## Assignment 4: Markov Decision Processes, Reinforcement Learning, and Classification

##### CS 440 Fall 2015 3 Credits

|  |  |  |
| --- | --- | --- |
| **Name** | **Email** | **Contributions** |
| Jaehwan “JJ” Park | park155@illinois.edu | part 1, report |
| Ruifan Li | li158@illinois.edu | part 2, report |

### 

### 1. MDPs and Reinforcements learning

### Part 1.1: Grid World MDP

### **Implementation**

We chose value iteration for part 1.1. First of all, we created a State class and assigned x, y, reward, utility, isWall boolean, isTerminal boolean, number of steps taken, etc. Then when the main function is called, we set up the grid environment same as the one provided in the mp document.

For terminal states, as suggested in the mp document, we set the number of iterations to 25 (<50) since it really gives a close value to the reference utility value, but for non-terminal states, it was impossible to find convergence with that number. Therefore, instead of 25, 1100 iterations are run to find convergence in the non-terminal states.

And while iterating we applied Bellman equation to calculate the new utility. When we calculate the optimal policy / utility of the terminal states, we turn on the isTerminal boolean on to terminal states, and we did the opposite when we calculate those of the non-terminal states.

### 

### **Results**

#### 1. Terminal State

Table 1. Optimal policy of terminal state

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | T | R | R | R | D |
| D | D | D | W | T | D |
| R | R | D | W | D | T |
| R | R | D | W | R | U |
| U | U | R | R | U | U |
| T | T | U | W | T | T |

T – Terminal W – Wall L – Left U – Up D – Down R – Right

Table 2. Utility values of terminal state

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1.6663804523839938 | -1.0 | 1.8122075602386967 | 1.8358641737025434 | 1.9095493274109672 | 2.3478585151480855 |
| 2.0712235021399112 | 2.140515242644923 | 2.2100463651313094 | 0.0 | -1.0 | 2.4827968923418426 |
| 2.13921926939236 | 2.218017116312072 | 2.2971474700701404 | 0.0 | 2.743906705883558 | 3.0 |
| 2.1969625360068323 | 2.2905610521705486 | 2.386548226692007 | 0.0 | 2.7970453813316927 | 2.9000083160378867 |
| 2.1318794828939 | 2.23061931329826 | 2.4791848727524455 | 2.6293469432240304 | 2.7130508199438714 | 2.802884148113836 |
| 1.0 | -1.0 | 2.024988205927515 | 0.0 | -1.0 | -1.0 |

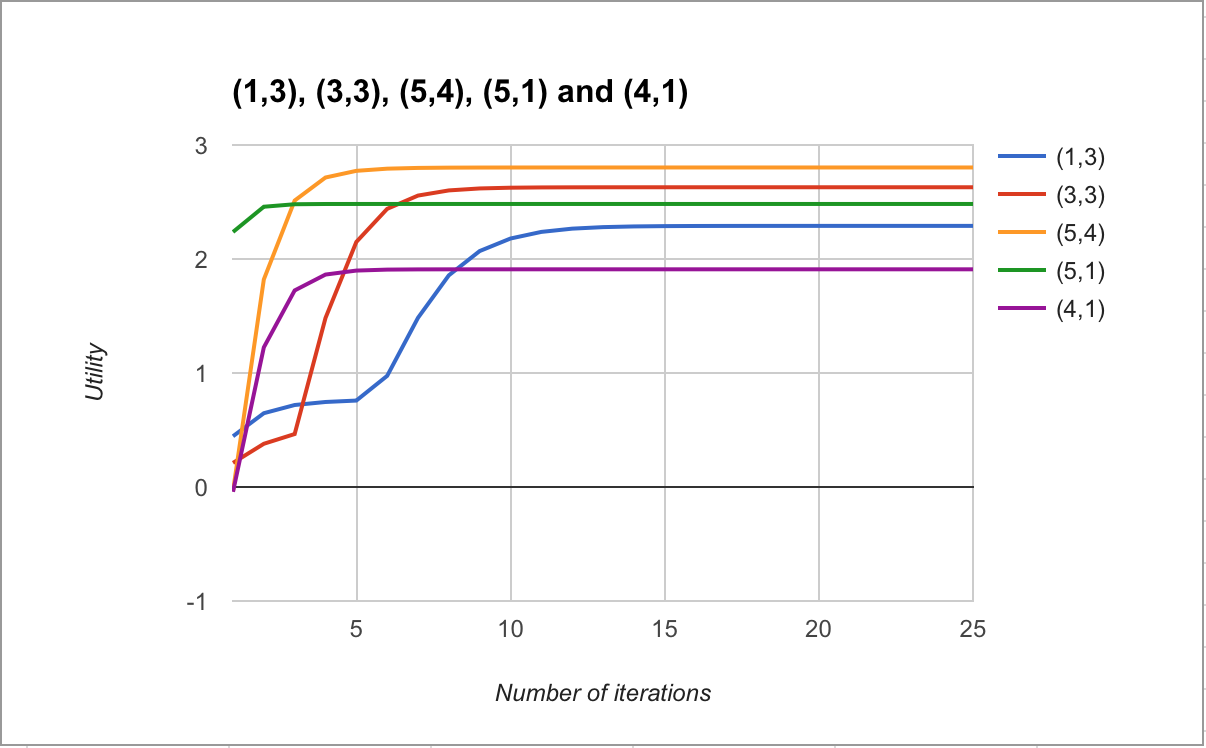


Figure 1. Utility estimate graph v. number of iterations

#### 2. Non-terminal State

Optimal policy:

Table 3. Optimal policy of non-terminal state

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| R | R | R | R | R | D |
| R | R | U | W | R | D |
| R | R | U | W | R | R |
| R | R | D | W | R | U |
| R | R | R | R | U | U |
| U | R | U | W | U | U |

W – Wall L – Left U – Up D – Down R – Right

Utility values:

Table 4. Utility values of non-terminal state

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 210.67753556741476 | 213.58230858759913 | 217.7583951724094 | 220.94720497192614 | 223.7874525460688 | 226.41805902128206 |
| 209.1239877689451 | 211.84863333600606 | 214.64779138449148 | 0.0 | 225.75147086443417 | 229.6562042649398 |
| 207.3009878187002 | 209.52462275795506 | 211.65814071615927 | 0.0 | 229.3764038375247 | 233.09450105200062 |
| 209.032479402365 | 211.7006490840825 | 214.40310615105176 | 0.0 | 226.8299704976011 | 229.77470390404537 |
| 210.93708299049447 | 214.04500036752026 | 217.4985287772971 | 221.08165189066804 | 223.9235971469972 | 226.53717859003885 |
| 209.68920157850695 | 210.78590941882968 | 214.30259278990937 | 0.0 | 220.1401627839023 | 222.2101197046723 |

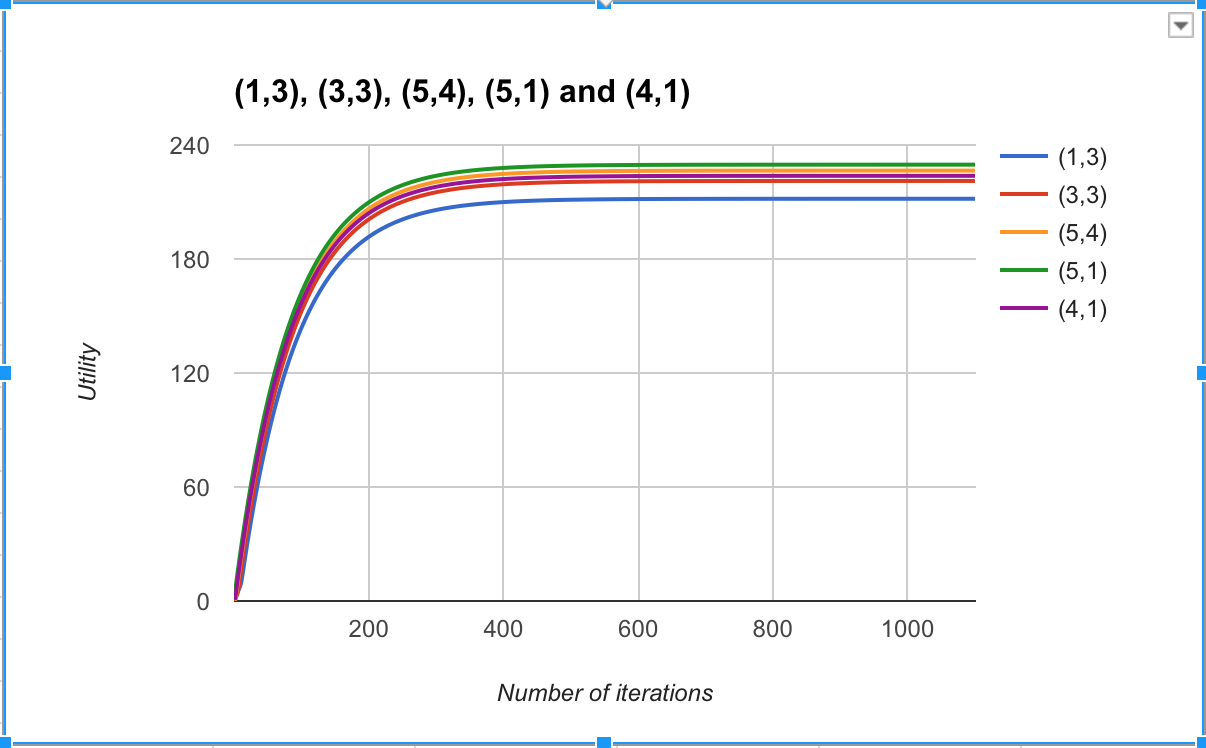


Figure 2. Utility estimate graph v. number of iterations

### Part 1.2: Grid World Reinforcement Learning

**Implementation**

The basic environment is same as what we’ve done in part 1.1. For TD Q-learning, we used the same equation as the one provided from lecture 21:

*Qnew*(*s*,*a*)=*Q*(*s*,*a*)+α(*R*(*s*)+γ max*a*' *Q*(*s*',*a*')−*Q*(*s*,*a*))

Where *Q*(*s*,*a*) is the Q of current state, α is α(t) = 60/(59+t), *R*(*s*) is the reward of current state, γ is discount factor, max*a*' *Q*(*s*',*a*') is the max number of successor state.

For α, we followed the spec from mp document. And for exploration function, we gave the tendency to generate a higher utility to unexplored grid based on the number of steps taken. So, less the number of step, more likely the program chooses the unexplored grid.

**Results:**

Utility estimates:

Table 5. Utility Estimates

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1.5201144280936234 | -1.0 | 2.6083692364441142 | 2.7037039162009524 | 2.629836604540901 | 2.670274004108638 |
| 2.130166096804794 | 2.205468726505333 | 2.503843982416898 | 0.0 | -1.0 | 2.7835419605312266 |
| 2.19756494273209 | 2.2686081918653 | 2.3679170151219306 | 0.0 | 2.7468749999999997 | 3.0 |
| 2.2502482230961074 | 2.3135070322861795 | 2.36945315058047 | 0.0 | 2.7944325618470245 | 2.9263051702395964 |
| 2.311537926567163 | 2.3807473334940332 | 2.471103222870406 | 2.592237014494566 | 2.6939963165945144 | 2.7670303681805963 |
| 1.0 | -1.0 | 2.3232982278623675 | 0.0 | -1.0 | -1.0 |

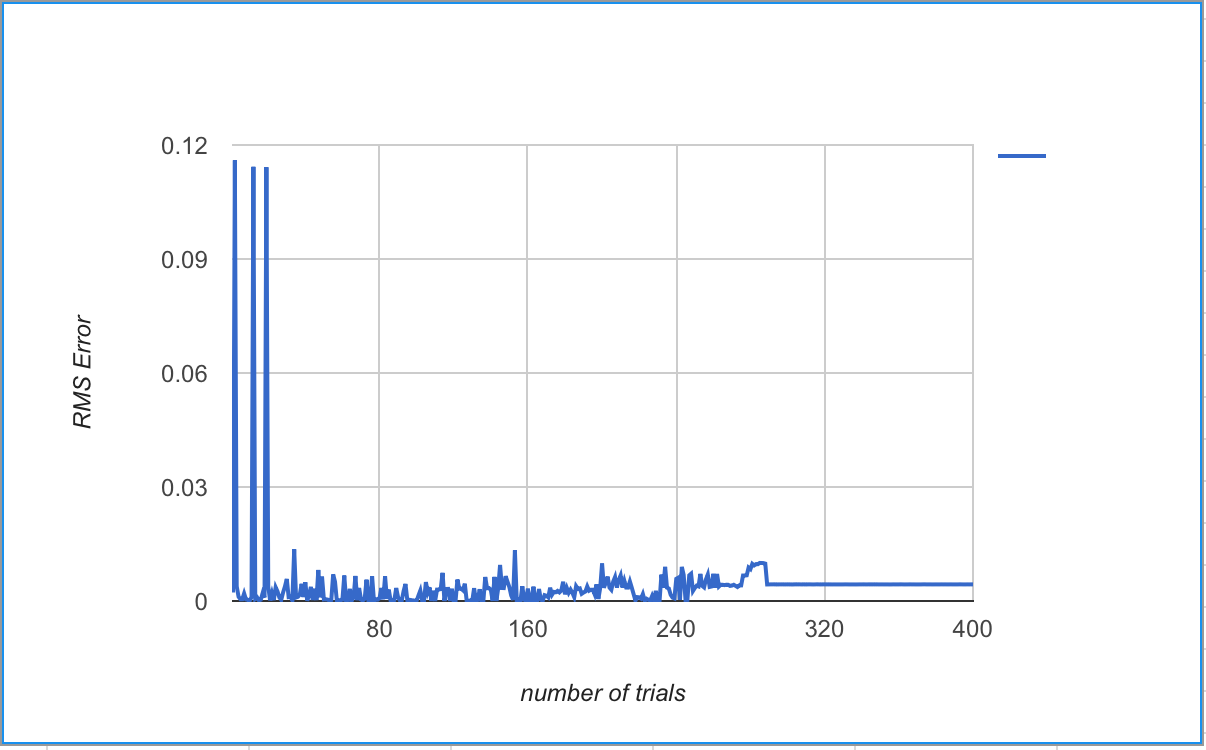


Figure 3. RMS Error v. function of number of trials

### Part 2.1: Digit Classification with Perceptrons

**Implementation**

In this part, the digit identification problem is solved by applying multi-class perceptrons. 10 weight factor vectors of size 28x28 for each class are tuned during training and applied to identify images during testing.

Algorithm outline:

Training

* Reading file
* Processing file
* Epoch loop (t)

Sample loop (5000 times)

Identification loop (10 times)

Go through all 10 class, and pick the best fit for this sample:

Return identification result c’

End identification loop

Compare with true result c (test sample)

If c’ != c:

Wc = Wc + α\*x

Wc’ = Wc’ - α\*x

bc = bc + α\*x

bc’ = bc’ - α\*x

End sample loop

t++

Shuffle the order of the training sample if needed

End epoch loop

* Build training curve

Testing

* Reading file
* Get next testing image x
* Calculate sgn(w\*x+b) for each class, and pick out the one, c with best value
* Compare c with true label of the image
* Calculate overall accuracy, build confusion matrix and start comparison with mp3 based on the result

Parameter tuning:

First we tune parameters without random factors (i.e. Random sampling and w random initialization), then choose the ones with good result to add parameters involving random factors. The tuning result is shown below.

Table 6. Parameters Tuned

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Trail | α | Bias? | W initialization | Sample Ordering | # epoch | Result(Rate) |
| 1 | 1 | N | 0 | FIXED | 22 | 80.90% |
| 2 | 0.1/（0.1+t） | N | 0 | FIXED | 10 | 84.10% |
| 3 | 10/（10+t） | N | 0 | FIXED | any | 79.00% |
| 4 | 1 | Y | 0 | FIXED | 22 | 81.90% |
| 5 | 0.08/（0.08+t） | Y | 0 | FIXED | 10 | 84.40% |
| 6 | 0.1/（0.1+t） | Y | 0 | FIXED | 10 | 84.20% |
| 7 | 0.4/（0.4+t） | Y | 0 | FIXED | 4 | 83.70% |
| 8 | 0.5/（0.5+t） | Y | 0 | FIXED | 12 | 82.60% |
| 9 | 10/（10+t） | Y | 0 | FIXED | any | 78.10% |
| 10 | 1 | Y | 0 | RANDOM | 22 | 82.70% |
| 11 | 0.08/（0.08+t） | Y | 0 | RANDOM | 10 | 84.10% |
| 12 | 0.1/（0.1+t） | Y | 0 | RANDOM | 10 | 84.40% |
| 13 | 0.5/（0.5+t） | Y | 0 | RANDOM | 12 | 83.10% |
| 14 | 1/（1+t） | Y | 0 | RANDOM | any | 78.10% |
| 15 | 10/（10+t） | Y | 0 | RANDOM | any | 78.10% |
| 16 | 0.1/（0.1+t） | Y | [-1,1] | RANDOM | 10 | 84.90% |
| 17 | 0.1/（0.1+t） | Y | [-2,2] | RANDOM | 10 | 85.10% |
| 18 | 0.1/（0.1+t） | Y | [-2,2] | RANDOM | 30 | 85.60% |

Best result of several trials

Parameter used in result part

From the result table above, it can be seen that all five factor have effect on the classification rate. Adding bias term and shuffling the order of training samples in most cases can increase the identification rate. Random initialization of weight factors can significantly improve the accuracy, while number of epoch has obvious effect especially when the training sample order is randomized.

**Results**

The overall accuracy is about **85.6%.**

Accuracy vs. epoch

Table 7. Training Curve Table

|  |  |
| --- | --- |
| epoch | Accuracy |
| 0 | 0.7804 |
| 1 | 0.8998 |
| 2 | 0.9178 |
| 3 | 0.9206 |
| 4 | 0.9268 |
| 5 | 0.9272 |
| 6 | 0.9298 |
| 7 | 0.929 |
| 8 | 0.9306 |
| 9 | 0.9312 |
| 10 | 0.9338 |
| 11 | 0.935 |
| 12 | 0.9348 |
| 13 | 0.937 |
| 14 | 0.9336 |
| 15 | 0.9368 |
| 16 | 0.936 |
| 17 | 0.9348 |
| 18 | 0.9372 |
| 19 | 0.9376 |
| 20 | 0.9376 |
| 21 | 0.9376 |
| 22 | 0.9374 |
| 23 | 0.9376 |
| 24 | 0.9364 |
| 25 | 0.9384 |
| 26 | 0.9378 |
| 27 | 0.9374 |
| 28 | 0.9394 |
| 29 | 0.9376 |

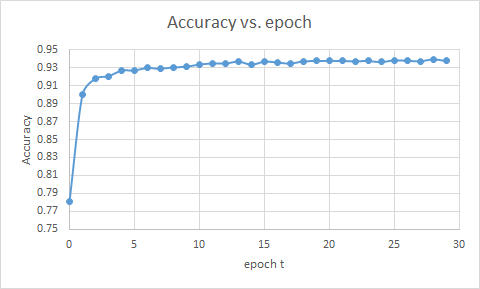
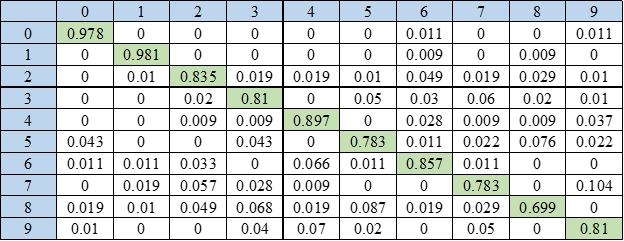


Figure 4. Training Curve

Confusiong Matrix:

Table 8. Confusion Matrix of Multi-Class Perceptron Classifier



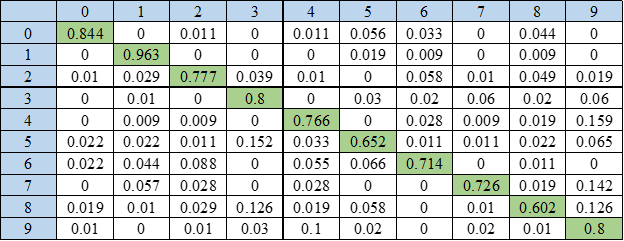
Comparison with Naice Bayes Model:

|  |  |
| --- | --- |
| Model | Accuracy |
| Multi-perceptron | 0.856 |
| Naive-Bayes | 0.766 |

The overall accuracy, by using a multi-class perceptron classifier, is increased by 9% compared to that of a Naive Bayes model.

Recall the confusion matrix of Naïve Bayes Model below

Table 9. Confusion Matrix of Naïve Bayes Classifier



Comparing two confusion matrices, one can conclude that the two classifiers share the same tendency of performance on each digit, like the digit with lowest accuracy is 8, and misclassifying 8 as 3 happens most for both classifiers. Multi-class perceptron works much better when dealing with digits with confusing figures. For example, the classification rate of digit 5 in Naive Bayes model is 65.2%, suggesting its confusing nature, while the multi-class perceptron increase this accuracy over 15% to 78.3%. As a result, the multi-class perceptron performs better than the Naive Bayes model. One reason for this can be that the former one goes through training samples many times to improve the classifier while the latter only uses these samples once.

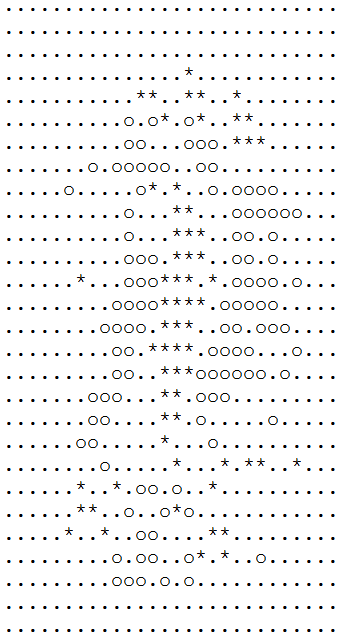
**Extra Credit:**

Visualization of Weight Factors

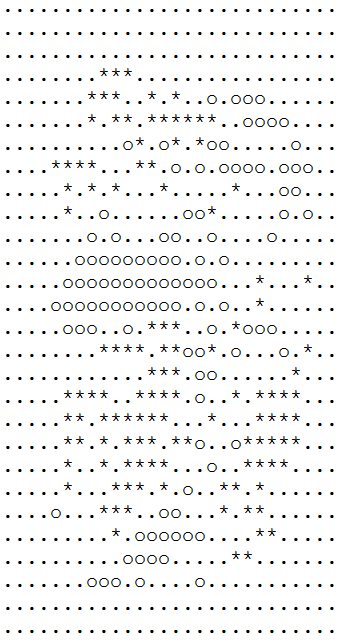
**To visualize weight factors, We visually print out the weight factor: [ \* ] for elements with l value larger than 3; [ . ] for [-3 , 3] ; [ o ] for less than -3. The results are shown below.**



Weight Factor of 0



Weight Factor of 1



Weight Factor of 2



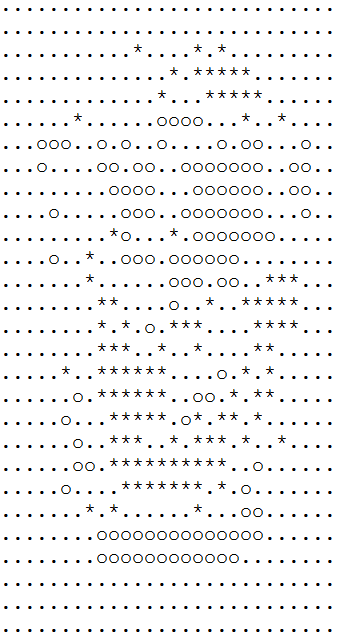
Weight Factor of 3



Weight Factor of 4



Weight Factor of 5



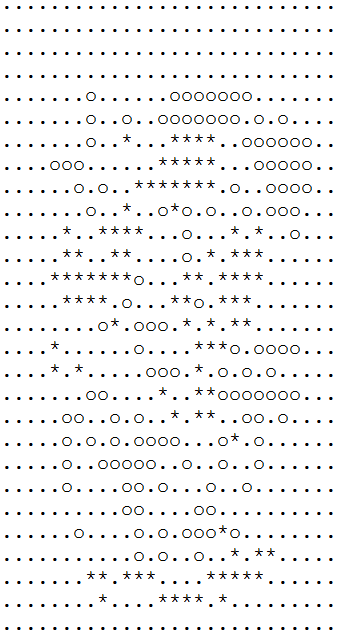
Weight Factor of 6



Weight Factor of 7



Weight Factor of 8



Weight Factor of 9

From the results above, it can be seen that, unlike Naive Bayes model who focuses on the feature of the digit, the multi-class perceptron tries to catch the special feature of each digit with positive emphasize (value large than 0) and the shared features with other digits with negative confusion rejection (value less than 0). By doing this, the classifier is more likely to distinguish the special characteristic of given digit while reduce confusions by paying less attention to common features. As a result, it works better than Naive Bayes model classifier, which does not avoid common features among digits.