Exercise 1

1. Recall that the objective of Logistic Regression is

$$\min_{w\in\mathbb{R}^d} \sum_i \log(1+e^{-y_i w^T x_i}) + \lambda\|w\|_2^2$$

For Kernel logistic regression, we want to compute this over the feature map of x, $\Phi(x)$, ie

$$\min_{w\in\mathbb{R}^d} \sum_i \log(1+e^{-y_i w^T \Phi(x_i)}) + \lambda\|w\|_2^2$$

By Representer Theorem, we have that the optimal w is

$$w = \sum_i \alpha_i \Phi(x_i)$$

So our objective is

$$\min_{\alpha\in\mathbb{R}^n} \sum_i \log\left(1+e^{-y_i (\sum_j \alpha_j \Phi(x_j))^T \Phi(x_i)}\right) + \lambda\|\sum_i \alpha_i \Phi(x_i)\|_2^2$$

Consider

$$(\sum_j \alpha_j \Phi(x_j))^T \Phi(x_i) = \sum_j \alpha_j \Phi(x_j)^T \Phi(x_i)$$

$$= \sum_j \alpha_j k(x_i, x_j) = \alpha^T \sum_j K(x_i, x_j) = \alpha^T K_i, \quad (K_i = i^{th} \text{ row/col of Kernel matrix})$$

Consider

$$\|\sum_i \alpha_i \Phi(x_i)\|_2^2 = \sum_j^n (\sum_i \alpha_i \Phi(x_i))^2 = \sum_j^n \sum_k \sum_\ell \alpha_k \Phi(x_k) \Phi(x_\ell) \alpha_\ell$$

$$= \sum_j^n \sum_k \sum_\ell \alpha_k K(x_k, x_\ell)\alpha_\ell = \sum_j^n \alpha^T K \alpha = n\alpha^T K \alpha$$

So our objective simplifies to

$$\min_{\alpha\in\mathbb{R}^n} \sum_i \log(1+\exp(-y_i \alpha^T K_i)) + \lambda n \alpha^T K \alpha$$

Which we can further simplify by setting $\lambda = \lambda n$, giving

$$\min_{\alpha\in\mathbb{R}^n} \sum_i \log(1+\exp(-y_i \alpha^T K_i)) + \lambda \alpha^T K \alpha$$

nt

Scanned with CamScanner

2. First, we need the gradient of our objective function:

$$\nabla \log(1+\exp(-y_i \alpha^T K_i)) + \lambda \alpha^T K \alpha$$

$$= \frac{1}{1+\exp(t)} \exp(t)(-y_i K_i) + 2\lambda \alpha K \Big|_{t = y_i \alpha^T K_i}$$

$$= -\sigma(-t)(-y_i K_i) + 2\lambda\alpha k,$$ where $\sigma(t)$ is our sigmoid function $\frac{1}{1+e^{-t}} = \frac{e^t}{1+e^t}$

$$= -y_i K_i + \sigma(t)(-y_i K_i) + 2\lambda\alpha k$$

$$= \begin{cases} (p_i - 0)K_i + 2\lambda\alpha k, & y_i = -1 \\ (p_i - 1)K_i + 2\lambda\alpha k & y_i = 1 \end{cases}$$

① $$= K_i^T\left(p_i - \frac{y_i+1}{2}\right) + 2\lambda\alpha k,$$ where $p_i = \frac{1}{1+\exp(-y_i\alpha^T K_i)}$

, similar to the gradient in regular logistic regression (note 4.12)

Then, our stochastic gradient descent function is:

$\vec{\alpha} = \vec{0}$ //intialize α

for $t = 0 \ldots$ maxIter:

  sample a minibatch $I = \{i_1 \ldots i_m\} \subseteq \{1, \ldots, n\}$

  $g = 0$

  for i in I

   $g += g_i$, where g_i is our gradient, as derived above in ①

  $\alpha = \alpha - \eta g$

  if $\|\eta g\| \le tol$, break

return α

where η is our step size

 α is the weights we compute

 tol is our tolerance

 g is our stochastic gradient.

3.

In the end, I never got my SGD algorithm to work. I suspect it might have something to do with how I derived/implemented my gradient, but after hours of debugging and checking my work, I'm stumped as to where my mistake might be.

Some peculiar behavior:

- My logistic loss actually INCREASES as I run SGD.
- If the step size is too small, then the gradient is always positive
- My test score sometimes dips BELOW 0.5

Anyways, I'm submitting my code anyways because I still spent a fuckload of time implementing it. If you want to take a look, and tell me where I went wrong, or give me pity marks for the scaffolding stuff I did implement, I'd appreciate it?

To run, make sure the training and testing datasets (csv) are in the same directory as **Exercise1.py**, and run **python Exercise1.py**

Some results from briefly letting it run 50 minutes before the deadline (I doubt I have enough time to let it completely finish running against all kernels, lambdas, and sigmas):

linear kernel
lambda = 0
Accuracy: 0.502
lambda = 10
Accuracy: 0.4965
lambda = 20
Accuracy: 0.5
lambda = 30
Accuracy: 0.5
lambda = 40
Accuracy: 0.5
lambda = 50
Accuracy: 0.5
lambda = 60
Accuracy: 0.5

Exercise 2

1. To show that $k(x,x') = \lim_{n\to\infty} k_n(x,x')$ is a kernel, we need to show that it's kernel matrix is symmetric and PSD.

Symmetric: Consider $k(x_i, x_j) - k(x_j, x_i)$

$$= \lim_{n\to\infty} k_n(x_i, x_j) - \lim_{n\to\infty} k_n(x_j, x_i)$$

$$= \lim_{n\to\infty} (k_n(x_i, x_j) - k_n(x_j, x_i))$$

, since we are given that the limit of k_n always exists & is finite, and that $\lim_{n\to\infty}(\alpha a_n + \beta b_n) = \alpha \lim_{n\to\infty} a_n + \beta \lim_{n\to\infty} b_n$ when limits exist (hint).

$= 0$, since we are given that k_n is a kernel, so $k_n(x_i, x_j) = k_n(x_j, x_i)$

$$k_n(x_i, x_j) - k_n(x_j, x_i) = 0$$

$\therefore$ $k(x_i, x_j) - k(x_j, x_i) = 0$

so $k(x_i, x_j) = k(x_j, x_i)$

so k is symmetric

PSD: Let K be the kernel matrix of $k(x_i, x_j)$ $i, j \in m$, m = training set size

Consider $\alpha^T K \alpha$, $\forall \alpha \in \mathbb{R}^m$.

$$= \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j K_{ij}$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k(x_i, x_j)$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \lim_{n\to\infty} k_n(x_i, x_j)$$

$$= \lim_{n\to\infty} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k_n(x_i, x_j)$$, by the given hint

Since we are given that k_n is a kernel, $\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k_n(x_i, x_j) \geq 0$, by definition.

Let this value be C_n

Then, $\lim_{n \to \infty} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k_n(x_i, x_j)$

$$= \lim_{n \to \infty} C_n$$

Since $C_n \geq 0$, and is guaranteed to exist & be finite,

$$\lim_{n \to \infty} C_n = C \geq 0$$

$\therefore$ $\alpha^T K \alpha \geq 0$

$\therefore$ K is PSD.

Since K is symmetric and PSD, $K(x_i, x_j) = \lim_{n \to \infty} k_n(x_i, x_j)$ is a kernel

2. Consider the Taylor expansion of e^k.

$$e^k = \sum_{n=0}^{\infty} \frac{k^n}{n!}$$

Since k is a kernel, k^n is also a kernel, since k^n is just repeated products of kernels, and products of kernels are also kernels.

$\frac{k^n}{n!}$ is also a kernel, since the product of a kernel & a constant $\lambda \geq 0$ (here $\lambda = 1/n!$) is a kernel

$\sum_{n=0}^{m} k^n/n!$, $m \in \mathbb{N}$ is also a kernel, since the sums of kernels are also kernels

c. Consider $\lim_{m\to\infty} \sum_{n=0}^{m} \frac{K^n}{n!}$. This is, by definition, the taylor expansion of e^K, which we know converges to e^K. Since $\sum_{n=0}^{m} K^n/n!$ is a kernel, and this limit converges, we can apply the theorem in part 1.

$\therefore \lim_{m\to\infty} \sum_{n=0}^{m} \frac{K^n}{n!} = \sum_{n=0}^{\infty} \frac{K^n}{n!} = e^K$ is a kernel.

3. Let us prove that $K(x, x') = e^{-\|x-x'\|_2^2/\sigma}$ is a kernel by showing that it can be constructed from base kernels & kernel calculus.

First, consider $\|x - x'\|_2^2 = \sum_{i=1}^{d} (x_i - x_i')^2$

$$= \sum_{i=1}^{d} x_i^2 - 2x_i' x_i + x_i'^2$$

$$= \sum_{i=1}^{d} x_i^2 - 2\sum_{i=1}^{d} x_i' x_i + \sum_{i=1}^{d} x_i'^2$$

$$= x^T x - 2x^T x' + x'^T x'$$

So $$e^{-\|x-x'\|_2^2/\sigma} = e^{\frac{2}{\sigma}x^T x' - \frac{1}{\sigma}x^T x - \frac{1}{\sigma}x'^T x'}$$

$$= e^{-x^T x/\sigma}\, e^{\frac{2}{\sigma}x^T x'}\, e^{-x'^T x'/\sigma} \quad (\text{since } a^{b+c} = a^b a^c)$$

Note that $x^T x'$ is a polynomial kernel, with $p=1$ (i.e. a linear kernel), so it is a valid kernel
Then, $\frac{2}{\sigma} x^T x'$ is also a valid kernel (scaling by a constant)
From part 2, we proved that if k is a valid kernel, e^k is also a valid kernel

$\therefore e^{\frac{2}{\sigma}x^T x'}$ is also a valid kernel

Let $\Phi_1(x)$ be the feature map of this kernel $e^{\frac{2}{\sigma}x^T x'}$, i.e.

$$\Phi_1(x)^T \Phi_1(x') = e^{\frac{2}{\sigma}x^T x'}$$

Then, we can define another feature map

$$\Phi_2(x) = e^{-x^T x/\sigma} \Phi_1(x)$$

Then, the kernel of this feature map

$$\Phi_2(x)^T \Phi_2(x') = e^{-x^T x/\sigma} \Phi_1(x)^T \Phi_1(x') e^{-x'^T x'/\sigma}$$

$$= e^{-x^T x/\sigma} e^{2x^T x'/\sigma} e^{-x'^T x'/\sigma}$$

$$= e^{-\|x-x'\|_2^2/\sigma}$$

which is our gaussian density function.

$\therefore$ $k(x, x') = e^{-\|x-x'\|_2^2/\sigma}$ is a kernel, $\forall \sigma > 0$.

Scanned with CamScanner

Exercise 3:

1. For the graph in Figure 1, there are 2 paths:

1) $s \to a \to b \to d \to t$

2) $s \to a \to c \to d \to t$

Then, $K_{p_1}(x,z) = K_{s\to a}(x,z)\, K_{a\to b}(x,z)\, K_{b\to d}(x,z)\, K_{d\to t}(x,z)$

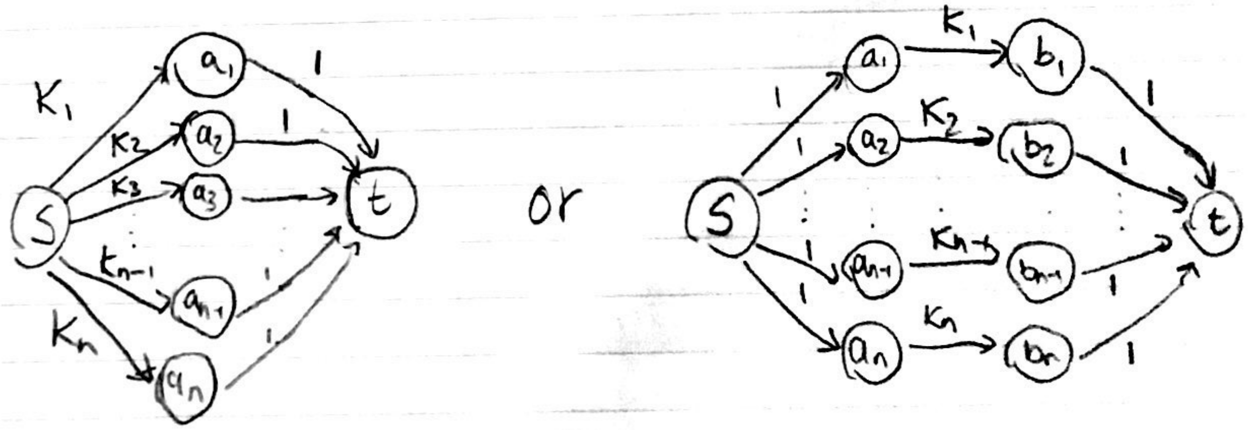
$$K_{p_1}(1,-1) = (1)\cdot(1\cdot -1)\cdot(e^{-(1-(-1))^2})\cdot(1)$$
$$= (1)\cdot(-1)\cdot(e^{-4})(1)$$
$$= -e^{-4}$$
$$\approx 0.0183156$$

$$K_{p_2}(x,z) = K_{s\to a}(x,z)\, K_{a\to c}(x,z)\, K_{c\to d}(x,z)\, K_{d\to t}(x,z)$$
$$K_{p_2}(1,-1) = 1\cdot(1+(1)(-1))^2\,(e^{-|1-(-1)|})\cdot 1$$
$$= 0$$

$\therefore K_G(x,z) = \sum_P K_P(x,z) = K_{p_1}(x,z) + K_{p_2}(x,z)$

$$K_G(1,-1) = -e^{-4} + 0 = \boxed{-e^{-4}}$$

2. Since we want $K_G = \sum k_i$, and by definition of $K_G = \sum_{Paths} k_i$, we simply place each n kernel on a different path, e.g.



3. Since we want $K_G = \prod k_i$, and by definition, $K_P = \prod_{i \in P} k_i$, we simply construct a graph w/ one path, and all kernels on that path, i.e.:

$$s \xrightarrow{k_1} a_1 \xrightarrow{k_2} a_2 \xrightarrow{k_3} \cdots \xrightarrow{k_{n-1}} a_{n-1} \xrightarrow{k_n} t$$

4. Note that $K_G(x, z)$ is essentially the sum of the products of the values of the base kernels in every path.
$\therefore$ we can reduce this problem to finding the sum of the products of the edge weights ($k_i(x,z)$) of all paths. We can achieve this with topological sorting & dynamic programming.
We define $SP[v]$ as the sum of the products of the edge weights in all the paths up to node v.
Then,

$$SP[v] = \sum_{\substack{\text{all incoming} \\ \text{edges } (u,v)}} SP[u] \cdot \underbrace{K_{(u,v)}(x,z)}_{\text{edge weight}}$$

This is true since if $SP[u]$ is the sum of i paths $(P_1 + P_2 + P_3 \cdots + P_i)$, then $SP[u] \cdot K_{(u,v)}(x,z)$ $= (P_1 + P_2 + \cdots P_i) K_{u,v}(x,z) = P_1 K_{u,v} + P_2 K_{u,v} + \cdots + P_i K_{u,v}$ $= (P_1' + P_2' + \cdots P_i')$, so the sum of products property is preserved.
Then, $SP[s] = 1$, since s is our source node w/ no incoming edges, and $SP[t] = K_G$.
If we sort the nodes by topological order, and fill in SP in this order, we are guaranteed that $\forall$ incoming edges (u, v), $SP[u]$ is calculated (by defn of topological order).
Since topological sorting is $O(|V| + |E|)$, and we go through all $|V|$ nodes and $|E|$ edges (each edge must be used once to calculated $SP[t]$), the total runtime is $O(|V| + |E|)$

In PseudoCode:

CalculateKG($G = \{V, E\}$):

    TopologicalSort(V)

    $SP[s] = 1$

    for v in V:

        $SP[v] = 0$

        for edge (u,v) in all incoming edges to v:

$$SP[v] \mathrel{+}= \underbrace{SP[u]}_{\substack{\text{guaranteed} \\ \text{to exist b/c} \\ \text{topological ordering}}} \cdot \underbrace{K_{(u,v)}(x,z)}_{\text{edge weight}}$$

    return $SP[t]$