1.

I.

Suppose we have a prediction $\hat{y}$ here. We have two choices, one is to output the prediction $\hat{y}$, the other is to output 'unsure'.

The expected penalty for output prediction $\hat{y}$ is

$$\mathrm{P}_{prediction} = P(Y = \hat{y}|X = x) \cdot 0 + \big(1 - P(Y = \hat{y}|X = x)\big) \cdot \lambda_s = \big(1 - P(Y = \hat{y}|X = x)\big) \cdot \lambda_s$$

And the expected penalty for output 'unsure' is

$$\mathrm{P}_{unsure} = \lambda_u$$

Thus the minimum penalty we could yield is $\min[\mathrm{P}_{prediction}, \mathrm{P}_{unsure}]$.

$$Penalty = \begin{cases} \big(1 - P(Y = \hat{y}|X = x)\big) \cdot \lambda_s, & when\ P(Y = \hat{y}|X = x) \geq \dfrac{\lambda_s - \lambda_u}{\lambda_s} \\ \lambda_u, & otherwise \end{cases}$$

II.

If $\lambda_u = 0$, as given above, the classifier will always output 'unsure' so that it could achieve zero penalty.

If $\lambda_u \geq \lambda_s$, the classifier will always output the prediction $\hat{y}$, because $\frac{\lambda_s - \lambda_u}{\lambda_s} \leq 0$ and $P(Y = \hat{y}|X = x) \geq 0$.

2.

I.

$$L(x, \lambda) = ||x - x_a||^2 + \lambda(\omega x + \omega_0)$$

Let $\frac{\partial L}{\partial x} = 2(x - x_a) + \lambda\omega = 0$, then we have

$$\lambda = \frac{2(x_a - x)}{\omega}$$

Let $\frac{\partial L}{\partial x} = \omega x + \omega_0 = 0$, then we have

$$x = -\frac{\omega_0}{\omega}$$

The origin Lagrange equation becomes

$$L(x, \lambda) = || - \frac{\omega_0}{\omega} - x_a||^2 + \frac{2(x_a - x)}{\omega}(-\omega\frac{\omega_0}{\omega} + \omega_0)$$

$$L(x, \lambda) = ||\frac{\omega_0 + \omega x_a}{\omega}||^2 = \frac{||g(x_a)||^2}{||\omega||^2} = \frac{||g(x_a)||}{||\omega||}$$

II.

A problem is a convex optimization if it satisfies for any x, x', $f(\lambda x + (1-\lambda)x') \leq \lambda f(x) + (1-\lambda)f(x')$

Now that the function is given,

$$f(\lambda x + (1-\lambda)x') = ||\lambda x + (1-\lambda)x'||$$

$$\leq \|\lambda x\| + \|(1-\lambda)x'\| = \lambda\|x\| + (1-\lambda)\|x'\| = \lambda f(x) + (1-\lambda)f(x')$$

Thus, this optimization problem is a convex optimization.

III.

The constraint of this problem could be rewrote as $5 - \sum x_i \leq 0$

The Lagrange function is $L(x, \lambda) = \big||x|\big| + \lambda(5 - \sum x_i)$

$$p^* \coloneqq \min_x \max_{\lambda \geq 0} L(x, \lambda)$$

The dual is

$$d^* \coloneqq \max_{\lambda \geq 0} \min_{x'} L(x', \lambda)$$

IV.

The function is already proven to be a convex function. Now if the constraint is also a convex set, the strong duality will hold.

To prove the constraint is a convex set, we need to prove for any $x, x' \in S$ and any $\beta \in [0, 1]$

$$\beta x + (1-\beta)x' \in S$$

In this problem, the constraint is $\sum x_i \geq 5$, arbitrarily choose two point x, x' in this set and an arbitrary $\beta \in [0, 1]$

$$x'' = \beta x + (1-\beta)x'$$

$$\sum x''_i = \sum \beta x_i + (1-\beta)x'_i = \beta \sum x_i + (1-\beta) \sum x_i'$$

$$\geq 5\beta + 5(1-\beta) = 5$$

x'' is also in the constraint set. Thus the constraint set is a convex set, the strong duality holds.

3.

I.

For any $0 < \sigma < \frac{\min(|x_i - x_j|)}{2}$ $(i, j \in N[1, n] \; and \; i \neq j)$, the mapping result can be linearly separated. (As n points are distinct, we could say that $\frac{\min(|x_i - x_j|)}{2} > 0$)

To prove it, we simply need to prove that there exists a $w(\alpha)$ so that for all x_i $(0 < i \leq n)$ we have

$$\int_{-\infty}^{+\infty} w(\alpha) \max\{0, 1 - \left|\frac{\alpha - x_i}{\sigma}\right|\} d\alpha = y_i$$

First, let's prove that for all intervals $[x_i - \sigma, \; x_i + \sigma]$, there is no overlap between each other.

As defined above, $0 < \sigma < \frac{\min(|x_i - x_j|)}{2}$, so we have $2\sigma < \min(|x_i - x_j|)$. Thus, apparently there is no overlap between adjacent intervals.

Then, we could find there exists $w(\alpha) = \begin{cases} \frac{y_i}{\sigma}, & \alpha \in [x_i - \sigma,\ x_i + \sigma] \\ 0, & others \end{cases}$ $(i \in N[0, n])$ such that the equation above will hold for all x_i.

For any x_i $(0 < i \leq n)$, we have

$$\int_{-\infty}^{+\infty} w(\alpha)\max\{0, 1 - \left|\frac{\alpha - x_i}{\sigma}\right|\}d\alpha = \int_{x_i - \sigma}^{x_i + \sigma} w(\alpha)(1 - \left|\frac{\alpha - x_i}{\sigma}\right|)d\alpha$$

$$= \int_{x_i - \sigma}^{x_i + \sigma} \frac{y_i}{\sigma}(1 - \left|\frac{\alpha - x_i}{\sigma}\right|)d\alpha = y_i$$

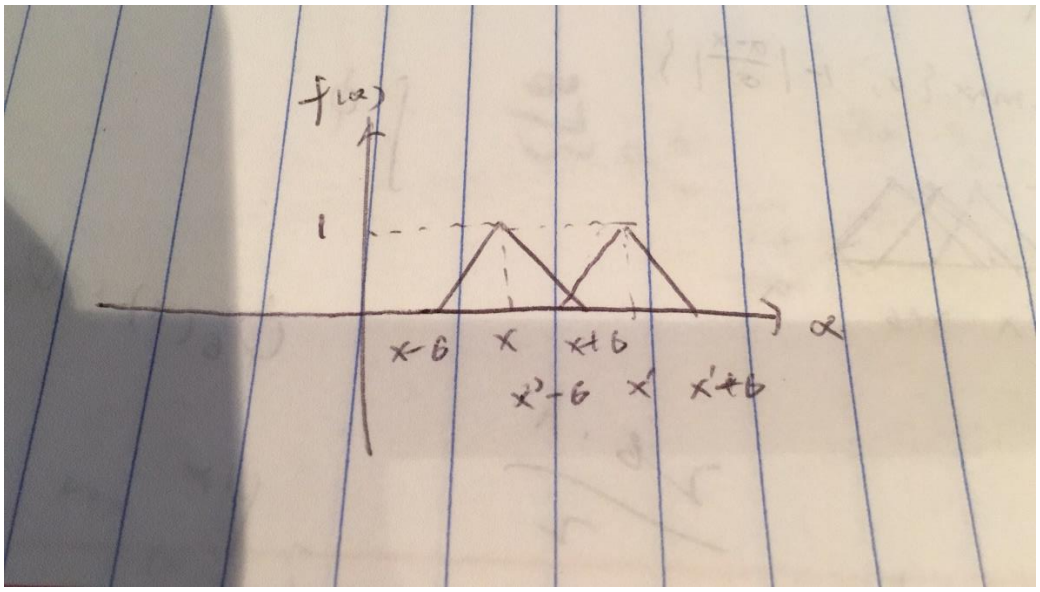
To conclude, for any distinct points set $(x_1, x_2 \ldots x_n)$, $0 < \sigma < \frac{\min(|x_i - x_j|)}{2}$ $(i, j \in N[1, n]\ and\ i \neq j)$ will ensure that the output of transformation is linearly separable.

II.

$$\Phi_\sigma(x) \cdot \Phi_\sigma(x') = \max\{0, 1 - \left|\frac{\alpha - x}{\sigma}\right|\}_{\alpha \in R} \cdot \max\{0, 1 - \left|\frac{\alpha - x'}{\sigma}\right|\}_{\alpha \in R}$$

$$= \int_{-\infty}^{+\infty} \max\{0, 1 - \left|\frac{\alpha - x}{\sigma}\right|\}\max\{0, 1 - \left|\frac{\alpha - x'}{\sigma}\right|\}d\alpha$$

Let's first take a look at the shape of the two functions $f_x(\alpha)\max\{0, 1 - \left|\frac{\alpha - x}{\sigma}\right|\}$ and $f_{x'}(\alpha)\max\{0, 1 - \left|\frac{\alpha - x'}{\sigma}\right|\}$



According to the graph, we could see that we could separate solution of $\Phi_\sigma(x) \cdot \Phi_\sigma(x')$ in three circumstances. Let

$$d = |x - x'|$$

a. $d \geq 2\sigma$

$$\Phi_\sigma(x) \cdot \Phi_\sigma(x') = \int_{-\infty}^{+\infty} \max\left\{0, 1 - \left|\frac{\alpha - x}{\sigma}\right|\right\}\max\left\{0, 1 - \left|\frac{\alpha - x'}{\sigma}\right|\right\}d\alpha$$

$$= \int_{\min[x,\,x']-\sigma}^{\min[x,\,x']+\sigma} \left(1 - \left|\frac{\alpha - \min[x,\,x']}{\sigma}\right|\right) \cdot 0 d\alpha + \int_{\max[x,\,x']-\sigma}^{\max[x,\,x']+\sigma} \left(1 - \left|\frac{\alpha - \max[x,\,x']}{\sigma}\right|\right) \cdot 0 d\alpha = 0$$

b. $\sigma < d < 2\sigma$

$$\Phi_\sigma(x) \cdot \Phi_\sigma(x') = \int_{-\infty}^{+\infty} \max\left\{0, 1 - \left|\frac{\alpha - x}{\sigma}\right|\right\} \max\left\{0, 1 - \left|\frac{\alpha - x'}{\sigma}\right|\right\} d\alpha$$

$$= \int_{\min[x,\,x']+d-\sigma}^{\min[x,\,x']+\sigma} \left(1 - \left|\frac{\alpha - \min[x,\,x']}{\sigma}\right|\right)\left(1 - \left|\frac{\alpha - \max[x,\,x']}{\sigma}\right|\right) d\alpha = \int_{d-\sigma}^{\sigma} \left(1 - \left|\frac{\alpha}{\sigma}\right|\right)\left(1 - \left|\frac{\alpha - d}{\sigma}\right|\right) d\alpha$$

$$= \int_{d-\sigma}^{\sigma} \left(1 - \frac{\alpha}{\sigma}\right)\left(1 + \frac{\alpha - d}{\sigma}\right) d\alpha = \int_{d-\sigma}^{\sigma} -\frac{\alpha^2}{\sigma^2} + \frac{\alpha d}{\sigma^2} + \left(1 - \frac{d}{\sigma}\right) d\alpha = \frac{2}{3}\sigma - \frac{d}{2} + \frac{(d-\sigma)^2}{6\sigma^2}(4\sigma - d)$$

$$= \frac{1}{6\sigma^2}(8\sigma^3 - 12\sigma^2 d + 6\sigma d^2 - d^3)$$

c. $0 \le d \le \sigma$

$$\Phi_\sigma(x) \cdot \Phi_\sigma(x') = \int_{-\infty}^{+\infty} \max\left\{0, 1 - \left|\frac{\alpha - x}{\sigma}\right|\right\} \max\left\{0, 1 - \left|\frac{\alpha - x'}{\sigma}\right|\right\} d\alpha$$

$$= \int_{\min[x,\,x']+d-\sigma}^{\min[x,\,x']+\sigma} \left(1 - \left|\frac{\alpha - \min[x,\,x']}{\sigma}\right|\right)\left(1 - \left|\frac{\alpha - \max[x,\,x']}{\sigma}\right|\right) d\alpha = \int_{d-\sigma}^{\sigma} \left(1 - \left|\frac{\alpha}{\sigma}\right|\right)\left(1 - \left|\frac{\alpha - \mathrm{d}}{\sigma}\right|\right) d\alpha$$

$$= \int_{d-\sigma}^{0} \left(1 + \frac{\alpha}{\sigma}\right)\left(1 + \frac{\alpha - d}{\sigma}\right) d\alpha + \int_{0}^{d} \left(1 - \frac{\alpha}{\sigma}\right)\left(1 + \frac{\alpha - d}{\sigma}\right) d\alpha + \int_{d}^{\sigma} \left(1 - \frac{\alpha}{\sigma}\right)\left(1 - \frac{\alpha - d}{\sigma}\right) d\alpha$$

$$= \frac{1}{6\sigma^2}(4\sigma^3 - 6\sigma d^2 + 3d^3)$$

Thus, we have $\Phi_\sigma(x) \cdot \Phi_\sigma(x') = \begin{cases} 0, d \ge 2\sigma \\ \frac{1}{6\sigma^2}(8\sigma^3 - 12\sigma^2 d + 6\sigma d^2 - d^3), \sigma < d < 2\sigma \\ \frac{1}{6\sigma^2}(4\sigma^3 - 6\sigma d^2 + 3d^3), 0 \le d \le \sigma \end{cases}$ where $d = |x - x'|$. Our computation amount is independent of x and x'. We could compute $\Phi_\sigma(x) \cdot \Phi_\sigma(x')$ in a relatively efficient way.

4.

I.

As the pseudo code of a 2-way classifier is already given, what we need to do is to implement a 10-way classifier based on ten 2-way classifiers.

Here, we used a simple but effective way, to use $f(x) \coloneqq sign\left(\underset{i}{\operatorname{argmax}}\, w_i^T \cdot x\right) \cdot \underset{i}{\operatorname{argmax}}\, w_i^T \cdot x$ as the classification.

That is, to compute the weight vector that gives the maximum dot production with test data, and then examine if it is larger than 0. If it is, then we choose it as the classification, if it is not, then set it to a negative value that does not match any label.

Core part of perceptron v0:

```
%% training
for i = 1:iter_num
    index = mod(i, train_size) + 1;
    for j = 1:classes_num
        if train_label(index, j)*(train_data(index, :)*W(:, j))<=0
            W(:, j) = W(:, j) + (train_label(index, j)*train_data(index, :)).';
        end
    end
end
```

```
%% testing.
cnt = 0;
for i = 1:test_size
    prediction = test_data(i, :)*W;
    prediction = find(prediction==max(prediction));
    if test_label(i, prediction)==1 && prediction>0
        cnt = cnt + 1;
    end
end
test_accuracy = cnt / test_size;
```

Core part of perceptron v1:

```
%% training
for j = 1:classes_num
    for iter_count = 1:iter_num
        temp = train_label(:, j).*(train_data*W(:, j));
        min_temp = min(temp);
        if min_temp>0
            break;
        end
        index = find(temp==min_temp);
        index = index(1);
        W(:, j) = W(:, j) + train_label(index, j)*train_data(index, :).';
    end
end
%% testing.
cnt = 0;
for i = 1:test_size
    prediction = test_data(i, :)*W;
    prediction = find(prediction==max(prediction));
    if test_label(i, prediction)==1 && prediction>0
        cnt = cnt + 1;
    end
end
test_accuracy = cnt / test_size;
```

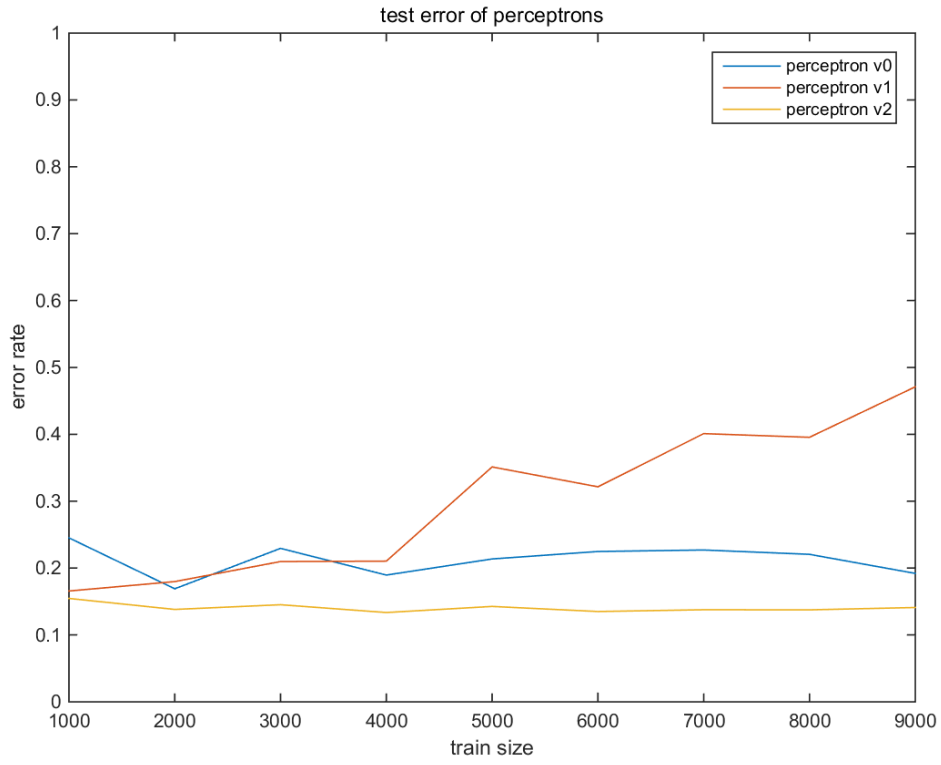
Core part of perceptron v2:

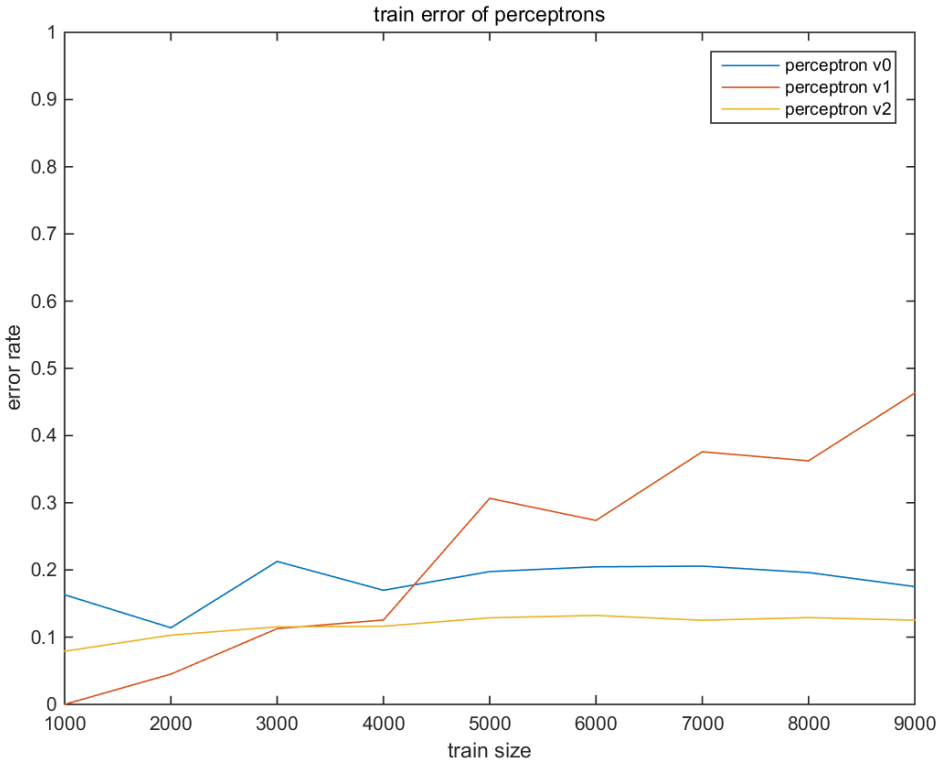
```
%% training
for j = 1:classes_num
    k = 1;
    for i = 1:iter_num
        index = mod(i, train_size) + 1;
        if train_label(index, j)*(train_data(index, :)*W(:, j, k))<=0
            W(:, j, k+1) = W(:, j, k) + (train_label(index, j)*train_data(index, :)).';
            c(k+1, j) = 1;
            k = k + 1;
        else
            c(k, j) = c(k, j) + 1;
        end
    end
end
%% testing.
cnt = 0;
for i = 1:test_size
    prediction = zeros(classes_num, 1);
    for j = 1:classes_num
        index = 2;
        while c(index, j)~=0
```

```
            prediction(j) = prediction(j) + c(index, j) * test_data(i, :) * W(:, j,
index);
            index = index + 1;
        end
    end
    prediction = find(prediction==max(prediction));
    if test_label(i, prediction)==1 && prediction>0
        cnt = cnt + 1;
    end
end
test_accuracy = cnt / test_size;
```

II.

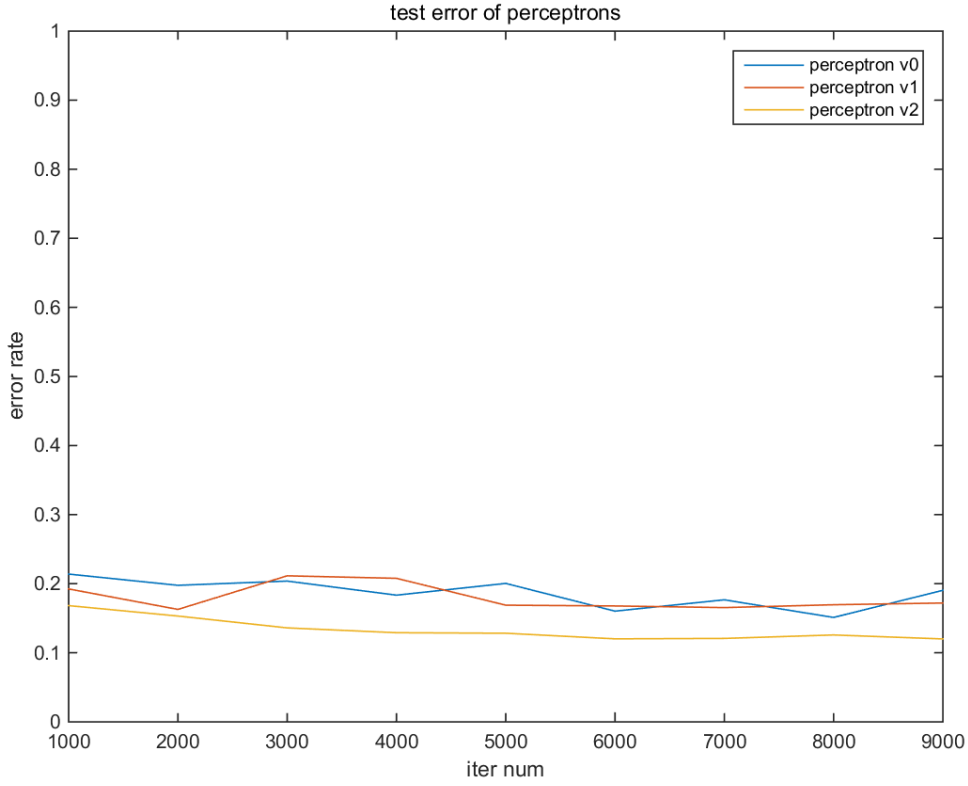
Performance of three perceptron versions over different training data size (iteration number is set to 2000)

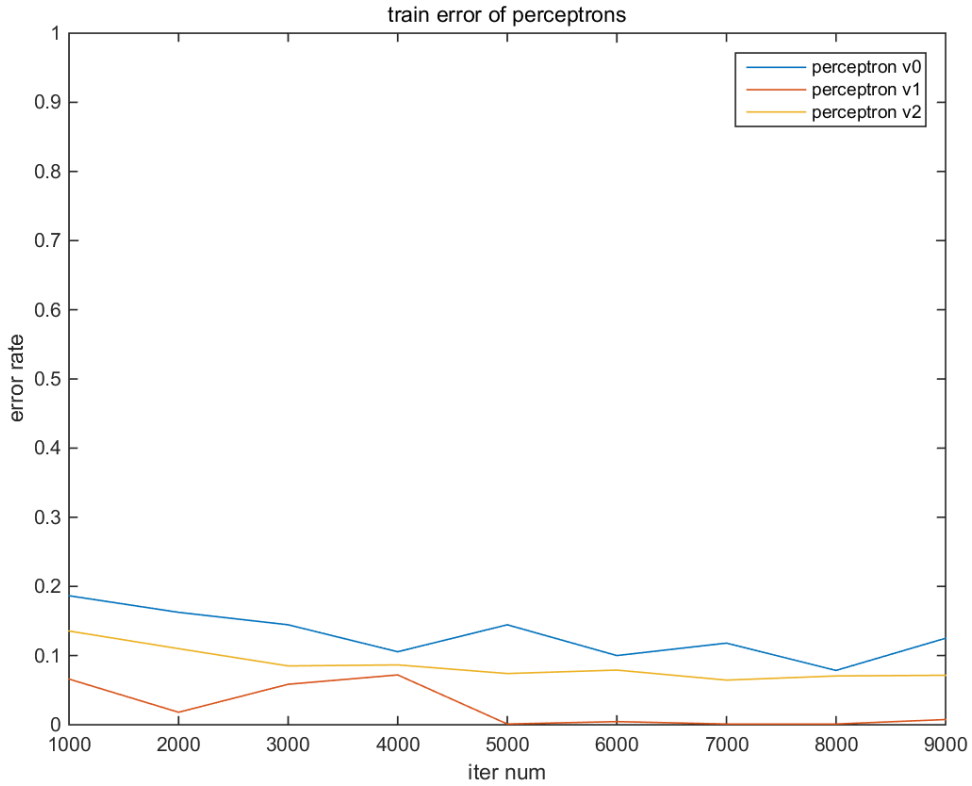




It is obvious that as training data size grows, performance of perceptron v1 degenerates greatly while the other two methods remain almost the same.

Performance of three perceptron versions over different iteration number (training data size is set to 2000)





As iteration numbers grows, accuracy of all three kinds of perceptron increases slightly.

According to all four graphs above, perceptron v1 has a good accuracy over training data, but accuracy of test data is not so promising, it is likely that this method is easy to get overfitting and thus more sensitive to noise in training data. It has a good accuracy when training data size is small, and it performs well even when iteration number is really small. But when training data size is big, it performs poorly. It is suitable for small size data and small iteration times.

Perceptron v2 has the best overall performance. It has a good accuracy no matter how large the training data size and the iteration numbers is.

Perceptron v0 behaves similar to v2 though it is not as accurate as v2. But it has a simpler structure and takes less time to train the perceptron. Thus when we need the perceptron to be really accurate, we should choose perceptron v2, when we do not necessarily need the method to be that accurate and would like to have a simpler perceptron, we should choose perceptron v0.

III.

Choose function $k(x, x') = (< x, x' > +1)^6$ as the kernel function.

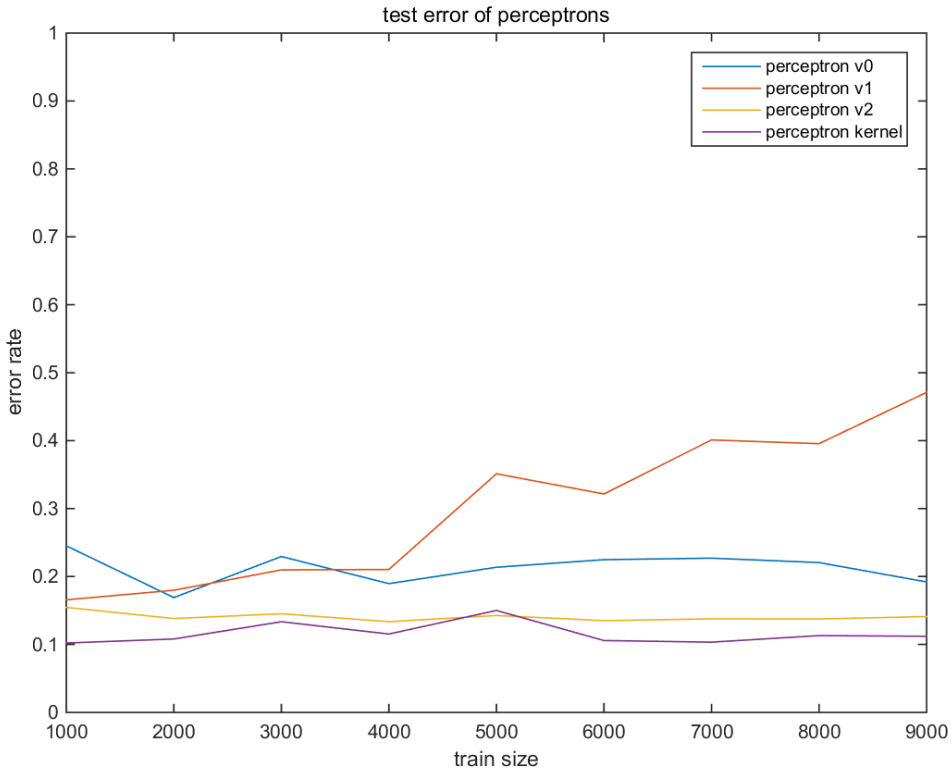
Core part of kernel perceptron:

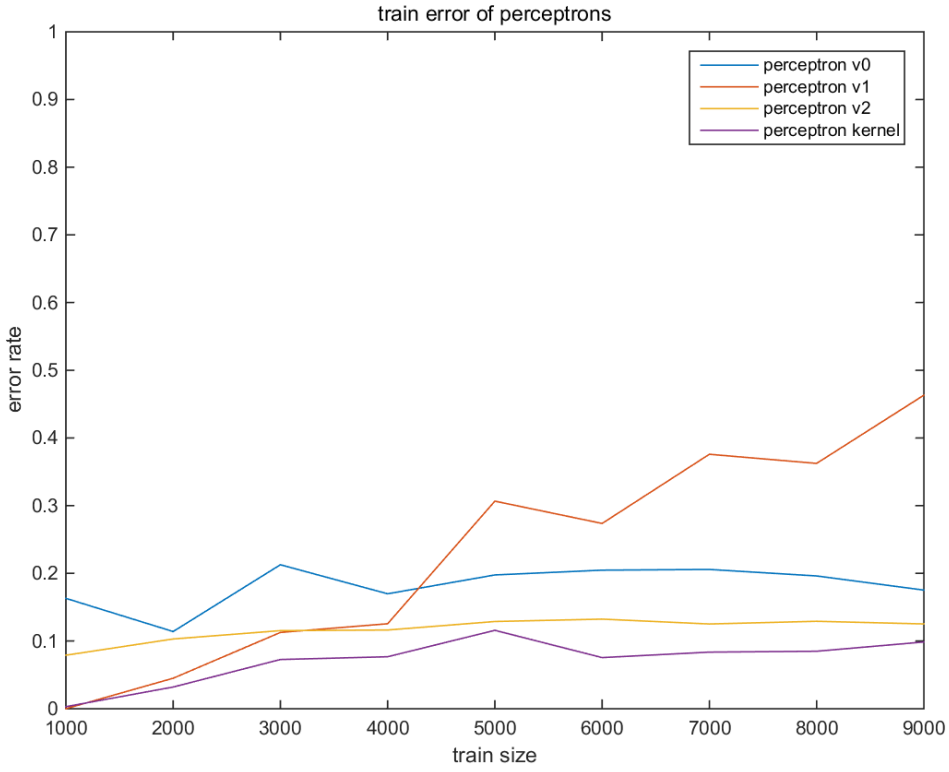
```
%% training
for i = 1:iter_num
    index = mod(i, train_size) + 1;
    temp = ((train_data*train_data(index, :).'+1).^6).'*(W.*train_label);
    for j = 1:classes_num
        if temp(j)*train_label(index, j)<=0
            W(index, j) = W(index, j) + 1;
        end
    end
end
```

```
%% testing.
cnt = 0;
for i = 1:test_size
    prediction = ((train_data*test_data(i, :).'+1).^6).'*(W.*train_label);
    prediction = find(prediction==max(prediction));
    if test_label(i, prediction)==1 && prediction>0
        cnt = cnt + 1;
    end
end
test_accuracy = cnt / test_size;
```

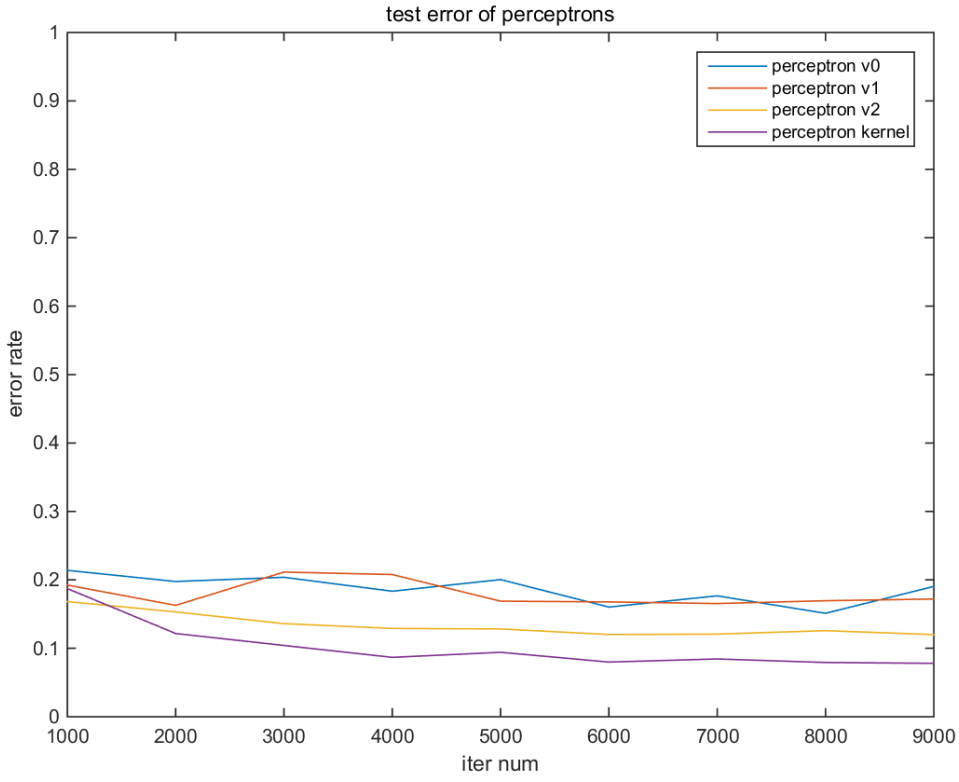
Comparison of four kinds of perceptron

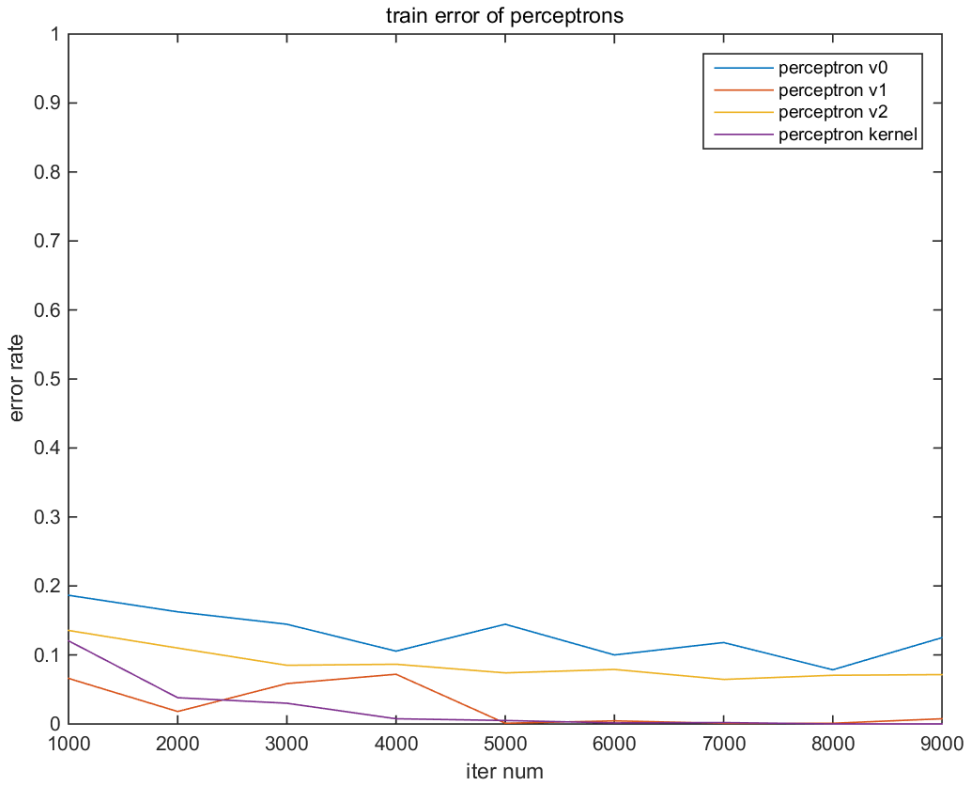
Performance of 4 perceptron over different training size (iteration times is set to 2000)





Performance of 4 perceptron over different iteration times (training size is set to 2000)





According to graphs above, it is obvious that this kernel version perceptron has the best performance regardless of the training size and iteration times. Besides, as we could use dot production of x and x' in the origin space to compute dot production in the transformed space, we could have the training process really fast, which makes it a very good version of perceptron.

5.

I.

$$\frac{\partial \sigma}{\partial x} = \left(-\frac{1}{(1+e^{-x})^2}\right)\cdot(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}}\cdot\left(1-\frac{1}{1+e^{-x}}\right) = \sigma(x)(1-\sigma(x))$$

II.

$$E(W,b) = \frac{1}{2n}\sum_{i=1}^{n}||\mathcal{F}_{I-layer}(x_i) - y_i||^2 = \frac{1}{2n}\sum_{i=1}^{n}||\sigma(W^T x_i + b) - y_i||^2$$

$$\frac{\partial E}{\partial W} = \frac{1}{2n}\sum_{i=1}^{n} 2\left|\left|\mathcal{F}_{I-layer}(x_i) - y_i\right|\right|\frac{\partial \sigma}{\partial W}$$

$$= \frac{1}{n}\sum_{i=1}^{n}||\mathcal{F}_{I-layer}(x_i) - y_i||x_i \times [\sigma(W^T x_i + b)(1-\sigma(W^T x_i + b))]^T$$

$$\frac{\partial E}{\partial b} = \frac{1}{2n}\sum_{i=1}^{n} 2\left|\left|\mathcal{F}_{I-layer}(x_i) - y_i\right|\right|\frac{\partial \sigma}{\partial b}$$

$$= \frac{1}{n}\sum_{i=1}^{n} ||\mathcal{F}_{I-layer}(x_i) - y_i||\sigma(W^T x_i + b)\big(1 - \sigma(W^T x_i + b)\big)$$

III.

As given the pseudo code, the main work of this question is to compute the derivatives of parameters, to choose an appropriate step size to update the parameters as well as to decide how many neurons to use.

Initialization of parameters:

```
neurons_num = 8;                                                    % number of
neurons in middle layer
threshold = 2e-5;                                                   % threshold for
Error function
updating_rate = 0.1;                                                % updating rate
of parameters
data_size = size(X, 1);
W_1 = rand(neurons_num, 1);                                         % initialize
parameters with random value
W_2 = rand(neurons_num, 1);
b_1 = rand(neurons_num, 1);
b_2 = rand(1, 1);
```

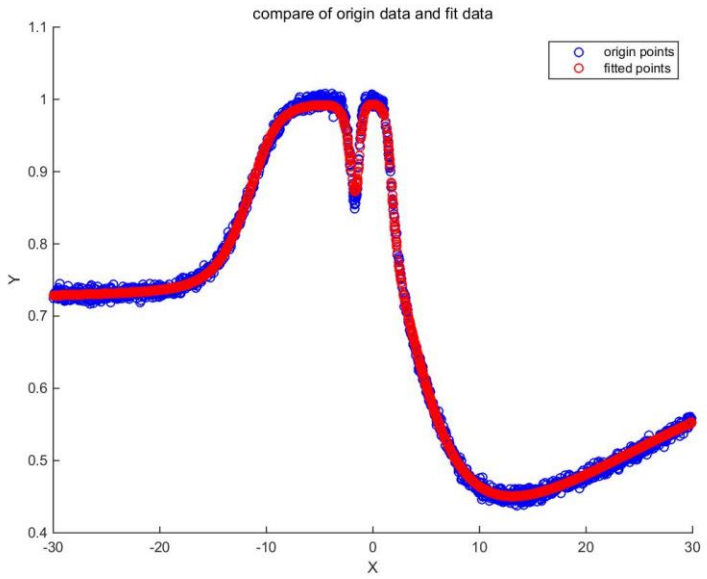
Core part of code:

```
while Err>threshold
    midNuerons = zeros(neurons_num, 1);                             % initialize
middle layer neurons
    dW_1 = zeros(size(W_1));                                        % initialize
derivative of parameters
    dW_2 = zeros(size(W_2));
    db_1 = zeros(size(b_1));
    db_2 = zeros(size(b_2));
    Err = 0;
    for i = 1:data_size
        for j = 1:neurons_num
            midNuerons(j) = 1/(1+exp(-(W_1(j)*X(i)+b_1(j))));       % calculate value
of middle layer
        end
        out = 1/(1+exp(-(W_2.'*midNuerons+b_2)));                   % calculate
output
        dout = (out-Y(i))*out*(1-out);                              % calculate
derivative of output
        Err = Err + (out-Y(i))^2;                                   % calculate Error
function value
        db_2 = dout;                                                % derivative of
b_2
        dW_2 = dout*midNuerons;                                     % derivative of
W_2
        dmidNuerons = dout*W_2;
        dmid = zeros(neurons_num, 1);                               % calculate
derivative of middle layer
        for j = 1:neurons_num
            dmid(j) = midNuerons(j)*(1-midNuerons(j));              % calculate
derivative of middle layer
        end
        db_1 = dmidNuerons.*dmid;                                   % derivative of
b_1
```

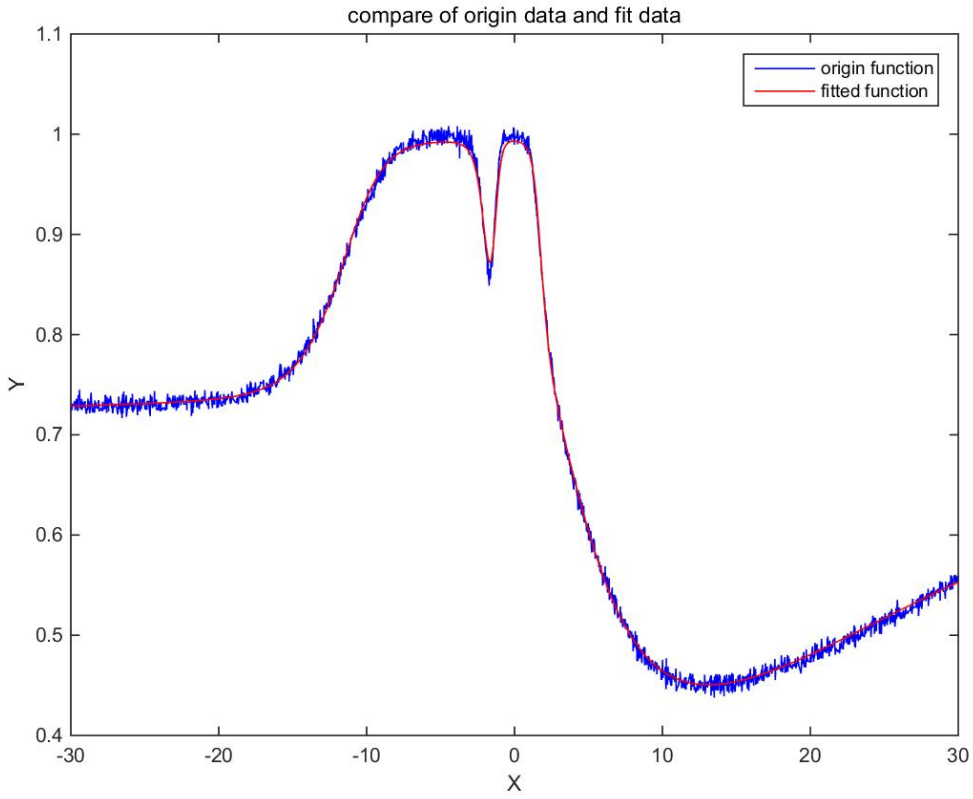
```
        dW_1 = dmidNuerons.*dmid*X(i);                                  % derivative of
W_1
        dW_1 = dW_1 * updating_rate;                                    % regulation of
derivative
        dW_2 = dW_2 * updating_rate;
        db_1 = db_1 * updating_rate;
        db_2 = db_2 * updating_rate;
        W_1 = W_1 - dW_1;                                               % updating
parameters
        W_2 = W_2 - dW_2;
        b_1 = b_1 - db_1;
        b_2 = b_2 - db_2;
    end
    Err = Err / data_size / 2;
end
```

IV.

Scatter graph of origin points (marked blue) and fitted points (marked red).



Plot graph of origin function and fitted function.



As the graph shows, the fitted function is very close to the origin function. The neuron network successfully approximated the origin function.