COMS 4771 Machine Learning (Spring 2018)
Problem Set #2

Yufei Zhao - yz3170@columbia.edu
Minghao Li - ml4025@columbia.edu
Zhuoran Xiong - zx2214@columbia.edu

March 3, 2018

Problem 1

(i)

OvR:
Training algorithm is described below:

Suppose we already have a binary classifier called C, the dataset has K labels called Y, each with some samples as X. Then for each class k, we can construct a new label vector with $n_i = 1$ and $n_{else} = 0$, which correspond to one v.s. all. Finally we can apply C to current X and n to obtain f_k. Using this algorithm, we can output a list of K classifiers analyzed as below:

One binary classifier is needed for each label, so, for each new test sample, we need to call k times binary classifier.

$$f : \mathbb{R}^d \to \{\text{class } i\}, i \in \{1, 2, \ldots, k\}.$$

$$f(x) = \arg\max_i f_i(x)$$

If we encounter a tie for some specific data point, we set a consistent rule to get the label. The rule is described below:

We set a priority array based on the alphabet order of the first item of label. For example:

$$[\text{"}blue\text{"}, \text{"}red\text{"}, \text{"}yellow\text{"}]$$

Then if for the current point, the votes for both red and blue are equal to each other, we will label it as blue since blue has higher priority in this array.

OvO:
One binary classifier is needed for each combination of two distinct labels, so, $\frac{k(k-1)}{2}$ calls of binary classifier are necessary for each test sample.
Let x to be the new test sample,

$$\text{for } i \in [k], \text{count}(i) = 0$$

$$\forall\, i, j \in [k], \text{if } f_{ij}(x) = \text{class } i, \text{then count}(i)\text{+=1}, \text{else count}(j)\text{+=1}$$

$$f(x) = \arg\max_i \text{count}(i)$$

More clearly, can write $f(x)$ as:

$$f(x) = \arg\max_i \big(\sum_j g_{ij}(x)\big)$$

where $g_{ij}(x) = \begin{cases} 1 \text{ , if } f_{ij}(x) = i \\ 0 \text{ , if } f_{ij}(x) = j \end{cases}$, notice that $g_{ij}(x) = 1 - g_{ji}(x)$, so $\binom{k}{2}$ calls will be enough.

(ii)

- OvR gives better accuracy over OvO

We suppose that under the assumption of this problem, OvR can't be better than OvO. I will prove by the following affirm: even OvR has the 100% accuracy, OvO can be no worse than it.

Firstly, let's assume that we can indeed classify two labels very well, between which are one class and the other all, which is ploted below:

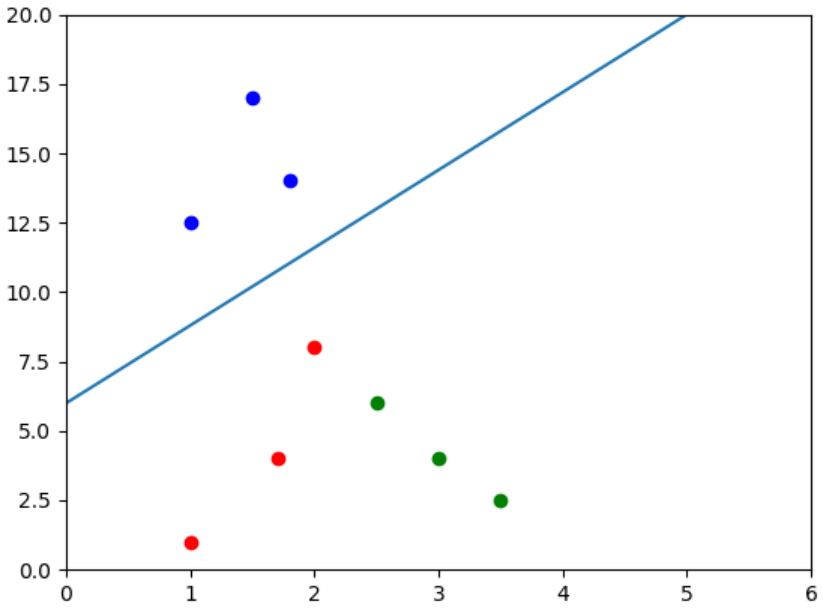

Figure 1: Data points

Three steps to prove the impossibility:

(a) If OVR is perfect, OVO can also be definitely perfect.

Based on OVR, the dataset is linearly separable. It's like the blue line can split blue points and the others. But based on OVO, the red and green points are both in the below part, so if we consider whatever like the blue V.S. red or blue V.S. green, we can also get the right labels.

(b) If OVR is not perfect enough because of the nonlinearity, OVO can still be perfect classifier.

If we want to classify out the red points in Figure1, using OVR, point(2,7.5) should be green since there's no straight line which can split out all the pure red points from the others. However, based on the OVO, as we just analyzed above, each pair of classes is linearly separable so we definitely will get 100% accuracy.

(c) Lastly, if based on OVO and OVR, the classes are all nonlinearly separable, it's also impossible for OVR to have a better performance than OVO does. The thoughts is that we can imagine OVO as sub-problems of OVR. For a specific class, if it's linearly separable based on OVR, it means this class can be linearly separable on OVO, which compares this class and another one in OVR. So under the OVR case, it can only make the situations more complex and data points more difficult to classified.

In summary, we don't think under the given assumption, OVR can be better than OVO.

- OvO gives better accuracy over OvR

class 1: (4, 4), (1, 5), (5, 1)
class 2: (1, 6), (4, 5), (7, 8)
class 3: (6, 1), (5, 4), (8, 7)
each combination of two classes is linear separable, but each OvR space is not linear separable.

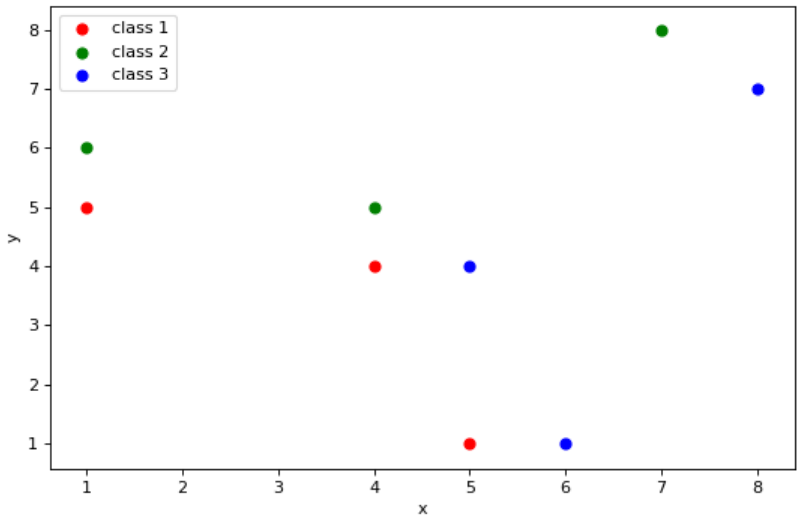

Figure 2: Data points

- OvO and OvR give accuracy of at most ϵ

For this part, we are going to construct a dataset depending on ϵ.

First I choose randomly 5 points on the plane. Then set initially 1 data on each point as the first class. Next I can put another class in: it also contains 5 data with each positioned exactly on the given 5 points. The picture is shown below:

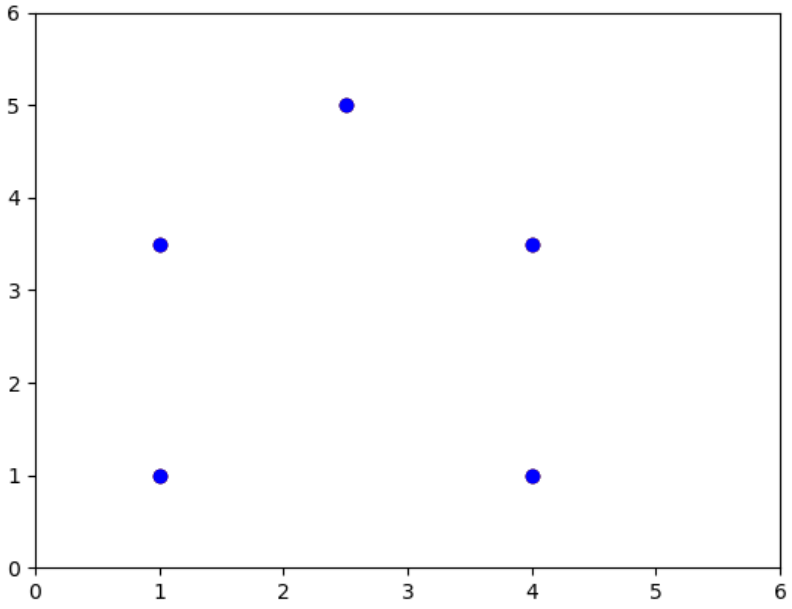

Figure 3: K classes positioned on 5 points

Now, we can suppose for some best circumstances, we can only get one correct label for each point. Obviously, the accuracy is $\frac{1}{K}$ no matter which OVR or OVO is used since we can only get one label for one point. Therefore, according to the requirement of this question, we can get the following equation:

$$\epsilon < \frac{1}{K}$$

where K is the class number. From the equation, we can see that for any given ϵ, we can get a class number K, such that $\epsilon < \frac{1}{K}$.

- OvO and OvR give accuracy of at least $1 - \epsilon$

This part is quite direct, we just apply the two kinds of classifiers to totally linearly separable dataset as below:

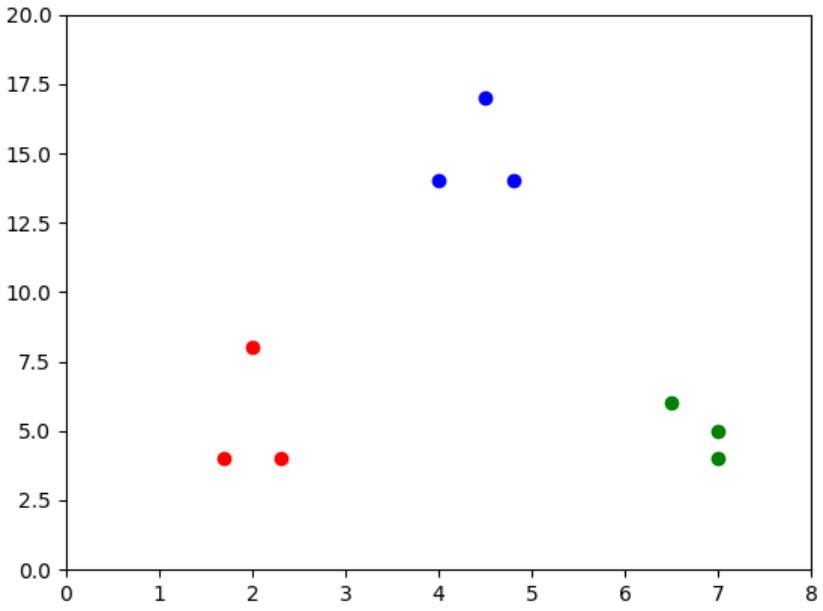

Figure 4: Data points

It's obvious that we can split the three classes very clearly using whatever the OVR or OVO. So the accuracy should be 1. That is given any $\epsilon > 0$, the accuracy is always

$$1 > 1 - \epsilon$$

which is the required conclusion.

(iii)

Follow the idea of binary search tree.

Step 1: Build a binary classifier to distinguish class $1, 2, \ldots, [\frac{k}{2}]$ from class $[\frac{k}{2}+1], \ldots, k$(assigning samples from class $1, 2, \ldots, [\frac{k}{2}]$ label 1 and others label 0)

Step 2: Build binary classifier to distinguish class $1, 2, \ldots, [\frac{k}{4}]$ from class $[\frac{k}{4} + 1], \ldots, [\frac{k}{2}]$ and class $[\frac{k}{2} + 1], \ldots, [\frac{3k}{4}]$ from class $[\frac{3k}{4} + 1], \ldots, k$

Repeat above process until only one class left in each branch.
Then, during test time, $\log_2 k$ calls of binary classifier will be made for each test case.

Proof of minimization:
Suppose we only call $c^* < \log_2 k$ times binary classifier.
Because binary classifier can only produce two results at one time, it can at most result in $2^{c^*} < 2^{\log_2 k} = k$ different classes. This means that for some data points belongs to some

classes, we can never give the correct answer, which is not acceptable.
So, $\log_2 k$ minimize the number of calls made to binary classifier during test time.

Problem 2

In this problem, we are supposed to prove that if we an find the minimize $||x - x_a||^2$ under constraint $g(x) = w \cdot x + x_0 = 0$, we can get the distance from hyperplane $g(x) = w \cdot x + x_0 = 0$ to x_a.
The function $g(x)$ is a convex function so the hyperplane is a convex set. Therefore we can find the minimum of Lagrange function to optimize it. We have

$$L(\vec{x}, \vec{\lambda}) = (x - x_a)^\top (x - x_a) + \lambda(w^\top x + x_0)$$

get the partial derivative of this function respect to both $\vec{x}$ and $\vec{\lambda}$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \lambda} = 0$$

we can get

$$x = x_a - \lambda w/2$$

and

$$w^\top x + w_0 = 0$$

combine this two solution together we get

$$\lambda = 2g(x_a)/w^\top w$$

take this λ into $x = x_a - \lambda w/2$ and get x^* that is the point on this hyperplane that has minimum distance to x_a under the constrain. Then get the distance from x^* to x_a by getting the l_1 norm of the difference.

$$\begin{aligned} |x^* - x_a| &= \lambda w/2 \\ &= \frac{|g(x_a)|}{||w||} \end{aligned}$$

Problem 3

(i)

Data points:(x_1, x_2, x_0) with $x_0 = 1$ as constant lifting
class 1: n1(1, 1, 1), n2(2, 2, 1), n3(1, 2, 1), n4(2, 1, 1)
class 2: n5(4, 4, 1), n6(5, 5, 1), n7(4, 5, 1), n8(5, 4, 1)

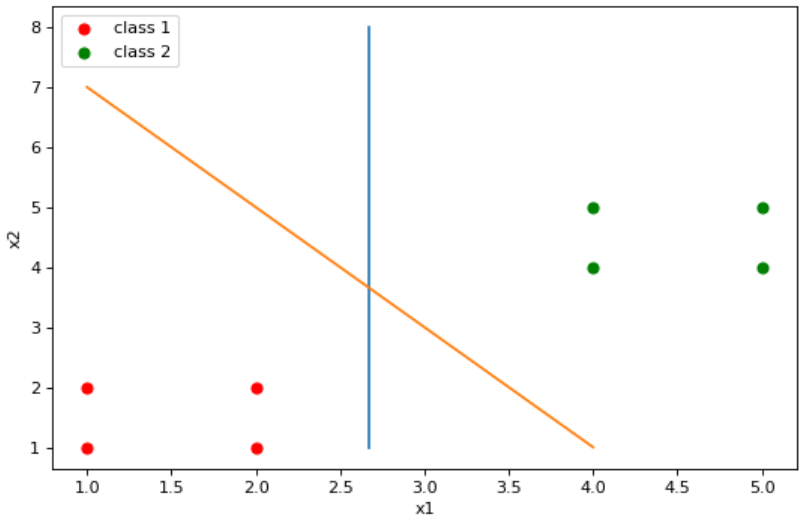

Figure 5: example training dataset S

When finding data point that violates the classifier from n1 to n8, the Perceptron algorithm will return $(w_1, w_2, w_0) = (-2, -1, 9)$, and has a margin of $\frac{3}{\sqrt{5}}$

When scanning data point that violates the classifier from n8 back to n1, the Perceptron algorithm will return $(w_1, w_2, w_0) = (-3, 0, 8)$, and has a margin of $\frac{2}{3}$

(ii)

(a)

a margin of $\frac{\gamma}{2}$ is guaranteed by MTA.
Proof:
when $w_t \neq \vec{0}$

$$2y(w_t \cdot x) \leq \gamma \|w_t\| \iff 2y\frac{w_t \cdot x}{\|w_t\|} \leq \gamma$$

Because $y = \{-1, +1\}$ and $\gamma > 0$, so

if (x, y) is not correctly classified

$$2y\frac{w_t \cdot x}{\|w_t\|} < 0 \leq \gamma$$

else

$$2y\frac{w_t \cdot x}{\|w_t\|} = 2\frac{|w_t \cdot x|}{\|w_t\|} \leq \gamma \iff \frac{|w_t \cdot x|}{\|w_t\|} \leq \frac{\gamma}{2}$$

So, MTA will only terminate when

$$\forall i \in \{1, 2, \ldots, n\}, \frac{|w_T \cdot x_i|}{\|w_T\|} > \frac{\gamma}{2}$$

$$\Rightarrow \text{margin} = \min_i \frac{|w_T \cdot x_i|}{\|w_T\|} > \frac{\gamma}{2}$$

(b)

i. Let w^* be linear separator that can separate the dataset with margin γ and $\|w^*\| = 1$.

$$\begin{aligned} w_t \cdot w^* &= (w_{t-1} + yx) \cdot\ w^* \\ &\geq w_{t-1} \cdot w^* + \gamma \\ &\geq t\gamma \end{aligned}$$

$$\Rightarrow T\gamma \leq \vec{w_T} \cdot w^* \leq \|w_T\|\|w^*\| = \|w_T\|$$

ii.

$$\begin{aligned} \|w_t\|^2 &= \|w_{t-1} + yx\|^2 \\ &= \|w_{t-1}\|^2 + 2yw_{t-1} \cdot x + \|x\|^2 \\ &\leq \|w_{t-1}\|^2 + \gamma\|w_{t-1}\| + R^2 \\ &\leq \|w_{t-1}\|^2 + \gamma\|w_{t-1}\| + \frac{\gamma^2}{4} + R^2 \\ &= (\|w_{t-1} + \frac{\gamma}{2}\|)^2 + R^2 \end{aligned}$$

iii.

$$\|w_t\|^2 \leq (\|w_{t-1}\| + \frac{\gamma}{2})^2 + R^2$$

$$\begin{aligned} &\Rightarrow \|w_t\|^2 - (\|w_t\| + \|w_{t-1}\| + \frac{\gamma}{2})^2 \leq R^2 \\ &\Rightarrow (\|w_t\| + \|w_t\| + \|w_{t-1}\| + \frac{\gamma}{2})(\|w_t\| - (\|w_t\| + \|w_{t-1}\| + \frac{\gamma}{2})) \leq R^2 \\ &\Rightarrow \|w_t\| \leq (\|w_{t-1}\| + \frac{\gamma}{2}) + \frac{R^2}{\|w_t\| + \|w_{t-1}\| + \gamma/2} \end{aligned}$$

iv. Because

$$\|w_{t-1}\| \geq \frac{4R^2}{\gamma} \text{ or } \|w_t\| \geq \frac{4R^2}{\gamma}$$

So

$$\frac{R^2}{\|w_t\| + \|w_{t-1}\| + \gamma/2} \leq \frac{R^2}{4R^2/\gamma} = \frac{\gamma}{4}$$

$$\Rightarrow \|w_t\| \leq \|w_{t-1}\| + \frac{3}{4}\gamma$$

v.

$$\begin{aligned} 4R &= 4 \max_i \|x_i\| \\ &\geq \|x_j\| \\ &= \|w^*\| \|x_j\| \\ &\geq \min_i w^* \cdot x_i \\ &= \gamma \end{aligned}$$

$$w_0 = \vec{0} \Rightarrow \|w_0\| \leq R \leq \frac{4R^2}{\gamma}$$

$$\|w_t\| \geq t \cdot \gamma \geq \frac{4R^2}{\gamma}, \forall t \geq \frac{4R^2}{\gamma^2}$$

So there must be a largest t_0 such that $\|w_{t_0-1}\| \leq 4R^2/\gamma$ and $\|w_{t_0}\| \geq 4R^2/\gamma$

vi.

$$\|w_t\| \leq \|w_{t-1}\| + \frac{3}{4}\gamma$$

$$\begin{aligned} \Rightarrow \|w_T\| &\leq \|w_{t_0-1}\| + \frac{3}{4}(T - t_0 + 1)\gamma \\ &\leq \|w_{t_0-1}\| + \frac{3T}{4}\gamma \end{aligned}$$

$$\Rightarrow T\gamma \leq \|w_T\| \leq \|w_{t_0-1}\| + \frac{3T}{4}\gamma \leq \frac{4R^2}{\gamma} + \frac{3T}{4}\gamma$$

$$\Rightarrow T \leq \frac{16R^2}{\gamma^2}$$

Problem 4

(i)

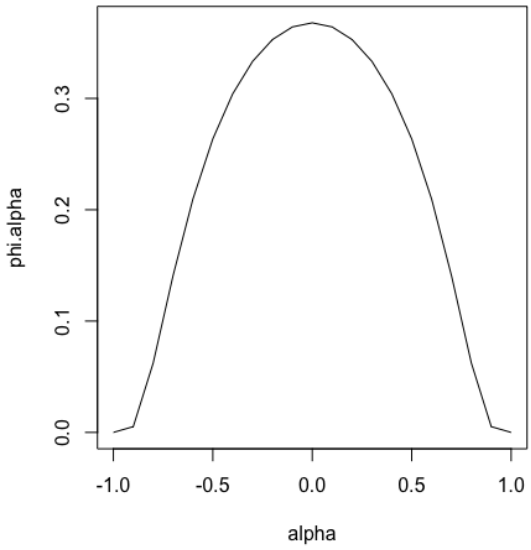

Figure 6: $\Phi_\sigma(\alpha)$ when $x = 0$ and $\sigma = 1$

We can first draw the figure of $\Phi_\sigma(\alpha)$ when $x = 0$ and $\sigma = 1$ as above figure shows. It is easy to infer for the figure and the function that when α is bigger than $x + \sigma$ or smaller than $x - \sigma$, $\Phi_\sigma(\alpha)$ equal to 0. Otherwise, $\Phi_\sigma(\alpha)$ equal to $e^{\frac{-1}{1-(\frac{|\alpha-x|}{\sigma})^2}}$ as the above figure shows.

We suppose x and x' are most closet distinct points and $x < x'$. Then, the distance between x and x' is $x' - x$ is the smallest gap between all points. So we can set the $/sigma$ to be smaller then $\frac{x'-x}{2}$ so that the $\Phi_{\sigma,x}(\alpha)$ of all the points will not overlap.

Then we can set the weight of feature to be $w_i(\alpha)$ and for $w_i(\alpha)$, α can only be in interval $(x_i - \sigma, x_i + \sigma)$ so that for each x_i we have

$$\int_{x_i-\sigma}^{x_i+\sigma} w_i(\alpha) \cdot \Phi_{\sigma,x_i}(\alpha) \mathrm{d}\alpha = \mathrm{y_i}$$

in order to satisfied this function, we only need to set the value of $w(\alpha)$ in $(x_i - \sigma, x_i + \sigma)$. Than we can sum up the value of $w(\alpha)$ of all of interval and get the final $w(\alpha)$.

Therefore, for each data x_i in $x_1, x_2, ..., x_n$, we can compute the label by $\int_{-\infty}^{\infty} w_i(\alpha) \cdot \Phi_{\sigma,x_i}(\alpha) \mathrm{d}\alpha$. This function is equal to $\int_{x_i-\sigma}^{x_i+\sigma} w_i(\alpha) \cdot \Phi_{\sigma,x_i}(\alpha) \mathrm{d}\alpha$ and can get the result y_i. So the n points are separated. Since the weight is given on linear model of features, these n points can be separated by linear model.

(ii)

If we compute the value of $\Phi_\sigma(\alpha, x) \cdot \Phi_\sigma(\alpha, x')$ for arbitrary points x and x'. Suppose $x < x'$, we have

$$\begin{aligned}\Phi_\sigma(\alpha, x) \cdot \Phi_\sigma(\alpha, x') &= \int_{-\infty}^{\infty} \Phi_\sigma(\alpha, x) \cdot \Phi_{\sigma,x'}(\alpha) \mathrm{d}\alpha \\ &= \int_{x-\alpha}^{x'+\alpha} \Phi_\sigma(\alpha, x) \cdot \Phi_{\sigma,x'}(\alpha) \mathrm{d}\alpha\end{aligned}$$

Before we set σ to distinguish every point, there is overlap between point. If we also suppose that $x - \alpha < x' - \alpha < x + \alpha < x' + \alpha$. Since the one of $\Phi_\sigma(\alpha, x)$ and $\Phi_\sigma(\alpha, x')$ is zero in the non-overlap area($(x - \alpha, x' - \alpha)$ and $(x + \alpha, x' + \alpha)$), we can also change this equation to

$$\begin{aligned}\Phi_\sigma(\alpha, x) \cdot \Phi_\sigma(\alpha, x') &= \int_{x-\alpha}^{x'+\alpha} \Phi_\sigma(\alpha, x) \cdot \Phi_{\sigma,x'}(\alpha) \mathrm{d}\alpha \\ &= \int_{x'+\alpha}^{x-\alpha} \Phi_\sigma(\alpha, x) \cdot \Phi_{\sigma,x'}(\alpha) \mathrm{d}\alpha\end{aligned}$$

(iii)

(a) Since $c_y = \frac{1}{m_y} \sum_{i:y_i=y} \phi(x_i)$, we can build coordinates based on $\phi(x)$ and y. There are two point in this coordinates, $(c_+, +1)$ and $(c_-, -1)$.
Suppose we link the point of $(c_+, +1)$ and $(c_-, -1)$ by a straight line $\langle \beta \cdot \phi(x) \rangle + m = y$ and expand this line. So we have

$$\langle \beta \cdot c_+ \rangle + m = 1$$

and

$$\langle \beta \cdot c_- \rangle + m = -1$$

Solve this two equation, we get

$$\begin{cases} \beta = \frac{2}{c_+ - c_-} \\ m = \frac{c_+ - c_-}{c_- - c_-} \end{cases}$$

So we can make the plug in β and m and get this

$$y = \frac{2}{c_+ - c_-} \phi(x) + \frac{c_+ - c_-}{c_- - c_-}$$

multiply this equation by $\frac{(c_+ - c_-)^2}{2}$ and get

$$\begin{aligned}\frac{(c_+ - c_-)^2}{2} \cdot y &= (c_+ - c_-) \cdot \phi(x) + \frac{(c_+ - c_-)(c_- - c_-)}{2} \\ &= (c_+ - c_-) \cdot \phi(x) + \frac{\|c_+\|^2 + \|c_-\|^2}{2} \\ &= \langle w, \phi(x) \rangle + b \end{aligned} \tag{1}$$

The mid-point of $(c_+, +1)$ and $(c_-, -1)$ is $(\frac{c_+ - c_-}{2}, 0)$ and divide the line into two part. Suppose point on this line is smaller or same distance to $(c_+, +1)$ compared to distance to $(c_+, +1)$. Then, $h(x)$ is equal to 1. Also, the y on the left of equal mark of (1) will be bigger than 0, since it is in the same part of $(c_+, +1)$ the y value of the threshold is 0. So if $\|\phi(x) - c_+\| \leq \|\phi(x) - c_-\|$, $(\phi(x), y)$ will be on the upper part of the coordinate system or on x-axis and y will be ≤ 0. According to (1), the right part is bigger than zero. so $sign(\langle w, \phi(x)\rangle + b)$ is equal to 1, the same as $h(x)$ under this situation. And it is easy to get when $\|\phi(x) - c_+\| < \|\phi(x) - c_-\|$, $y < 0$. So $sign(\langle w, \phi(x)\rangle + b)$ is equal to -1, which is the same as $h(x)$ under this situation.
So $h(x)$ and $sign(\langle w, \phi(x)\rangle + b)$ is proved the same under any situation.

(b) According to (a), we first have $h(x) = sign(\langle w, \phi(x)\rangle + b)$. We also have $K(x, x') = \phi(x) \cdot \phi(x')$. If we can get $w = c_+ - c_-$ in form of $\phi(x)$, we can take this kernel to compute $\langle w, \phi(x)\rangle$.
Since

$$c_+ = \frac{1}{m_+} \sum_{i:y_i=+1} \phi(x_i)$$

and

$$c_- = \frac{1}{m_-} \sum_{i:y_i=-1} \phi(x_i)$$

So

$$w = \frac{1}{m_+} \sum_{i:y_i=+1} \phi(x_i) - \frac{1}{m_-} \sum_{i:y_i=-1} \phi(x_i)$$

Then,

$$\begin{aligned}
\langle w, \phi(x)\rangle &= \langle \frac{1}{m_+} \sum_{i:y_i=+1} \phi(x_i), \phi(x)\rangle - \langle \frac{1}{m_-} \sum_{i:y_i=-1} \phi(x_i), \phi(x)\rangle \\
&= \frac{1}{m_+} \sum_{i:y_i=+1} \langle \phi(x_i), \phi(x)\rangle - \frac{1}{m_-} \sum_{i:y_i=-1} \langle \phi(x_i), \phi(x)\rangle \\
&= \frac{1}{m_+} \sum_{i:y_i=+1} K(x_i, x) - \frac{1}{m_-} \sum_{i:y_i=-1} K(x_i, x)
\end{aligned}$$

Therefore

$$\begin{aligned}
h(x) &= sign(\langle w, x\rangle + b) \\
&= sign(\frac{1}{m_+} \sum_{i:y_i=+1} K(x_i, x) - \frac{1}{m_-} \sum_{i:y_i=-1} K(x_i, x) + b) \qquad (2)
\end{aligned}$$

We also have

$$b = \frac{1}{2}(\|c_-\|^2 - \|c_+\|^2)$$

So

$$\begin{aligned} b &= \frac{1}{2}(c_-^T \cdot c_- - c_+^T \cdot c_+) \\ &= \frac{1}{2m_-^2} \sum_{i:y_i=-1} \phi^T(x_i) \cdot \sum_{j:y_j=-1} \phi^T(x_j) - \frac{1}{2m_+^2} \sum_{i:y_i=+1} \phi^T(x_i) \cdot \sum_{j:y_j=+1} \phi^T(x_j) \\ &= \frac{1}{2m_-^2} \sum_{i:y_i=-1} \sum_{j:y_j=-1} \phi^T(x_i) \cdot \phi^T(x_j) - \frac{1}{2m_+^2} \sum_{i:y_i=+1} \sum_{j:y_j=+1} \phi^T(x_i) \cdot \phi^T(x_j) \\ &= \frac{1}{2m_-^2} \sum_{i:y_i=-1} \sum_{j:y_j=-1} K(x_i, x_j) - \frac{1}{2m_+^2} \sum_{i:y_i=+1} \sum_{j:y_j=+1} K(x_i, x_j) \end{aligned} \tag{3}$$

Plug in (3) into (2) and get the final result.

$$\begin{aligned} h(x) = sign(&\frac{1}{m_+} \sum_{i:y_i=+1} K(x_i, x) - \frac{1}{m_-} \sum_{i:y_i=-1} K(x_i, x) \\ &+ \frac{1}{2m_-^2} \sum_{i:y_i=-1} \sum_{j:y_j=-1} K(x_i, x_j) - \frac{1}{2m_+^2} \sum_{i:y_i=+1} \sum_{j:y_j=+1} K(x_i, x_j)) \end{aligned}$$

Problem 5

(i)

(a) One potential issue about unigram models is their independence assumption. That is, for a general n-gram model, it lacks the ability to analyze long range dependency. Here's one example:

I love the dog.

I hate the dog.

I extremely like a clever and handsome animal called dog.

Here only using the n-gram, especially the unigram model, it's very easy to classify the first two sentences as one kind because they share more common words and words' proportion than other combinations. The reason is just because it does ignore the sentence structure, with only some counts and MLE method. So it definitely becomes less accurate when compared to tree parser model, which considers more on the sentence grammar structures.

(b) Another potential problem is the stopwords. Say if the training corpus have lots of stopwords like "the","is",etc. The models will give high score result or low perplexity(low perplexity means high similarity of two documents) mainly based on these stopwords' presence. So a unigram model accuracy can depend much on the original corpus. That's why we can use the TF-IDF method to help compensate the errors.

(ii)

online-preceptron:

1. Unigram
   Instead of using numbers to index through dimensions, we decided to directly use the word itself to represent each dimension of $\vec{w}$. With built-in type dictionary in python, we can implement this algorithm easily and efficiently (both insert and search only take constant time).
   Initially, the classifier is a empty dictionary.
   As for each line of text, we also represent its feature vector $\vec{x}$ in the same manner, but add a key value pair ("BIAS", 1) for lifting. After extracting feature from each line, we get a dictionary contains all words in that line as its key and the times the word appears as its value. Then, we can traverse the feature and compute dot product with the classifier and decide whether to update the corresponding dimension in classifier or not.

2. tf-idf
   The extra effort we should do is to traverse all the review texts in the beginning and form a dictionary called *word_count_in_doc* that stores the total times of appear(appearing in one review counts as 1) for all the words. Then, when compute the

feature for each review, we firstly divide the review numbers by $word\_count\_in\_doc[word]$ to get the idf parameter for each word:

$$idf(t; D) := \frac{|D|}{word\_count\_in\_doc[word]}$$

Then we can get tfidf feature by:

$$tfidf = tf(t; d) * \log_{10}(idf(t; D))$$

3. Bigram
   The only difference between bigram and unigram is that we use a tuple (word1, word2) as index of features, where word1 and word2 appear continuously in a line.

(iii)

In general, we suppose the bigram model is a best feature representation model in this problem. And we test their performances in three dimensions below:

(a) For the original analysis, that is the whole test data applied on the whole training data model, we got the accuracies as below:

```
origin:
    unigram : 87.5%
    bigram : 87.7%
    tfidf : 85.3%
```

Figure 7: Original results

(b) The second dimension is relied on the training data sampling. We obtain 20%, 50% and 80% sample data from the whole dataset and execute the three feature representation methods on them. Here are our results below:

```
full test with train sample
first 20w:                          last 20w:
    unigram : 91%                       unigram : 85.5%
    bigram : 86.9%                      bigram : 88.2%
    tfidf : 85.5%                       tfidf : 85.3%

first 50w:                          last 50w:
    unigram : 87.9%                     unigram : 87%
    bigram : 87.9%                      bigram : 85.2%
    tfidf : 85.7%                       tfidf : 85.2%

first 80w:                          last 80w:
    unigram : 87.1%                     unigram : 86.9%
    bigram : 87.9%                      bigram : 88.0%
    tfidf : 85.4%                       tfidf : 85.1%
```

Figure 8: Trainsample results

Note: for the general consideration, we sample the training data from head and from tail respectively.

(c) The third dimension is the test data sampling. Here we obtained 20%,50% and 80% sample data from the whole test dataset. The results are listed below:

```
full train with test sample
first 20%:
    unigram : 86.4%
    bigram : 88.0%
    tfidf : 85.1%

first 50%:
    unigram : 83.5%
    bigram : 87.3%
    tfidf : 85.0%

first 80%:
    unigram : 83.6%
    bigram : 87.9%
    tfidf : 85.6%
```

Figure 9: Testsample results

Note: since we have tested influences by the different parts in same size in (b), we only obtained data from head rather than from both head and tail.

Comparison and Analysis:

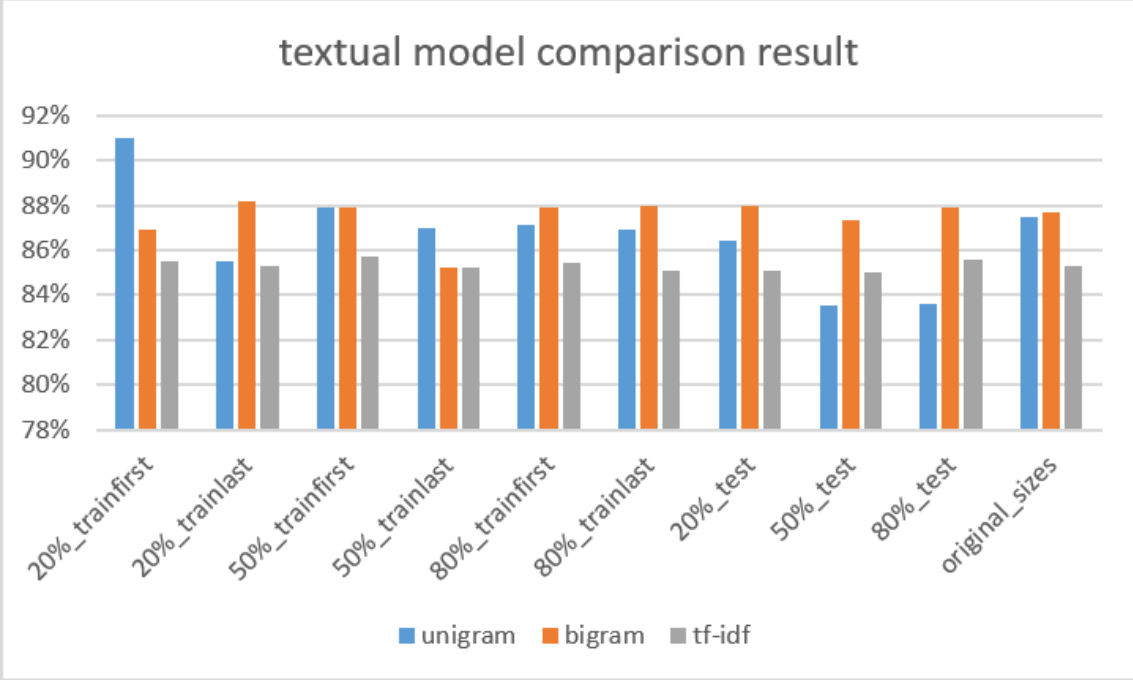

Figure 10: Comparison results

Through the results, we can see unigram accuracies change relatively dramatically according to the size of datasets(91% → 87.9% → 87.1% in training samples), or vary a lot in different parts data(91% → 85.5% for both 20% training data sample), while the other two feature representation models, have quite stable accuracies varying in the range of 1%. On the other hand, from the results we can see that bigram models' accuracies are all a little higher than tfidf models(nearly 88% > 85.5%). So that's the reason why we think the bigram model should be the best one to represent text features.

Theoretically, we think it's reasonable to view the bigram as the best text representation model in this problem. Firstly, the unigram model, as mentioned in the first part, lacks many features like the dependencies, because it just ignores the whole relations among words, which is therefore worse than any other n-gram models. And that can also help explain why unigram results change dramatically. Secondly, on the basis of unigram models, tfidf model compensates the potential errors caused by some high or low frequency words, which makes the accuracy better. Lastly, bigram model introduces more features in, like not only some words, but also some combinations of words, which obviously make accuracy higher. Specially, trigram model should have best performance and 7-gram model will cause over-fitting in theory. So the conclusion that bigram model is better than unigram is correct theoretically. Besides, tfidf, as analyzed, can indeed improve the performance of pure unigram, but it is another improvement way when compared to bigram. So theoretically it's hard to compare they two. But for this specific dataset, bigram does have higher accuracy.

(iv)

We designed feature sort and export algorithm to find the results. It's fairly direct since we have obtained the weights for each word in part(ii), using unigram model, therefore we only need to sort the weights and export the first 10 words with lowest weights and last 10 words with highest weights. Our results are listed below:

Table 1: Top 10 words with lowest weights

rank	words	weights
1	worst	-280.1044976602669
2	mediocre	-251.8375650533237
3	flavorless	-249.08249979695222
4	meh	-241.13114999906318
5	horrible	-219.3268722549529
6	lacked	-211.9581534539957
7	bland	-207.3571794503355
8	poisoning	-206.93256008646708
9	tasteless	-201.37647209501384
10	underwhelmed	-197.68080919149577

Table 2: Top 10 words with highest weights

rank	words	weights
1	disappoint	221.36364573506356
2	perfection	211.2971648309081
3	gem	178.84403758567026
4	heaven	174.8382179294144
5	superb	173.8517565178276
6	incredible	168.3170135135978
7	worried	154.9455488844878
8	minor	151.30382166798918
9	phenomenal	148.33180162563022
10	fantastic	141.85802912639556

From the above two tables, we can see that most of the words are reasonably positioned, except for a very strange one: disappoint.

If a word has a very low weight, it means the word contribute a lot to make the comment a negative one, otherwise to make the comment a positive one. So the results are saying the same story: for the lowest weights words, they are all negative ones.

Plus, we want to add a little explanations to the wired "disappoint". Many people left the comments like: "they didn't disappoint me." so that we can easily get the positive "disappoint" word here. I also list some specific examples below for proof:

my wife and i went to check full house bbq and we were not disappointed

do yourself a favor and don t bother getting the pommes frites sans truffles upgrade and you won t be disappointed

Problem 6

(i)

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\begin{aligned} \Rightarrow \frac{\partial \sigma(x)}{\partial x} &= -\frac{1}{(1+e^{-x})^2} \cdot -e^{-x} \\ &= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\ &= \sigma(x)(1-\sigma(x)) \end{aligned}$$

(ii)

$$\begin{aligned} E(W,b) &= \frac{1}{2n}\sum_i ||\sigma(W^T x_i + b) - y_i||^2 \\ &= \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{d_O}(\sigma(\vec{w_j}^T x_i + b_j) - y_{ij})^2 \end{aligned}$$

$$\frac{\partial E}{\partial \vec{w_j}} = \frac{1}{n}\sum_i (\sigma(\vec{w_j}^T x_i + b_j) - y_{ij}) \cdot \sigma(\vec{w_j}^T x_i + b_j) \cdot (1 - \sigma(\vec{w_j}^T x_i + b_j)) \cdot \vec{x_i} \tag{4}$$

$$\Rightarrow \frac{\partial E}{\partial W} = \begin{pmatrix} \frac{\partial E}{\partial \vec{w_1}} \\ \vdots \\ \frac{\partial E}{\partial \vec{w_O}} \end{pmatrix}^T, \text{ where } \frac{\partial E}{\partial \vec{w_j}} \text{ is shown in } (1), j = 1, 2, \ldots, d_O$$

$$\frac{\partial E}{\partial b_j} = \frac{1}{n}\sum_i (\sigma(\vec{w_j}^T x_i + b_j) - y_{ij}) \cdot \sigma(\vec{w_j}^T x_i + b_j) \cdot (1 - \sigma(\vec{w_j}^T x_i + b_j)) \tag{5}$$

$$\Rightarrow \frac{\partial E}{\partial b} = \begin{pmatrix} \frac{\partial E}{\partial b_1} \\ \vdots \\ \frac{\partial E}{\partial b_O} \end{pmatrix}, \text{ where } \frac{\partial E}{\partial \vec{b_j}} \text{ is shown in } (2), j = 1, 2, \ldots, d_O$$

(iii)

Update through iterations for hidden layer is not completely the same with output layer: Suppose layer $\mathcal{N}^1$ contains k neutrons

Let h_i denotes the output of the i-th neutron in $\mathcal{N}^1$ and h denotes output of layer $\mathcal{N}^1$

$$\hat{y} = \sigma(W_2^T h + b_2)$$
$$h_i = \sigma(W_{1i} \cdot x + b_{1i})$$

Then

$$\begin{aligned} \Rightarrow \frac{\partial E}{\partial W_{1i}} &= (\hat{y} - y) \cdot \hat{y}(1 - \hat{y}) \cdot W_{2i} \cdot h_i(1 - h_i) \cdot x \\ &= \frac{\partial E}{\partial b_2} \cdot W_{2i} \cdot \sigma(W_{1i} \cdot x + b_{1i})(1 - \sigma(W_{1i} \cdot x + b_{1i})) \cdot x \\ \frac{\partial E}{\partial b_{1i}} &= \frac{\partial E}{\partial b_2} \cdot W_{2i} \cdot \sigma(W_{1i} \cdot x + b_{1i})(1 - \sigma(W_{1i} \cdot x + b_{1i})) \end{aligned}$$

Then, just follow the step in pseudocode to update parameter in each iterations to implement this neural network.

(iv)

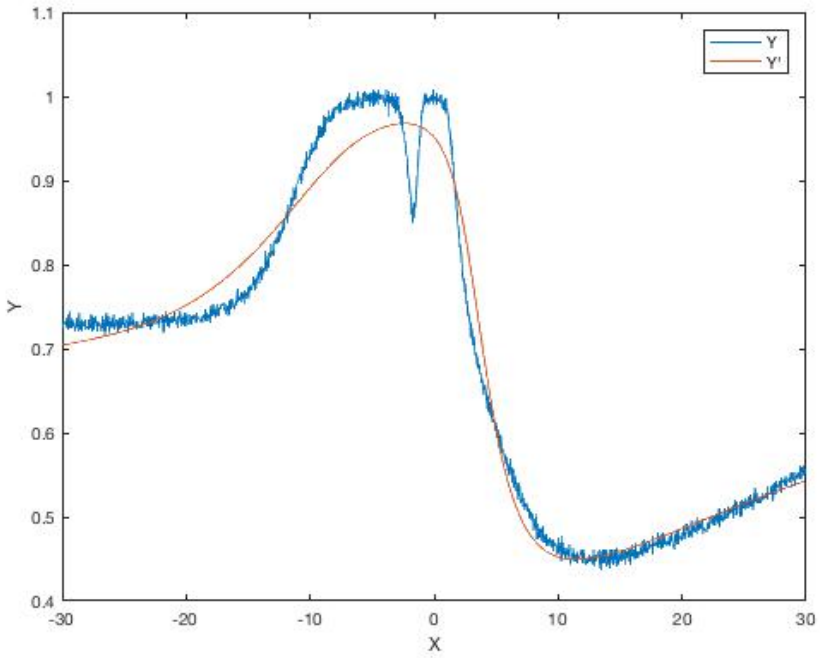

Figure 11: network output Y' and given Y for each X