1.(a)

$$
\begin{aligned}
&p(F=1,T=0,S=1,A=1)\\
&=p(F=1)p(T=0)p(S=1|F=1)p(A=1|F=1,T=0)\\
&=0.1\times(1-0.01)\times0.95\times0.95\\
&=0.0893475\approx0.08935
\end{aligned}
$$

$$
\begin{aligned}
&p(F=1,T=1,S=0,A=1)=\\
&=p(F=1)p(T=1)p(S=0|F=1)p(A=1|F=1,T=1)\\
&=0.1\times0.01\times(1-0.95)\times0.8\\
&=0.00004
\end{aligned}
$$

(b)(sorry but I couldn't type{} in Typora's equation)

i.

$$
\begin{aligned}
&p(F=1,T=0,S=0,A=1)\\
&=p(F=1)p(T=0)p(S=0|F=1)p(A=1|F=1,T=0)\\
&=0.1\times(1-0.01)\times(1-0.95)\times0.95\\
&=0.0047025\approx0.00470
\end{aligned}
$$

$$
\begin{aligned}
&p(F=1,T=1,S=1,A=1)\\
&=p(F=1)p(T=1)p(S=1|F=1)p(A=1|F=1,T=1)\\
&=0.1\times0.01\times0.95\times0.8\\
&=0.00076
\end{aligned}
$$

$$
\begin{aligned}
P(F=1,A=1)&=\sum_{T\in0,1}\sum_{S\in0,1}P(F=1,A=1,T,S)\\
&=0.0893475+0.00004+0.0047025+0.00076\\
&=0.09485
\end{aligned}
$$

ii.

$$
\begin{aligned}
&p(F=0,T=1,S=0,A=1)=\\
&=p(F=0)p(T=1)p(S=0|F=0)p(A=1|F=0,T=1)\\
&=(1-0.1)\times0.01\times(1-0.01)\times0.8\\
&=0.0071280000000000015\approx0.00713
\end{aligned}
$$

$$
\begin{aligned}
&p(F=0,T=0,S=0,A=1)=\\
&=p(F=0)p(T=0)p(S=0|F=0)p(A=1|F=0,T=0)\\
&=0.9\times0.99\times(1-0.01)\times0.0001\\
&=0.000088209\approx0.00009
\end{aligned}
$$

$$
\begin{aligned}
&p(F=0,T=1,S=1,A=1)= \\
&=p(F=0)p(T=1)p(S=1|F=0)p(A=1|F=0,T=1) \\
&=(1-0.1)\times 0.01\times 0.01\times 0.8 \\
&=0.000072\approx 0.00007
\end{aligned}
$$

$$
\begin{aligned}
&p(F=0,T=0,S=1,A=1)= \\
&=p(F=0)p(T=0)p(S=1|F=0)p(A=1|F=0,T=0) \\
&=(1-0.1)\times (1-0.01)\times 0.01\times 0.0001 \\
&=0.0000891\approx 0.00009
\end{aligned}
$$

$$
\begin{aligned}
P(A=1)&=P(F=1,A=1)+P(F=0,A=1) \\
&=0.09485+\sum_{T\in 0,1}\sum_{S\in 0,1}P(F=0,A=1,T,S) \\
&=0.09485+7.128\times 10^{-3}+8.8209\times 10^{-5}+7.2\times 10^{-5}+8.91\times 10^{-5} \\
&=0.10213910\approx 0.10214
\end{aligned}
$$

(c)

$$
P(F=1|A=1)=\frac{P(A=1,F=1)}{P(A=1)}=\frac{0.09485}{0.10213910}\approx 0.92864
$$

2.(a)

Yes, we can. No, it is impossible.

SVM for Linearly Inseparable Data allows data points to be on the wrong side of the margin boundary.

Assume slack variable $ \xi=\left\{ \begin{aligned} &0 , y_i(w^Tx_i+w_0)\geq1 \\ &1-y_i(w^Tx_i+w_0) , y_i(w^Tx_i+w_0)\lt1 \end{aligned} \right. $

s.t. $y_i(w^Tx_i+w_0)\geq 1-\xi=\begin{cases}1, y_i(w^Tx_i+w_0)\geq 1\\ y_i(w^Tx_i+w_0), y_i(w^Tx_i+w_0)<1\end{cases}$ holds

We could get the separating hyperplane by finding the minimum of$||w||^2+C\sum_i\xi_i$ where C control the trade-off between the penalty and the margin. Smaller C allows more mistake while larger C allows less mistake.

The goal of training a linear SVM is to find the hyperplane that maximizes the distance between the hyperplane and the closest data points (i.e., the support vectors). These distances are represented by the slack variables in the SVM optimization problem. To be specific, there exists a separating hyperplane between the blue points and orange points

such that $y_i(w^T x_i + w_0) \geq 1 - \xi$ holds and $||w||^2 + C\sum_i \xi_i$ gets its minimum.

If all slack variables are zero, it means that the hyperplane is able to perfectly separate the data points into the two classes, with the maximum margin possible. This is often the desired outcome when training a linear SVM, as it indicates that the model is able to accurately classify the data with a clear separation between the two classes. There isn't such line such that they could be separated into two sides of the graph. Therefore, slack variables cannot be all zeros, which will cause the soft margin SVM change into hard margin SVM

(b)

Yes, I agree

Increasing the dimensionality of the data can improve the performance of a support vector machine (SVM), but it is not always the case that it will lead to a highly accurate predictor. The ability of an SVM to handle high-dimensional data is one of the advantages of this algorithm, but the "curse of dimensionality" can make it difficult to find the best hyperplane when dealing with high-dimensional data. Additionally, increasing the number of dimensions can also make the training process more computationally intensive. Therefore, it is important to consider the trade-offs between model performance and computational complexity when deciding how many dimensions to use in training an SVM.

As for the example, if we use eight dimensionality to predict, it should be more accurate. The graph of the example is possible to be separated by an Elliptic equation $a^2x^2 + b^2y^2 + cxy + dx + ey = 1$, where a, b, c, d, e are constant and could be the parameter of the training SVM. The space contains $[1, x_1^i, x_2^i, x_1^i x_2^i, (x_1^i)^2, (x_2^i)^2, (x_1^i)^4, (x_2^i)^4]^T$, which contains all the item of elliptic equation. The extra items give a more accurate answer for the prediction. What's more, the dimension is not high so the speed is acceptable.

Therefore, it would give a highly accurate predictor.

3.(a)

The degrees of freedom for the test is 9.

Reason:

In a k-fold cross-validation,the dataset D is separated into k parts randomly: $D_1, D_2, \ldots, D_k$. D_i is test set, where $i \in \{1, 2, \ldots, k\}$. The other $(k-1)$ sets are train set. Once the $(k-1)$ sets is determined,the last is also settled. Therefore, the degrees of freedom for the test is $k - 1 = 9$.

(b)

$error = 1 - value$

	1	2	3	4	5	6	7	8	9	10
error(A1)	0.092	0.038	0.122	0.044	0.061	0.045	0.056	0.067	0.119	0.051
error(B1)	0.551	0.415	0.619	0.567	0.525	0.570	0.480	0.410	0.435	0.557

$\overline{errA_1} = \frac{1}{10}\sum_{i=1}^{10} errA_1^i = 0.0695$

$\overline{errB_1} = \frac{1}{10}\sum_{i=1}^{10} errB_1^i = 0.5129$

$$
\begin{aligned}
var(A_1 - B_1) &= \frac{1}{9}\sum_{i=1}^{10}[err(A_1^i) - err(B_1^i) - (\overline{errA_1} - \overline{errB_1})]^2 \\
&\approx 0.00572782
\end{aligned}
$$

$$
\begin{aligned}
t &= \frac{\overline{errA_1} - \overline{errB_1}}{\sqrt{(\frac{var(A_1 - B_1)}{k})}} \\
&= \frac{0.0695 - 0.5129}{\sqrt{(\frac{0.00572782}{10})}} \\
&\approx -18.52682 \notin [-2.262, 2.262]
\end{aligned}
$$

Therefore, I reject the null hypothesis.

$$
\begin{aligned}
p_{val} &= 2 \times (1 - t.cdf(abs(t_{start}, df))) \\
&\approx 1.781023195590592 \times 10^{-8} < 0.05
\end{aligned}
$$

Therefore, two algorithms performed significantly different with significance level $\alpha = 5$ %

4.(a)

i.

①assume $y^i \leq \hat{y}^i = g(a^i)$

$$
\begin{aligned}
w_j^{new} &= w_j - \eta \frac{\delta}{\delta w_j} L \\
&= w_j - \eta \frac{\delta}{\delta w_j} \frac{1}{2}(y^i - g(a^i))^2 \\
&= w_j - \eta |y^i - g(a^i)| (\frac{\delta}{\delta w_j}(y^i - g(a^i)) \\
&= w_j - \eta (g(a^i) - y^i) g'(a^i) (\frac{\delta}{\delta w_j} a^i) \\
&= w_j + \eta (y^i - g(a^i)) g'(a^i) (\frac{\delta}{\delta w_j} x_j^i w_j) \\
&= w_j + \eta g'(a^i)(y^i - \hat{y}^i) x_j^i
\end{aligned}
$$

②Similarly, if $y^i > \hat{y}^i$, $w_j^{new} = w_j + \eta g'(a^i)(y^i - \hat{y}^i) x_j^i$

Therefore, $w_j^{new} = w_j + \eta g'(a^i)(y^i - \hat{y}^i) x_j^i$

ii.

① if $a^i \geq 0$

$g(a^i) = max(a^i, 0) = a^i$

$g'(a^i) = \frac{\delta}{\delta a^i} a^i = 1$

② otherwise, $a^i < 0$

$g(a^i) = max(a^i, 0) = 0$

$g'(a^i) = \frac{\delta}{\delta a^i} 0 = 0$

Therefore,$g'(a^i) = \begin{cases} 1, if\ a^i \geq 0 \\ 0, otherwise \end{cases}$

(b)

(2) changes as follows:

$w_j^{new} = w_j + \eta g'(a^i)(y^i - \hat{y}^i) x_j^i = w_j^{new} = w_j + \eta (y^i - \hat{y}^i) x_j^i$

(3) changes as follows:

$g'(a) = \frac{\delta}{\delta a^i} a^i = 1$

If you use the function f(x) = x instead of a ReLU function in a neural network, the gradient of the function would be 1 for all values of x. This means that the weights of the network would be updated by the same amount, regardless of the input. In contrast, the gradient of the ReLU function is 0 for negative input values and 1 for positive input values. This means that the weights of the network would only be updated for positive input values

In general, using the identity function as the activation function will result in a slower and less stable convergence during training, compared to using the ReLU function or another non-linear activation function. This is because the identity function does not introduce non-linearity into the network, which limits the network's ability to model complex relationships between the input data and the output. Using the identity function as the activation function will not introduce any regularization into the network, which can lead to overfitting

5.(a)

The complexity of K-Medoids is $O(K(N-K)^2)$, where N is the number of samples, and K is the number of clusters.

Reasons:

PAM works as follows:①Build step: Starts from an initial set of medoids.②Swap step: for all medoids already selected, compute the cost of swapping this medoid with any non-medoid point. ③Loop② and stop when there is no change anymore.

Since we have K medoids and (N - K) non-medoids, the cost would be O(K(N-K)). What's more, loop times are (N - K). Therefore, the total computational cost would be $O(K(N-K)^2)$.

(b)

Yes, it is.

Reasons:

The computational complexity of k-mean clustering algorithm is $O(tKn)$, where n represent of objects, K represent of clusters, and t represent of iterations.

The complexity of K-Medoids is $O(K(N-K)^2)$, where N is the number of samples, and K is the number of clusters.

Normally, $K << n, t << n$. Therefore, $O(tKn) << O(K(N-K)^2)$. So the k-medians algorithm is more computationally demanding than k-means.

(c)

i.

	a	b	c	d	e
a	0				
b	2.2	0			
c	4	6.1	0		
d	7.8	8.6	6.1	0	
e	7.3	7.2	7.3	3.2	0

$$\begin{aligned} distance((a,b),c) &= max(distance(a,c), distance(b,c)) \\ &= max(4, 6.1) \\ &= 6.1 \end{aligned}$$

$$\begin{aligned} distance((a,b),d) &= max(distance(a,d), distance(b,d)) \\ &= max(7.8, 8.6) \\ &= 8.6 \end{aligned}$$

$$\begin{aligned} distance((a,b),e) &= max(distance(a,e), distance(b,e)) \\ &= max(7.3, 7.2) \\ &= 7.3 \end{aligned}$$

	a,b	c	d	e
a,b	0			
c	6.1	0		
d	8.6	6.1	0	
e	7.3	7.3	3.2	0

$$\begin{aligned} distance((a,b),(d,e)) &= max(distance(d,(a,b)), distance(e,(a,b))) \\ &= max(8.6, 7.3) \\ &= 8.6 \end{aligned}$$

$$\begin{aligned} distance(c,(d,e)) &= max(distance(c,d), distance(c,e)) \\ &= max(6.1, 7.3) \\ &= 7.3 \end{aligned}$$

	a,b	c	d,e
a,b	0		
c	6.1	0	
d,e	8.6	7.3	0

$$
\begin{aligned}
distance((d,e),(c,(a,b))) &= max(distance((d,e),c,distance((d,e),(a,b))) \\
&= max(8.6,7.3) \\
&= 8.6
\end{aligned}
$$

	a,b,c	d,e
a,b,c	0	
d,e	8.6	0

	a,b,c,d,e	
a,b,c,d,e	0	