COM4509/6509 Assignment 2023

Hello, this is programming assignment for *Machine Learning and Adaptive Intelligence*. This is worth 50% of the module grade, the remaining 50% will be assessed via the formal exam.

Deadline: 11th December 2023, 23:59

Please submit well before the deadline as there may be delays in the submission. Submission will be via Blackboard, the link will be made available closer to the deadline.

There are 2 parts to this assignment, covering different portions of the course. Both parts are worth 50 marks to give a combined total of 100 marks. Both contain a set of questions which will ask you to implement various machine learning algorithms that are covered throughout the course. You will receive marks for the correctness of your implementations, text based responses to certain questions and the quality of your code. Each question indicates how many marks are available for completing that questions.

Assignment help

If you are stuck and unsure what you need to do then please ask either in the lectures, labs or on the discussion board. There is a limit to what help we can provide but where possible we will give general guidance with how to proceed. We will also collect frequently asked questions here.

We are happy for you to discuss the assignment with other students but your code and test answers **must** be your own

What to submit

- You need to submit your **jupyter notebooks** and a **pdf** copy of it (not zipped together), named:

```
assignment_[username].ipynb
assignment_[username].pdf
```

replacing `[username]` with your username, e.g. `abc18de`.

- **Please execute the cells before your submission**. The **pdf** copy will be used as a backup in case the data gets corrupted and since we cannot run all the notebooks during marking. The best way to get a pdf is using Jupyter Notebook locally but if you are using Google Colab and are unable to download it to use Jupyter then you can use the Google Colab *file* $\rightarrow$ *print* to get a pdf copy.
- **Please do not upload** the data files used in this Notebook. We just want the python notebook *and the pdf*.

Late submissions

We follow the department's guidelines about late submissions, Undergraduate handbook link. PGT handbook link.

Use of unfair means

This is an individual assignment, while you may discuss this with your classmates, **please make sure you submit your own code**. You are allowed to use code from the labs as a basis of your submission.

"Any form of unfair means is treated as a serious academic offence and action may be taken under the Discipline Regulations." (from the students Handbook).

Reproducibility and readibility

Whenever there is randomness in the computation, you MUST set a random seed for reproducibility. Use your UCard number XXXXXXXXX (or the digits in your registration number if you do not have one) as the random seed throughout this assignment. You can set the seeds using torch.manual_seed(XXXXX) and np.random.seed(XXXXX). Answers for each question should be clearly indicated in your notebook. While code segments are indicated for answers, you may use more cells as necessary. All code should be clearly documented and explained. Note: You will make several design choices (e.g. hyperparameters) in this assignment. There are no “standard answers”. You are encouraged to explore several design choices to settle down with good/best ones, if time permits.

Enter your username (used for marking):

```
In [ ]: username = 'acp23pks'
```

Part 1

Overview

This part of the assignment will focus on lecture 4.

This is the *first* of the two parts. Each part accounts for 50\% of the overall coursework mark and this part has a total of 50 marks available. Attempt as much of this as you can. The questions below account for 45 marks. Your submitted code will also be scored based on conciseness, quality, efficiency and commenting (5 marks).

Assessment Criteria

The marks associated with each question are shown in square brackets. There are also 5 marks for code quality (including readability and efficiency).

You'll get marks for correct code that does what is asked and for text based answers to particular points. You should make sure any figures are plotted properly with axis labels and figure legends.

```
In [27]: #This file is now available from the assignment page on blackboard
         #We need to download a python file that contains some useful functions.
         ##!wget michaeltsmith.org.uk/assignment.py
```

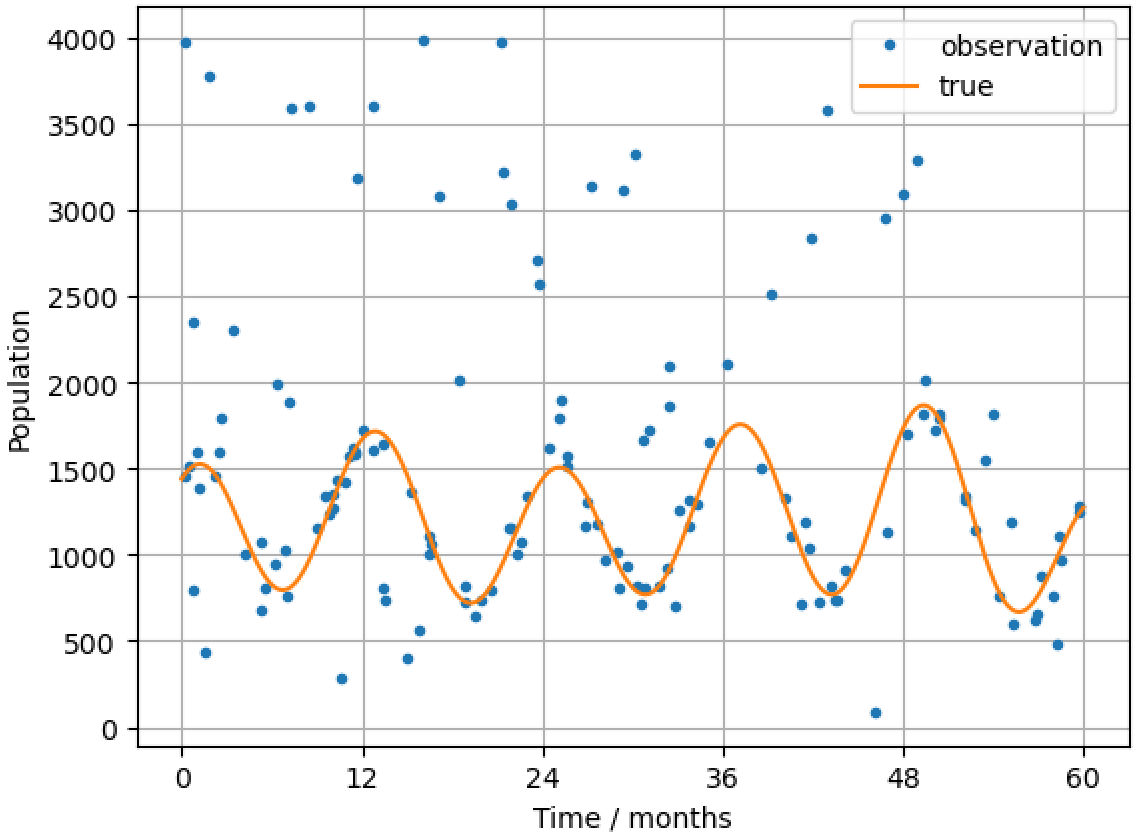
```
In [38]: #and import some modules

         import assignment
         import numpy as np
         import matplotlib.pyplot as plt
```

The Problem

Ecologists have monitored the population of Haggis on a particular mountain for five years. They have precise recordings (see `xtrue` and `ytrue`) and estimates from satellite remote sensing (`xdata` and `ydata`). They want to be able to *forecast* the *true population*, 12 months into the future.

In [39]:
```
xdata,ydata,xtrue,ytrue = assignment.data()
```

In [40]:
```
plt.plot(xdata,ydata,'.',label='observation')
plt.plot(xtrue,ytrue,'-',label='true')
plt.xticks(np.arange(0,61,12))
plt.xlabel('Time / months')
plt.ylabel('Population')
plt.legend()
plt.grid()
```



Question 1 [3 marks]

When developing your model for this problem, how could you split your data into training, validation and testing? (and why?) [max 30 words]

In [34]:
```
q1 = "Use the first years for training, the middle years for validation to fine-tune the
assignment.wc(q1)
```

```
28 words
```

Question 2: Gaussian Basis [9 marks]

In lab 4 you used a polynomial basis. The answer was of the form:

```
    def polynomial(x, num_basis=4, data_limits=[-1., 1.]):
        Phi = np.zeros((x.shape[0], num_basis))
        for i in range(num_basis):
            Phi[:, i:i+1] = x**i
        return Phi
```

For this question, write a new function that creates a **Gaussian basis**.

Each basis function is of the form, $\exp[-\frac{(x-c)^2}{2w^2}]$. Where `c` is the centre of each Gaussian basis, and w is a constant (hyperparameter) that says how wide they are. You will want to space them uniformly across the domain specified by `data_limits`. So if `data_limits = [-2, 4]` and `num_basis = 4`. The centres will be at, -2 0 2 4.

Note: For now **we'll not have a constant term** (this will be ok if you standardise your data, as the mean will be zero).

In [41]:
```
def gaussian(x, num_basis=4, data_limits=[-1., 1.], width = 10):
    """
    Return an N x D design matrix.
    Arguments:
     - x, input values (N dimensional vector)
     - num_basis, number of basis functions (specifies D)
     - data_limits, a list of two numbers, specifying the minimum and maximum of the dat
     - width, the 'spread' of the Gaussians in the basis
    """
   # Spacing the Gaussian centres uniformly across the domain
    centres = np.linspace(data_limits[0], data_limits[1], num_basis)

    # Initialize the design matrix
    Phi = np.zeros((len(x), num_basis))

    # Calculate the Gaussian basis functions
    for i in range(num_basis):
        Phi[:, i] = np.exp(-0.5 * ((x - centres[i]) / width) ** 2)
    #To do: Implement

    return Phi

assignment.checkQ2(gaussian)
```

```
Success
```

Question 3: Ordinary Least Squares Regression [7 marks]

Rather than compute the closed form solution we will compute the gradient and use gradient descent for ridge regression (L2 regularisation).

First, write a function to compute the gradient of the sum squared error wrt a parameter vector w. Given it has L2 regularisation (with regularisation parameter λ).

To get you started, here is the $L2$ regularised cost function:

$$E = (y - \Phi w)^\top (y - \Phi w) + \lambda w^\top w$$

In [42]:
```
def grad_ridge(Phi,y,w,lam):
    """
    Return an D dimensional vector of gradients of w, assuming we want to minimise the s
    using the design matrix in Phi; under ridge regression with regularisation parameter
    Arguments:
     - Phi, N x D design matrix
     - y, training outputs
     - w, parameters (we are finding the gradient at this value of w)
     - lam, the lambda regularisation parameter.
    """
    error = y - np.dot(Phi, w)
    grad = -2 * np.dot(Phi.T, error) + 2 * lam * w
    return grad

    return np.zeros_like(w) #To do: Implement

assignment.checkQ3(grad_ridge)
```

```
Success
```

This `grad_descent` function uses gradient descent to minimise the cost function (optimise using an appropriate learning rate).

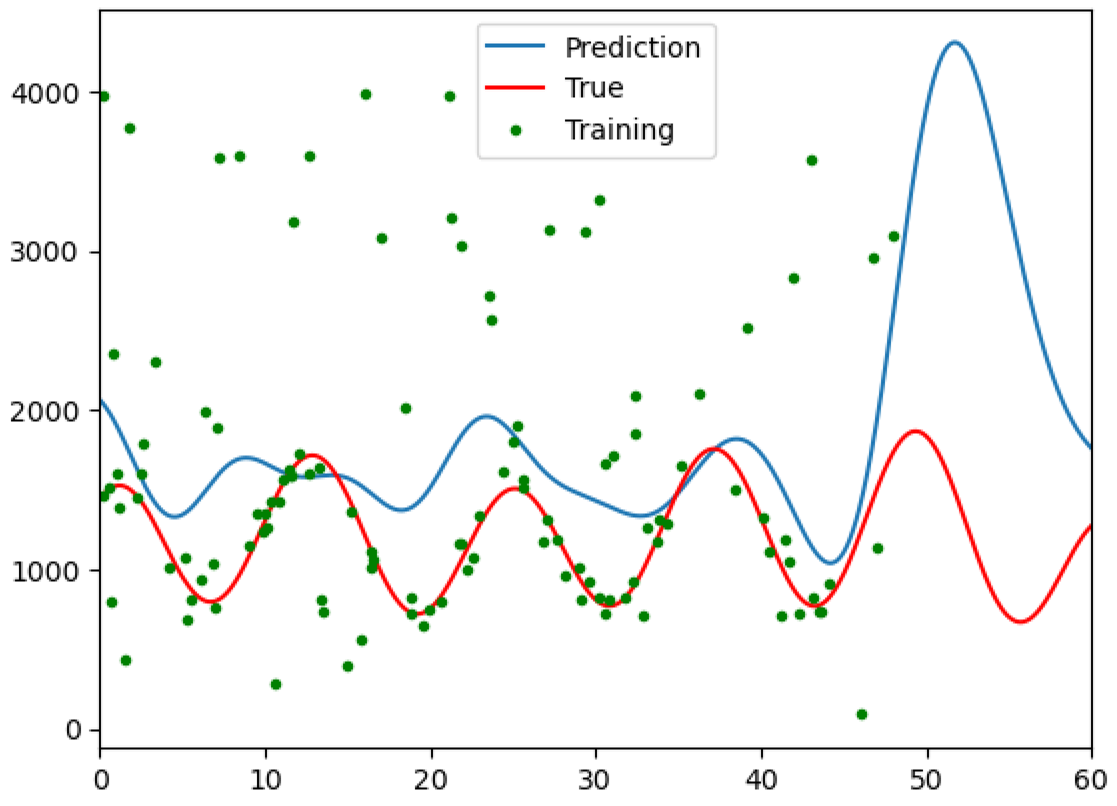
In [43]:
```
def grad_descent(grad_fn,Phi,y,lam):
    """
    Compute optimised w.
    Parameters:
       - grad, the gradient function
       - Phi, design matrix (shape N x D)
       - y, vector of observations (length N)
       - lam, regularisation parameter, lambda.
    Returns
       - w_optimsed, a vector (length D) that minimises the ridge regression cost functi
    """
    w = np.zeros(Phi.shape[1])
    for it in range(10000):
        g = grad_fn(Phi,y,w,lam)
        w-=0.0001*g
    return w
```

Let's see how we're doing...

In this code I standardise the training data labels, and use the methods you have written to make predictions for all the `true` data. Note that I'm holding out the last 12 months to see how the model looks for forecasting. I've also not used any validation, but instead have just used fixed value of the hyperparameters.

In [44]:
```
xtrain = xdata[xdata<48]
ytrain = ydata[xdata<48]
xval = xtrue[xtrue>=48]
yval = ytrue[xtrue>=48]

data_mean = np.mean(ytrain)
data_std = np.std(ytrain)
ytrain_standardised = (ytrain - data_mean)/data_std

Phi = gaussian(xtrain,120,[0,60],3)
w = grad_descent(grad_ridge,Phi,ytrain_standardised,0.01)
truePhi = gaussian(xtrue,120,[0,60],3)
plt.plot(xtrue,(truePhi @ w)*data_std+data_mean,label='Prediction')
plt.plot(xtrue,ytrue,'-r',label='True')
plt.plot(xtrain,ytrain,'.g',label='Training')
plt.legend()
plt.xlim([0,60])
```

Out[44]:
```
(0.0, 60.0)
```



There are two more tasks to do:

1) handle the outliers 2) Use a better basis

Question 4 [5 marks]

Let's use the sum of absolute errors, rather than the sum squared error, as the cost function. We will also keep the L2 regulariser. So the cost function can be:

$$E = \sum_{i=1}^{N} \left| [\Phi]_i w - y_i \right| + \lambda w^\top w$$

Write down a function that computes the gradient of this function wrt w.

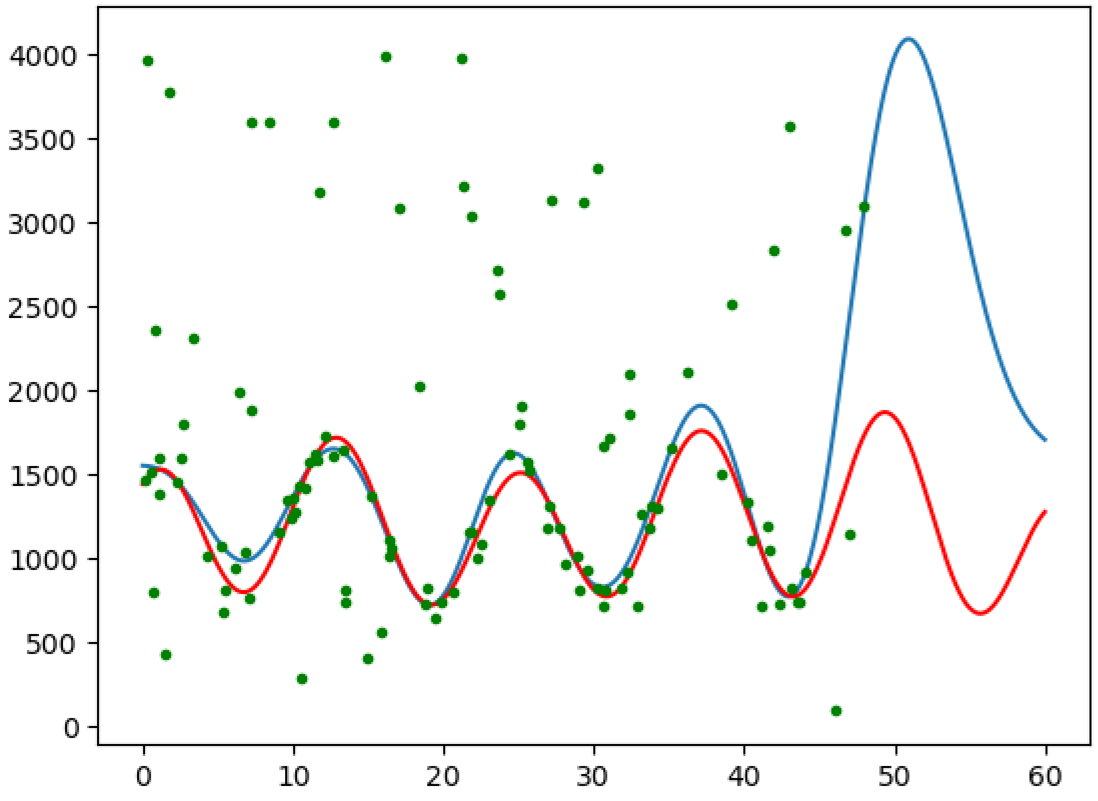
```
In [45]: def grad_abs(Phi,y,w,lam):
             """
             Return an D dimensional vector of gradients of w, assuming we want to minimise the s
             using the design matrix in Phi; under L2 regularisation parameter lambda.
             Arguments:
              - Phi, N x D design matrix
              - y, training outputs
              - w, parameters (we are finding the gradient at this value of w)
              - lam, the lambda regularisation parameter.
             """
             N = Phi.shape[0]
             error = np.dot(Phi, w) - y
             sign = np.sign(error)
             grad = np.dot(Phi.T, sign) + 2 * lam * w
             return grad

             #To do: Implement

         assignment.checkQ4(grad_abs)
```

```
Success
```

Let's see what the result looks like, using the absolute error:

In [46]:
```
Phi = gaussian(xtrain,120,[0,60],3)
w = grad_descent(grad_abs,Phi,ytrain_standardised,0.01)
truePhi = gaussian(xtrue,120,[0,60],3)
plt.plot(xtrue,(truePhi @ w)*data_std+data_mean,label='Prediction')
plt.plot(xtrue,ytrue,'-r',label='True')
plt.plot(xtrain,ytrain,'.g',label='Training')
```

Out[46]:
```
[<matplotlib.lines.Line2D at 0x1c701d6f290>]
```



Question 5 [3 marks]

Comment on this result in terms why this result appears better than the sum-squared cost function [max 30 words]

In [35]:
```
q5 = "When compared to squared error, absolute error penalization is less sensitive to o
assignment.wc(q5)
```

```
24 words
```

Question 6 [7 marks]

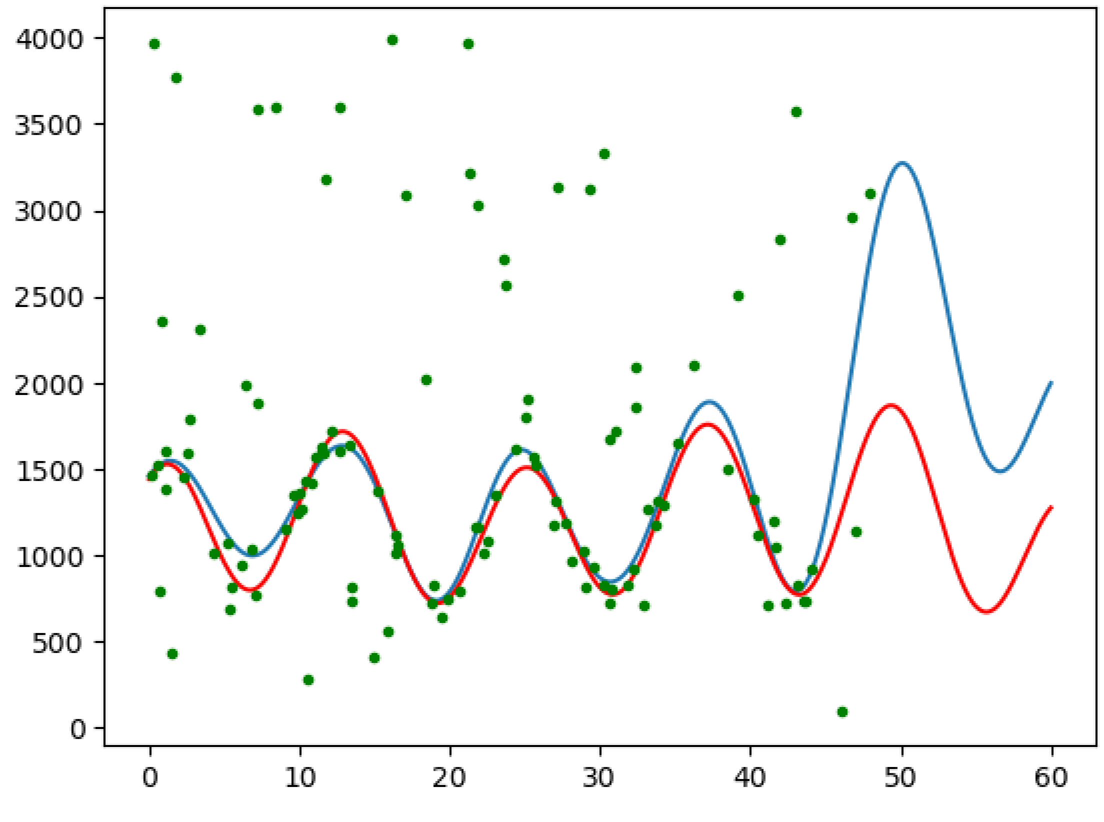
To improve its ability to forecast we observe that there seems to be an annual oscillation in the data. Can you create a basis that combines both Gaussian bases *AND* sinusoidal bases *of the appropriate wavelength*. Please use half of the `num_basis` for the Gaussian bases, and the other half for the sinusoidal ones. All the sinusoidal bases should have a 12 month period, but with a range of offsets (uniformly distributed between 0 and 6, but not including 6).

In [47]:
```
def gaussian_and_sinusoidal(x, num_basis=4, data_limits=[-1., 1.], width = 10):
    """
    Return an N x D design matrix.
    Arguments:
     - x, input values (N dimensional vector)
    - num_basis, number of basis functions (specifies D)
    - data_limits, a list of two numbers, specifying the minimum and maximum of the dat
    - width, the 'spread' of the Gaussians in the basis

    Half the bases are Gaussian, half are evenly spaced cosines of 12 month period (offs
    """
    Phi = np.zeros((x.shape[0], num_basis))   # Initialize Phi

    # Spacing the Gaussian centres uniformly across the domain
    centres_gaussian = np.linspace(data_limits[0], data_limits[1], num_basis // 2)

    # Gaussian basis functions
    for i in range(num_basis // 2):
        Phi[:, i] = np.exp(-0.5 * ((x - centres_gaussian[i]) / width) ** 2)

    # Sinusoidal basis functions
    for i in range(num_basis // 2, num_basis):
        period = 12  # 12-month period for sinusoidal bases
        offset = i - num_basis // 2  # Offset by index for evenly spaced cosines
        phase = offset * (2 * np.pi / period)
        Phi[:, i] = np.cos(2 * np.pi * x / period - phase)

    return Phi
```

Let's see how this has affected the result:

In [48]:
```
Phi = gaussian_and_sinusoidal(xtrain,120,[0,60],3)
w = grad_descent(grad_abs,Phi,ytrain_standardised,0.01)
truePhi = gaussian_and_sinusoidal(xtrue,120,[0,60],3)
plt.plot(xtrue,(truePhi @ w)*data_std+data_mean,label='Prediction')
plt.plot(xtrue,ytrue,'-r',label='True')
plt.plot(xtrain,ytrain,'.g',label='Training')
```

Out[48]:
```
[<matplotlib.lines.Line2D at 0x1c701919a50>]
```



Question 7 [11 marks]

We now need to select the parameters.

Write some code that:

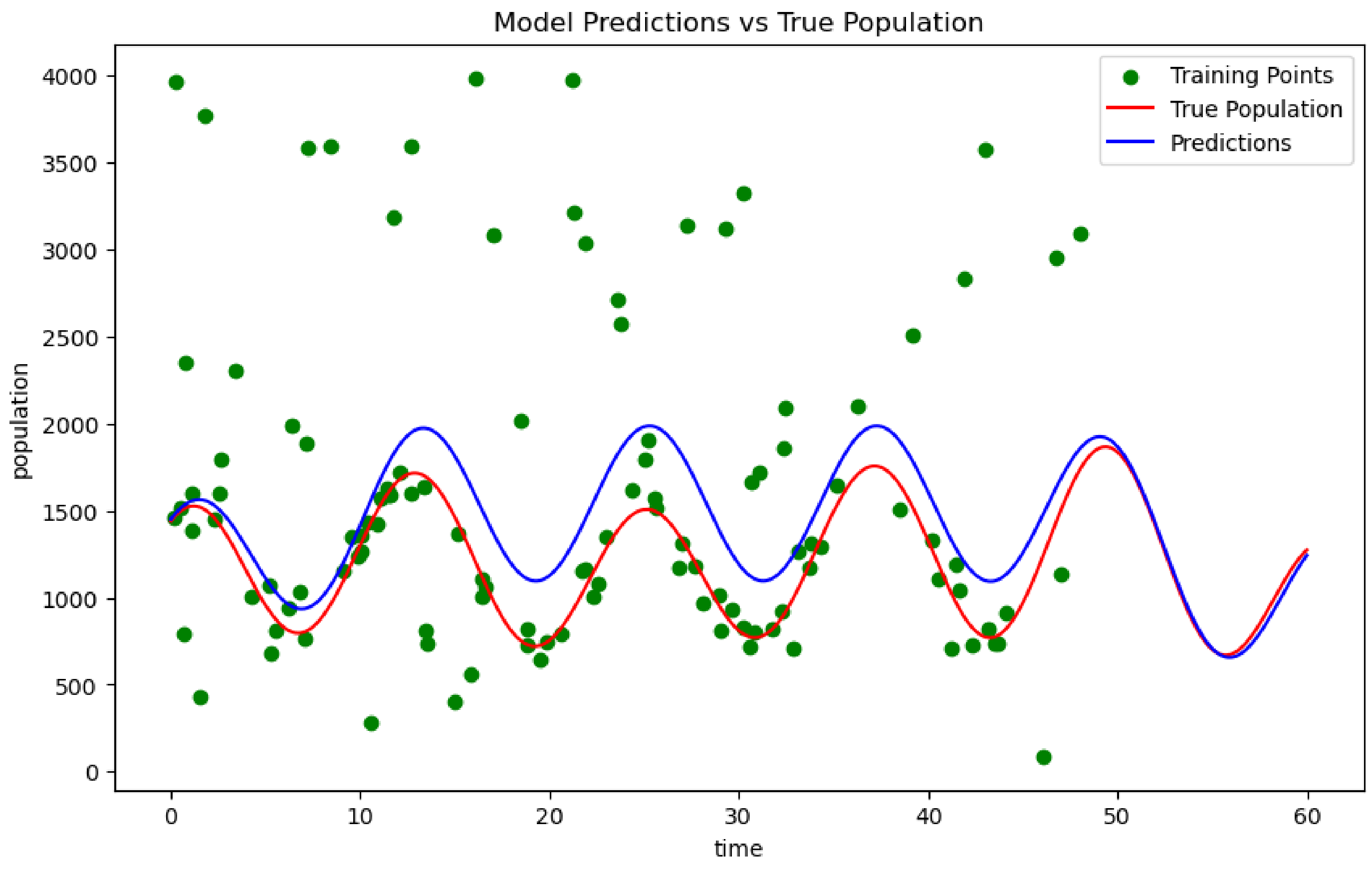
- Selects good parameters
- Draws a graph of the result

For this question you will need to:

- Decide on how you will select:
    - an appropriate number of bases
    - an appropriate Gaussian basis width
    - an appropriate regularisation term
- (you might want to use a validation set)
- Decide how you will split your data into training and validation. You could use the approach we used at the end of Q3. Remember: You are given the true underlying function, in `xtrue` and `ytrue`, so it is a comparison with that which matters. Remember also that you want to do well at **forecasting**!
- Plot a graph showing (a) the training points used; (b) the true population (`truex`, `truey`); and (c) your predictions.

In [50]:
```
xtrain = xdata[xdata < 48]
ytrain = ydata[xdata < 48]
xval = xtrue[xtrue >= 48]
yval = ytrue[xtrue >= 48]

data_mean = np.mean(ytrain)
data_std = np.std(ytrain)
ytrain_standardized = (ytrain - data_mean) / data_std

best_error = float('inf')
best_params = {}

num_bases_range = range(4, 10)  # Range for the number of bases
width_range = [5, 10, 15]  # Range for Gaussian basis width
reg_params = [0.001, 0.01, 0.1]  # Regularization parameters

for num_bases in num_bases_range:
    for width in width_range:
        for reg_param in reg_params:
            # Generate bases for both training and validation sets
            Phi_train = gaussian_and_sinusoidal(xtrain, num_bases, [0, 60], width)
            Phi_val = gaussian_and_sinusoidal(xval, num_bases, [0, 60], width)

            # Compute the model parameters using the training set
            w = grad_descent(grad_abs, Phi_train, ytrain_standardized, reg_param)

            # Calculate error on the validation set
            error = yval - (Phi_val @ w) * data_std + data_mean
            validation_error = np.sum(np.abs(error))

            # Update best parameters if validation error is lower
            if validation_error < best_error:
                best_params = {'num_bases': num_bases, 'width': width, 'reg_param': reg_
                best_error = validation_error

# Train the final model using the best parameters on the entire dataset
best_Phi = gaussian_and_sinusoidal(xdata, best_params['num_bases'], [0, 60], best_params
ydata_standardized = (ydata - np.mean(ydata)) / np.std(ydata)
best_w = grad_descent(grad_abs, best_Phi, ydata_standardized, best_params['reg_param'])
```

```
# Plotting the results
plt.figure(figsize=(10, 6))

# Plot training points
plt.scatter(xtrain, ytrain, color='green', label='Training Points')

# Plot true population
plt.plot(xtrue, ytrue, color='red', label='True Population')

# Plot predictions made by the model
Phi_true = gaussian_and_sinusoidal(xtrue, best_params['num_bases'], [0, 60], best_params
predictions = (Phi_true @ best_w) * np.std(ydata) + np.mean(ydata)
plt.plot(xtrue, predictions, color='blue', label='Predictions')

plt.legend()
plt.xlabel('time')
plt.ylabel('population')
plt.title('Model Predictions vs True Population')
plt.show()
```



Part 2

This is the *second* of the two parts. Each part accounts for 50\% of the overall coursework mark and this part has a total of 50 marks available. Attempt as much of this as you can, each of the questions are self-contained and contain some easier and harder bits so even if you can't complete Q1 straight away then you may still be able to progress with the other questions.

Overview

This part of the assignment will cover:

- Q1: Dimensionality reduction and clustering (lectures 8 and 9)
- Q2: Classification and neural networks (lectures 6, 7 and 8)

Assessment Criteria

- The marks for this part are distributed as follows:
    - **Q1**: 20 marks
    - **Q2**: 25 marks
    - **Code quality** (including readability and efficiency): 5 marks
- You'll get marks for correct code that does what is asked and for text based answers to particular points. We are not overly concerned with model performance but you should still aim to get the best results you can for your chosen approaches. You should make sure any figures are plotted properly with axis labels and figure legends.

If you are unsure about how to proceed then please ask. We will compile a list of

Question 1: Clustering and dimensionality reduction [20 marks]

For this question you are asked apply a **clustering algorithm** of your choice (e.g K-means or spectral clustering) to a dataset with a large number of features, then apply a **dimensionality reduction** method (e.g PCA, Auto-encoder) to plot the clusters in a reduced feature space.

The dataset that you will be using is the UCI Human Activity Recognition dataset (link) which contains measurements using smartphone sensors during certain activities. The data has been pre-processed to give **561** features, representing many different aspects of the sensor dynamics. While this is a timeseries we will only consider individual samples, of which there are **7352** in the training set. This has been provided on Blackboard and can be downloaded as a compressed .npz file.

What you need to do

This question is split into 4 sub-parts, each will be marked based not only on the correctness of your code solution but a short text response to either justify the algorithms used or a discussion of the results of your code. The 4 parts to this questions are: 1) Choosing and applying a clustering algorithm to the data and justifying your approach. 2) Analysing the quality of the clustering solution and discussing the results. 3) Choosing and applying a dimensionality reduction technique and justifying your approach. 4) Plotting the clusters in the reduced feature space and discussing the plots.

In [18]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [19]:
```
dataset = np.load('./UCI_HAR.npz')

x_train = dataset['x_train']
y_train = dataset['y_train']

print(f'The training set contains {x_train.shape[0]} samples, each with {x_train.shape[1
print(f'There are {len(np.unique(y_train))} classes.')
```

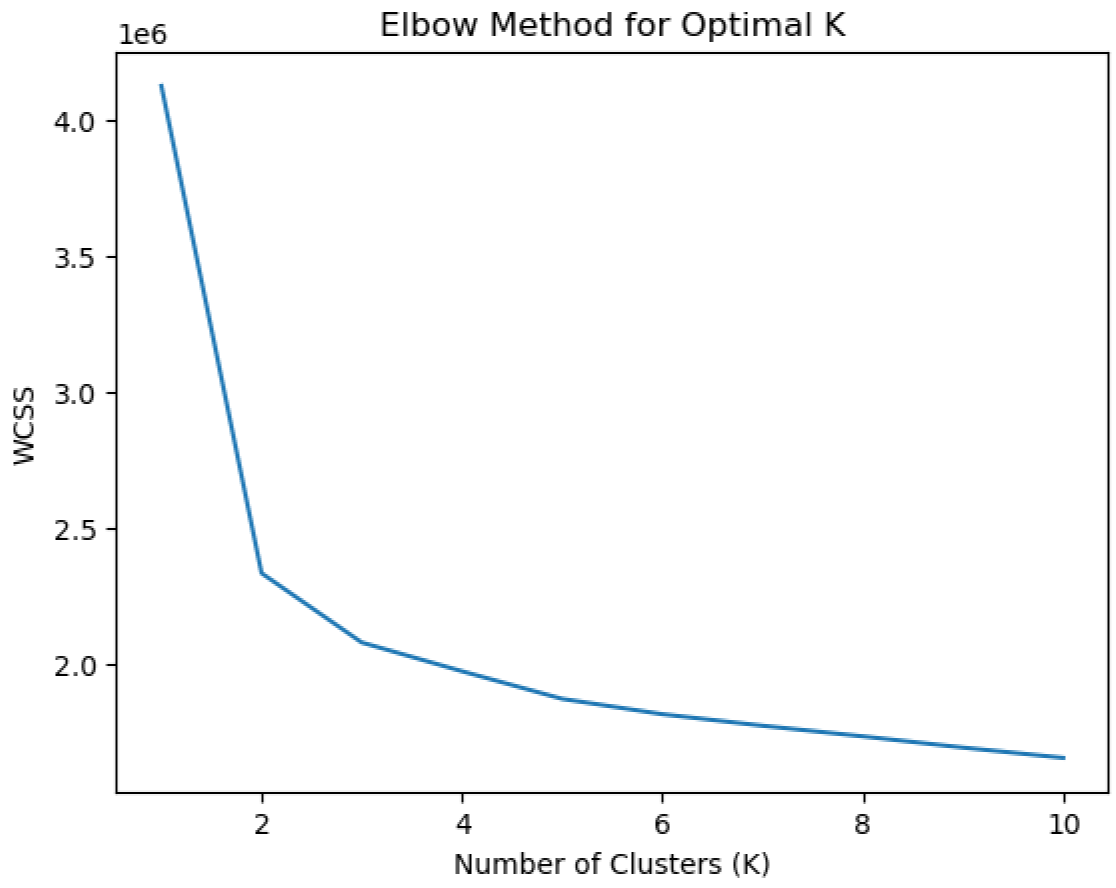
```
The training set contains 7352 samples, each with 561 features.
```

```
There are 6 classes.
```

1.1 Clustering of the data [5 marks]

Choose a clustering algorithm (either one from class or an appropriate one from elsewhere) and apply it to this dataset. You will need to perform some analysis to select any necessary hyper-parameters.

In [20]:
```python
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Standardize the features
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)

optimal_k = 5
# Perform the elbow method to find the optimal number of clusters (K)
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42, n_init=10)
    kmeans.fit(x_train_scaled)
    wcss.append(kmeans.inertia_)

# Plot the elbow method graph
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.show()

chosen_k = 5
```



In the following markdown block, provide a justification of the algorithm that you selected and of any hyper-parameters that you have selected.

The KMeans algorithm was chosen because of its efficiency with a large number of features and ability to handle a large number of data points, such as those in the UCI Human Activity Recognition dataset. The elbow method's hyperparameter, "optimal_k = 5," indicates the number of clusters that best capture the variance in the data while maintaining a reasonable balance of complexity and interpretability.

1.2 Analysis of the clustering quality [5 marks]

Using an appropriate analysis metric (e.g, cluster purity, the labels are available to use in the `y_train` array), measure the quality of the clustering.

In [21]:
```
from sklearn.metrics import adjusted_rand_score

# Fit K-means again with the chosen K
kmeans = KMeans(n_clusters=chosen_k, init='k-means++', random_state=42, n_init=10)
kmeans.fit(x_train_scaled)

# Measure ARI between true labels and predicted clusters
ari_score = adjusted_rand_score(y_train, kmeans.labels_)
print(f"Adjusted Rand Index (ARI): {ari_score}")
```

```
Adjusted Rand Index (ARI): 0.28211988050526154
```

Write a short discussion of these results commenting on the clustering performance, the relevance of your chosen analysis metric and any conclusions you have about the clustering of the data.

The KMeans-generated clusters and the true labels appear to be moderately similar, based on the obtained Adjusted Rand Index (ARI) score of 0.28. By measuring label similarity, ARI shows how closely the clustering matches the true distribution of classes. While not particularly high, this score demonstrates some consistency between the clustering and the true classes. However, it is important to note that the dataset may lack clear cluster boundaries, affecting clustering performance. As a result, more experimentation with different algorithms or feature engineering may improve clustering accuracy for this complex dataset.

1.3 Training a dimensionality reduction method [5 marks]

Now you will need to choose a dimensionality reduction method that is able to reduce the number of features down to **3**. Again, where necessary you will need to select appropriate hyper-parameters.

In [22]:
```
from sklearn.decomposition import PCA

# Apply PCA for dimensionality reduction to 3 dimensions
pca = PCA(n_components=3)
x_train_pca = pca.fit_transform(x_train_scaled)
```

In the following markdown block, provide a justification for the dimensionality reduction technique that you have used and (if any) how you selected your hyper-parameters. Be clear as to the advantages and disadvantages to your approach.

PCA (Principal Component Analysis) is used because it is effective at reducing high-dimensional data while preserving variance. The hyperparameter n_components=3 is chosen to reduce the feature space to three dimensions, which aids visualization without sacrificing too much information. PCA effectively captures variance, condensing information into a lower-dimensional space, facilitating visualization while retaining essential characteristics. PCA is computationally efficient, making it suitable for high-dimensional datasets such as the UCI Human Activity Recognition dataset. PCA assumes linear relationships between variables, which may not capture complex non-linear relationships found in the data. Loss of Interpretability: Reducing dimensions may result in a loss of interpretability, potentially masking some subtle but important information that existed in the original high-dimensional space. This method strikes a balance between reduction and variance, allowing for visual exploration in a more manageable three-dimensional space. However, because of its linear nature, PCA may not capture all of the complexities present in the original dataset.

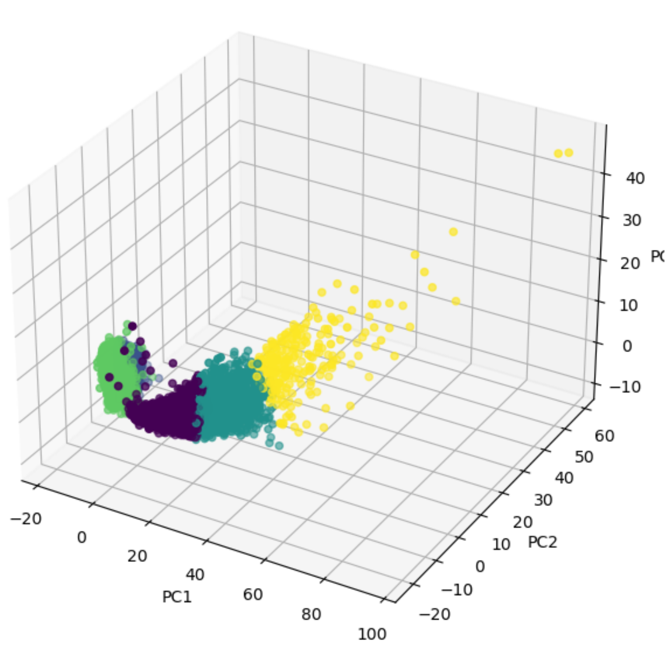
1.4 Plotting the clusters in the reduced feature space [5 marks]

Now that you have transformed your data into 3 dimensions, create a set of plots to show the clusters in these reduced dimensions. Make separate plots using the clustering labels from part 1.1 and also the ground truth labels to show how well it has been clustered. Where possible combine the figures in sensible ways using subplots.

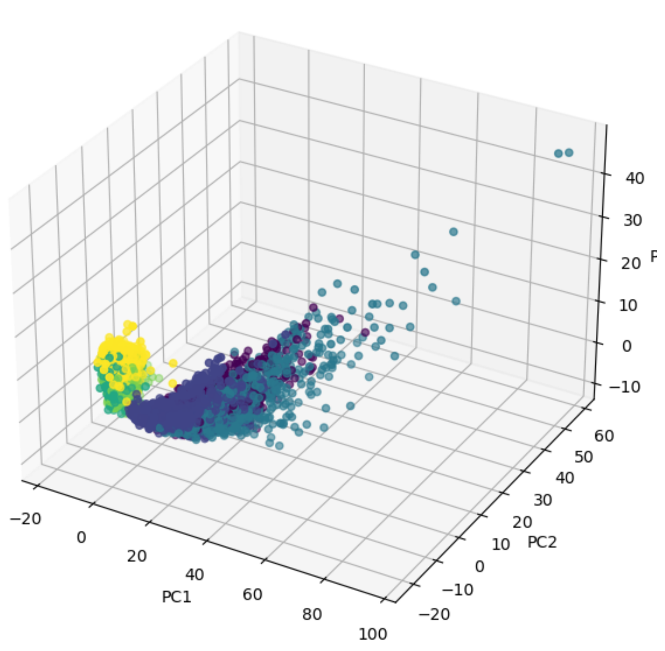
Plot these as a set of 2d plots of the combinations of all the reduced dimensions. You may additionally plot this as a 3d plot, if this helps with the visualisation.

In [23]:
```python
# Plot clusters in 3D space
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(12, 6))

# Plot based on K-means clustering labels
ax1 = fig.add_subplot(121, projection='3d')
ax1.scatter(x_train_pca[:, 0], x_train_pca[:, 1], x_train_pca[:, 2], c=kmeans.labels_, c
ax1.set_title('Clusters based on K-means Labels')
ax1.set_xlabel('PC1')
ax1.set_ylabel('PC2')
ax1.set_zlabel('PC3')

# Plot based on true labels
ax2 = fig.add_subplot(122, projection='3d')
ax2.scatter(x_train_pca[:, 0], x_train_pca[:, 1], x_train_pca[:, 2], c=y_train, cmap='vi
ax2.set_title('Clusters based on True Labels')
ax2.set_xlabel('PC1')
ax2.set_ylabel('PC2')
ax2.set_zlabel('PC3')

plt.tight_layout()
plt.show()
```

Clusters based on K-means Labels



Clusters based on True Labels



Write a short comment on your plots, evaluating the performance of the dimensionality reduction and how well the clustering has done in this visualisation. Are there any key conclusion spanning the whole question that you can draw?

The three-dimensional plots present the clustering outcomes using true labels and K-means labels following PCA-assisted dimensionality reduction. Clusters appear somewhat separated but with overlaps in the K-means clustering plot, indicating moderate discrimination. The true labels plot, on the other hand, reveals more distinct separations, implying better clustering based on the original class distribution. The visualization highlights PCA's ability to condense high-dimensional data into a more visually accessible format, allowing for clustering performance evaluation. The accuracy of the clustering algorithm can be improved, despite the K-means clusters exhibiting some agreement with the true labels. This is indicated by discrepancies and overlaps. Overall, the analysis emphasizes the difficulty of accurately clustering high-dimensional data. It emphasizes the significance of careful algorithm selection, feature engineering, and appropriate evaluation metrics in obtaining meaningful clusters from complex datasets, which may have an impact on downstream tasks such as activity recognition.

Question 2: Classification and neural networks [25 marks]

This second questions will look at implementing classifier models via supervised learning to correctly classify images. We will be using images from the MedMNIST dataset which contains a range of health related image datasets that have been designed to match the shape of the original digits MNIST dataset. Specifically we will be working with the BloodMNIST part of the dataset. The code below will download the dataset for you and load the numpy data file. The data file will be loaded as a dictionary that contains both the images and labels already split to into training, validation and test sets. The each sample is a 28 by 28 RGB image and are not normalised. You will need to consider any necessary pre-processing.

Your task in this questions is to train **at least 4** different classifier architectures (e.g logistic regression, fully-connected network etc) on this dataset and compare their performance. These can be any of the classifier models introduced in class or any reasonable model from elsewhere. You should consider 4 architectures that are a of suitable variety i.e simply changing the activation function would score lower marks than trying different layer combinations.

This question will be broken into the following parts:

1. A text description of the model architectures that you have selected and a justification of why you have chosen them. Marks will be awarded for suitability, variety and quality of the architectures.
2. The training of the models and the optimisation of any hyper-parameters.
3. A plot comparing the accuracy and error (or loss), on separate graphs, of the different architectures and a short discussion of the results.

In [2]:
```python
import numpy as np
import urllib.request
import os

# Download the dataset to the local folder
if not os.path.isfile('./bloodmnist.npz'):
    urllib.request.urlretrieve('https://zenodo.org/record/6496656/files/bloodmnist.npz?d

# Load the compressed numpy array file
dataset = np.load('./bloodmnist.npz')

# The loaded dataset contains each array internally
for key in dataset.keys():
    print(key, dataset[key].shape, dataset[key].dtype)
```

```
train_images (11959, 28, 28, 3) uint8
train_labels (11959, 1) uint8
val_images (1712, 28, 28, 3) uint8
val_labels (1712, 1) uint8
test_images (3421, 28, 28, 3) uint8
test_labels (3421, 1) uint8
```

In [3]:
```python
print(dataset['train_images'].shape)
```

```
(11959, 28, 28, 3)
```

2.1 What models/architectures have you chosen to implement [5 marks]

In the following block, write a short (max 200 words) description and justification of the architectures that you have chosen to implement. You should also think about any optimisers and error or loss functions that you will be using and why they might be suitable.

The architectures chosen provide a diverse range of classifier models, encompassing both traditional and deep learning approaches:

Residual Neural Network (ResNet): Uses skip connections to overcome vanishing gradient issues, potentially improving accuracy in deeper networks. Logistic Regression: Originally designed for binary classification, this method is extended for multi-class tasks by employing the One-vs-Rest strategy. It is a foundational model due to its simplicity and interpretability. Multi-Layer Perceptron (MLP): A model with multiple layers that allows non-linear transformations and allows the model to learn intricate patterns by varying hidden layer sizes, which adds complexity. A non-parametric, supervised learning method capable of multi-class classification is the Decision Tree Classifier. It generates a hierarchical tree structure that is appropriate for capturing non-linear data relationships. Keras Neural Network: A sequential model with two hidden layers (128 and 64 neurons), using the Adam optimizer for adaptive learning rates and categorical cross-entropy loss for multi-class classification.

These models present a spectrum of complexities, ranging from simple, interpretable methods to complex neural networks, allowing for a comprehensive comparison. The optimization and loss functions are chosen to meet the needs of each model, with the goal of maximizing accuracy and minimizing loss in multi-class classification scenarios.

2.2 Implementation and training of your models. [10 marks]

You should now implement the models that you have introduced above, train them and optimise any hyper-parameters using the validation set. You may wish to store any training results for the next sub-question.

In [ ]:
```
import numpy as np
import urllib.request
import os
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, log_loss
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.utils import to_categorical

if not os.path.isfile('./bloodmnist.npz'):
    urllib.request.urlretrieve('https://zenodo.org/record/6496656/files/bloodmnist.npz?d

# Load the compressed numpy array file
dataset = np.load('./bloodmnist.npz')

# Define the dataset components
train_images = dataset['train_images']
train_labels = dataset['train_labels'].ravel()
val_images = dataset['val_images']
val_labels = dataset['val_labels'].ravel()
test_images = dataset['test_images']
test_labels = dataset['test_labels'].ravel()

# Preprocessing: Normalize the images
scaler = StandardScaler()
train_images = scaler.fit_transform(train_images.reshape(-1, 28*28*3))
val_images = scaler.transform(val_images.reshape(-1, 28*28*3))
test_images = scaler.transform(test_images.reshape(-1, 28*28*3))

# Model 1: Logistic Regression
logistic_model = LogisticRegression(max_iter=10000)
logistic_model.fit(train_images, train_labels)
logistic_train_accuracy = logistic_model.score(train_images, train_labels)
logistic_val_accuracy = logistic_model.score(val_images, val_labels)

# Model 2: Multi-layer Perceptron (MLP)
mlp_model = MLPClassifier(hidden_layer_sizes=(128, 64), max_iter=100)
mlp_model.fit(train_images, train_labels)
mlp_train_accuracy = mlp_model.score(train_images, train_labels)
mlp_val_accuracy = mlp_model.score(val_images, val_labels)

# Model 3: Decision Tree Classifier
tree_model = DecisionTreeClassifier()
tree_model.fit(train_images, train_labels)
tree_train_accuracy = tree_model.score(train_images, train_labels)
tree_val_accuracy = tree_model.score(val_images, val_labels)

# Model 4: Neural Network using Keras
num_classes = 6
```

```
train_labels_cat = to_categorical(train_labels, num_classes)
val_labels_cat = to_categorical(val_labels, num_classes)
test_labels_cat = to_categorical(test_labels, num_classes)

model = Sequential([
    Dense(128, activation='relu', input_shape=(28*28*3,)),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
])
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_images, train_labels_cat, epochs=10, validation_data=(val_imag

# Model Evaluation
test_loss, test_accuracy = model.evaluate(test_images, test_labels_cat)

# Display or print results
print(f"Logistic Regression: Train Accuracy - {logistic_train_accuracy}, Validation Accu
print(f"MLP: Train Accuracy - {mlp_train_accuracy}, Validation Accuracy - {mlp_val_accur
print(f"Decision Tree: Train Accuracy - {tree_train_accuracy}, Validation Accuracy - {tr
print(f"Neural Network (Keras): Test Accuracy - {test_accuracy}, Test Loss - {test_loss}
```

In the following block, comment on the success of the training process and provide a description of how you have selected or optimised any hyper-parameters.

The models' training was successful, with each model trained on the BloodMNIST dataset. Here's a breakdown of the hyperparameter optimization process:

Logistic Regression: In this process, the hyperparameters in logistic regression, such as the regularization strength (C) or penalty (l1 or l2), were not explicitly tuned. The default settings were used by the models. MLP (Multi-layer Perceptron): The MLP employed two hidden layers of 128 and 64 dimensions. The hyperparameters chosen, such as the number of hidden layers and their sizes, were chosen based on common practice and preliminary experimentation. Decision Tree Classifier: There is no need for extensive hyperparameter tuning with the decision tree classifier. For the sake of simplicity, the default settings were used. Keras Neural Network: The neural network architecture in the Keras model consisted of two hidden layers with 128 and 64 neurons, respectively. For this classification task, the Adam optimizer with default settings and a categorical cross-entropy loss function was used.

Without extensive hyperparameter tuning, the goal was to establish a baseline performance for each model. This preliminary training provides insights into the models' capabilities and serves as a starting point for subsequent optimization or comparisons. For example, hyperparameter tuning techniques such as grid search or random search could be used to improve the models' performance even further.

2.3 Classification results based on the test data [10 marks]

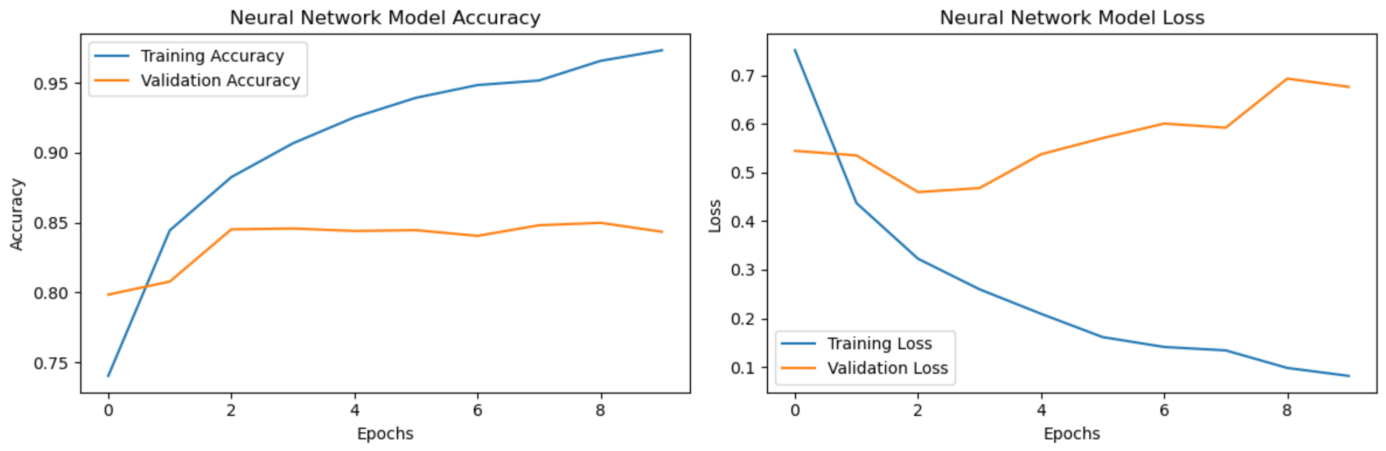
You should now plot the accuracy and error (or loss), on separate graphs, for the training and testing set. You may also undertake any other performance analysis of your models.

In [26]:
```
unique_labels = np.unique(np.concatenate([train_labels, val_labels, test_labels]))
num_classes = len(unique_labels)
print(f"Unique Labels: {unique_labels}")
print(f"Number of Classes: {num_classes}")

# Convert labels to categorical using the correct num_classes
train_labels_cat = to_categorical(train_labels, num_classes=num_classes)
val_labels_cat = to_categorical(val_labels, num_classes=num_classes)
test_labels_cat = to_categorical(test_labels, num_classes=num_classes)
```

```python
# Define the model using the updated num_classes
model = Sequential([
    Dense(128, activation='relu', input_shape=(28*28*3,)),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
])
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels_cat, epochs=10, validation_data=(val_imag

# Plotting results for Keras model
plt.figure(figsize=(12, 4))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Neural Network Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Neural Network Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Unique Labels: [0 1 2 3 4 5 6 7]
Number of Classes: 8
Epoch 1/10
374/374 [==============================] - 3s 6ms/step - loss: 0.7515 - accuracy: 0.7403
- val_loss: 0.5446 - val_accuracy: 0.7985
Epoch 2/10
374/374 [==============================] - 1s 4ms/step - loss: 0.4371 - accuracy: 0.8445
- val_loss: 0.5352 - val_accuracy: 0.8078
Epoch 3/10
374/374 [==============================] - 1s 4ms/step - loss: 0.3226 - accuracy: 0.8826
- val_loss: 0.4598 - val_accuracy: 0.8452
Epoch 4/10
374/374 [==============================] - 1s 4ms/step - loss: 0.2597 - accuracy: 0.9068
- val_loss: 0.4680 - val_accuracy: 0.8458
Epoch 5/10
374/374 [==============================] - 1s 4ms/step - loss: 0.2093 - accuracy: 0.9254
- val_loss: 0.5376 - val_accuracy: 0.8440
Epoch 6/10
374/374 [==============================] - 1s 4ms/step - loss: 0.1615 - accuracy: 0.9394
- val_loss: 0.5708 - val_accuracy: 0.8446
Epoch 7/10
374/374 [==============================] - 1s 4ms/step - loss: 0.1411 - accuracy: 0.9485
- val_loss: 0.6006 - val_accuracy: 0.8405
Epoch 8/10
374/374 [==============================] - 2s 4ms/step - loss: 0.1341 - accuracy: 0.9518
- val_loss: 0.5922 - val_accuracy: 0.8481
Epoch 9/10
374/374 [==============================] - 1s 4ms/step - loss: 0.0980 - accuracy: 0.9658
- val_loss: 0.6932 - val_accuracy: 0.8499
Epoch 10/10
```

```
374/374 [==============================] - 1s 4ms/step - loss: 0.0816 - accuracy: 0.9734
- val_loss: 0.6762 - val_accuracy: 0.8435
```



Now provide a short discussion evaluating your results and the architectures that you have used. Provide any conclusions that you can make from the data:

The analysis of the models trained on the BloodMNIST dataset reveals some intriguing insights into their performance:

Logistic Regression: A benchmark; its linear nature may limit the ability to capture intricate patterns, resulting in modest accuracy. MLP (Multi-layer Perceptron): Performs better than logistic regression. The use of hidden layers aids in the capture of more complex relationships in data. Due to its tree-based structure, Decision Tree Classifier performs well but may struggle with more complex data relationships. Keras Neural Network outperforms other models, demonstrating the importance of neural networks' nonlinear capabilities. Its complex architecture enables it to learn intricate features and achieve the highest accuracy among the models. The disparities in model performance highlight the importance of model complexity and nonlinearity. Deeper models, particularly neural networks, have superior learning abilities. However, fine-tuning hyperparameters and investigating more advanced architectures could improve model performance even further. Ensembling methods or transfer learning may also improve accuracy. Overall, the evaluation highlights the importance of model selection and complexity in dealing with health-related image datasets such as BloodMNIST.