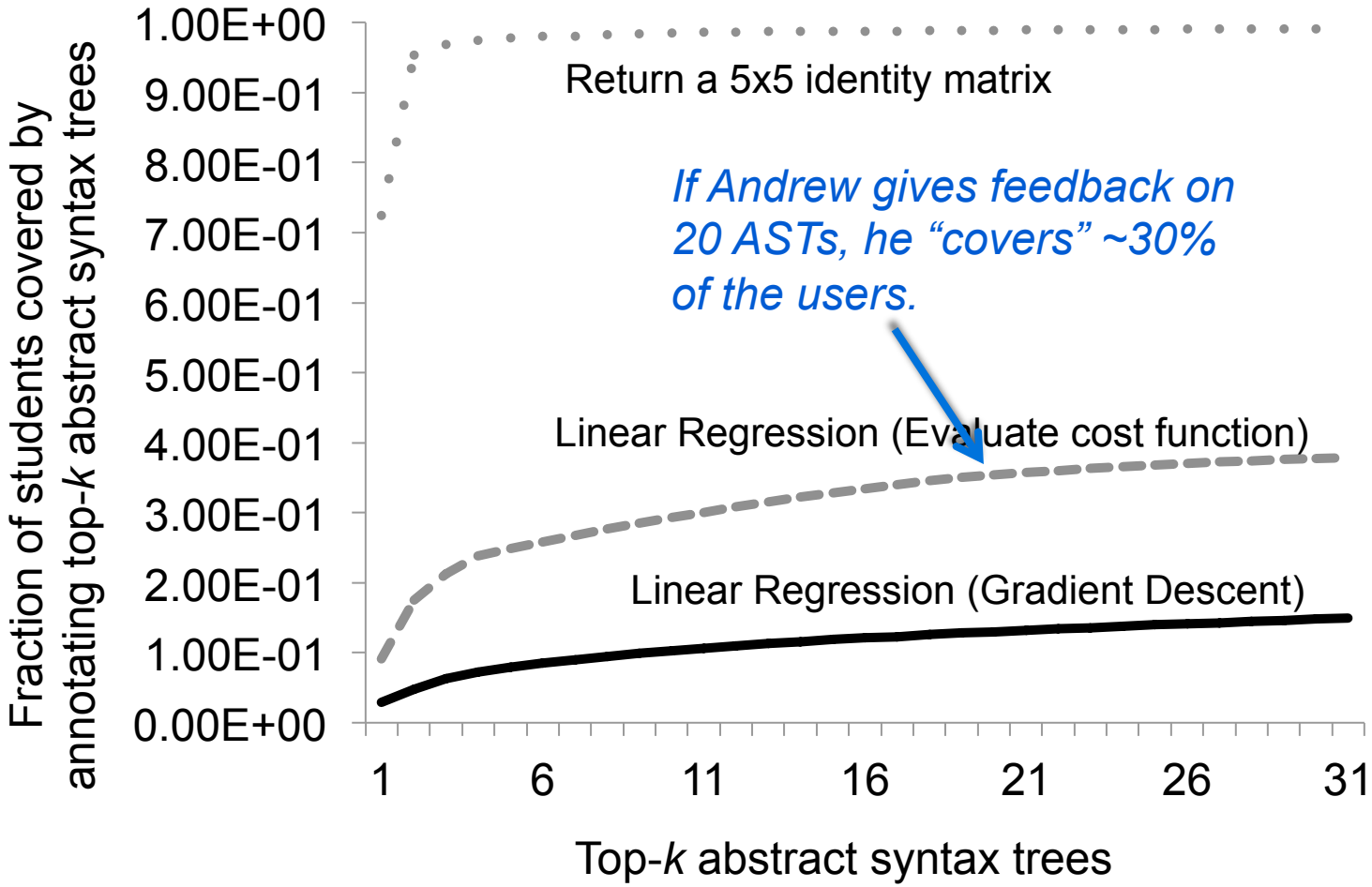
- **Whitespace**
- **Comments**
- **Differences in variable names**
- **…**

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!!

Core Subproblem: Code Correspondence

```
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
```

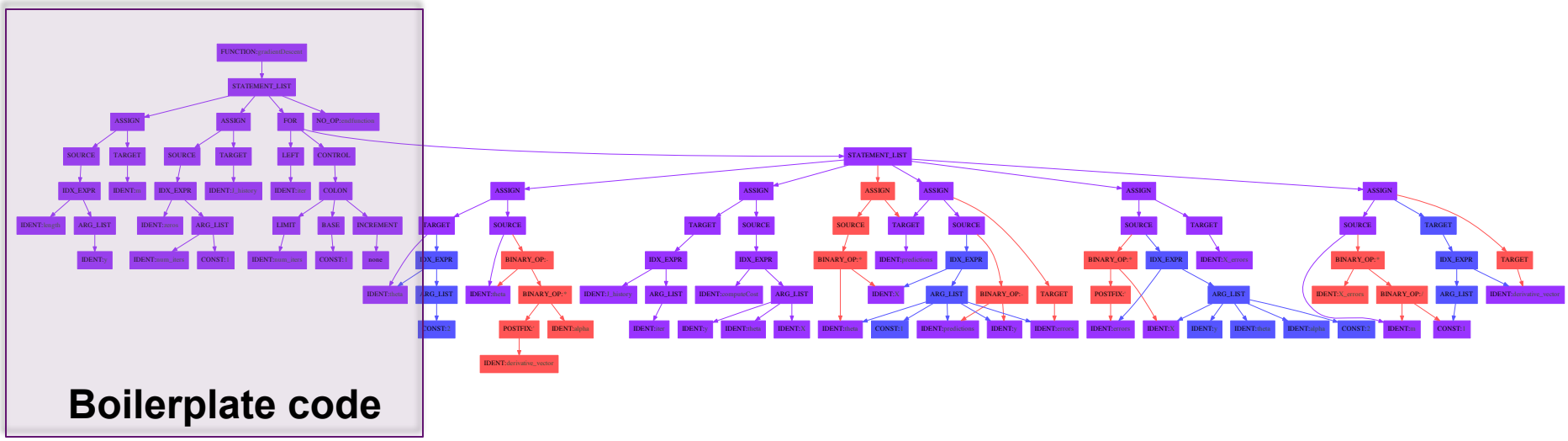
```
function A = warmUpExercise()

  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)
```

See [Shasha et al, ‘94] for $O(n^4)$ dynamic programming algorithm

Matched ASTs for two submitted octave implementations

Gradient descent for one-dimensional linear regression

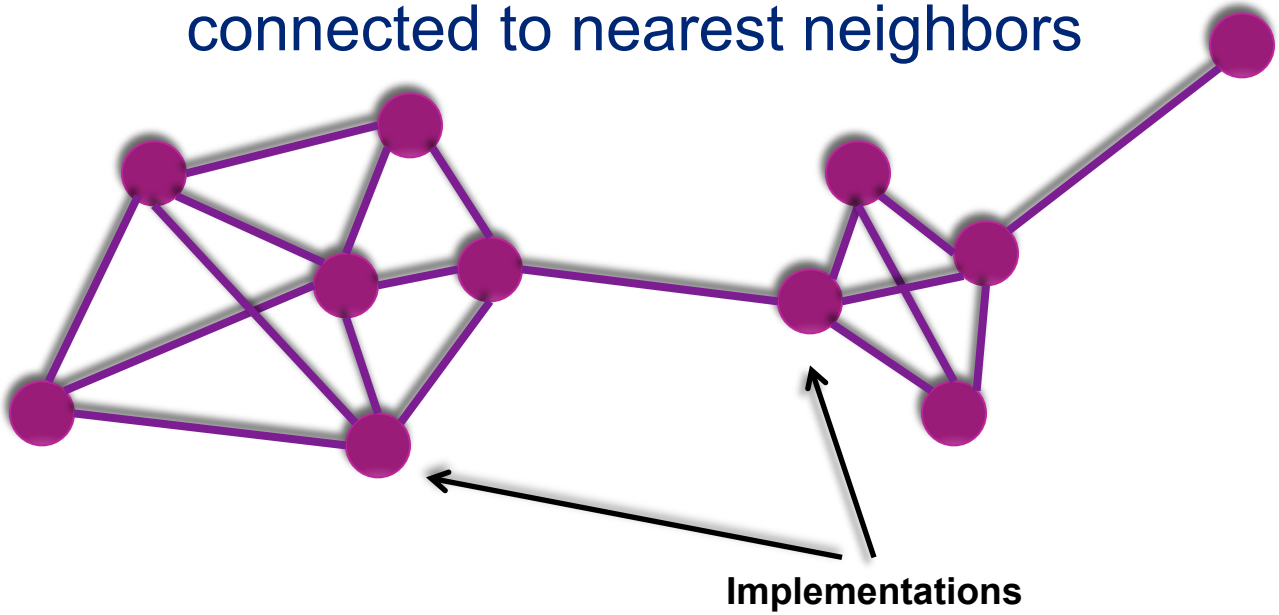


Purple – tokens common to both implementations

Red/**Blue** – tokens distinct to one of the implementations

Code webs

Graphs of code submissions with nodes connected to nearest neighbors

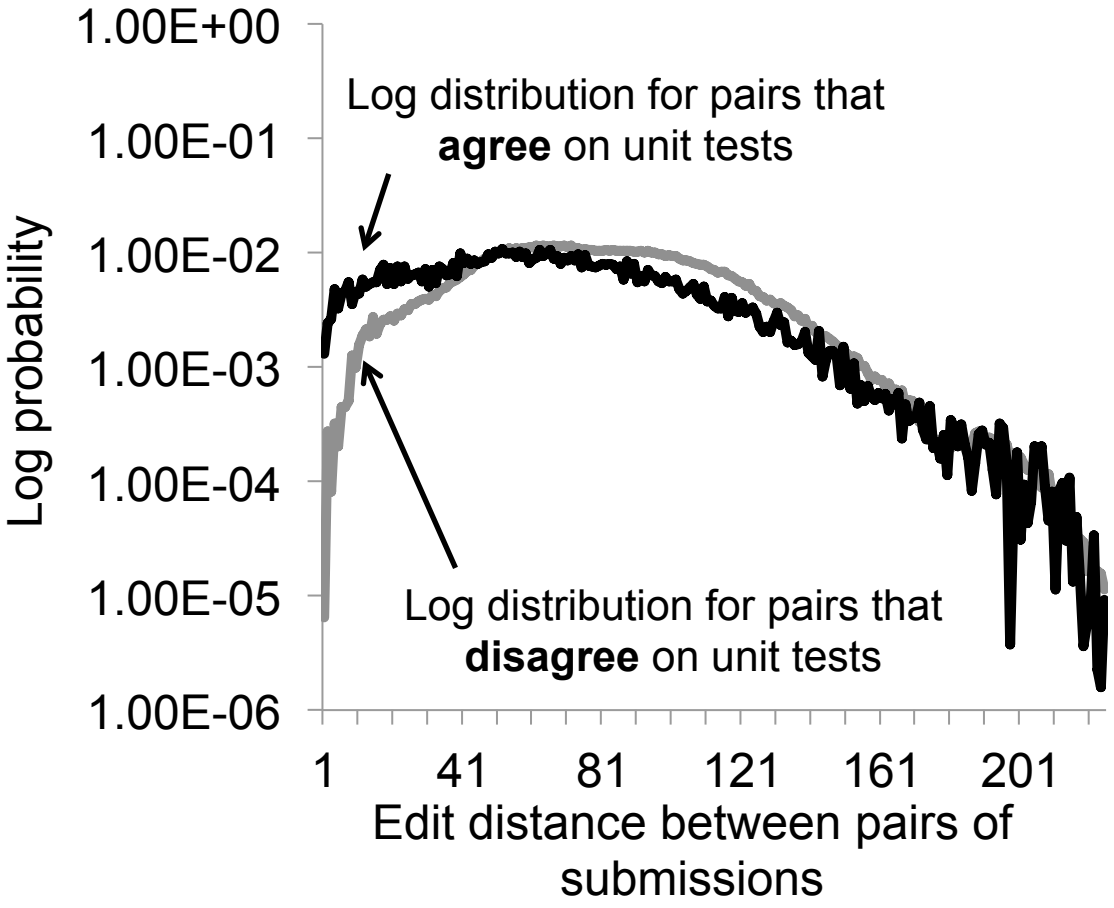


- **Codewebs can act as a backbone for:**
  - Clustering
  - Outlier identification
  - Automatic suggestions for alternative (better) implementations
  - Propagating complex and informative feedback/annotations

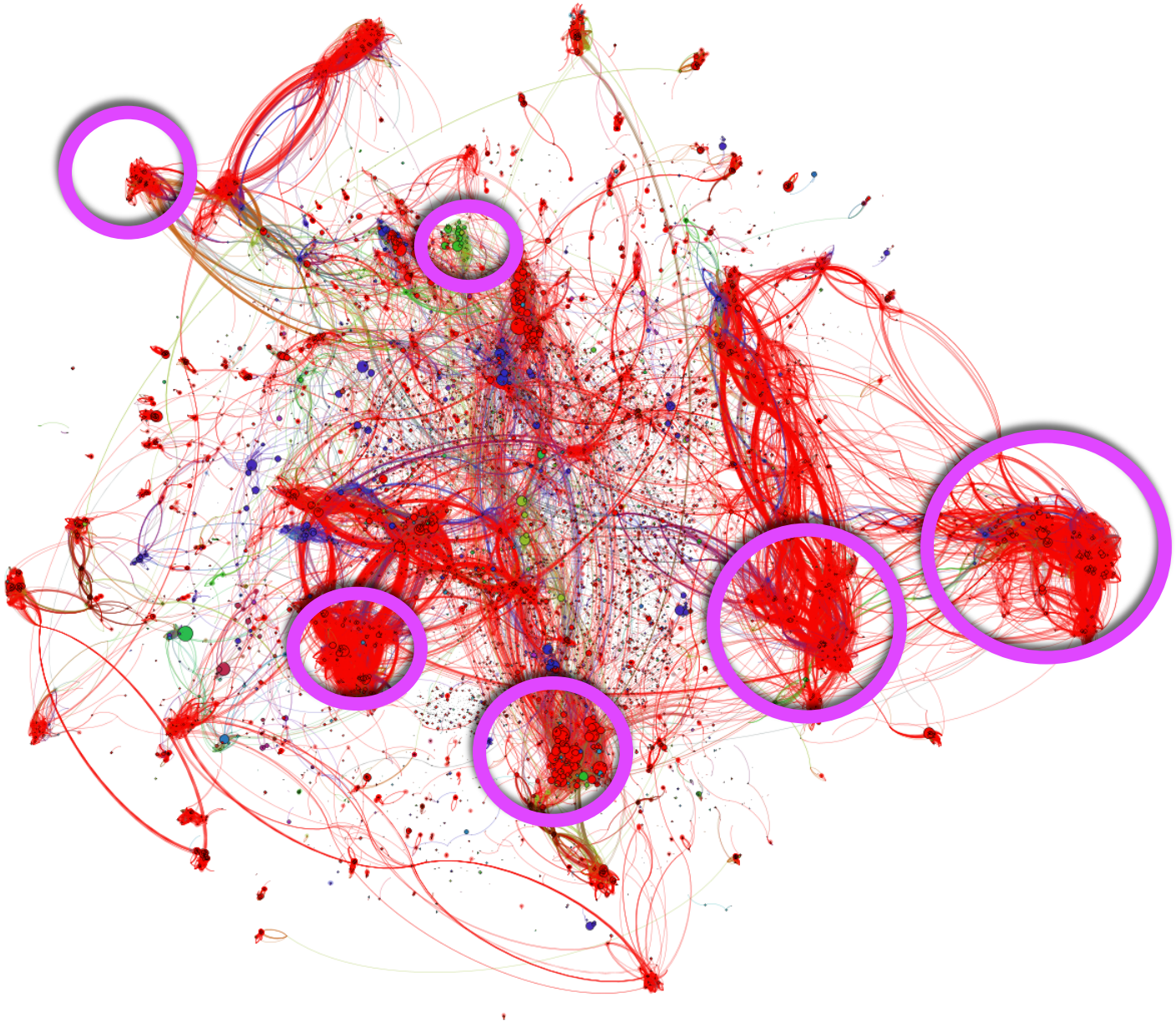
Labeled code webs

Gradient descent for linear regression
~40,000 submissions

Interplay between functional and syntactic similarity



Finding Prototypical Submissions



(we use affinity propagation [Frey, Dueck, 2007])

(Correct) Gradient Descent Prototypes

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

#1

```
m = length(y);
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
```

Most common implementation, vectorized, one-line update

#2

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

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

Vectorized, with temporary variable

#3

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

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
```

Unvectorized, synchronous update

Finding Outlier Submissions

```
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
```

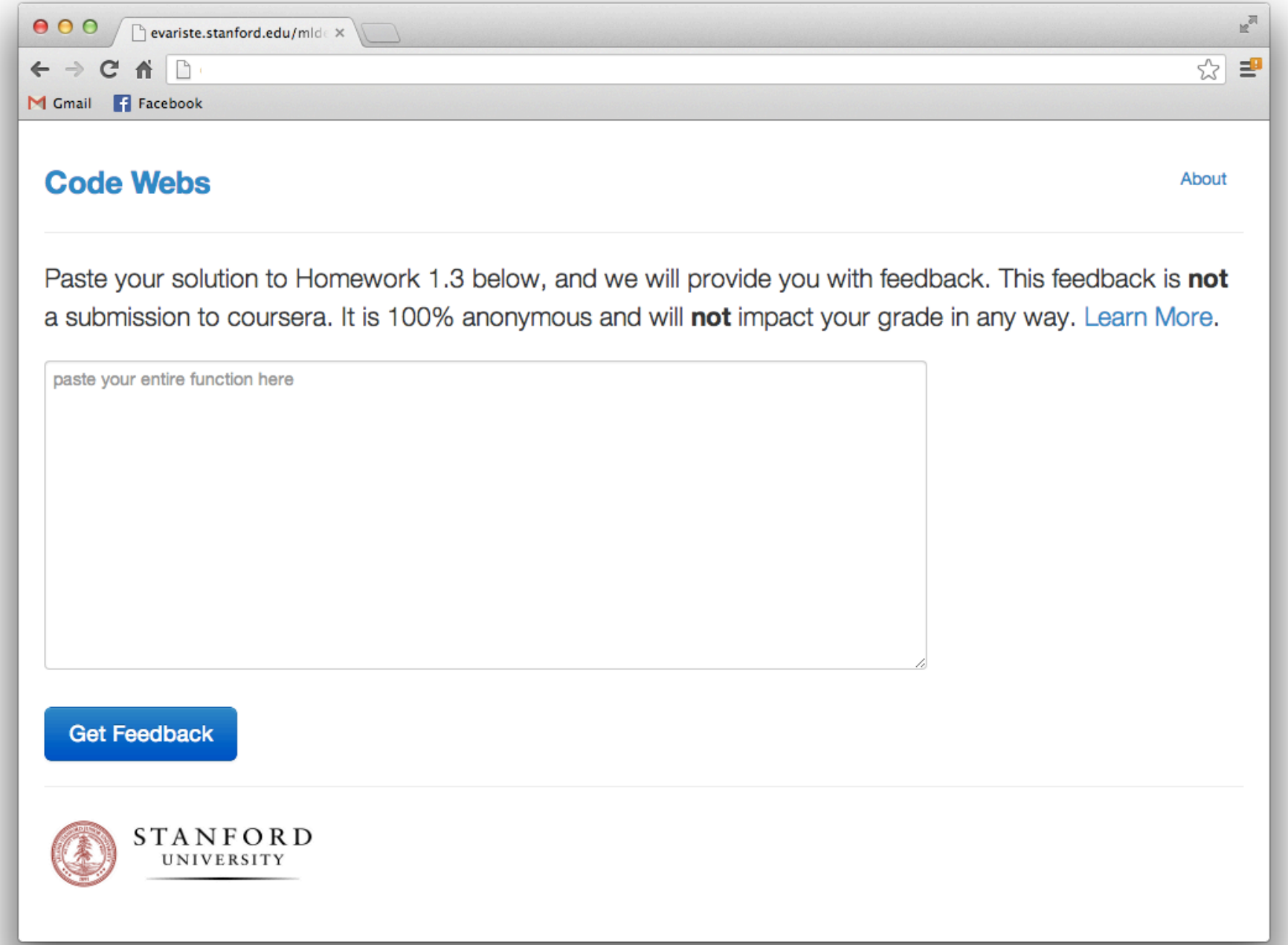
```
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
m = length(y);
J_history = zeros(num_iters, 1);
for iter = 1:num_iters
a=0;
h=zeros(m,1);
h=X*theta;
k=(h-y);
for i=1:m,
a=k(i)+a;
end;
a=(alpha*a)/m;
theta(1)=theta(1)-a;
a=0;
x1=X(:,2);
for i=1:m,
a=k(i)*x1(i)+a;
end;
a=(alpha*a)/m;
theta(2)=theta(2)-a;
    J_history(iter) = computeCost(X, y, theta);
end
end
```

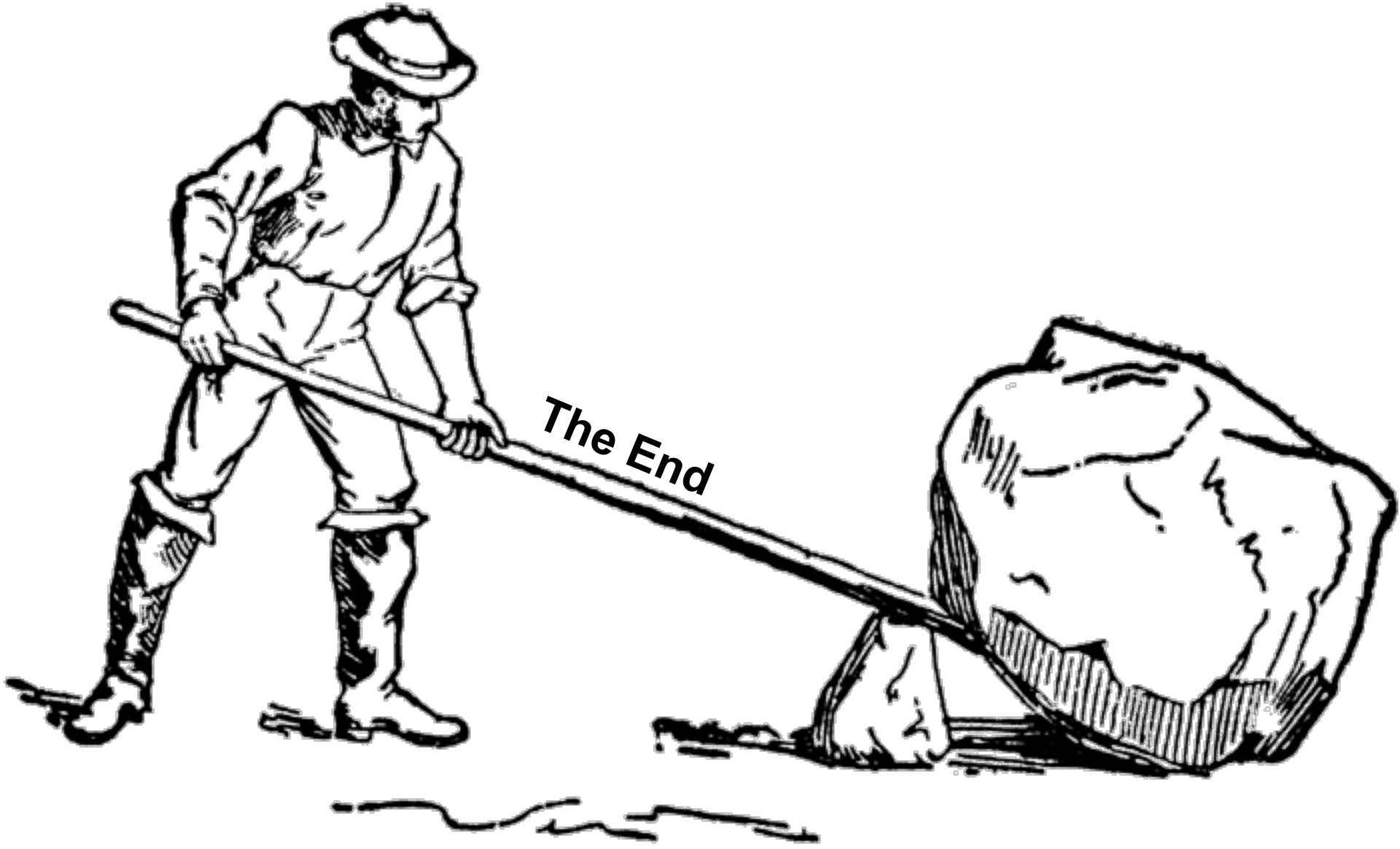
Finding Outlier Submissions

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

C = 1;
sigma = 0.3;
mean(double(predictions ~= yval))
%
d = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30];
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);
```

```
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);
```


evariste.stanford.edu/mlc
Gmail
Facebook
Code Webs
About
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.
paste your entire function here
Get Feedback
STANFORD
UNIVERSITY


The End