Question 1 Let $\{\vec{x}_i, y_i\}_{i=1}^N$ be our training dataset, where $y_i \in \{0, 1, \dots, C-1\}$, indicates the class of $\vec{x}_i$. Suppose we want to train a regularized logistic regression for this multi-class classification problem. Then, we can define,

$$P(y=c \mid \vec{x}_i) = \begin{cases} \dfrac{\exp(-\vec{w}^{(c)} \cdot \vec{x})}{1 + \sum_{k=1}^{C-1} \exp(-\vec{w}^{(k)} \cdot \vec{x})}, & \text{for } c > 0 \\ \dfrac{1}{1 + \sum_{k=1}^{C-1} \exp(-\vec{w}^{(k)} \cdot \vec{x})}, & \text{for } c = 0 \end{cases}$$

Here, $\vec{w}^{(c)}$ is the weight associated with class $c = 1, 2, \dots, C-1$.

We can fix $\vec{w}^{(0)} = \vec{0}$ (hence it is not a parameter) and write it more compactly as,

$$P(y=c \mid \vec{x}_i) = \frac{\exp(-\vec{w}^{(c)} \cdot \vec{x}_i)}{\sum_{k=0}^{C-1} \exp(-\vec{w}^{(c)} \cdot x_i)}$$

Similarly, we define the likelihood of $W = [\vec{w}^{(1)} \dots \vec{w}^{(C-1)}]$ as,

$\mathcal{L}(W) = \prod_{i=1}^{N} P(y_i \mid \vec{x}_i ; W)$ to which we wish to maximize.

This is equivalent to maximizing $\ln \mathcal{L}(W)$.

$$\therefore \hat{W} = \underset{W}{\operatorname{argmax}} \ln \mathcal{L}(W) = \underset{W}{\operatorname{argmin}} (-\ln \mathcal{L}(W)) \quad (\because 0 \le \mathcal{L}(W) \le 1 \Rightarrow \ln \mathcal{L}(W) < 0)$$

$$= \underset{W}{\operatorname{argmin}} \left[\sum_{i=1}^{N} -\ln P(y_i \mid \vec{x}_i ; W) \right] = \underset{W}{\operatorname{argmin}} \left[\sum_{i=1}^{N} -\ln \left(\frac{\exp(-\vec{w}^{(y_i)} \cdot \vec{x}_i)}{\sum_{k=0}^{C-1} \exp(-\vec{w}^{(k)} \cdot \vec{x}_i)} \right) \right]$$

$$= \underset{W}{\operatorname{argmin}} \left[\sum_{i=1}^{N} (\vec{w}^{(y_i)} \cdot \vec{x}_i) + \ln \left(\sum_{k=0}^{C-1} \exp(-\vec{w}^{(k)} \cdot \vec{x}_i) \right) \right]$$

Since we want to train a regularized model, we add a L2 regularization term,

$$\therefore \hat{W} = \underset{W}{\operatorname{argmin}} \left\{ \left[\sum_{i=1}^{N} (\vec{w}^{(y_i)} \cdot \vec{x}_i) + \ln \left(\sum_{k=0}^{C-1} \exp(-\vec{w}^{(k)} \cdot \vec{x}_i) \right) \right] + \frac{\lambda}{2} \sum_{j=1}^{C-1} \| \vec{w}^{(j)} \|_2^2 \right\}$$

Denote $\ell_R(\vec{x}_i, y_i ; W) = \left[\sum_{i=1}^{N} (\vec{w}^{(y_i)} \cdot \vec{x}_i) + \ln \left(\sum_{k=0}^{C-1} \exp(-\vec{w}^{(k)} \cdot \vec{x}_i) \right) \right] + \frac{\lambda}{2} \sum_{j=1}^{C-1} \| \vec{w}^{(j)} \|_2^2$,

i.e. $\hat{W} = \underset{W}{\operatorname{argmin}} \ell_R(\vec{x}_i, y_i ; W)$. In order to train our model, we use gradient descent to update each $\vec{w}^{(\alpha)}$, for $\alpha = 1, 2, \dots, C-1$ (again, $\vec{w}^{(0)} = 0$ is fixed)

The update at time step $(t+1)$ is given to be:

$$\vec{w}_{t+1}^{(\alpha)} = \vec{w}_t^{(\alpha)} - \eta \frac{\partial \ell_R(\vec{x}_i, y_i ; W)}{\partial \vec{w}^{(\alpha)}}$$

Here, we have

$$\frac{\partial L_R(\vec{x}_i, y_i; W)}{\partial \vec{w}^{(\alpha)}} = \frac{\partial}{\partial \vec{w}^{(\alpha)}} \left\{ \left[\sum_{i=1}^{N} (\vec{w}^{(y_i)} \cdot \vec{x}_i) + \ln\left(\sum_{k=0}^{C-1} \exp(-\vec{w}^{(k)} \cdot \vec{x}_i) \right) \right] + \frac{\lambda}{2} \sum_{j=1}^{C-1} \| \vec{w}^{(j)} \|_2^2 \right\}$$

$$= \sum_{\substack{1 \le i \le N \\ y_i = \alpha}} \vec{x}_i + \left[\sum_{i=1}^{N} \frac{1}{\sum_{k=0}^{C-1} \exp(-\vec{w}^{(k)} \cdot \vec{x}_i)} \left(\exp(-\vec{w}^{(\alpha)} \cdot \vec{x}_i) \right) (-\vec{x}_i) \right] + \frac{\lambda}{2} (2 \vec{w}^{(\alpha)})$$

$$= \lambda \vec{w}^{(\alpha)} + \sum_{\substack{1 \le i \le N \\ y_i = \alpha}} \vec{x}_i - \sum_{i=1}^{N} \underbrace{\frac{\exp(-\vec{w}^{(\alpha)} \cdot \vec{x}_i)}{\sum_{k=0}^{C-1} (-\vec{w}^{(k)} \cdot \vec{x}_i)}}_{P(\hat{y}_i = \alpha \mid \vec{x}_i ; W)} \vec{x}_i$$

$$= \lambda \vec{w}^{(\alpha)} + \left[\sum_{\substack{1 \le i \le N \\ y_i = \alpha}} \left(1 - P(\hat{y}_i = \alpha \mid \vec{x}_i ; W) \right) \vec{x}_i - \sum_{\substack{1 \le i \le N \\ y_i \neq \alpha}} P(\hat{y}_i = \alpha \mid \vec{x}_i ; W) \vec{x}_i \right]$$

$\therefore$ The update rule for the training procedure at time step $(t+1)$ is:

$$\vec{w}_{t+1}^{(\alpha)} = \vec{w}_t^{(\alpha)} - \eta \left[\lambda \vec{w}^{(\alpha)} + \sum_{\substack{1 \le i \le N \\ y_i = \alpha}} \left(1 - P(\hat{y}_i = \alpha \mid \vec{x}_i ; W) \right) \vec{x}_i - \sum_{\substack{1 \le i \le N \\ y_i \neq \alpha}} P(\hat{y}_i = \alpha \mid \vec{x}_i ; W) \vec{x}_i \right]$$

for all $\alpha \in \{1, 2, \cdots, C-1\}$ and $\vec{w}^{(0)} = 1$ is fixed.

Question 2

We first read the training dataset as $(\boldsymbol{X}, y)$ and the test dataset as $(\boldsymbol{X}_{test}, y_{test})$.

```
#Read data
X, y = load_svmlight_file('./a9a.txt')  #Training set
X_test, y_test = load_svmlight_file('./a9a.t')  #Test set
```

The training dataset was preprocessed and stored as a `.txt` file (i.e., `a9a.txt`), while the test dataset was preprocessed and stored as a `.t` file (i.e., `a9a.t`).

Since we are performing 3-fold cross validation in the training of both the linear SVM and non-linear SVM, we this as `kf`, as follows,

```
#Perform K-fold Validation, K=3
kf = KFold(n_splits = 3)
```

Here, 3-fold cross validation means that we partition our training dataset (i.e., $(\boldsymbol{X}, y)$) into 3 non-overlapping subsets, say $(\boldsymbol{X}_1, y_1)$, $(\boldsymbol{X}_2, y_2)$, and $(\boldsymbol{X}_3, y_3)$. With these non-overlapping subsets, we train the SVM model for 3 iterations, where in the i^{th} iteration, the subsets $(\boldsymbol{X}_j, y_j)$, for $j \neq i$, will be used to train the SVM model. Then, after the training, the subset $(\boldsymbol{X}_i, y_i)$ will be used as the validation set, where we test it against the SVM model to evaluate its accuracy. Here, accuracy is defined as,

$$\text{Accuracy}_i = \frac{n(\text{SVM}(\mathbf{x}_i) = y_i)}{|\boldsymbol{X}_i|}, \text{for all } \mathbf{x}_i \in \boldsymbol{X}_i,$$

where $n(\text{SVM}(\mathbf{x}_i) = y_i)$ denotes the number of correct classifications by the trained SVM.

Because we perform 3 iterations, we will obtain 3 accuracy scores corresponding to the evaluation on the 3 different subsets. We therefore take the average of the three accuracy scores as our accuracy score:

$$\text{Accuracy score} = \frac{1}{3}\sum_{i=1}^{3} \text{Accuracy}_i$$

The purpose of performing 3-fold cross validation is to select appropriate hyperparameters. For the linear SVM, the hyperparameter we are experimenting with is the value of C, which controls the trade-off between the L2 regularization term and the loss term. In particular, the higher the values of C, the weaker the regularization strength. To implement this, we first define a dictionary of parameters for the linear SVM as `params_linear`, then we prepare to exhaustively search over the different hyperparameter values, with the 3-fold cross validation scheme (i.e., `kf`), using the `GridSearchCV` function. We would also evaluate the accuracy of the different trained models (i.e., `scoring = ['accuracy']`). Finally, we train the model on our training set $(\boldsymbol{X}, y)$ by using the `.fit` method.

```
params_linear = {'C': [0.01, 0.05, 0.1, 0.5, 1], 'kernel': ['linear']}

svm_linear = GridSearchCV(svm.SVC(), params_linear, scoring = ['accuracy'], n_jobs = -2, refit=False, cv=kf, verbose=3)
svm_linear.fit(X,y)
```

We tabulate the accuracy score (to 4 significant figures) for the linear SVM in Table 1 below, for the different values of C.

$C = 0.01$	$C = 0.05$	$C = 0.1$	$C = 0.5$	$\boldsymbol{C = 1}$
0.8439	0.8462	0.8464	0.8467	$\mathbf{0.8472}$

Table 1: Accuracy score of linear SVM model for different values of C.

From the result above, we see that the linear SVM model with $C = 1$ performs the best.

We wish to also investigate the use of non-linear SVM model for our dataset. This is typically done if the distribution of the data is not linearly separable. Therefore, by fitting a non-linear SVM model, we attempt to implicitly project the data onto a higher dimensional space such that they could then be linearly separated. The implicit projection is characterized by a known kernel function.

In this example, we use a common kernel function called the Radial Basis Function (RBF). The function is defined as,

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\sigma^2}\right)$$
$$= \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|_2^2),$$

where $\mathbf{x}, \mathbf{x}' \in \boldsymbol{X}$ the original input data space, and $\gamma = \frac{1}{2\sigma^2}$.

The implementation of this non-linear SVM is similar to the linear SVM, where we perform 3-fold cross validation. But now, on top of the hyperparameter C, we have another hyperparameter γ as seen in the equation above. This value scales the value of the kernel function.

```
params_rbf = {'C': [0.01, 0.05, 0.1, 0.5, 1], 'kernel': ['rbf'], 'gamma':[0.01, 0.05, 0.1, 0.5, 1]}

svm_rbf = GridSearchCV(svm.SVC(), params_rbf, scoring = ['accuracy'], n_jobs = -2, refit=False, cv=kf, verbose=3)
svm_rbf.fit(X,y)
```

Similarly, we tabulate the accuracy scores (in 4 significant figures) for the 25 different permutations of C and γ values in the table below.

	$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.1$	$\gamma = 0.5$	$\gamma = 1$
$C = 0.01$	0.7592	0.8197	0.8195	0.7592	0.7592
$C = 0.05$	0.8309	0.8358	0.8341	0.7892	0.7592
$C = 0.1$	0.8379	0.8398	0.8390	0.8062	0.7620
$C = 0.5$	0.8428	0.8452	0.8465	0.8323	0.7899
$C = 1$	0.8444	$\mathbf{0.8469}$	0.8467	0.8363	0.7985

Table 2: Accuracy score of non-linear SVM model for different values of C and γ.

From the result above, we see that for the non-linear SVM with RBF kernel, the hyperparameters $(C, \gamma) = (1, 0.05)$ gives the best accuracy. Notably, however, the best of the non-linear SVM model performs worse than the best of the linear SVM model. As such, the best hyperparameter settings will be a linear SVM with $C = 1$.

We then take these hyperparameters for our final SVM model, train it on the entire training dataset $(\boldsymbol{X}, y)$ and evaluate it on our holdout test set $(\boldsymbol{X}_{test}, y_{test})$. Because feature 123 is 0 for all the test data instances, we will add an additional zero column in the matrix $\boldsymbol{X}_{test}$.

```
#Train final SVM on entire training set
params_final = {'C': 1, 'kernel': 'linear'}
svm_final = svm.SVC(**params_final)
svm_final.fit(X,y)

#Append zero column because feature 123 is 0 for all test data
X_test = np.array(X_test)
zeros_column = np.array([0] * X_test.shape[0])
X_test = np.hstack((X_test, zeros_column[:,np.newaxis]))

#Calculate accuracy
y_pred = svm_final.predict(X_test)
print(f"The accuracy on the test set is {accuracy_score(y_test, y_pred)}")
```

```
The accuracy on the test set is 0.8497635280388183
```

The accuracy of the test set is therefore 0.8498.

Question 3

In our initial formulation, the optimization problem of linear soft-margin SVM for a given training dataset $\{\vec{x}_i, y_i\}_{i=1}^{N}$ is stated as:

$$\min_{\vec{w}, b, \xi_i} \left[\frac{\|\vec{w}\|_2^2}{2} + C\left(\sum_{i=1}^{N} \xi_i\right) \right], \text{ s.t. } y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0, \ \forall\, i = 1, 2, \ldots, N.$$

Specifically, the constraint $y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \xi_i$ can be rewritten as $\xi_i \geq 1 - y_i(\vec{w} \cdot \vec{x}_i + b)$. Together with the constraint $\xi_i \geq 0$, we can compactly write $\xi_i \geq \max(0, 1 - y_i(\vec{w} \cdot \vec{x}_i + b))$. Because the objective is a summation of two nonnegative summands, one of which is $C\sum_{i=1}^{N} \xi_i$, that means we want to minimize ξ_i. Therefore, we take $\xi_i = \max(0, 1 - y_i(\vec{w} \cdot \vec{x}_i + b))$

Hence, our optimization problem can be written as:

$$\min_{\vec{w}, b} \left[\frac{\|\vec{w}\|_2^2}{2} + C \sum_{i=1}^{N} \max(0, 1 - y_i(\vec{w} \cdot \vec{x}_i + b)) \right] \quad \text{——— (*)}$$

This can then be interpreted as an instance of empirical structural risk minimization:

$$\min_{\vec{\theta}} \left[\sum_{i=1}^{N} \ell(f(\vec{x}_i; \vec{\theta}), y_i) + \lambda \Omega(\vec{\theta}) \right],$$

where ℓ is the loss function and $\Omega(\vec{\theta})$ is the structural regularization term of $\vec{\theta}$.

In particular, $\max(0, 1 - y_i(\vec{w} \cdot \vec{x}_i + b))$ in (*) corresponds to the loss function (ie. hinge loss, $\ell_{\text{hinge}}(f(\vec{x}_i; \vec{\theta}), y_i) = \max[0, 1 - y_i(f(\vec{x}_i; \vec{\theta})])$, while $\frac{\|\vec{w}\|_2^2}{2}$ in (*) corresponds to the structural regularization term (ie. L2-regularization, $\Omega_{L2}(\vec{\theta}) = \frac{\|\vec{\theta}\|_2^2}{2}$).

Hence, the initial formulation can be re-formulated as an instance of empirical structural risk minimization, involving hinge loss and L2 regularization.

Question 4

Under empirical structural risk minimization, applying the kernel trick on a regularized linear regression model to solve nonlinear regression problems yields the following optimization problem:

$$\min_{\vec{w}} \left[\frac{1}{2} \left(\vec{w} \cdot \varphi(\vec{x}_i) - y_i \right)^2 + \frac{\lambda}{2} \|\vec{w}\|_2^2 \right],$$

for a given training dataset $\{\vec{x}_i, y_i\}_{i=1}^N$ and non-linear transformation, $\varphi : \mathbb{R}^d \to \mathbb{R}^m$, where $m > d$ and $d = \dim(\vec{x}_i)$. In particular, φ is characterized by a known kernel function, $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, as $K(\vec{z}, \vec{z}') = \varphi(\vec{z}) \cdot \varphi(\vec{z}')$.

Note that the optimization problem is still unconstrained and the objective is still convex but this time with respect to $\varphi(\vec{x}_i)$ instead of $\vec{x}_i$.

Let $\vec{a}_i = \varphi(\vec{x}_i) \in \mathbb{R}^m$, $\forall i = 1, 2, \dots, N$. Then, we can use the closed-form solution for regularized linear regression, but in terms of $\vec{a}_i = \varphi(\vec{x}_i)$. Denote $A = [\vec{a}_1 \cdots \vec{a}_N]$, then the solution to the optimization problem is given to be,

$$\vec{w} = (AA^T + \lambda I)^{-1} A\vec{y}, \quad \text{where } \vec{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

$$\Rightarrow AA^T\vec{w} + \lambda\vec{w} = A\vec{y}$$

Now, we rearrange the terms to get

$$\vec{w} = A\left(\frac{1}{\lambda}(\vec{y} - A^T\vec{w})\right)$$

Denote $\vec{\alpha} = \frac{1}{\lambda}(\vec{y} - A^T\vec{w})$ ——— (*)

Then, $\vec{w} = A\vec{\alpha}$. Substituting this into (*) yields

$$\vec{\alpha} = \frac{1}{\lambda}(\vec{y} - A^T(A\vec{\alpha})) \Rightarrow \lambda\vec{\alpha} + A^TA\vec{\alpha} = \vec{y}$$

$$\therefore (A^TA + \lambda I)\vec{\alpha} = \vec{y}$$

Note that $(A^TA + \lambda I)$ is positive definite since, $\forall \vec{z} \neq \vec{0}$ we have,

$$\vec{z}^T(A^TA + \lambda I)\vec{z} = \vec{z}^TA^TA\vec{z} + \vec{z}^T\vec{z} = \underbrace{(A\vec{z})^T(A\vec{z})}_{\|A\vec{z}\|_2^2 \geq 0} + \underbrace{\vec{z}^T\vec{z}}_{\|\vec{z}\|_2^2 > 0 \ (\because \vec{z} \neq \vec{0})}$$

$\Rightarrow \vec{z}^T(A^TA + \lambda I)\vec{z} > 0$. By theorem, this means $A^TA + \lambda I$ is invertible.

$\therefore$ We have $\vec{\alpha} = (A^TA + \lambda I)^{-1}\vec{y}$. Substitute this into $\vec{w} = A\vec{\alpha}$, we have $\vec{w} = A(A^TA + \lambda I)^{-1}\vec{y}$

Note that, given a new data instance $\vec{x}^*$, the model predicts y^* as,

$$y^* = \vec{w} \cdot \phi(\vec{x}^*) = \vec{w}^T \varphi(\vec{x}^*)$$

$$= \vec{y}^T (A^T A + \lambda I)^{-1} A^T \varphi(x^*)$$

$$= \vec{y}^T \left(\begin{bmatrix} \varphi(\vec{x}_1)^T \\ \vdots \\ \varphi(\vec{x}_n)^T \end{bmatrix} \begin{bmatrix} \varphi(\vec{x}_1) \cdots \varphi(\vec{x}_n) \end{bmatrix} + \lambda I \right)^{-1} \begin{bmatrix} \varphi(\vec{x}_1)^T \\ \vdots \\ \varphi(\vec{x}_n)^T \end{bmatrix} \varphi(\vec{x}^*)$$

$$= \vec{y}^T \left(\begin{bmatrix} \varphi(\vec{x}_1)^T\varphi(\vec{x}_1) & \varphi(\vec{x}_1)^T\varphi(\vec{x}_2) & \cdots & \varphi(\vec{x}_1)^T\varphi(\vec{x}_n) \\ \vdots & \ddots & & \vdots \\ \varphi(\vec{x}_n)^T\varphi(\vec{x}_1) & \varphi(\vec{x}_n)^T\varphi(\vec{x}_2) & \cdots & \varphi(\vec{x}_n)^T\varphi(\vec{x}_n) \end{bmatrix} + \lambda I \right)^{-1} \begin{bmatrix} \varphi(\vec{x}_1)^T\varphi(\vec{x}^*) \\ \vdots \\ \varphi(\vec{x}_n)^T\varphi(\vec{x}^*) \end{bmatrix}$$

$$= \vec{y}^T (K + \lambda I)^{-1} \vec{k}^*, \text{ where } [K]_{ij} = k(\vec{x}_i, \vec{x}_j) \text{ and } \vec{k}^*_i = k(\vec{x}_i, \vec{x}^*)$$

Unlike the ordinary linear regression, $\vec{w}$ is not known explicitly, since the nonlinear transformation $\varphi : \mathbb{R}^d \to \mathbb{R}^m$ is not known $\Rightarrow$ matrix A is unknown. However, we still have a closed-form solution for y^*, since K and $\vec{k}^*$ can be found for a given kernel (i.e. known $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$) and $\vec{x}^*$.