Question 1(a)

We cannot train a hard margin linear SVM on the dataset since the two classes are not linearly separable, or no straight line can be drawn to separate all the tuples of the two classes.

Question 1(b)

Although we can train a soft margin linear SVM on the dataset, the accuracy is very low.

The soft-margin linear SVMs allows a small fraction of training tuples to be mis-classified. And the reason why the accuracy is very low is because no matter how we place the boundary, there are same proportion of dataset located on the two sides of the boundary, i.e., half of the two classes on one side and half of the other two classes on the other side. If you can endure such large amount of misclassification, then you can use soft-margin linear SVMs.

Question 1(c)

I agree with professor Kernel's claim. Since the centres of the two classes are nearly overlap and only the semi-minor axis and semi-major axis differs, we can map the data to a higher dimension and find a linear separating hyperplane. Then the SVM can perform non-linear classification well and thus training a linear SVM in that mapped space would give a highly accurate predictor.

Question 2(a)

Suppose that for each model, we did 10-fold cross-validation, say, 10 times, each time using a different 10-fold data partitioning. Each partitioning is independently drawn. We can average the 10 error rates obtained each for $A1$ and $B1$, respectively, to obtain the mean error rate for each model. For a given model, the individual error rates calculated in the cross-validations may be considered as different, independent samples from a probability distribution. In general, they follow a t-distribution with $k - 1$ degrees of freedom where, here, $k = 10$. This allows us to do hypothesis testing where the significance test used is the t-test, or Student's t-test.

Hence, the degree of freedom is $k - 1 = 10 - 1 = 9$.

Question 2(b)

We have accuracies of 10 folds of $A1$:

$$\begin{aligned}&\text{err}(A_1)_1 = 0.908, \text{err}(A_1)_2 = 0.962, \text{err}(A_1)_3 = 0.878, \text{err}(A_1)_4 = 0.956\\&\text{err}(A_1)_5 = 0.939, \text{err}(A_1)_6 = 0.955, \text{err}(A_1)_7 = 0.944, \text{err}(A_1)_8 = 0.933\\&\text{err}(A_1)_9 = 0.881, \text{err}(A_1)_{10} = 0.949\end{aligned}$$

We have accuracies of 10 folds of $B1$:

$$\begin{aligned}&\text{err}(B_1)_1 = 0.449, \text{err}(B_1)_2 = 0.585, \text{err}(B_1)_3 = 0.381, \text{err}(B_1)_4 = 0.433\\&\text{err}(B_1)_5 = 0.475, \text{err}(B_1)_6 = 0.430, \text{err}(B_1)_7 = 0.520, \text{err}(B_1)_8 = 0.590\\&\text{err}(B_1)_9 = 0.565, \text{err}(B_1)_{10} = 0.443\end{aligned}$$

Note that the error rate is $1 - accuracy(M)$, where M denotes the model. Hence, we have

$$\begin{aligned}\overline{err}(A_1) &= \frac{1}{10}\sum_{i=1}^{10}\text{err}(A_1)_i \approx 0.0695\\\overline{err}(B_1) &= \frac{1}{10}\sum_{i=1}^{10}\text{err}(B_1)_i \approx 0.5129\end{aligned}$$

Then $\overline{err}(A_1) - \overline{err}(B_1) = 0.0695 - 0.5129 = -0.4434$.

Then we have

$$\begin{aligned}var(A_1 - B_1) &= \frac{1}{10}\sum_{i=1}^{10}[\text{err}(A_1)_i - \text{err}(B_1)_i - (\overline{err}(A_1) - \overline{err}(B_1))]^2\\&= \frac{1}{10}\sum_{i=1}^{10}[\text{err}(A_1)_i - \text{err}(B_1)_i + 0.4434]^2\\&\approx 0.005155039999999994\end{aligned}$$

Then $var(A_1 - B_1)/10 \approx 0.0005155039999999994$.

Therefore,

$$t = \frac{\overline{err}(A_1) - \overline{err}(B_1)}{\sqrt{var(A_1 - B_1)/10}} = \frac{-0.4434}{\sqrt{0.0005155039999999994}} = -19.52898487361761.$$

By run the code below, we obtain p value, $1.120231307716324 \times 10^{-8}$, which is nearly 0.

```
from scipy.stats import t
p_val = (1 - t.cdf(abs(-19.52898487361761), 9)) * 2
```

Since p value is less than $alpha$, we reject the null hypothesis and conclude that one of the algorithms is significantly better than the other.

Question 2(c)

We have accuracies of 10 folds of A_1:

$$\begin{aligned}&\text{err}(A_1)_1 = 0.908, \text{err}(A_1)_2 = 0.962, \text{err}(A_1)_3 = 0.878, \text{err}(A_1)_4 = 0.956\\&\text{err}(A_1)_5 = 0.939, \text{err}(A_1)_6 = 0.955, \text{err}(A_1)_7 = 0.944, \text{err}(A_1)_8 = 0.933\\&\text{err}(A_1)_9 = 0.881, \text{err}(A_1)_{10} = 0.949\end{aligned}$$

We have accuracies of 10 folds of B_2:

$$\begin{aligned}&\text{err}(B_2)_1 = 0.968, \text{err}(B_2)_2 = 1.000, \text{err}(B_2)_3 = 0.950, \text{err}(B_2)_4 = 0.994\\&\text{err}(B_2)_5 = 0.989, \text{err}(B_2)_6 = 0.989, \text{err}(B_2)_7 = 1.000, \text{err}(B_2)_8 = 0.994\\&\text{err}(B_2)_9 = 0.966, \text{err}(B_2)_{10} = 0.966\end{aligned}$$

Note that the error rate is $1 - accuracy(M)$, where M denotes the model. Hence, we have

$$\begin{aligned}\overline{err}(A_1) &= \frac{1}{10}\sum_{i=1}^{10}\text{err}(A_1)_i \approx 0.0695\\\overline{err}(B_2) &= \frac{1}{10}\sum_{i=1}^{10}\text{err}(B_2)_i \approx 0.0184\end{aligned}$$

Then $\overline{err}(A_1) - \overline{err}(B_2) = 0.0695 - 0.0184 = 0.0511$.

Then we have

$$\begin{aligned}var(A_1 - B_2) &= \frac{1}{10}\sum_{i=1}^{10}[\text{err}(A_1)_i - \text{err}(B_2)_i - (\overline{err}(A_1) - \overline{err}(B_2))]^2\\&= \frac{1}{10}\sum_{i=1}^{10}[\text{err}(A_1)_i - \text{err}(B_2)_i - 0.0511]^2\\&\approx 0.00035868999999999897\end{aligned}$$

Then $var(A_1 - B_2)/10 \approx 3.58689999999999 \times 10^{-5}$.

Therefore,

$$t = \frac{\overline{err}(A_1) - \overline{err}(B_1)}{\sqrt{var(A_1 - B_1)/10}} = \frac{0.0511}{\sqrt{3.58689999999999 \times 10^{-5}}} = 8.532204687249001.$$

By run the code below, we obtain p value, $1.3184220974293837 \times 10^{-5}$, which is nearly 0.

```
from scipy.stats import t
p_val = (1 - t.cdf(abs(-19.52898487361761), 9)) * 2
```

Since p value is less than $alpha$, we reject the null hypothesis and conclude that one of the algorithms is significantly better than the other.

Question 3(a)

We first calculate the partial derivative of a^i with respect to w_j,

$$\frac{\partial a^i}{\partial w_j} = \frac{\partial \left(\sum_{j=1}^{d} w_j x_j^i \right)}{\partial w_j} = x_j^i$$

By chain rule, we have

$$\begin{aligned}
\frac{\partial L}{\partial w_j} &= \frac{\partial L}{\partial g} \frac{\partial g}{\partial a^i} \frac{\partial a^i}{\partial w_j} \\
&= -2 \cdot \frac{1}{2} \sum_{i=1}^{n} (y^i - g(\mathbf{w}^T \mathbf{x}^i)) \cdot g'(a^i) \cdot x_j^i \\
&= -\sum_{i=1}^{n} (y^i - g(\mathbf{w}^T \mathbf{x}^i)) \cdot g'(a^i) \cdot x_j^i \\
&= \sum_{i=1}^{n} -(y^i - \hat{y}^i) \cdot g'(a^i) \cdot x_j^i.
\end{aligned}$$

Note that SGD with the given learning algorithm yields the following algorithm:

1. Pick w_0 and set step size $\eta > 0$
2. For $i = 0, 1, \ldots,$:
   (a) Pick a uniformly random index $i \in [n]$ and set $v_i = -(y^i - \hat{y}^i) \cdot g'(a^i) \cdot x_j^i$
   (b) Set $w_j^{\text{new}} = w_j - \eta v_i = w_j + \eta (y^i - \hat{y}^i) \cdot g'(a^i) \cdot x_j^i$

We observe that the SGD update for parameter w_j with step size η is of the form

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

Question 3(b)

For the linear transfer function $g(a) = a$, the gradient is $g'(a) = 1$ for all a, which is a modification to (2).

Furthermore,

$$L(\mathbf{w}|\mathcal{Z}) = \frac{1}{2}\sum_{i=1}^{n}(y^i - g(\mathbf{w}^T\mathbf{x}^i))^2 = \frac{1}{2}\sum_{i=1}^{n}(y^i - \mathbf{w}^T\mathbf{x}^i)^2.$$

By chain rule, we have

$$\begin{aligned}\frac{\partial L}{\partial w_j} &= \frac{\partial L}{\partial g}\frac{\partial g}{\partial a^i}\frac{\partial a^i}{\partial w_j} \\ &= -2 \cdot \frac{1}{2}\sum_{i=1}^{n}(y^i - g(\mathbf{w}^T\mathbf{x}^i)) \cdot g'(a^i) \cdot x_j^i \\ &= -\sum_{i=1}^{n}(y^i - \mathbf{w}^T\mathbf{x}^i) \cdot 1 \cdot x_j^i \\ &= \sum_{i=1}^{n} -(y^i - \hat{y}^i) \cdot x_j^i.\end{aligned}$$

Note that SGD with the given learning algorithm yields the following algorithm:

1. Pick w_0 and set step size $\eta > 0$
2. For $i = 0, 1, \ldots,$:
   (a) Pick a uniformly random index $i \in [n]$ and set $v_i = -(y^i - \hat{y}^i) \cdot x_j^i$
   (b) Set $w_j^{\text{new}} = w_j - \eta v_i = w_j + \eta(y^i - \hat{y}^i)x_j^i$

We observe that the SGD update for parameter w_j with step size η is of the form

$$w_j^{\text{new}} = w_j + \eta(y^i - \hat{y}^i)x_j^i,$$

which is a modification to (1).

We conclude that we modify (1) to $w_j^{\text{new}} = w_j + \eta(y^i - \hat{y}^i)x_j^i$, and (2) to $g'(a) = 1$ for all a.

Question 4(a)

I use Partitioning Around Medoids (PAM), a k-medoids algorithm for partitioning based on medoid or central object.

Let $\boldsymbol{o_1}, \ldots, \boldsymbol{o_k}$ be the current set of representative objects (i.e., medoids). To determine whether a non-representative object, denoted by $\boldsymbol{o_{random}}$, which is a replacement for a current medoid $\boldsymbol{o_j}$ $(1 \leq j \leq k)$, we calculate the distance from every object $\boldsymbol{p}$ to the closest object in the set $\{\boldsymbol{o_1}, \ldots, \boldsymbol{o_{j-1}}, \boldsymbol{o_{random}}, \boldsymbol{o_{j+1}}, \ldots, \boldsymbol{o_k}\}$ and use the distance to update the cost function.

Suppose object $\boldsymbol{p}$ is currently assigned to a cluster represented by medoid $\boldsymbol{o_j}$, if $\boldsymbol{o_j}$ is being replaced by $\boldsymbol{o_{random}}$, object $\boldsymbol{p}$ needs to be reassigned to either $\boldsymbol{o_{random}}$ or some other cluster represented by $\boldsymbol{o_i}$ $(i \neq j)$, whichever is the closest. If, instead, $\boldsymbol{p}$ is currently assigned to a cluster represented by some other object $\boldsymbol{o_i}$, $i \neq j$, object $\boldsymbol{p}$ remains assigned to the cluster represented by $\boldsymbol{o_i}$ as long as $\boldsymbol{p}$ is still closer to $\boldsymbol{o_i}$ than to the $\boldsymbol{o_{random}}$.

Each time a reassignment occurs, a difference in absolute error, E, is contributed to the cost function. Therefore, the cost function calculates the difference in absolute-error value if a current representative object is replaced by a non-representative object. The total cost of swapping is the sum of costs incurred by all non-representative objects. If the total cost is negative, then $\boldsymbol{o_j}$ is replaced or swapped with $\boldsymbol{o_{random}}$ because the actual absolute-error E is reduced. If the total cost is positive, the current representative object, $\boldsymbol{o_j}$, is considered acceptable, and nothing is changed in the iteration.

The pseudocode is shown below. The input k means the number of clusters, and D means a data set containing n objects. The output is a set of k clusters.

Arbitrary choose k objects in D as the initial representative objects or seeds
repeat
    assign each remaining object to the cluster with the nearest representative object;
    randomly select a non-representative object, $\boldsymbol{o_{random}}$;
    compute the total cost, S, of swapping representative object, $\boldsymbol{o_j}$ with $\boldsymbol{o_{random}}$;
    if $S < 0$ **then** swap $\boldsymbol{o_j}$ with $\boldsymbol{o_{random}}$ to form the new set of k representative objects;
until no change;

Question 4(b)

Note that Big O notation denotes the upper bound of an algorithm. Hence, we will consider the worst case.

Assume that

- the first sets of medoids are the worst ones
- we choose the next worst one in all the possibilities every time

then

- the cost function calculated through these medoids is the maximum of all the possible sets of medoids
- we will exhaust all the remaining medoids $(n - k)$ to find the set of medoids that has the minimum cost function since we need to choose a random medoid that decreases the cost function every time.

Then we have

- To loop through all the medoids needs k times
- To loop through all the non-medoid data points needs $n - k$ times.
- To choose the random medoid needs $n - k$ times.

Hence, for such worst case, the computational complexity is $O(k \cdot (n-k) \cdot (n-k)) = O(k(n-k)^2)$.

Question 4(c)

k-means minimizes a l_2 norm error function while k-median minimizes a l_1 norm error function. Furthermore, k-means is sensitive to outliers while k-medians is robust to outliers and results in compact clusters. Hence, we conclude that we use k-median when

- we want to minimize the sum of the absolute differences of their Cartesian coordinates,
- There are outliers in the dataset.

The k-medians algorithm is more computational demanding than k-means when there are outliers in the dataset. Generally, the k-Medians clustering algorithm is much faster in calculating of time for the identification of outliers, against the k-means. Hence, , in errors minimization, the k-medians clustering algorithm is more effective probably because it uses median as robust for computing the clustering compare to the k-means which uses mean which is sensitive to outliers. When there are no outliers, generally k-medians is more computationally demanding than k-means since the coordinates of cluster data points in each dimension need to be sorted in k-medians.