10 11 2020 Week 3 - Logistic Regression Functions

Logistic Regression for Classification Problems

$h_\theta(x) = g(\theta^T x)$

$g(z) = \frac{1}{1+e^{-z}}$ → Sigmoid function

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

$0 \leq h_\theta(x) \leq 1$

$h_\theta(x)$ is going to be conceptually represented as the probability that $y = 1$

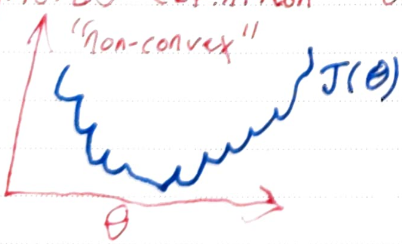
$y \in \{0, 1\}$

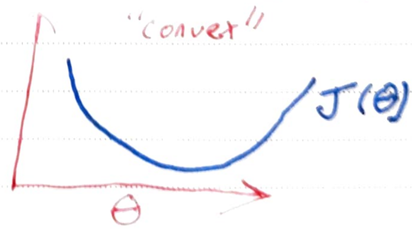
Cost Function:

Linear R

$$\text{cost}(h_\theta(x^{(i)}), y) = \tfrac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

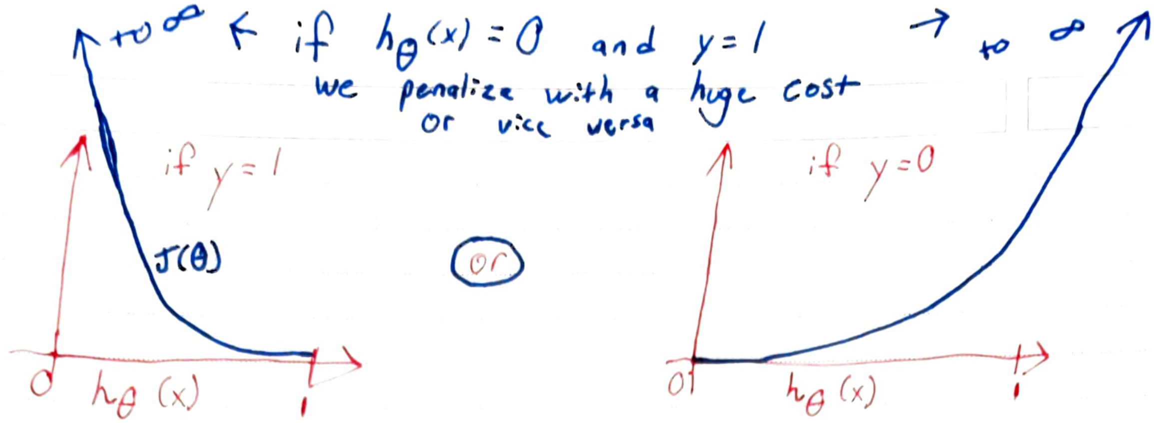
if using the linear regr... ...tion with the sigmoid definition of $(h_\theta(x^{i})$ ... would be "non-convex"





Logistic Regression:

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



Logistic Regression Cost Function:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))$$

Gradient Descent:

Uni Variate

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$\theta := \theta - \alpha \frac{1}{m} \sum_{i=1}^{m} [(h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}]$$

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - \vec{y})$$

Week 3 - Overfitting

Regularization helps with overfitting by making theta smaller.

Gradient Descent:

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]$$

Cost Function :

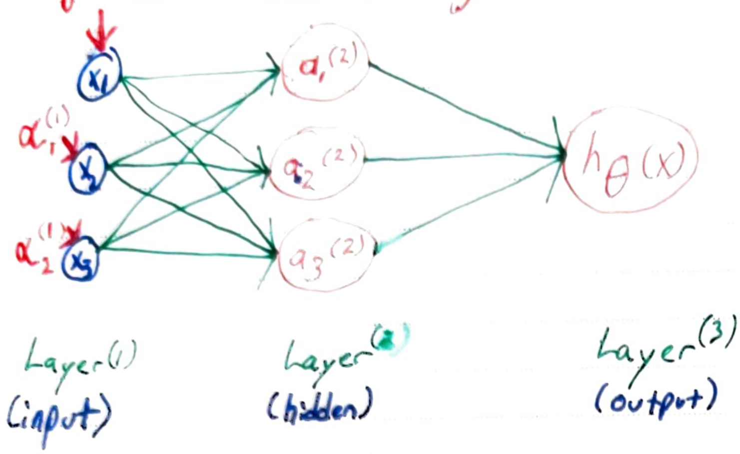
$$J(\theta) = \left[-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

We can find performance issues ... ie. low cost but not the <u>lowest</u> cost through <u>gradient checking</u>

<u>Gradient Checking</u>:

Week 4 - Neural Networks

$a_0^{(1)} = x$ * can be thaght of as the 1st activation layer



$a_i^{(j)}$ = activation of unit i in layer j

θ_j = matrix of weights controlling function mapping from layer j to $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

Vectorized Approach

Previously $g(z) = g(\Theta^T x)$ therefore:

now
$$z_1^{(2)} = \Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3$$

$$z_2^{(2)} = \Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3$$

$$z_3^{(2)} = \Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3$$

$z^{(2)} = \Theta^{(1)} x$ or $z^{(2)} = \Theta^{(1)} a^{(1)}$

$a^{(2)} = g(z^{(2)})$ ← elementwise sigmoid of z

*we also need to add a bias - $a_0^{(2)}$ *

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$\boxed{h_\Theta(x) = a^{(3)} = g(z^3)}$$

Week 5 - Neural Network Backpropagation

Cost Function

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log((h_\theta(x^{(i)}))_k)+(1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right]+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\theta_{j,i}^{(l)})^2$$

*not ith training example

L = total # of layers
S_l = # of units (not counting bias) in layer l
K = # of output units / classes
j = node in layer l

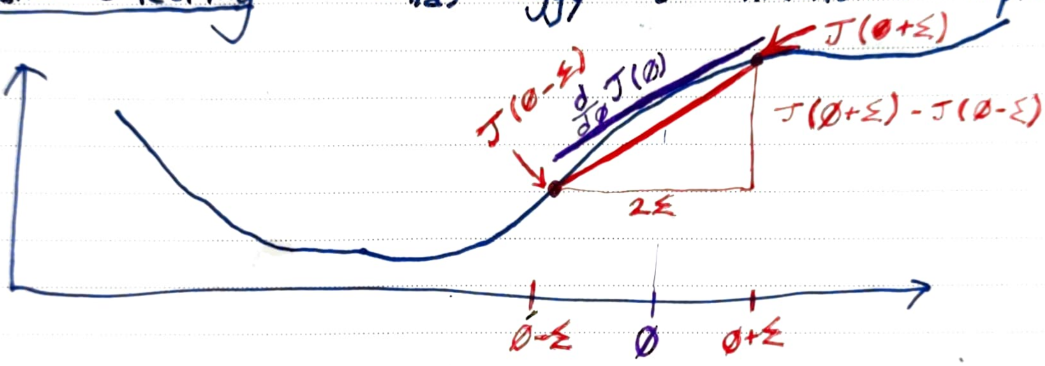
To calculate error for any given node:

$$\boxed{g'(z^{(3)}) = a^{(3)} .* (1-a^{(3)})}$$

$$\delta_j^{(l)} = a_j^{(4)} - y_j \quad \text{or} \quad \underbrace{\delta^{(4)} = a^{(4)} - y}_{\text{\# of output units}} \rightarrow \delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$\boxed{\text{NO } \delta^{(1)}}$

Gradient Checking: finds "buggy" or inefficient implementation



QED: $\frac{d}{d\theta} J\theta \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$ $\quad \varepsilon \approx 10^{-4}$

Vectorized: $\frac{d}{d\theta_1} J(\theta) \approx \frac{J(\theta_1+\varepsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1-\varepsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\varepsilon}$

$\frac{d}{d\theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2+\varepsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2-\varepsilon, \theta_3, \dots, \theta_n)}{2\varepsilon}$

$\vdots$

$\frac{d}{d\theta_n}$

continued →

Week 5 - Neural Network Backpropagation

Gradient Checking Implementation:

```
theta is a vector (unrolled)
 for i = (1:n);
    thetaPlus = theta
    thetaPlus(i) += Epsillon
    thetaMinus = theta
    thetaMinus(i) -= Epsillon
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus)) / (2*Epsillon)
end;
```

1. Implement backprop to compute DVec (unrolled deltas)
2. Implement Gradient Checking to compute gradApprox
3. Turn off gradient checking before training

Random Initialization:

How can we initialize theta?

* We cannot initialize all thetas to zero or all neurons will be the same.

aka Symmetry Breaking

Set each θ to a random value between $-\varepsilon$ and ε

10 x 11 matrix between 0 and 1

theta1 = rand(10, 11) * (2* epsilon) - epsilon*

* not the same as epsilon in gradient checking

Week 5 - Overview for Neural Networks

1. Network Architecture

   - input
   - output
   - hidden (usually 1 layer or same # of units per layer)
     - more units the better (usually more than # of inputs)

2. Training

   1. Randomly initialize weights for θ
   2. Implement foward propagation to get $h_\theta(x^{(i)})$ or a
   3. Implement cost function to get $J(\theta)$
   4. Implement backward propagation to get $\frac{\partial}{\partial\theta_{jk}^{(i)}} J(\theta)$

      for $i = 1:m$

      forward prop (); returns a (activations)

      backward prop (); returns $\frac{\partial}{\partial\theta_{jk}^{(i)}} J(\theta)$ (derivative)
   5. Use gradient checking to verify backprop, then disable it.
   6. Use a optimization function like gradient descent to minimize $J(\theta)$

English Translations

1. Where do we start on the hill?
2. What is our guess? hypothesis?
3. How wrong was our guess?
4. Which direction to the bottom of the hill? (compass)
5. Is our compass accurate?
6. Use all the info above to make better guesses... each iteration.

$$\frac{\partial}{\partial\theta_{jk}^{(i)}} J\theta = a_j^{(l)} \delta_i^{(l+1)} *$$

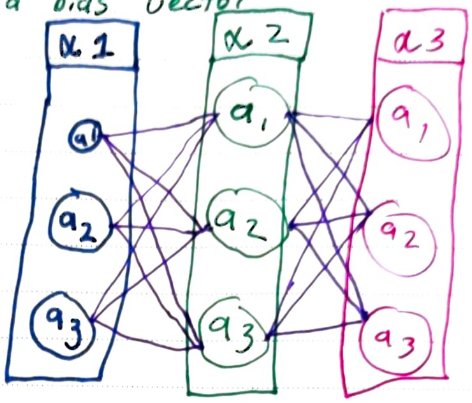
ignoring λ / if $\lambda = 0$

Week 5 - Mechanical Implementation of Backprop

Forward Prop:

y_bin = eye(num_labels)[y.T] # create an identity matrix for y

bias = np.ones((m, 1)) # create a bias vector
noBias1 = np.delete(Theta1, 0, 1)
noBias2 = np.delete(Theta2, 0, 2)



a1 = concat([bias, X], axis=1)
z2 = a1 @ Theta1.T
⋮ # work forward

Cost Function:

J = (-1/m) * np.sum(y_bin * log(h) + (1 - y_bin) * log(1 - h))

Regularization:

reg = (lambda_/(2*m)) * (np.sum(noBias1**2) + np.sum(noBias2**2))

J += reg

Backprop:

d3 = a3 - y_bin
d2 = d3 @ noBias2 * sigmoidgradient(z2)
d1 = "no such thing as error for layer 1"

Delta1 = d2.T @ a1 # use Delta1 (Δ) as an "accumulator" for our deltas (δ)
Delta2 = d3.T @ a2

Theta1_grad = (1/m) * Delta1 # Adjust our respective theta
Theta2_grad = (1/m) * Delta2
# Add our regularization term
Theta1_grad[:, 1:] = Theta1_grad[:, 1:] + 1 * lambda_/m * Theta1[:, 1:]
Theta2_grad[:, 1:] = Theta2_grad[:, 1:] + 1 * lambda_/m * Theta1[:, 1:]

grad = np.concatenate([Theta1_grad.ravel(), Theta2_grad.ravel()])

Week 6 - ML Diagnostics

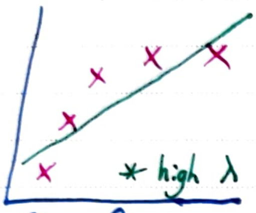
* Don't randomly adjust parameters or gather more data! *

Training Set: 60%
Cross Validation Set: 20%
Test Set: 20%

Plot! Plot! Plot!

1. Optimize Θ w/ the training set for each polynomial
2. Find the polynomial with the lowest cost using the CV set.
3. Estimate the general error w/ the test set using the best polynomial

* The most common problems will be under or over-fitting *



$\theta_0 + \theta_1 x$

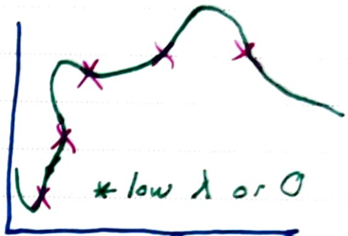
"underfit"

d = 1

$\theta_0 + \theta_1 x + \theta_2 x^2$

"Goldilocks"

d = 2



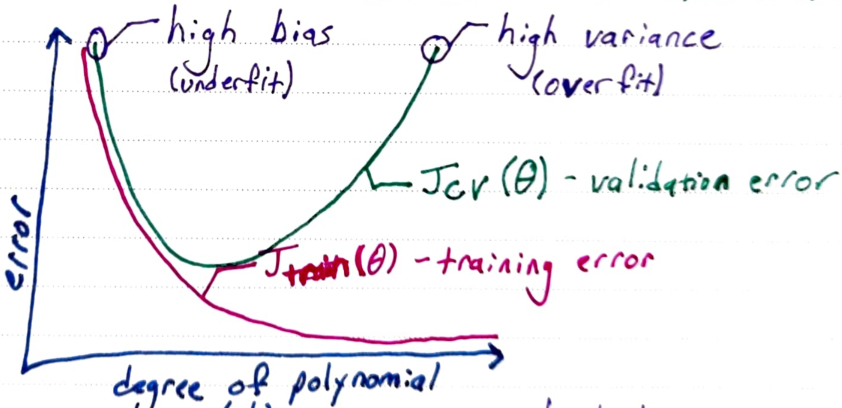
$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

"overfit"

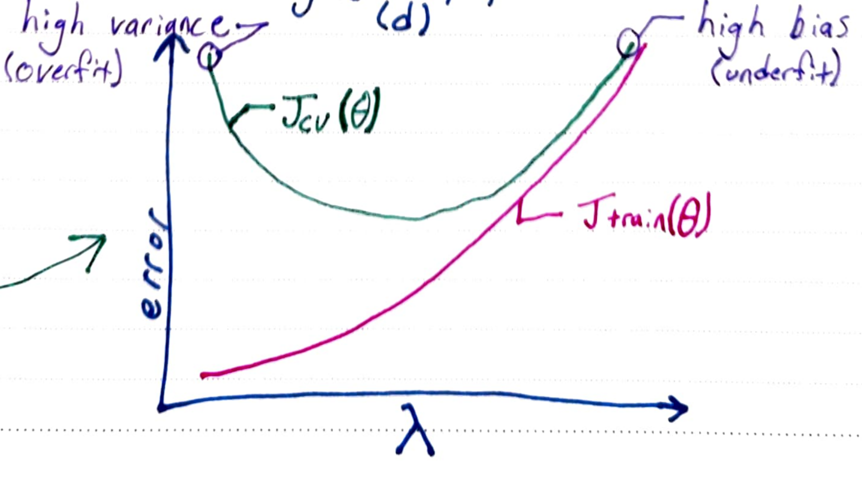
d = 4

* By plotting our error for Train and CV we can see how d affects J *

how does the # of polynomials affect our cost?



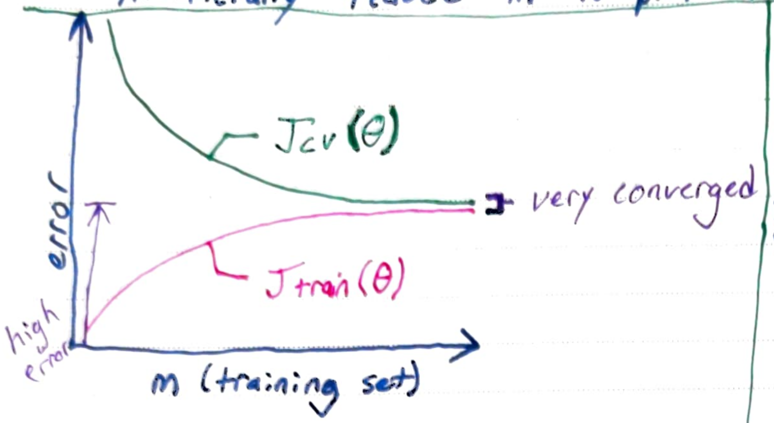
how does λ affect our cost?

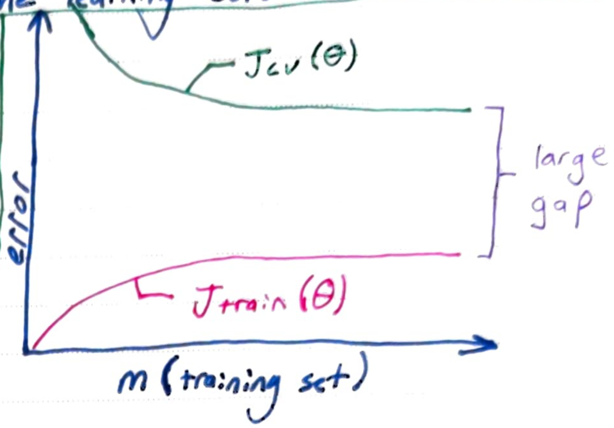


Week 6 - ML Diagnostics

* Learning curves can help diagnose under or over-fitting *

- Artificially reduce m to plot the learning curves





High Bias

- more training data won't help
- lower λ may help
- more polynomials may help
- more features may help

High Variance

- more training data may help
- higher λ may help
- less polynomials may help
- less features may help

Implementation of learning curves

```
for i in range(1, m+1):
    subX = X[:i :]
    subY = y[:i]

    theta = utils.trainLinearReg(linearRegCostFunction, subX, subY, lambda_)
    error_train[i-1], _ = linearRegCostFunction(subX, subY, theta, lambda_=0)
    error_val[i-1], _ = linearRegCostFunction(Xval, yval, theta, lambd_=0)
return error_train, error_val
```

Week 6 - Building a spam classifier

* It is difficult to determine which features to spend time on *

Error Analysis : A systematic approach to improving an algorithm

1. Start simple : Create a model in > 24 hours & test it on CV
2. Plot learning curves: decide on more data, more features, etc
3. Error Analysis : Manually review examples where the model was wrong
   * look for error trends / patterns

Error Analysis in practice:

1. Categorize the reviewed errors and count them
   ie. 12 : Pharma , 4 : Replica / fake, 64 : phishing, 3 other, etc.
   *how can we improve detection here
2. What features would have helped these errors?

* Our goal is to quickly find out which scenarios are the most difficult * to predict. This will allow us to focus our efforts on that.

3. Numerical Evaluation : implement Accuracy % on your CV [redacted]
   * this allows you to test features that can't be evaluated by * using error analysis. ie. natural language stemming

Skewed Classes : When your ratio of classes are skewed accuracy won't help
ie. 99% accuracy, but 0.5% of y=1 | 99.5% of y=0

Precision / Recall : A better way to measure accuracy.

Predicted ($h_\theta(x)$) \ Actual (y)	1	0
1	True Positive	False positive
0	False Negative	True Negative

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False positives}}$$

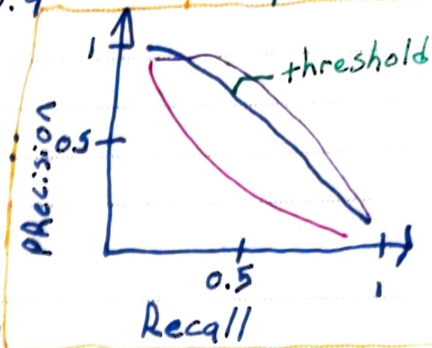
$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Week 6 - Prescision / Recall (continued)

* We can vary our threshold for predicting y = <some class> *
ie.

- if we want $y=1$ (cancer) only when really confident:
  - $y=1$ if $h_\theta(x) \geq 0.7$ or even 0.9
    → high precision, low recall

examples



- if we want to avoid missing positive sases:
  - $y=1$ if $h_\theta(x) \geq 0.3$
    → high recall, low precision

F_1 Score : a better way to measure precision/recall

F_1 Score $= \boxed{2\frac{PR}{P+R}}$

ie.

P	R	F_1
0.5	0.4	0.444
0.7	0.1	0.175
0.02	1.0	0.0392

← F_1 Score winner