元　　智　　大　　學

資 訊 工 程 學 系

專題製作成果報告

Privacy Preserving Local Support Vector Machine(SVM) with Secret Sharing

專 題 生:江仕瑄

學　　號:1071527

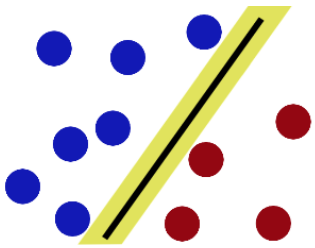
指導教授:陳昱圻 教授

中華民國　111 年 1 月

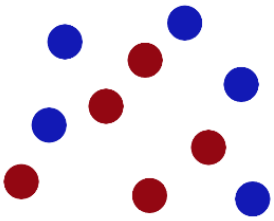
目錄

一、摘要

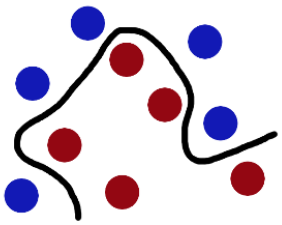
Support Vector Machine(SVM)是一種監督式的學習方法，用統計風險最小化的原則 來估計一個分類的超平面(hyperplane)，其基礎的概念非常簡單，就是找到一個決策邊界(decision boundary)讓兩類之間的邊界(margins)最大化，使其可以完美區隔開來，而會影響到邊界的點，即為 support vector。以下圖(一)而言，我們要透過 SVM 找到的就是那條黑色直線，使它能讓我們成功的分類成藍色區與紅色區。若是遇到圖(二)情況，我們可以用一曲線分割，而這條曲線在立體空間中，可視為用一平面分割，稱為超平面(hyperplane)。



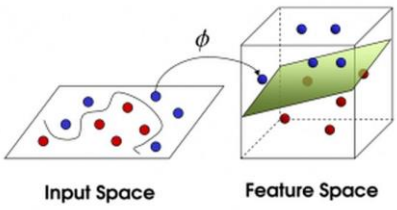
圖(一)



圖(二)



圖(三)



圖(四)

而我們打算在有一個完好的 SVM 的基礎上，添加演算法使其達到更好的 privacy preserving 效果。

二、研究目的

我們的目的是要使 SVM 達到一個更好的 Privacy Preserving 效果。假設在有一個不受信任的第三方下，如何安全的計算 global SVM model，又不會將有關雙方的數據外漏是我們本次要解決的問題。

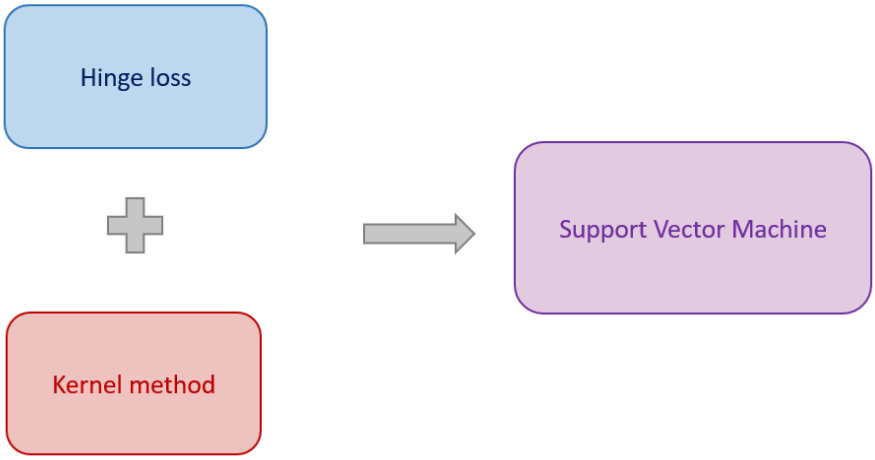
要達成這個目標，首先，我們先要有一個可以正常運作的 local SVM，原本我們打算利用市面上已經做好的 local SVM model 去增加防護，但很遺憾的是，我們大多看到的都是 global kernel 版本的 SVM，並沒有找到一個合適的選項，所以我們會先撰寫一個 local SVM model。我們將會使用 Python 語言於 Windows 環境下撰寫。以下是這個 local SVM model 的架構:每個參與者計算出各自的 local kernel，傳到 cloud 後，計算出 global Kernel，再透過 SVM 演算法計算出 SVM model。

有了這個利用 local Kernel 實作的 Linear SVM 後，我們將會在合併 global Kernel 時使用亂數矩陣及 Shamir’s Secret Sharing Algorithm 來達到更好的 privacy preserving。

三、文獻探討

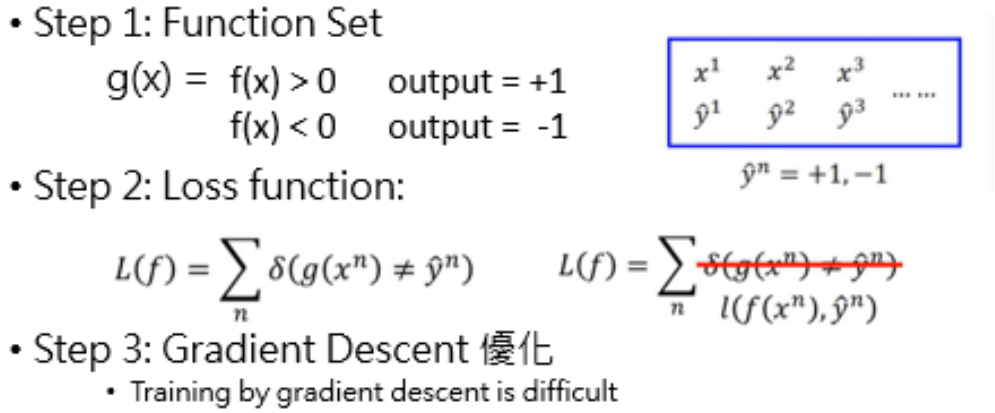
(一)Support Vector Machine (SVM)

SVM 中有兩個特色:Hinge loss & Kernel method



Binary Classification

而一個 machine learning 中，通常有三個步驟:



首先第一步，訂一個 function g(x) 裡面有個 f(x)，這邊可以假設他的 output 為一個大於 0 時為正 1 的 class，和另一個小於 0 時為負 1 的 class。在 Training data 中，每個 xˆn 都有個 yˆn hat，每個 yˆn hat 裡的值皆用正 1 和負 1 表示，分別代表兩個不同

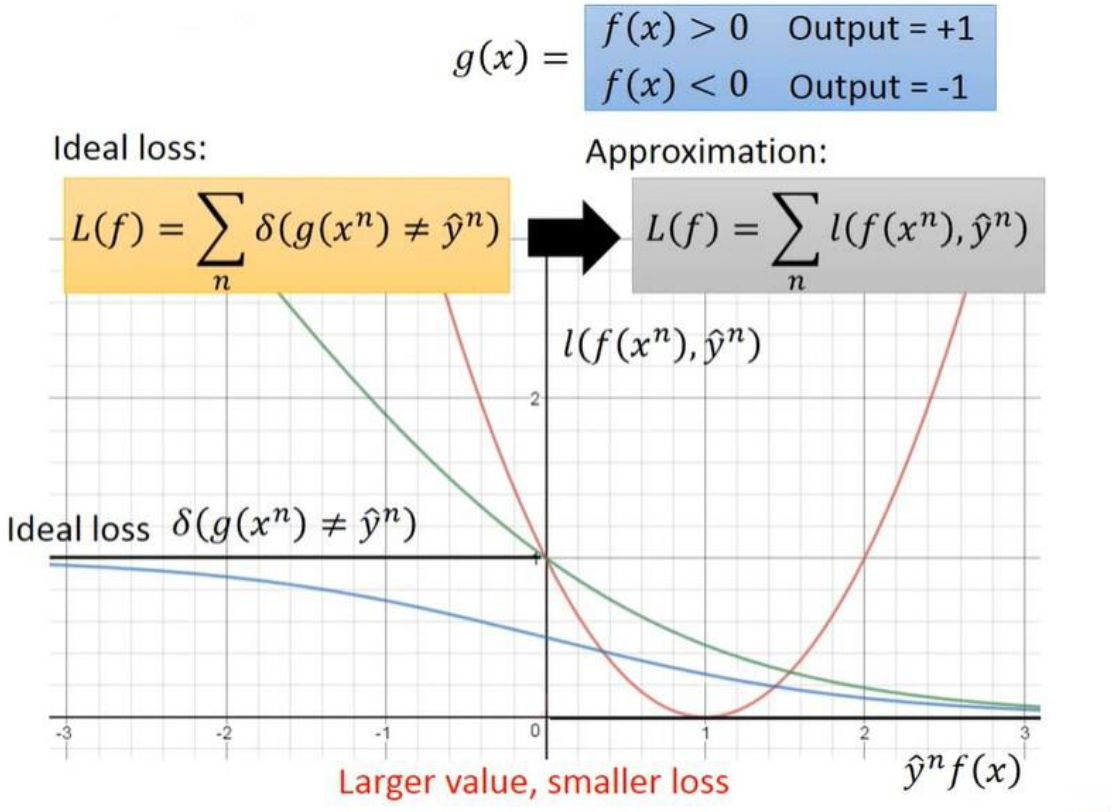
的 class。其中，這裡的正 1 及負 1 跟羅吉斯回歸的 1 及 0 是一樣的意思。

第二步，設定 loss function。當 g(xˆn)跟 yˆn hat(label)不一樣的時候，設定 loss 的值為 1，當 g(xˆn)跟 yˆn hat 一樣的時候，設定 loss 的值為 0。而這邊的 loss 是表示 g(xˆn)在 training set 上總共犯了多少次錯誤（越小越好）。

第三步，利用 Gradient descent 做優化。

Loss function

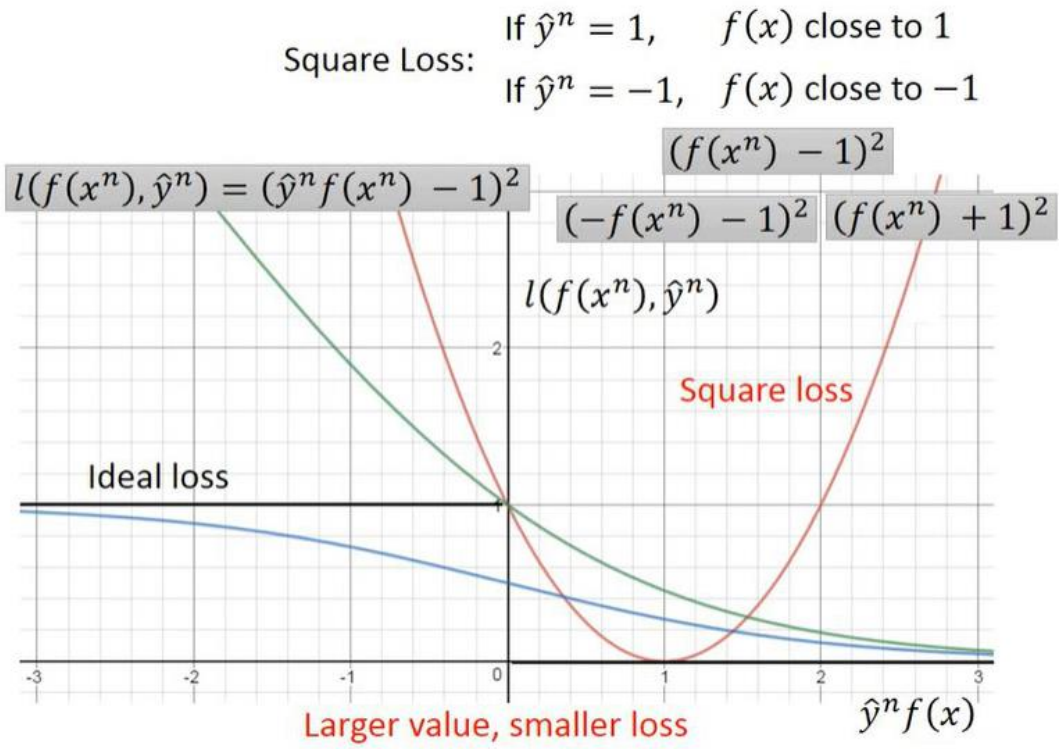
下圖是理想狀態下，縱軸為 loss，橫軸為 yˆn 乘上 f(x)的曲線圖:



由圖可知，在 yˆn 及 f(x)兩者同號時，相乘的結果越大越好，

越往右兩者相乘越大，其 loss 就越小，所以我們可以設，理想上當兩者不同的時候輸出為 0，相同的時候輸出為 1，但這種 loss 事實上是不可微分的，因此需要利用其它方式來表示 loss function。

Loss function - Square loss



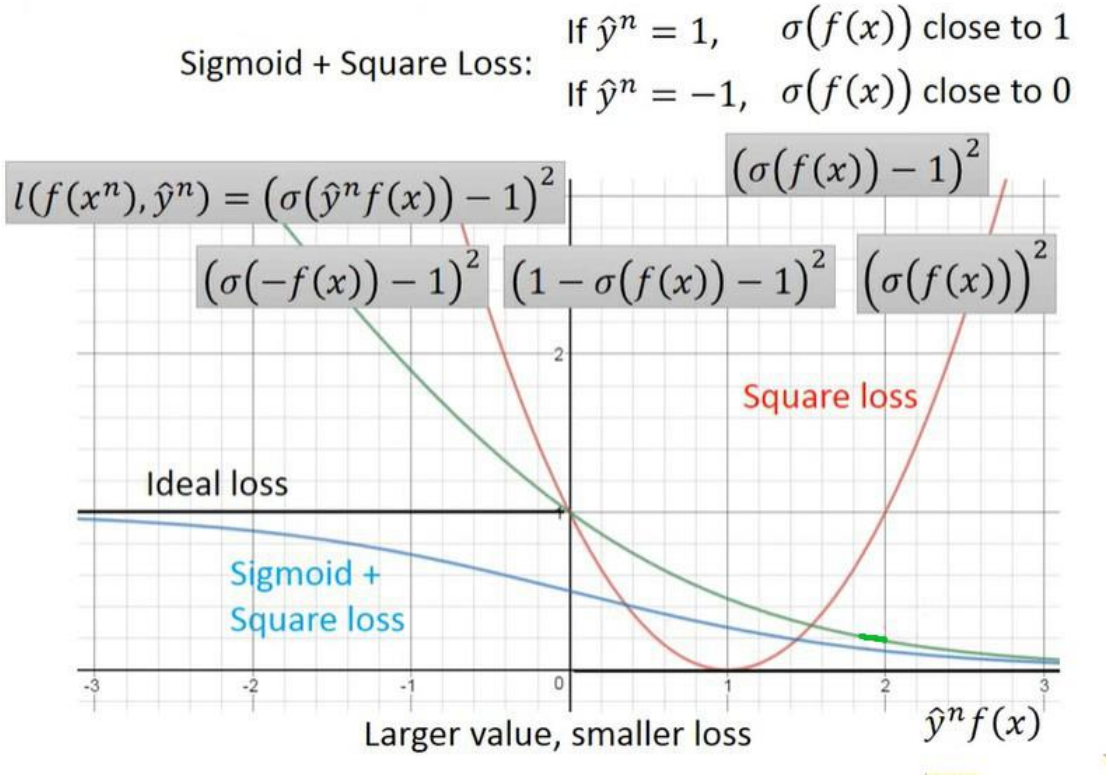
loss function: $l(f(x^n), \hat{y}^n) = (\hat{y}^n f(x^n) - 1)^2$

$\hat{y}^n = 1$，$f(x)$愈接近1愈好
- $(f(x^n) - 1)^2$

$\hat{y}^n = -1$，$f(x)$愈接近-1愈好
- $(-f(x^n) - 1)^2 = (f(x^n) + 1)^2$

此時發現使用 square loss(紅線)於 binary classification 是不合理的，橫軸 yˆnf(x)越大 竟然會有越大的 loss。

Loss function - Sigmoid + Square loss



loss function: $l(f(x^n), \hat{y}^n) = \sigma((\hat{y}^n f(x^n) - 1))^2$

$\hat{y}^n = 1$，$\sigma(f(x))$愈接近1愈好

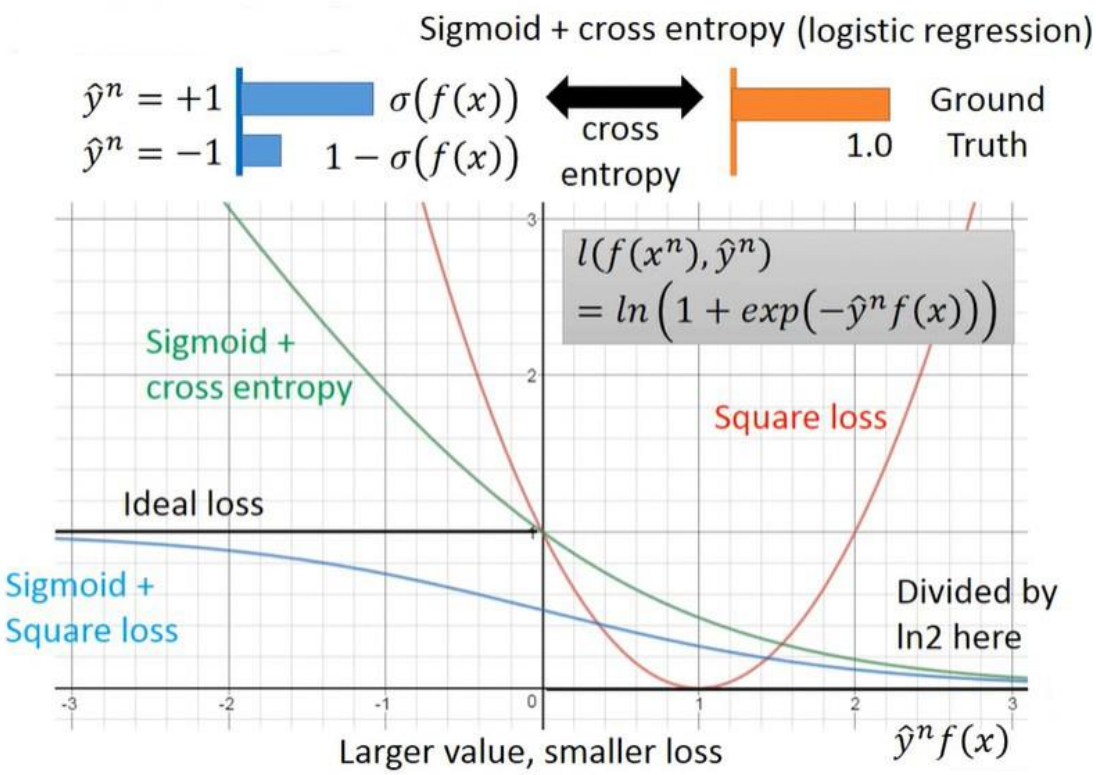
- $\sigma((f(x^n) - 1))^2$

$\hat{y}^n = -1$，$\sigma(f(x))$愈接近0愈好

- $\sigma((-f(x^n) - 1))^2 = (1 - \sigma(f(x)) - 1)^2 = (\sigma(f(x)))^2$

如藍線所示，但通常在 logistic regression 還是不會使用 square loss，因為 performance 不好，改用 cross entropy。

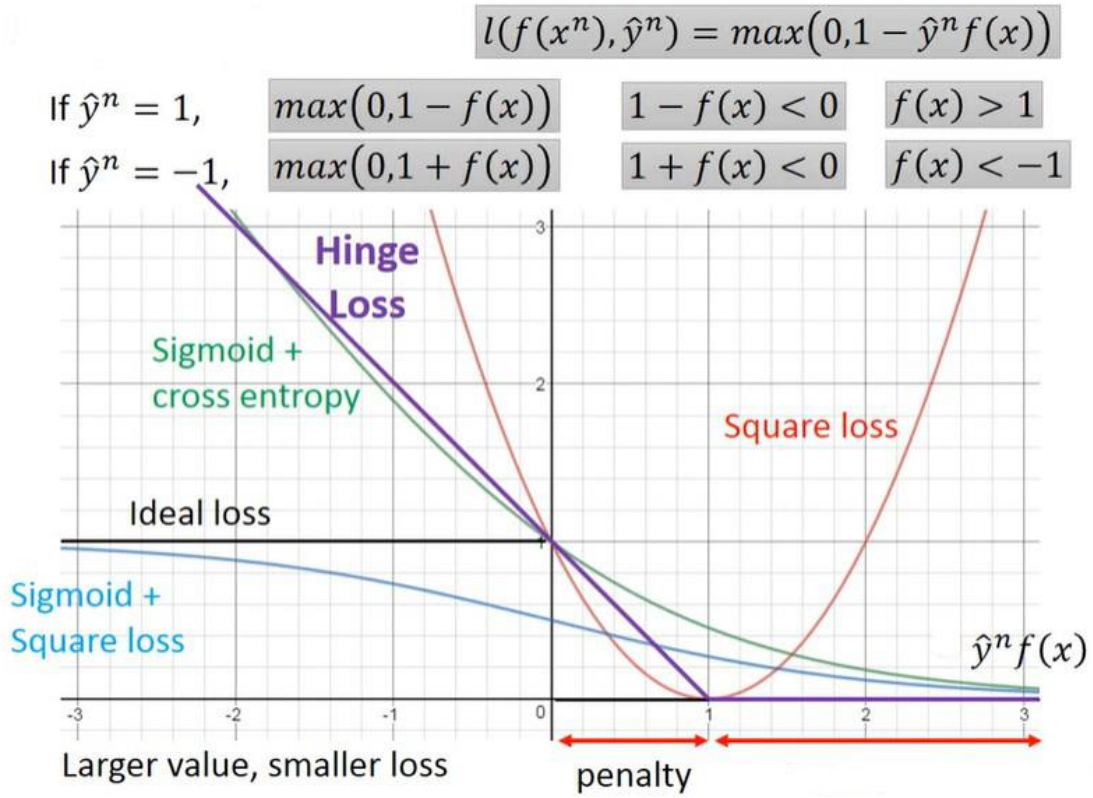
Loss function - Sigmoid + Cross entropy



loss function: $l(f(x^n), \hat{y}^n) = ln(1 + exp(-\hat{y}^n f(x)))$

在圖的上方，左邊藍色、右邊橘色，分別代表一個 distribution，而兩者之間的 cross entropy，就是要來 minimize 的 loss。(除以 log2 讓他變成 ideal 的 upper bound)minimize 他的 upper bound 時，用 cross entropy 而不是 square los 的原因在於，當值負很大的時候 gradient 的變化在 cross entropy 比較明顯，他才會樂意去調整值。

Loss function - Hinge loss



loss function: $l(f(x^n), \hat{y}^n) = max(0, 1 - \hat{y}^n f(x))$

$\hat{y}^n = 1$

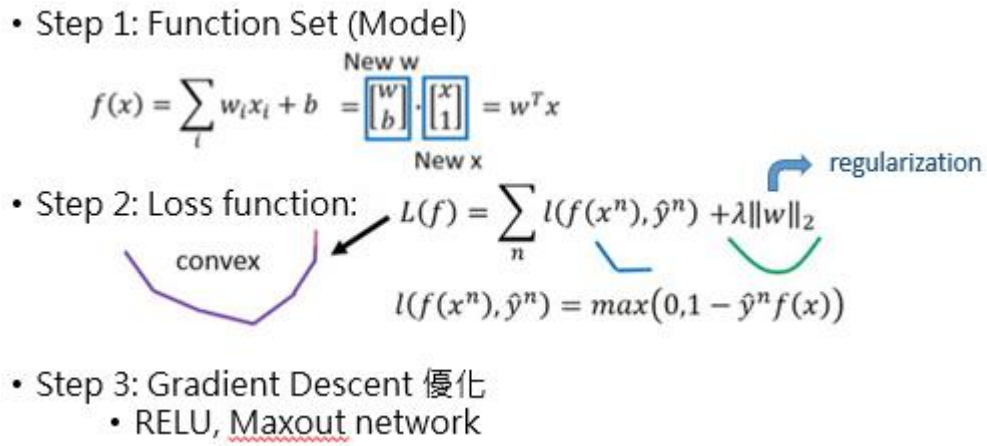
- $max(0, 1 - f(x))$
  - loss最小值為0
  - 當$1 - f(x) < 0$則loss為0
    - $f(x) > 1$
- $max(0, 1 + f(x))$
  - loss最小值為0
  - 當$1 + f(x) < 0$則loss為0
    - $f(x) < -1$

在這邊使用 1 去減的目的，是為了讓他成為 ideal loss 的一個 upper bound，並期待 minimize hinge loss 可能可以得到 minimize ideal 的效果。跟 Sigmoid + Cross entropy 比較 Sigmoid + Cross entropy 會想要一直變大、減少 loss，而 hinge

loss 則過了之後就都一樣了。一般來說，hinge loss 與 cross entroyp 之間的差異並沒有那麼顯著，但 hinge loss 會比較 robust。

Linear SVM

確認完 loss function 後，把前面提到的 machine learning 三步驟使用到 linear SVM 中:



第一步，x 裡面的每一個 feature xi 乘上他對應的 weight wi，summation 起來後再加上一個常數 b(誤差)，透過簡化可以看成兩個 vector 的 inner product。第一個 vector 用新的 w 表示，這是透過 training data 找出來的 model 參數，而另一個當作一個新的 feature。就可以把 Function 寫成 w 的 transpose 乘上 x(或為 w x 的 inner product)。

第二步，採用 hinge loss 再加上一個 regularisation 的 term。Regularisation 主要是當這個 model overfiting 的時候不

使他變得太奇怪，我在這邊把他理解成 margin。

第三步，這個 loss function 是一個 convex function（前項是 後項也是 疊加起來也是）Convex 做 gradient descent 就很簡單，不管在那個地方做 initialization，最後找出來的結果都會一樣。有些位子雖然不可微分，但可以用 gradient descent 去做 optimization。

Gradient descent:

Ignore regularization for simplicity

$$L(f)=\sum_n l(f(x^n),\hat{y}^n) \qquad l(f(x^n),\hat{y}^n)=max(0,1-\hat{y}^n f(x^n))$$

$$\frac{\partial l(f(x^n),\hat{y}^n)}{\partial w_i}=\frac{\partial l(f(x^n),\hat{y}^n)}{\partial f(x^n)}\frac{\partial f(x^n)}{\partial w_i}x_i^n \qquad f(x^n)=w^T\cdot x^n$$

$$\frac{\partial max(0,1-\hat{y}^n f(x^n))}{\partial f(x^n)}=\begin{cases}-\hat{y}^n & \text{If } \hat{y}^n f(x^n)<1 \\ & 1-\hat{y}^n f(x^n)>0 \\ 0 & \text{otherwise}\end{cases}$$

$$\frac{\partial L(f)}{\partial w_i}=\sum_n \underbrace{-\delta(\hat{y}^n f(x^n)<1)\hat{y}^n x_i}_{c^n(w)} \qquad w_i\leftarrow w_i-\eta\sum_n c^n(w)x_i^n$$

Linear SVM – Another formulation

將原式替代的另一個長相:

Minimizing loss function L:

$$L(f)=\sum_n \varepsilon^n +\lambda\|w\|_2$$

ε^n: slack variable

$$\varepsilon^n=max(0,1-\hat{y}^n f(x))$$

=

$$\varepsilon^n\geq 0$$

$$\varepsilon^n\geq 1-\hat{y}^n f(x) \Rightarrow \hat{y}^n f(x)\geq 1-\varepsilon^n$$

Quadratic Programming (QP) Problem

$$f(w)=\frac{1}{2}w^T w+C\sum_{i=1}^{m}\max(0,1-y_i w^T x_i)$$

s.t. $y_i(w^t x_i+b)\geq 1, i=1,2,\ldots,m.$

$$\arg\min_{\mathbf{w},b,\boldsymbol{\xi}}\frac{1}{2}\|\mathbf{w}\|^2+C\sum_{i=1}^{m}\xi_i$$

subject to $y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1-\xi_i, \xi_i\geq 0, i=1,\ldots,m$

Dual representation

對所有 training data 的 vector point xˆn 乘上 alpha(weight)，找出來的 model 就是 data point 的 linear combination。剛剛做完 gradient descent 後的式子 對 wl~wk 串成一個 vector w:

$$w^* = \sum_n \alpha_n^* x^n$$ Linear combination of data points

α_n^* may be sparse ➡ x^n with non-zero α_n^* are support vectors

$$w_1 \leftarrow w_1 - \eta \sum_n c^n(w) x_1^n$$
$$\vdots$$
$$w_i \leftarrow w_i - \eta \sum_n c^n(w) x_i^n$$
$$\vdots$$
$$w_k \leftarrow w_k - \eta \sum_n c^n(w) x_k^n$$

If w initialized as **0**

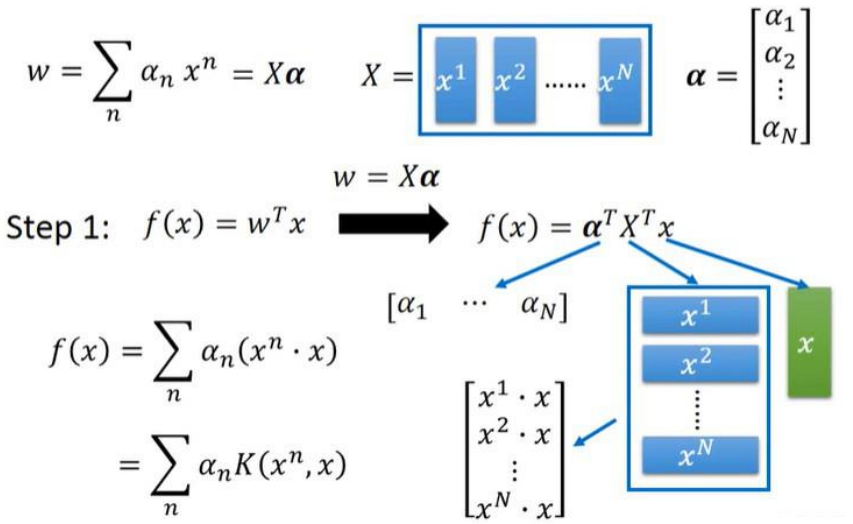
$$w \leftarrow w - \eta \sum_n c^n(w) x^n$$

$$c^n(w) = \frac{\partial l(f(x^n), \hat{y}^n)}{\partial f(x^n)}$$ Hinge loss: usually zero

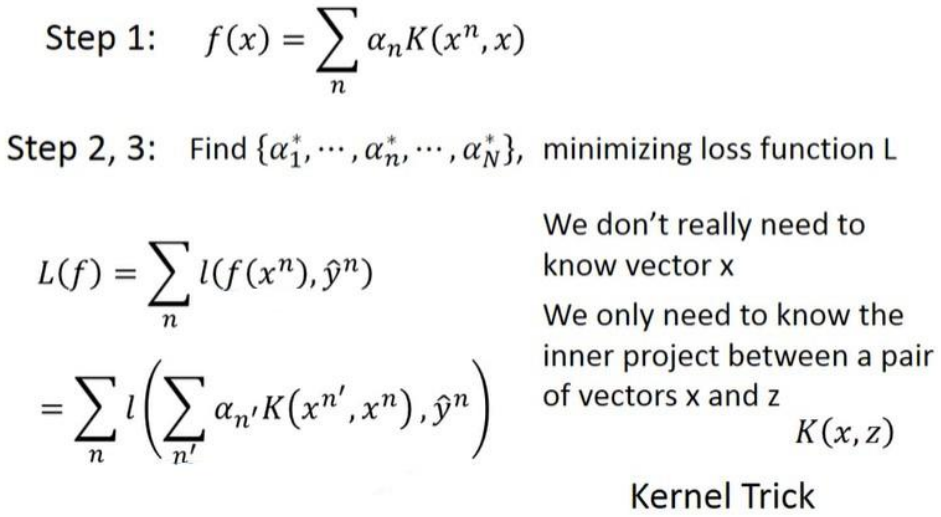
c.f. for logistic regression, it is always non-zero

每次 update w 的時候 都是加上 data point 的 linear combination，Cˆn(w)是 f(xˆn)對 loss function 的偏微分 如果他作用在 max=0 的 region，那他這項就會是 0，也就是說不是所有的 xˆn 都會被加到 w 裡面去，所以最後解出來的 w star 的這個 weight 有可能是 sparse(可能有很多的 data point 他對應的 alpha star 值等於 0)，而那些值不等於 0 的 xˆn 就是 support vector。

最後化簡後如下圖，X 是 x matrix、alpha 是 vector(X transpose: several rows,a transpose: vector)，三項乘完後會是一個 scalar。



寫成這樣的好處是可以使用 kernel trick:

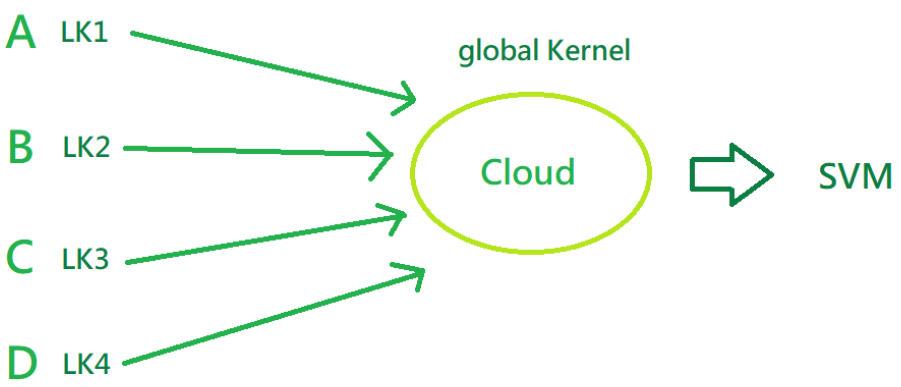


aˆn 是未知，若要找到一組最好的 aˆn 讓 total loss 最小，不需要知道 x 的 vector，要知道的只有這個(x,z)的 inner product 值。換句話說，只要知道 kernel function，就可以做所

有的 optimization。

四、研究方法

透過以上 SVM 推導，我們先使用 python 寫出了一個 Normal 的 SVM model(2-class)，其 kernel 為 global kernel。而我們需要的是能代表著不同參與者的 local data 所 training 出來的 local kernel 版本，架構如下:

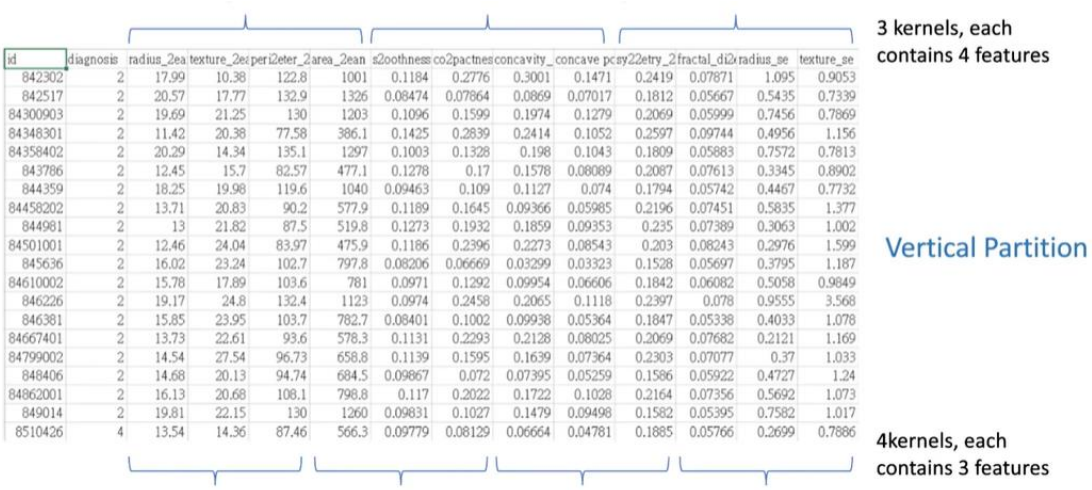


A、B、C、D 分別代表不同參與者，各自透過 training 後得到一個 local kernel(LK1, LK2, LK3, LK4)，再將 local kernel 傳送至 cloud，由 cloud 計算總合形成 global kernel，並使用這個 global kernel 去計算 alpha 值，有了 alpha 值後，就可用推導出的公式計算，並帶入 QP solution 解出 w 值及 b 值，完成 SVM model training。在計算 global kernel 的過程中，為了達到 privacy preserving，我們在每一個參與者生成 local kernel 時，再新增一個亂數矩陣，連同 local kernel 並使用 shamir’s

secret sharing(SSS)的方式傳給 cloud。

程式整體的流程為:

1. 資料分割: 將 dataset(m x n)透過垂直分割分成數筆 local data，代表不同的參與者所擁有的資料。如下圖，假設將資料切成 3 個參與者，每個參與者擁有 4 個特徵值；若將資料切成 4 個參與者，每個參與者擁有 3 個特徵值。



id	diagnosis	radius_2ea	texture_2ea	peri2eter_2	area_2ean	s2oothness	co2pactnes	concavity_	concave pc	sy22etry_2	fractal_di2	radius_se	texture_se
842302	2	17.99	10.38	122.8	1001	0.1184	0.2776	0.3001	0.1471	0.2419	0.07871	1.095	0.9053
842517	2	20.57	17.77	132.9	1326	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339
84300903	2	19.69	21.25	130	1203	0.1096	0.1599	0.1974	0.1279	0.2069	0.05999	0.7456	0.7869
84348301	2	11.42	20.38	77.58	386.1	0.1425	0.2839	0.2414	0.1052	0.2597	0.09744	0.4956	1.156
84358402	2	20.29	14.34	135.1	1297	0.1003	0.1328	0.198	0.1043	0.1809	0.05883	0.7572	0.7813
843786	2	12.45	15.7	82.57	477.1	0.1278	0.17	0.1578	0.08089	0.2087	0.07613	0.3345	0.8902
844359	2	18.25	19.98	119.6	1040	0.09463	0.109	0.1127	0.074	0.1794	0.05742	0.4467	0.7732
84458202	2	13.71	20.83	90.2	577.9	0.1189	0.1645	0.09366	0.05985	0.2196	0.07451	0.5835	1.377
844981	2	13	21.82	87.5	519.8	0.1273	0.1932	0.1859	0.09353	0.235	0.07389	0.3063	1.002
84501001	2	12.46	24.04	83.97	475.9	0.1186	0.2396	0.2273	0.08543	0.203	0.08243	0.2976	1.599
845636	2	16.02	23.24	102.7	797.8	0.08206	0.06669	0.03299	0.03323	0.1528	0.05697	0.3795	1.187
84610002	2	15.78	17.89	103.6	781	0.0971	0.1292	0.09954	0.06606	0.1842	0.06082	0.5058	0.9849
846226	2	19.17	24.8	132.4	1123	0.0974	0.2458	0.2065	0.1118	0.2397	0.078	0.9555	3.568
846381	2	15.85	23.95	103.7	782.7	0.08401	0.1002	0.09938	0.05364	0.1847	0.05338	0.4033	1.078
84667401	2	13.73	22.61	93.6	578.3	0.1131	0.2293	0.2128	0.08025	0.2069	0.07682	0.2121	1.169
84799002	2	14.54	27.54	96.73	658.8	0.1139	0.1595	0.1639	0.07364	0.2303	0.07077	0.37	1.033
848406	2	14.68	20.13	94.74	684.5	0.09867	0.072	0.07395	0.05259	0.1586	0.05922	0.4727	1.24
84862001	2	16.13	20.68	108.1	798.8	0.117	0.2022	0.1722	0.1028	0.2164	0.07356	0.5692	1.073
849014	2	19.81	22.15	130	1260	0.09831	0.1027	0.1479	0.09498	0.1582	0.05395	0.7582	1.017
8510426	4	13.54	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.04781	0.1885	0.05766	0.2699	0.7886

2. SVM + SSS:

(1) 分別計算 local data 的 local kernel。

```
LK[i] = y * x[i]                    #K = y * x
LK2[i] = np.dot(LK[i], LK[i].T)     #K = np.dot(K, K.T)
```

(2) 每個 local kernel 生成一個隨機的 random matrix(ex: 參與者 A 生成 r1, 參與者 B 生成 r2,……)。

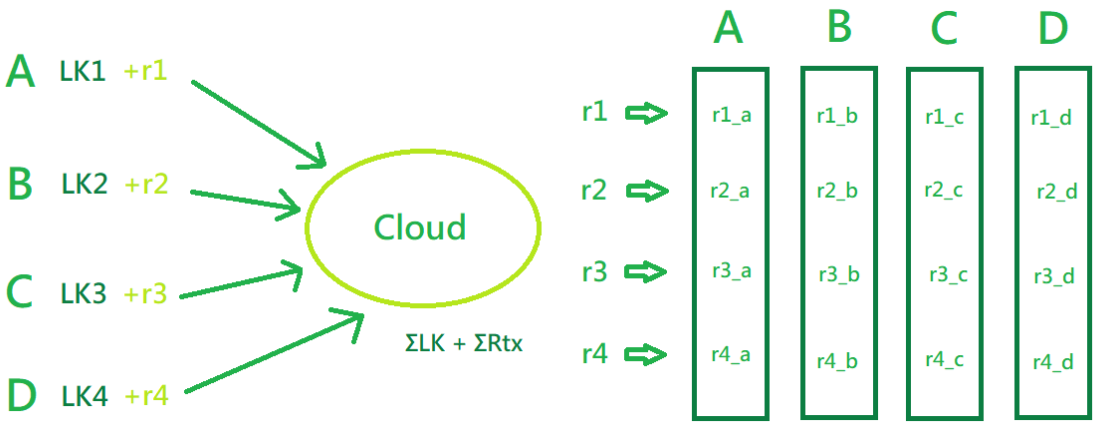
(3) 每個參與者將各自的 local kernel 和 random matrix 透過 Shamir’s Secret Sharing Algorithm 傳給自己和其他參與者(ex: A 將 r1 分給四個參與者，則 A 擁有 r1_a, B 擁有

r1_b, C 擁有 r1_c, D 擁有 r1_d；B 也將 r2 分給四個參與者，……)。

(4) 此時，每個參與者都擁有自己及其他參與者生成的 random matrix 的一部分(ex: A 有 r1_a +r2_a +r3_a +r4_a, B 有 A 有 r1_b +r2_b +r3_b +r4_b)。

(5) 參與者將擁有的部分 local kernel + random matrix 相加，傳給 cloud。

(6) 此時，cloud 擁有 local matrix 的總和及 random matrix 的總和，不需要知道參與者分別的 local kernel 或 random matrix，只要減掉 random matrix 的總和就可以得到 global kernel。



五、實驗結果與問題討論

本次實驗是在 windows 10 下使用 python 3.8，編譯器: PyCharm Community Edition 2020.3 x64，設備: four-core Intel® Core i5 8250 1.6 GHz CPU, 8GB RAM。Dataset 使用了 Heart Disease UCI、Breast Cancer Wisconsin (Diagnostic) Data Set、Red Wine Quality。

Dataset	Heart attack	Breast cancer	Red wine quality
Dataset size	303x14	559x32	1599x12

在 Shamir’s secret sharing 中，random matrix 裡值的範圍介於 0~1000，Prime=2^P-1，range of P:12~31(Prime can’t be neither too small nor too large.)，Coefficient field range: [0, Prime)。Training data & testing data will be randomly choosed by the fixed proportion(Training : Test = 70:30).

表(一)、表(二)為使用 Dataset - Breast cancer diagnostic 的結果，SVM accuracy: 0.934523。表(一)為在 3 個參與者、每個參與者有 10 feature 的情況；表(二)為在 10 個參與者、每個參與者有 3 feature 的情況。

以下記錄著三個階段的時間:

1. 各個 local 產生亂數 + 分割 shares 傳給各個 local

2. 各個 local 把 data 轉成 kernel matrix + 把拿到的 shares 加總傳給 cloud 的時間

3. Cloud 把 2. 收到的東西還原 secret 最後得到 global kernel 的時間。

表(一)、3 local kernels, 10 features in the local kernels

Prime(in secret sharing):	$2^{17}-1$	$2^{23}-1$	$2^{31}-1$
Each LK generate random matrix + sharing (on average)	5.370585(s)	4.891469(s)	5.059458(s)
Data to LK + Sum of LK & rMtx on cloud	0.008981(s)	0.007962(s)	0.007946(s)
Reconstruct secret + Get global kernel	3.235362(s)	2.771904(s)	3.811612(s)

表(二)、10 local kernels, 3 features in the local kernels

Prime(in secret sharing):	$2^{17}-1$	$2^{23}-1$	$2^{31}-1$
Each LK generate random matrix + sharing (on average)	13.161426(s)	12.712590(s)	13.333990(s)
Data to LK + Sum of LK & rMtx on cloud	0.024929(s)	0.024909(s)	0.025939(s)
Reconstruct secret + Get global kernel	32.686150(s)	27.258705(s)	32.175280(s)

Dataset 1 - Heart attack(with secret sharing)

Accuracy (on test): 0.824175 (random state=45)

3 local kernels, 5 features in the local kernels

Prime(in secret sharing):	$2^{13}-1$	$2^{23}-1$	$2^{31}-1$
Each LK generate random matrix + sharing (on average)	1.357200(s)	1.325701(s)	1.466364(s)
Data to LK + Sum of LK & rMtx on cloud	0.002961(s)	0.001962(s)	0.002027(s)
Reconstruct secret + Get global kernel	0.788410(s)	0.950865(s)	1.170856(s)

5 local kernels, 3 features in the local kernels

Prime(in secret sharing):	$2^{13}-1$	$2^{23}-1$	$2^{31}-1$
Each LK generate random matrix + sharing (on average)	1.982053(s)	2.011561(s)	2.088419(s)
Data to LK + Sum of LK & rMtx on cloud	0.003920(s)	0.001963(s)	0.003956(s)
Reconstruct secret + Get global kernel	1.692310(s)	1.753770(s)	1.822052(s)

Dataset 3 - red wine quality(with secret sharing)

Accuracy (on test): 0.860416 (random state=42)

3 local kernels, 4 features in the local kernels

Prime(in secret sharing):	$2^{17}-1$	$2^{23}-1$	$2^{31}-1$
Each LK generate random matrix + sharing (on average)	39.220014(s)	38.850112(s)	41.301888(s)
Data to LK + Sum of LK & rMtx on cloud	0.077290(s)	0.073311(s)	0.072805(s)
Reconstruct secret + Get global kernel	24.697184(s)	26.979782(s)	27.975812(s)

4 local kernels, 3 features in the local kernels

Prime(in secret sharing):	$2^{17}-1$	$2^{23}-1$	$2^{31}-1$
Each LK generate random matrix + sharing (on average)	48.064970(s)	52.498290(s)	51.773858(s)
Data to LK + Sum of LK & rMtx on cloud	0.091957(s)	0.099206(s)	0.100652(s)
Reconstruct secret + Get global kernel	39.503282(s)	43.901119(s)	41.064306(s)

我們成功用 SSS 演算法時做出一個 local SVM，但由表可知，參與者越多，所花費時間成本越長，所以我們對程式做了一些修改，著重在如何分配參與者、亂數矩陣生成，才不會使 cost 消耗太多。

我們將原本資料以相同方式切成 group，以第十六頁例子而言，將 12 個 feature 分成 3 個 group，再將 group 中的資料切成 2 等分，這 2 等分代表 2 個 local data。在 local kernel training 完後和亂數矩陣的傳遞將會發生在每一個 group 之中。修改後的實驗結果如下:

5 Groups with 15 local kernels (Breast Cancer)

```
Each LK generate random matrix + sharing:  0:00:02.896125
Data to LK + Sum of LK & rMtx on cloud:  0:00:00.020211
Reconstruct secret + Get global kernel:  0:00:01.658895
Accuracy (on test): 0.9345238095238095
```

Original: 5 local kernels with 6 features

```
Each LK generate random matrix + sharing:  0:00:04.615241
Data to LK + Sum of LK & rMtx on cloud:  0:00:00.006977
Reconstruct secret + Get global kernel:  0:00:03.870014
Accuracy (on test): 0.9345238095238095
```

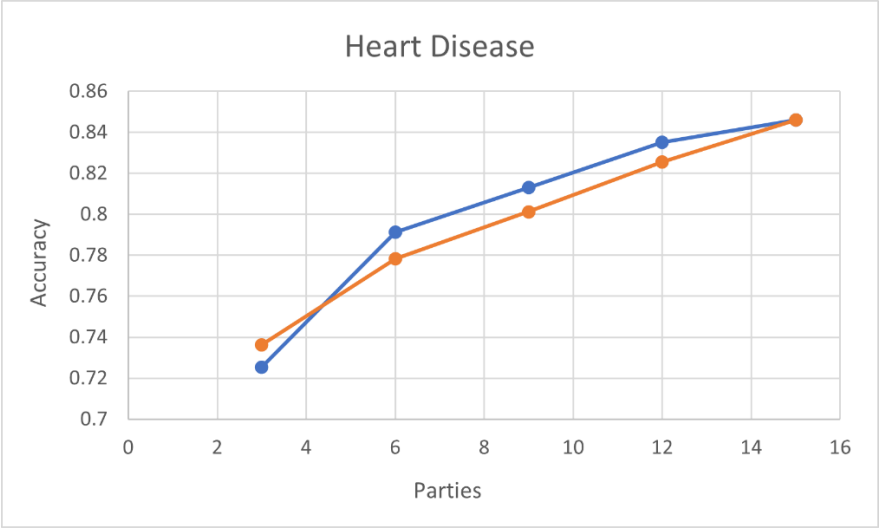
Sharing 執行的時間和參與 sharing 的 user 有關，參與 share 的 user 越多，shaing 時間越長，將參與者以分成多個 group 的形式去做 secret sharing 大大減少了執行時間。

我們將程式執行多次後，將原本沒有做 secret sharing 的 SVM 和使用 group method + secret sharing 的 SVM 做比較，並呈現在兩種不同座標軸的折線圖中。

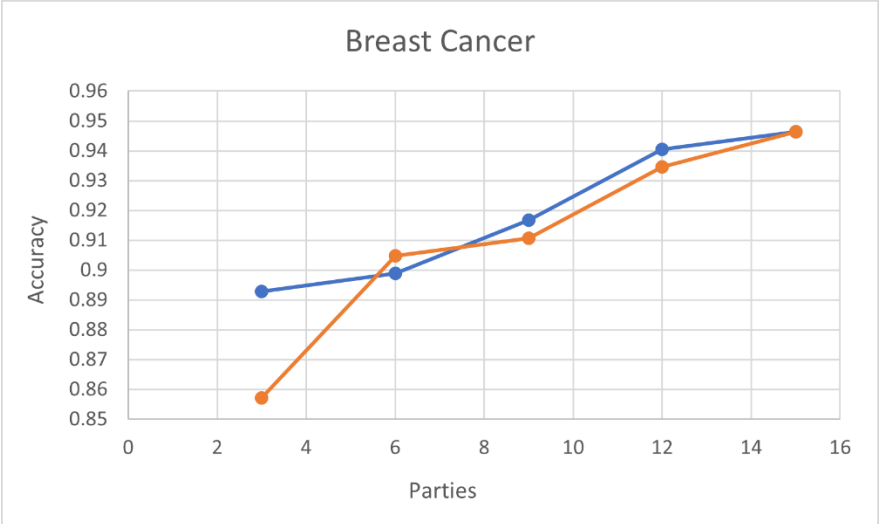
1. X-axis:參與者數量, Y-axis: Accuracy

橘線:SVM；藍線: SVM with secret sharing

(1) Heart Disease Dataset



(2)Breast Cancer Dataset



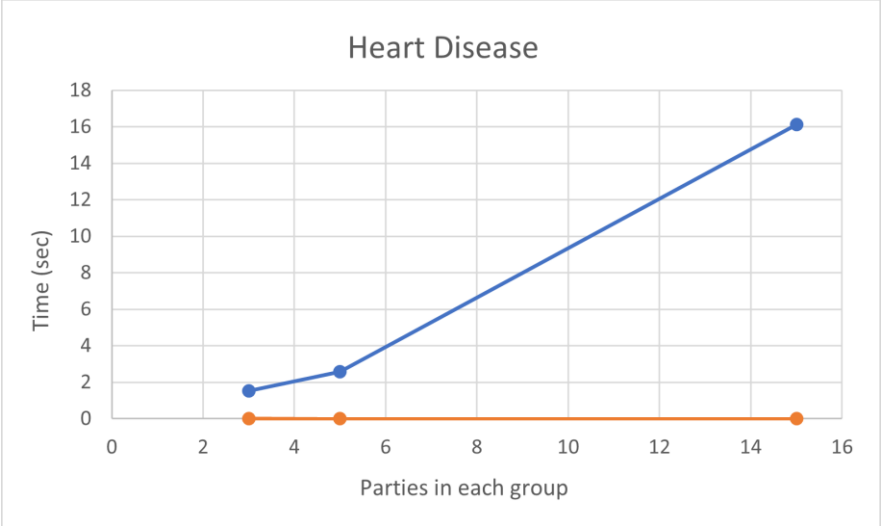
由表可知，參與 training 的 party 越多，準確性越高，有沒

有 secret sharing 的加入，並不太影響 SVM 的準確度。

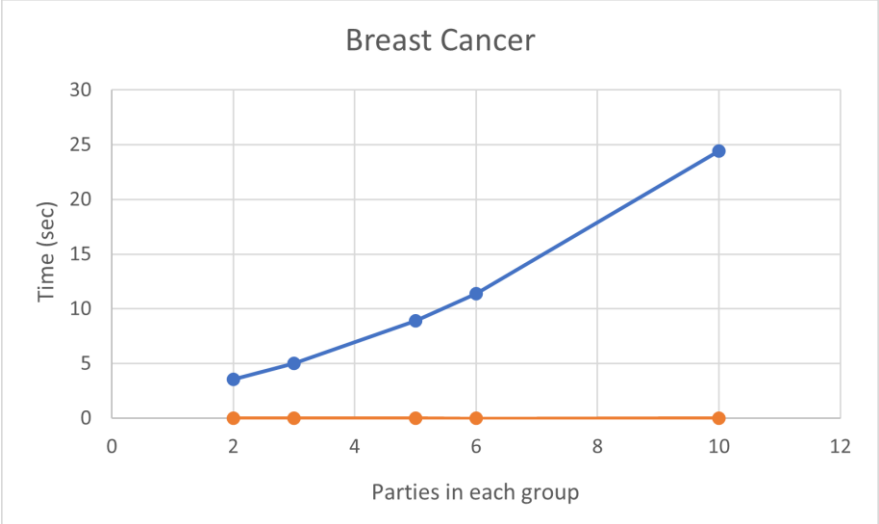
2. X-axis:參與者數量(Group 中), Y-axis: level-1 執行時間

橘線:SVM；藍線: SVM with secret sharing

(1) Heart Disease Dataset



(2)Breast Cancer Dataset



由表可知，參與者越多，sharing 次數越大，執行時間也越長。

六、參考資料

[1] Vaidya, J., Yu, H. & Jiang, X. Privacy-preserving SVM classification. *Knowl Inf Syst* **14**, 161–178 (2008).

https://doi.org/10.1007/s10115-007-0073-7

https://chih-sheng-huang821.medium.com/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E6%94%AF%E6%92%90%E5%90%91%E9%87%8F%E6%A9%9F-support-vector-machine-svm-%E8%A9%B3%E7%B4%B0%E6%8E%A8%E5%B0%8E-c320098a3d2e

https://medium.com/jameslearningnote/%E8%B3%87%E6%96%99%E5%88%86%E6%9E%90-%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E7%AC%AC3-4%E8%AC%9B-%E6%94%AF%E6%8F%B4%E5%90%91%E9%87%8F%E6%A9%9F-support-vector-machine-%E4%BB%8B%E7%B4%B9-9c6c6925856b

https://hackmd.io/@shaoeChen/B1CoXxvmm/https%3A%2F%2Fhackmd.io%2Fs%2FB1zzzspxE

https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing

https://github.com/cperales/SupportVectorMachine