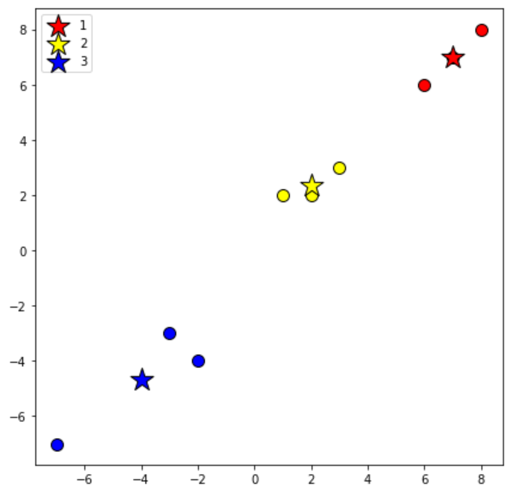
PATTERN RECOGNITION! S.Natchanon (MSc.)

HOMEWORK1 6470177521

PART: Hello Clustering

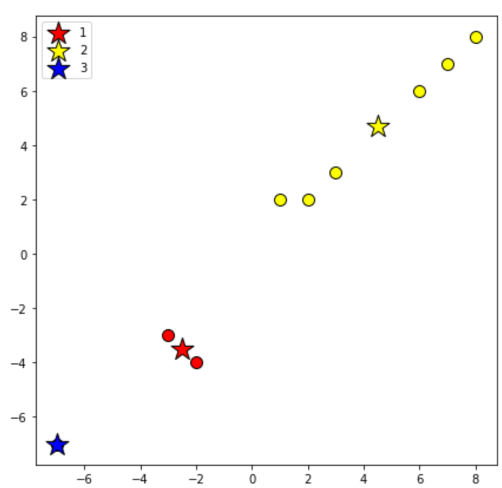
T1: What are the points assigned? What are the updated centroids?

- **Ans:** Coordinate of cluster centroids are (7, 7), (2, 2.33), (-4, -4.67) respectively. And each data point is assigned as following picture.



T2: What happens if we change starting point?

- **Ans**: Cluster centroid are changed, and each data point is assigned in different pattern compared to previous one.



T3: Which one do you think is better? How would you measure the "Goodness"?

- **Ans:** Starting point from T1. Is better than T2.

How?

1) Manually observe the pattern of data (if possible). We can evaluate if the outcome make sense.
2) The goodness can be determined by "Cost function". In the work, we are using the sum of squared distances: (minimize)

$$J(c_k) = \sum_{x_i \in c_k} ||x_i - \mu_k||^2$$

Which indicate how far each data point is from its assigned cluster centroid.

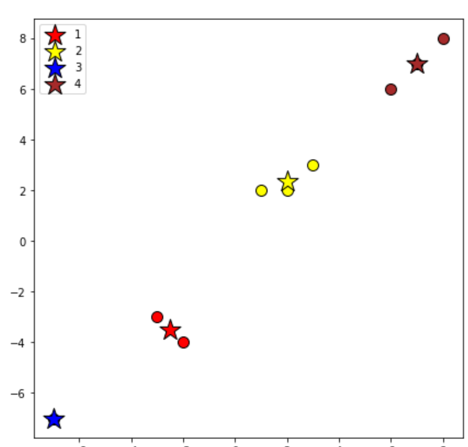
** lower is better **

T1 cost = 8.393945 (Detail in ipynb)

T2 cost = 9.765463 (Detail in ipynb)

OT1: What would be the best K for this question?

- **Ans**: The answer is K = 4. Reason: This dataset can be likely observed as 4 group in general because the first data point (the most left one) is too far to be group with next two points. And it got the lowest cost of squared distances.



Cost Function = 4.6329931618554525

PART: My heart will go on

T4: What is the median age of the training set?

- **Ans:** 28.0 (same for mode)

T5: What is the mode of "Embarked"?

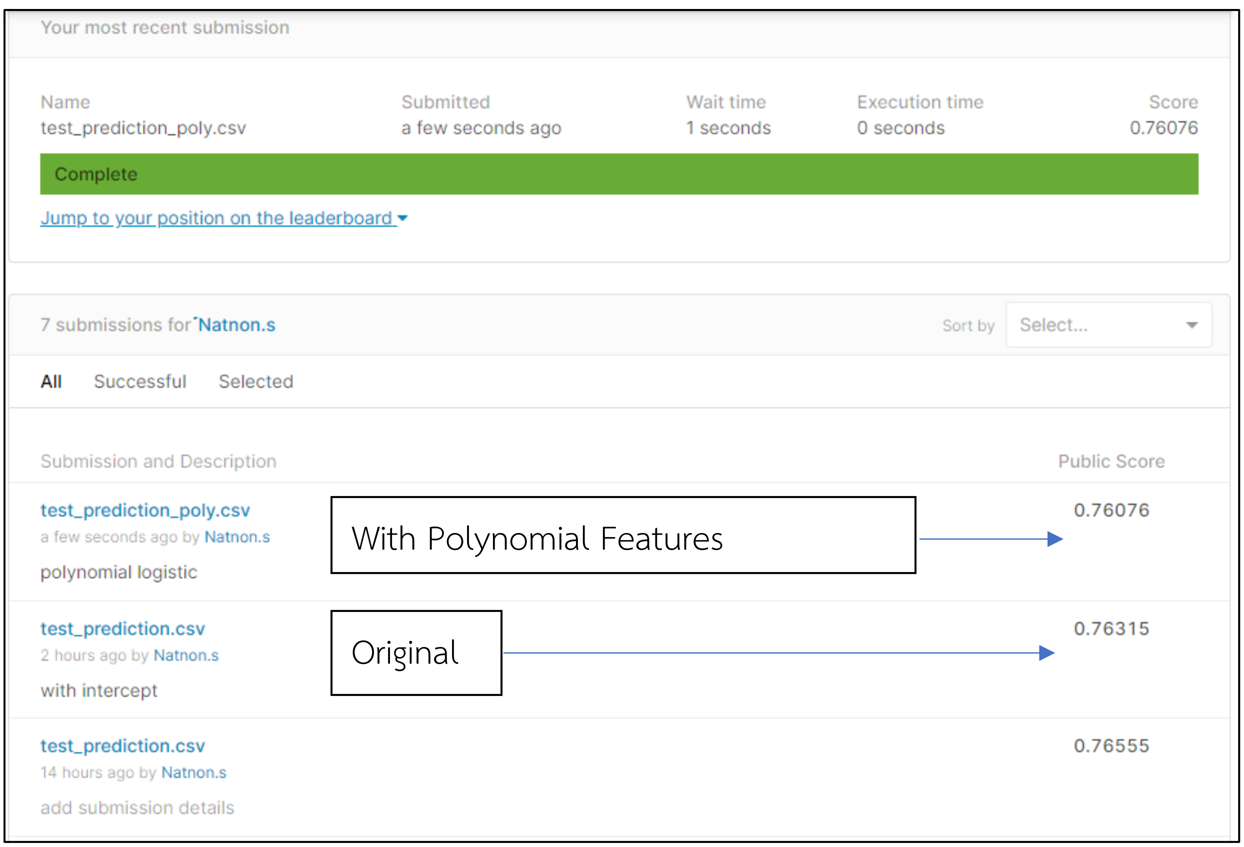
- **Ans:** "S" with 644 instances.

T6: Write a logistic regression using gradient descent?

- **Ans:** Code & ipynb file are provided in additional material.

T7: Submit Evaluation.

- **Ans:**



T8: How does it perform on higher order features?

- **Ans:** It converge faster than the original one but eventually there is no improvement of accuracy. And it gets slightly lower accuracy on test set.

(It’s a sign of “Overfitting” if training is improved but test is worsened)

Set up: Polynomial Features terms as follow:

[Pclass, Sex, Age, Embarked, Poly1, Poly2, . . .]

Poly1: ***Age***^{2}

Poly2: ***Age***^{3}

Poly3: ***Age***^{4}

Poly4: ***Age*Pclass***

Poly5: ***Age*Sex***

Poly6: ***Age*Embarked***

Poly	Original
Epoch: 10001 Logistic loss : 0.4492529271383494 Training accuracy = 78.79%	Epoch: 100001 Logistic loss : 0.45393831938173523 Training accuracy = 79.12%

Polynomial features loss is less than 0.45 since Epoch: 10,000, while Non-Polynomial Epoch is 100,000

T9: What happens if we reduce the number of features?

- **Ans:** It perform worsen than other. And its logistic loss likely to stuck greater than 0.5 (This is a sign of "Underfitting")

```
Epoch: 100001
Logistic loss :  0.5151997394638019
Training accuracy = 78.68%
```

OT2: Linear Regression?

- **Ans:** ipynb file are provided in additional material.

```
Epoch: 100001
Logistic loss :  0.07246764834112983
Training accuracy = 79.46%
```

OT3: Weight comparison from gradient descent and Normal equation?

- **Ans:** MSE = 0.00023402. (Almost identical)

<u>W_gradient</u>

```
array([[ 0.76183634],
       [-0.18509332],
       [ 0.49340374],
       [-0.00486948],
       [ 0.04927934]])
```

<u>W_Normal</u>

```
array([[ 0.77654442],
       [-0.18843944],
       [ 0.49086711],
       [-0.00505436],
       [ 0.04911346]])
```

MSE

```
array(0.00023402)
```

```python
def sigmoid(z):
    return 1/(1 + np.exp(-z))

# Compute forward path function
def random_init_param(X):
    #size of X is [m, n] where m=sample, n=features
    W = np.random.randn(len(X[0]), 1) # +1 for Bias term
    return W

def cost_function(X, Y, H, W):
    # X : training data
    # H : prediction (before sigmoid)
    # Y : training label
    # W : trainable parameters
    m = len(Y)
    devide_zeros_threshold = 1e-5 # Solve devide by zero problem
    #L2 loss
    L2loss = (np.dot((Y-sigmoid(H)).T, (Y-sigmoid(H))))/2 # /2 for derivative term
    # or Logistic loss
    Logistloss = np.dot(-Y.T, np.log(sigmoid(H)+devide_zeros_threshold))-np.dot((1-
Y).T, np.log((1-sigmoid(H))+devide_zeros_threshold))
    cost = (1/m)*Logistloss
    grad = (1/m)*(np.dot(X.T, (sigmoid(H)-Y)))
    return cost, grad

def main(X, Y, W):
    h = np.dot(X, W)
    cost, grad = cost_function(X, Y, h, W)
    #update parameters
    W = W - (LR)*grad
    return W, cost

########### Utility function ############
#round function uses threshold = 0.5
def predict(X, params):
    h = np.dot(X, params)
    return np.round(sigmoid(h))

def accuracy(pred, y):
    return np.squeeze(np.squeeze((sum(Y == pred)/len(X))*100))

def add_bias(X):
    Bias = np.ones((len(X), 1))
    res = np.concatenate((Bias, X), axis=1)
    return res
```

```python
################################## Training Function
#########################################


X = train_data
Y = train_label
# Add bias for training sample
X = add_bias(X)
W = random_init_param(X)
W_lowest = np.zeros((len(X[0]), 1))

LR = 0.001
epoch = 100000
best_loss = 1e6


for i in range(epoch+1):
    W, cost = main(X, Y, W)
    pred = predict(X, W)
    if cost < best_loss:
        W_lowest = W
    if (i%1000 == 0): # Just for logging
        print("Epoch:", i+1)
        acc = accuracy(pred, Y)
        print("Logistic loss : ", np.squeeze(cost))
        print("Training accuracy = {:.2f}%\n".format(acc))
```