**Question 1.** What is the basic idea of a Support Vector Machine?

**Model Solution 1.** SVMs are applied to classification problems. That is, given training data \((x^{(i)}, y^{(i)}), i = 1, \ldots, m\) consisting of inputs/features \(x^{(i)}\) and corresponding output \(y^{(i)}\) which is labelled either +1 or −1, given a new input \(x\) an SVM predicts the corresponding −1/ +1 label. The prediction is of the form \(\text{sign}(\theta^T x)\), where \(\theta\) are unknown parameters. This corresponds to assuming that the data is linearly separable since \(\theta^T x = 0\) defines a line and points on one side of the line (where \(\theta^T x > 0\)) are labelled +1 and those on the other side of the line (where \(\theta^T x < 0\)) are labelled −1.

In general there may be many choices of line that can separate the training data, and SVMs choose the line which maximises the margin i.e. the perpendicular distance between the line and the nearest training data points. E.g. see the following sketch:

![Sketch of linearly separable data with support vectors and margin](image)

**Question 2.** Consider the four training instances shown in the figure below.

![Figure with four training instances](image)

There are positive examples at \((0,0), (2,2)\) and negative examples at \((h,1), (0,3)\). Note that the horizontal position \(h\) of the point \((h,1)\) is a variable such that \(0 \leq h \leq 3\).

1. How large can \(h\) be made so that the training points are still linearly separable?

2. When the points are linearly separable, does the orientation (i.e., angle) of the maximum margin decision boundary change as \(h\) varies?

**Model Solution 2.**

1. To maintain linear separability the point \((h,1)\) can’t move to the right of the line joining the two points labelled ‘x’. The line between these two points intersects the \(x_2 = 1\) line when \(x_1 = 1\) i.e. at point \((1,1)\). So we can’t make \(h\) larger than 1.
2. No. An SVM selects the separating plane that maximises the margin i.e. the perpendicular distance between the line and the nearest training data points. The nearest data points in this example (the support vectors) are \((h, 1)\) marked ‘o’ and at least one of the two points marked ‘x’. A line parallel to the two ‘x’ points maximises the margin – if we rotate this line anti-clockwise while keeping the same margin between the \((h, 1)\) point and the \((0, 0)\) point then the distance to these points will decrease, if we rotate clockwise the distance to the \((2, 2)\) point will decrease, see:

![Graph showing SVM decision boundary](image)

**Question 3.** In the following image, circle the points such that removing that example from the training set and retraining a linear SVM, we would get a different decision boundary than training on the full sample. Also, give a one or two sentence explanation for why you circled the points.

![Image with labeled points](image)

**Model Solution 3.** See sketch:

![Sketch showing SVM decision boundary](image)

An SVM selects the separating plane which maximises the margin i.e. the perpendicular distance between the line and the nearest training data points (the ”support vectors”). The left-hand sketch indicates the SVM choice of separating hyperplane for the example data set. Only those points closest to the separating plane affect the position of the plane. By removing the indicated point (which is a support vector) we change the position of this plane.
**Question 4.** Consider the following one dimensional training data set, ‘x’ denotes negative examples and ‘o’ positive examples. The exact data points and their labels are given in the table below. Suppose a SVM is used to classify this data.

1. Indicate on the diagram which are the support vectors and mark the decision boundary

2. Give the value of the cost function and of the model parameter $\theta$ after training.

<table>
<thead>
<tr>
<th>$x$</th>
<th>1</th>
<th>1.5</th>
<th>2.5</th>
<th>3</th>
<th>4</th>
<th>4.5</th>
<th>5</th>
<th>5.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Model Solution 4.**

1. See sketch:

2. In an SVM given training data $(x^{(i)}, y^{(i)}), i = 1, \ldots, m$ consisting of inputs/features $x^{(i)}$ and corresponding output $y^{(i)}$ which is labelled either +1 or −1 the cost function used is $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y^{(i)}\theta^T x^{(i)}) + \lambda \theta^T \theta$, where the $\theta$ are parameters whose value is unknown. In this example there is a single input feature, although we also add a dummy constant feature so that there are two parameter values to estimate $\theta = (\theta_0, \theta_1)^T$.

Only the support vectors (the training data points closest to the decision boundary) contribute to the cost function, namely points (2.5,-1) and (3,1). So in this example the cost function simplifies to $J(\theta) = \max(0, 1 + (\theta_0 + \theta_1 \times 2.5) + \max(0, 1 - (\theta_0 + \theta_1 \times 3) + \lambda (\theta_0^2 + \theta_1^2))$. The SVM maximises the distance between the decision boundary and the support vectors, so in this example will place the decision boundary mid-way between the two support vectors i.e. at $x = 2.75$. So we have $\theta_0 + \theta_1 \times 2.75 = 0$ i.e. $\theta_1 = -\theta_0/2.75$. The cost function is therefore,

$$J(\theta) = \max(0, 1 + \theta_0(1 - 2.5/2.75) + \max(0, 1 - \theta_0(1 - 3/2.75) + \lambda \theta_0^2(1 + \frac{1}{2.75})^2$$

$$= \max(0, 1 + 0.0909\theta_0) + \max(0, 1 + 0.0909\theta_0) + 1.86\lambda \theta_0^2$$

$$= 1 + 0.0909\theta_0 + 1 + 0.0909\theta_0 + 1.86\lambda \theta_0^2$$

$$= 2 + 0.1818\theta_0 + 1.86\lambda \theta_0^2$$

We could stop at this point for full marks. But if you’re keen you can find the minimising value of $\theta_0$ by differentiating $J(\theta)$ to obtain $dJ/d\theta_0 = 0.1818 + 2 \times 1.86\theta_0$ and then and setting the derivative equal to zero i.e. $0.1818 + 2 \times 1.86\theta_0 = 0$ and so $\theta_0 = -0.1818/(2 \times 1.86) = -0.0489$. 

3
Question 5. 1. Describe how a $k$-nearest neighbour classifier makes predictions.

2. As we increase $k$, the $k$-nearest neighbour algorithm begins to overfit the training dataset. True or False. Explain your answer.

Model Solution 5. 1. We have training data $(x^{(i)}, y^{(i)})$, $i = 1, \ldots, m$ consisting of inputs/features $x^{(i)}$ and corresponding output $y^{(i)}$ which is labelled either $+1$ or $-1$ (since we are interested in a $k$NN classifier). Given a new input $x$ a $k$NN classifier finds the $k$ training points with $x^{(i)}$ closest to $x$. To decide which points are closest we need to decide how we would like to measure distance, with a common choice being Euclidean distance i.e. $\sum_{j=1}^{n} (x^{(i)}_j - x_{j})^2$. Amongst these $k$ closest points we take a majority vote i.e. if a majority of them are labelled $+1$ then the $k$NN classifier predicts $+1$ for input $x$ and if a majority of them are labelled $-1$ then the $k$NN classifier predicts $1$ for input $x$.

2. False. As we increase $k$ a $k$NN classifier tends to underfit (rather than overfit) the data. As we increase $k$ the model starts to use training points further away from input $x$ and this tends to smooth out small details in the training data, which is good when they are noise but becomes underfitting if carried too far. <illustrate with a sketch>.

Question 6. Describe how logistic regression can be extended to make use of kernels. Illustrate your answer with reference to the Gaussian kernel $K(x, y) = e^{-(x-y)^2}$.

Model Solution 6. Given training data $(x^{(i)}, y^{(i)})$, $i = 1, \ldots, m$ consisting of inputs/features $x^{(i)}$ and corresponding output $y^{(i)}$ which is labelled either $+1$ or $-1$, for a new input $x$ normal logistic regression makes predicts the output to be $\text{sign}(\theta^T x)$. The parameter values $\theta$ are selected to minimise the cost function $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \log(1 + e^{-y^{(i)} \theta^T x^{(i)}})$. In kernel logistic regression with a Gaussian kernel we replace $\theta^T x$ by $\sum_{j=1}^{m} \alpha_j y^{(j)} K(x, x^{(j)})$. So the model prediction is now $\text{sign}(\sum_{j=1}^{m} \alpha_j y^{(j)} K(x, x^{(j)}))$ and the cost function is $J(\alpha) = \frac{1}{m} \sum_{i=1}^{m} \log(1 + e^{-y^{(i)} \sum_{j=1}^{m} \alpha_j y^{(j)} K(x^{(i)}, x^{(j)})})$.

When the parameters $\alpha_j = 1$, the prediction is $\text{sign}(\sum_{j=1}^{m} y^{(j)} K(x, x^{(j)}))$, which for a Gaussian kernel and a single feature becomes $\text{sign}(\sum_{j=1}^{m} y^{(j)} e^{-(x-x^{(j)})^2})$. The model has no parameters to estimate and the prediction is a weighted combination of the training data points nearest to input $x$. 