Using Regression Methods for Spectral Reconstruction of Monochromatic Light

Mahtab Hosseinidolama, Daniel Paz

October 2023

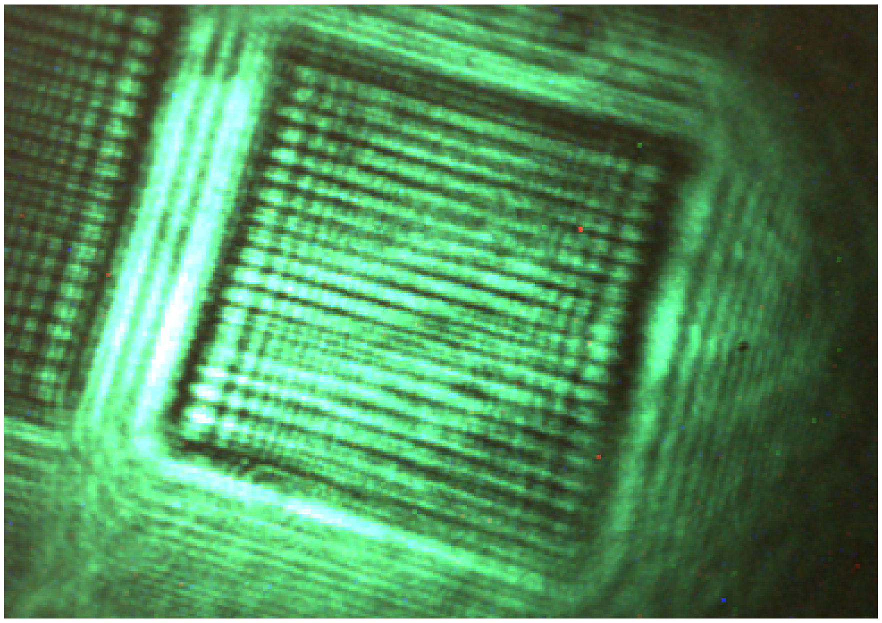


Figure 1: Ti-Au optical micro-array device used for data collection, exposed to laser light (515 *nm*)

Contents

1 Introduction

1.1 Background

Computational spectrometry is a recently emerged field, which utilises computational techniques, including Machine Learning, to decompose incident light, based on certain device responses. Traditional bench-top laboratory spectrometer systems offer exceptional ultra-fine resolution and broad spectral coverage. However, they come with substantial physical footprints, cost, and power requirements. In this sense, computational spectrometry offers a promising alternative due to its potential for producing inexpensive spectrometers of sufficient resolution with minimal footprint [11].

1.2 Objective

The primary objective of this project is to study regression-based algorithms for the reconstruction of monochromatic light, based on image samples taken from micro-devices fabricated in Micronova.

1.3 Report Outline

This report offers a comprehensive overview of our project, covering the background, the machine learning problem, the methodology employed, and our results.

Section 2, 'Problem Formulation', provides a formal framework for the machine learning problem. It contextualises the problem within the realm of the field and existing literature. Additionally, it details the construction of our model, outlining the creation of data points, features, and labels.

Section 3, 'Methods', delves into the specifics of the data-set used in the project. Additionally, it provides a detailed rationale for the two choices of the machine learning models explored in the report: Linear Regression and Multi-Layer Perceptron, offering a comprehensive explanation for the design decisions made.

Sections 4 and 5 of the report discuss the results obtained from the applied methods and provide a conclusion regarding the data.

2 Problem Formulation

2.1 Data Source

The data collection process is conducted with an experimental setup situated in a laser laboratory. In this configuration, a laser is precisely directed and focused onto a specific area of a Ti-Au optical filter micro-array. Positioned behind the filter array is a broadband camera sensor, tasked with capturing intensity measurements at various points across the array.

For each wavelength in a predefined set of values within the visible spectrum range, the laser light is filtered to produce monochromatic light. Subsequently, the sensor generates an image of the filter array, which is then collected as data for the project.

In the experiment, the filter array has been intentionally designed to exhibit randomly varying light transmission characteristics at distinct locations. This implies that the transmission measured at different points of the device will fluctuate in an unknown manner, further depending upon the wavelength of incident light. The use of regression, in this context, seeks to 'learn' how the fluctuating transmittance at various points correlates with the incident wavelength.

For the purposes of this project, only narrow-band monochromatic light has been considered for the data-set, such that each spectrum can be approximated as a single wavelength.

This research was conducted with the authorisation of the Photonics Group at the Department of Electrical Engineering, Aalto University.

2.2 Machine Learning Problem

Based on the experimental configuration, the ith (labelled) data point $d^{(i)} := (x^{(i)}, y^{(i)})$, corresponds to a n-dimensional feature vector $x^{(i)} := (x_1, x_2, \dots, x_m)^T$, containing the measured transmissions (after normalisation) at m fixed pixels of the array image, for the signal i. Moreover, it contains the label $y^{(i)} := \lambda^{(i)}$, representing the wavelength of light of the signal, in nm. Note that the same pixels are to be taken as features for each data point, and both the features and labels are continuous data.

The objective is to use machine learning methods to develop and optimise an algorithm capable of accurately predicting the incident wavelength, based on training and testing conducted with the data obtained for intensities measured at different pixels.

3 Methodology

3.1 Raw Data and Pre-processing

The output from the camera sensor consists of data in the form of Bitmap Image Files (with a '.bmp' file extension). These images are in RGB format and are centred around a specific section of the filter array. To conduct regression analysis, the images must undergo pre-processing. This pre-processing involves cropping the images to include only the pixels within the array upon which the laser is incident (see Figure 1) and converting the RGB values to grayscale. The grayscale conversion process involves summing the red, green, and blue components of each pixel into a single metric, which is taken as a measure of the transmission intensity.

The data-set comprises 271 files, each with the '*.bmp*' file extension. Furthermore, the images were captured at wavelengths ranging from 470 to 740 nm, with a one nanometre increment.

Due to the small size of the data-set, data augmentation is conducted through linear interpolation. Data is artificially created in-between existing points, by a factor of five. This value has been chosen due to considerations of achieving a sizeable data sample while maintaining a reasonable data-set that is physically relevant (with a wavelength step size $\Delta\lambda = 0.2nm$).

3.2 Feature Selection Process

To select the m fixed pixel points to be considered for obtaining the features in all data-points, a set of linearly spaced pixels $\{p_1, p_2, \dots, p_m\}$ along the cropped image is to be constructed. Then, for each data-point, the grayscale values for each of the pixels are taken as the features, following a normalisation process conducted using the 'MinMaxScaler' function from the 'Scikit-learn' library.

3.3 Linear Regression (model justification and hypothesis space)

The choice of linear regression as the machine learning method stems from the assumption that, in this configuration, the relationship between transmittance intensities and wavelength can be linearly approximated. This choice is grounded in both practical and theoretical reasons.

Firstly, the use of linear regression in spectral reconstruction algorithms, based on spectral-encoded information as features, has shown promising results in a variety of configurations [12, 13, 2]. Moreover, within the specific project set-up, the data can be assumed to meet the conditions of independence between observations, homoscedasticity, normality, and no multicollinearity between the feature variables - all of which are requirements for linear regression models [5, 10].

On the other hand, selecting linear regression offers the advantage of serving as a simpler baseline model. In contrast with more complex algorithms, such as the Multi-Layer Perceptron, linear regression enables a more meaningful discussion within the context of the trade-offs between interpretability, transparency, and speed versus reconstruction capabilities. Given this choice of model, the hypothesis space can be characterised as:

$$\mathcal{H}^{(i)} := \{h^{(w)} : \mathbb{R}^m \to \mathbb{R} \mid h^{(w)}(x) := x^T w, \text{ where } w \in \mathbb{R}^m\} \tag{1}$$

Meaning that our hypothesis space is constituted by linear maps [6].

3.4 Multi-layer Perceptron (model justification and hypothesis space)

The selection of the Multi-layer Perceptron (MLP) as the secondary model is based on its capability to introduce non-linearity into the modeling process. Considering that Linear Regression does not account for non-linear patterns, MLP networks are employed in the project to provide a more balanced approach. These networks are characterised by multiple layers and non-linear activation functions, offering the capacity to establish flexible decision boundaries that can adapt to non-linear patterns in data [1].

Furthermore, MLP models are renowned for their ability to autonomously identify important patterns within raw data. In the context of this project, grayscale transmission images might harbour intricate patterns not readily apparent to human observers. By leveraging MLP models, the necessity for extensive manual feature engineering can be avoided.

MLP models also offer adaptability in choosing the number of hidden layers and neurons per layer based on the data-set characteristics. Given the relatively small data-set size at our disposal, a decision has been made to employ a single hidden layer for this model [8]. For the number of neurons, a linear space, ranging from 20 to 100 neurons with 20-neuron increments, has been implemented. The Rectified Linear Unit (Re-LU) is adopted as the activation function for the hidden layer, while Stochastic Gradient Descent (SGD) is designated as the optimisation algorithm for the model.

The hypothesis space of this model encompasses a collection of functions that involve linear combinations of input features transformed by the Re-LU activation function. Through the manipulation of the number of neurons within the hidden layer, the model exhibits the capacity to capture varying levels of data complexity. The incorporation of Re-LU activation introduces non-linearity into the model, enabling it to effectively model intricate data patterns and relationships.

3.5 Loss function

For this method, the mean squared error (MSE) served as the primary metric to assess model performance on the testing data-set. This choice ensures maximum consistency between the optimisation carried out during training and model validation. This is attributed to the use of the Scikit-learn library, which employs regression techniques to find the analytical solution that minimises the sum of squared residuals [3].

Additionally, MSE offers several advantages over other loss functions, including high interpretability, sensitivity to outliers [9], and high-suitability to the model's continuous label space, serving as an appropriate metric for the accuracy of our model.

3.6 Model Validation

After data augmentation, the data-set is divided into training and testing sets using an $80 : 20$ ratio, with elements randomly selected for each set to minimise model over-fitting. This ratio strikes a balance between having a sufficiently large training set for accurate predictions and a substantial testing set for rigorous evaluation, based on both empirical and statistical analysis [4].

Once the data is split, k-fold cross validation is performed within the training set, selecting the model iteration with minimum MSE on the validation set. For the project, $k = 5$ has been chosen, to obtain a balance in computational performance as well as minimising the bias of the performance estimator [7].

4 Results

When comparing both methodologies used, the results seem to indicate a trade-off between minimising testing and validation errors. On average, Linear Regression computed a training loss on all of its folds more than 20 times smaller than that of the MLP model. This result may be attributed to a much larger variation in the obtained losses for the predictions of the MLP on the various validation sets, ranging from over 6800 to under 22, compared to Linear Regression, which ranged from 75.3 to 93.9 (Table 1).

These results suggest a high degree of robustness for the linear regression model employed, regarding the folds used for training. In contrast, the MLP shows much greater sensitivity to the choice of folds, as well as to the number of neurons located in the hidden layer. Nevertheless, when comparing the loss function on the testing data-set, computed after the validation process, the data suggests the MLP model is more efficient at predicting unseen data.

With these results in mind, the choice of the optimal model should be made based on the computational capabilities available for the algorithm's implementation. Although MLP can be trained over a large parameter space with exhaustive validation, the high degree of variability and sensitivity to parameters may require extensive cross-validation techniques for optimal results. In contrast, linear regression has been shown to provide a highly robust model, consistently providing 'approximate' results, with a testing error of 78.7, and all MSE values computed in the range of 75-90. This model may, therefore, be more favourable for small computational chips and similar devices with limited computational power, which may be used for optical spectrometry.

Table 1: MSE scores (5 SF)

Model	Mean Training Error	Mean Validation Error	Testing error
Linear Regression	64.904	82.498	78.693
Multi-Layer Perceptron	1870.5	1887.9	22.807

5 Conclusion

Overall, when considering the models of Linear Regression and MLP, as well as their results on the testing data-set, it can be concluded that, under appropriate training, both approaches are able to provide optimal approximations of monochromatic light based on experimental data collected, under a certain margin of error.

As seen in the Appendix, plotting the predictions of both models against their actual labels indicates a linear trend, reflecting the capability of both models to make accurate predictions within a reasonable degree of accuracy, consistently for the entire testing set.

Nevertheless, some areas for improvement to be considered include the limited scope of the training and validation, both in terms of the data-set size and the lack of cross-validation for hyper-parameter selection. Moreover, the absence of regularisation, as well as the applicability of image-processing techniques (such as the use of convolution), indicate areas for future research that would be expected to further optimise the models' prediction capabilities. Similarly, as discussed in the results, the trade-offs observed between model simplicity and sensitivity to hyper-parameters in both models indicate the need for further considerations in the project domain to make optimal algorithm choices.

Bibliography

[1] Amanatulla. *Unraveling the Magic: How Multilayer Perceptron Captures Non-Linearity in Data.* Medium. 2023. URL: https://medium.com/@amanatulla1606/unraveling-the-magic-how-multilayer-perceptron-captures-non-linearity-in-data-6a4d385f7592#:~:text=By%20incorporating%20multiple%20layers%20and,boundary%20on%20non%20linear%20datasets (visited on 10/10/2023).

[2] W. Deng et al. "Electrically tunable two-dimensional heterojunctions for miniaturized near-infrared spectrometers". In: *Nature Communications* 13.1 (). DOI: 10.1038/s41467-022-32306-z.

[3] S. ES. *7 Cross-validation Mistakes That Can Cost You a Lot [Best Practices in ML].* Accessed on 20 September 2023. 2023. URL: https://neptune.ai/blog/cross-validation-mistakes.

[4] A. Gholamy, V. Kreinovich, and O. Kosheleva. *Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation.* Accessed on 20 September 2023. 2018. URL: https://scholarworks.utep.edu/cs_techrep/1209/.

[5] M. Gupta. *Linear regression in machine learning.* Accessed on 20 September 2023. 2023. URL: https://www.geeksforgeeks.org/ml-linear-regression/.

[6] A. Jung. *Machine learning: the basics.* Springer Verlag, Singapore, 2023. Chap. 2.2.

[7] V. Lyashenko. *Cross-validation in Machine Learning: How To Do It Right.* Accessed on 20 September 2023. 2023. URL: https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right.

[8] H.S. Sachdev. *Choosing Number of Hidden Layers and Number of Hidden Neurons in Neural Networks.* Accessed: 10 October 2023. 2020. URL: https://www.linkedin.com/pulse/choosing-number-hidden-layers-neurons-neural-networks-sachdev.

[9] G. Seif. *Understanding the 3 Most Common Loss Functions for Machine Learning Regression.* Accessed on 20 September 2023. 2022. URL: https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3.

[10] *Testing the assumptions of linear regression.* Accessed on 20 September 2023. URL: https://people.duke.edu/~rnau/testing.htm.

[11] Z. Yang et al. "Miniaturization of optical spectrometers". In: *Science* 371.6528 (2021). DOI: 10.1126/science.abe0722.

[12] H.H. Yoon et al. "Miniaturized spectrometers with a tunable van der Waals junction". In: *Science* 378.6617 (2022), pp. 296–299. DOI: 10.1126/science.add8544.

[13] S. Yuan et al. "A wavelength-scale black phosphorus spectrometer". In: *Nature Photonics* 15.8 (2021), pp. 601–607. DOI: 10.1038/s41566-021-00787-x.

optical-spectrometry-machine-learning

November 6, 2023

Appendix

[1]:
```
# Import necessary packages and modules
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.neural_network import MLPRegressor
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
import os
```

[3]:
```
# File path-related constants
General_Path = # Your path here
Image_Path = os.path.join(General_Path, 'images')

# Image-related constants
Image_Height = 484
Image_Width = 644
File_Extension = '.bmp'
Cropped_Left = int(Image_Width*0.45)
Cropped_Right = int(Image_Width*0.85)
Cropped_Top = int(Image_Height*0.1)
Cropped_Bottom = int(Image_Height*0.65)

# Signal related constants
Min_Wavelength = 470
Max_Wavelength = 740
Num_Wavelengths = 271
```

[4]:
```
# Function used to return the Gray values (i.e. intensity) of a specific pixel␣
 ↪for a given image
def gray_values(wavelength, pixel):
    this_image_path = os.path.join(Image_Path, str(int(wavelength)) + '.bmp')
    this_image = Image.open(this_image_path)
    rgb = this_image.getpixel(pixel)
    gray = sum(rgb[:3])
    return gray
```

[5]:
```
# Parameters
Sampling_Factor = 10                    # Factor equal to the number of␣
 ↪equally-spaced divisions of the image, used for sampling
Testing_Ratio = 0.2                     # Proportion of data points to be␣
 ↪considered for testing
seed = 30                               # Seed for initialising random state
Num_Folds = 5                           # Number of folds for k-cross validation
int_factor = 5                          # Interpolation factor                  ␣
 ↪
show_folds = True                       # Parameter
```

[6]:
```
# Constants
Num_Pixels = Sampling_Factor**2                         # Number of pixels of the␣
 ↪image to sample = number of features
datapoints_intp_num = Num_Wavelengths*int_factor         # Number of data points␣
 ↪after interpolation
```

<h2>Preprocessing</h2>

[7]:
```
# Construct NumPy array containing an evenly spaced set of points corresponding␣
 ↪to pixels on the image
x_coords = np.linspace(Cropped_Left, Cropped_Right, Sampling_Factor)
y_coords = np.linspace(Cropped_Top, Cropped_Bottom, Sampling_Factor)
x_grid, y_grid = np.meshgrid(x_coords, y_coords)
pixels = np.column_stack((x_grid.flatten(), y_grid.flatten()))
```

<h3>Interpolation</h3>

[8]:
```
# Equally spaced NumPy array containing the labels
y = np.linspace(Min_Wavelength, Max_Wavelength, Num_Wavelengths)

# Construct the feature matrix
X = np.zeros((Num_Wavelengths, Num_Pixels))
for i in range(Num_Wavelengths):
  for j in range(Num_Pixels):
      X[i, j] = gray_values(y[i], tuple(pixels[j]))

# defining arrays for the features and interpolated labels
y_interpolated = np.linspace(Min_Wavelength, Max_Wavelength,␣
 ↪datapoints_intp_num)
X_interpolated= np.zeros((datapoints_intp_num, Num_Pixels))
```

[9]:
```
# Interpolate the feature data for the desired wavelengths
for r in range(Num_Pixels):
   X_interpolated[:,r] = sp.interpolate.UnivariateSpline(y, X[:,␣
 ↪r])(y_interpolated[:])

# Convert the result to a NumPy array
X_interpolated = np.array(X_interpolated)
```

<h3>Spliting the data</h3>

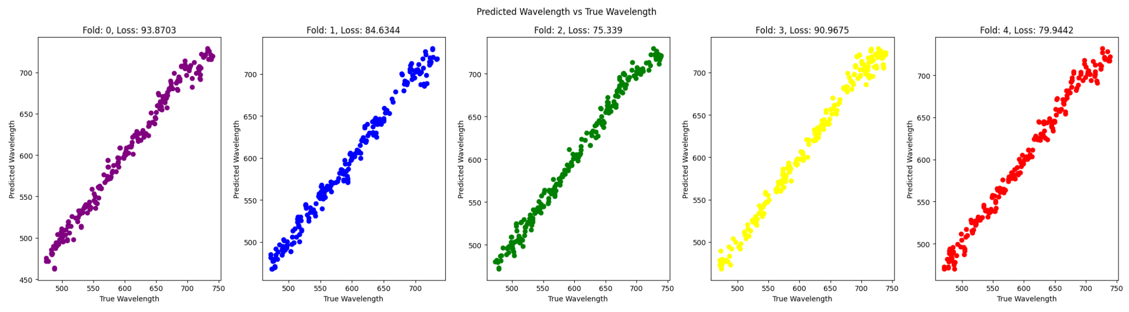
[10]:
```
# Separate the labels of the dataset into two arrays for training and testing
np.random.seed(seed)
X_trainNVal, X_test, y_trainNVal, y_test = train_test_split(X_interpolated,␣
 ↪y_interpolated, test_size=0.2, random_state=seed)
```

<h3>Normalization</h3>

[11]:
```
# Normalizing the data

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit the scaler on training data and transform it
X_train_scaled = scaler.fit_transform(X_trainNVal)

# Transform test data using the same scaler
X_test_scaled = scaler.transform(X_test)
```

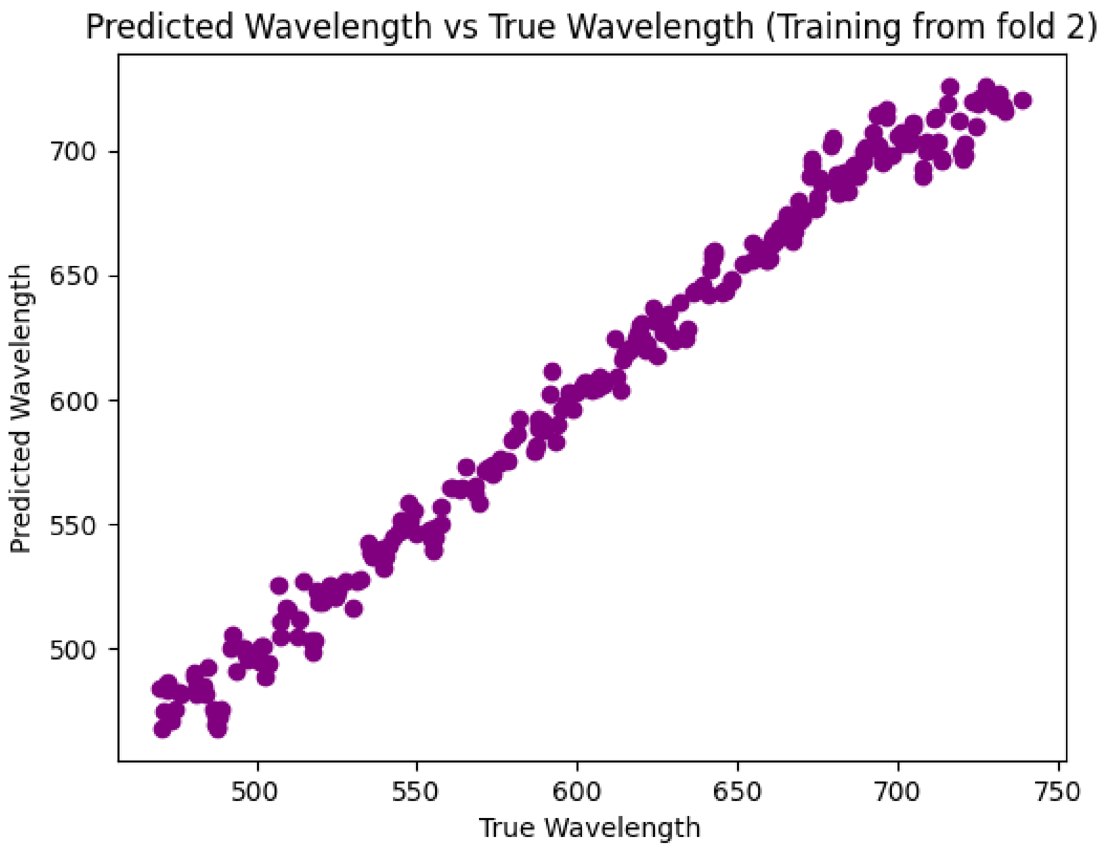
<h2>Linear Regression</h2>

[18]:
```
# Perform KFold cross-validation
kf = KFold(n_splits=Num_Folds)
reg_val_scores = []
reg_train_scores = []

# Create a figure with subplots
colours = ('purple', 'blue', 'green', 'yellow', 'red')
fig, axs = plt.subplots(nrows=1, ncols=Num_Folds, figsize=(22, 6))

for fold_num, (train_index, val_index) in enumerate(kf.split(X_train_scaled)):
    X_train, X_val = X_train_scaled[train_index], X_train_scaled[val_index]
    y_train, y_val = y_trainNVal[train_index], y_trainNVal[val_index]

    # Create a Linear Regression model
    model = LinearRegression()

    # Fit the model on the training data
    model.fit(X_train, y_train)

    #make predictions on the training data
    y_train_pred = model.predict(X_train)

    # Make predictions on the validation data
    y_val_pred = model.predict(X_val)

    # Caclulate Mean Squared Error (MSE) for this fold
    mse_train = mean_squared_error(y_train, y_train_pred)
    reg_train_scores.append(mse_train)
    mse_val = mean_squared_error(y_val, y_val_pred)
    reg_val_scores.append(mse_val)

    if mse_val == min(reg_val_scores):
       best_fold = fold_num
       y_test_pred = model.predict(X_test_scaled)

    # Add the figure to the plot
    axs[fold_num].scatter(y_val, y_val_pred, color=colours[fold_num])
    axs[fold_num].set_xlabel('True Wavelength')
    axs[fold_num].set_ylabel('Predicted Wavelength')
    axs[fold_num].set_title(f'Fold: {fold_num}, Loss: {round(mse_val, 4)}')

plt.suptitle('Predicted Wavelength vs True Wavelength')
plt.tight_layout()
plt.show()

mean_mse_train1 = np.mean(reg_train_scores)
mean_mse_val1 = np.mean(reg_val_scores)
test1_mse = mean_squared_error(y_test, y_test_pred)
print('Mean MSE across all folds in training set:', mean_mse_train1)
print('Mean MSE across all folds in validation set:', mean_mse_val1)
print('MSE for test data:', test1_mse)
```



```
Mean MSE across all folds in training set: 64.71325185088597
Mean MSE across all folds in validation set: 84.95109234853764
MSE for test data: 79.8801843960225
```

[13]:
```
plt.scatter(y_test, y_test_pred, color='purple')
plt.xlabel('True Wavelength')
plt.ylabel('Predicted Wavelength')
plt.title(f'Predicted Wavelength vs True Wavelength (Training from fold␣
  ↪{best_fold})')
plt.show()
```

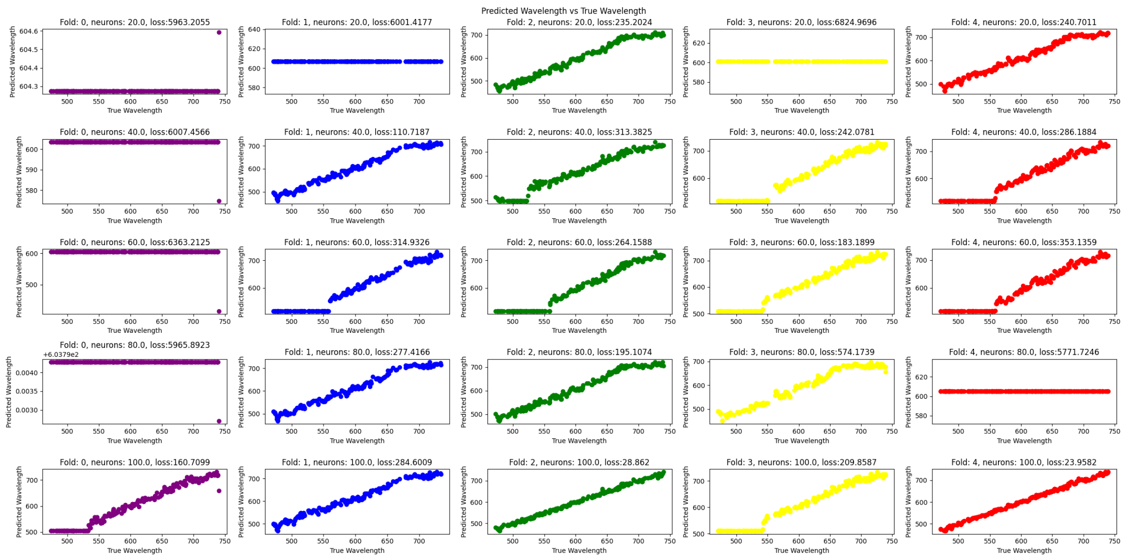


<h2>MLP Regression</h2>

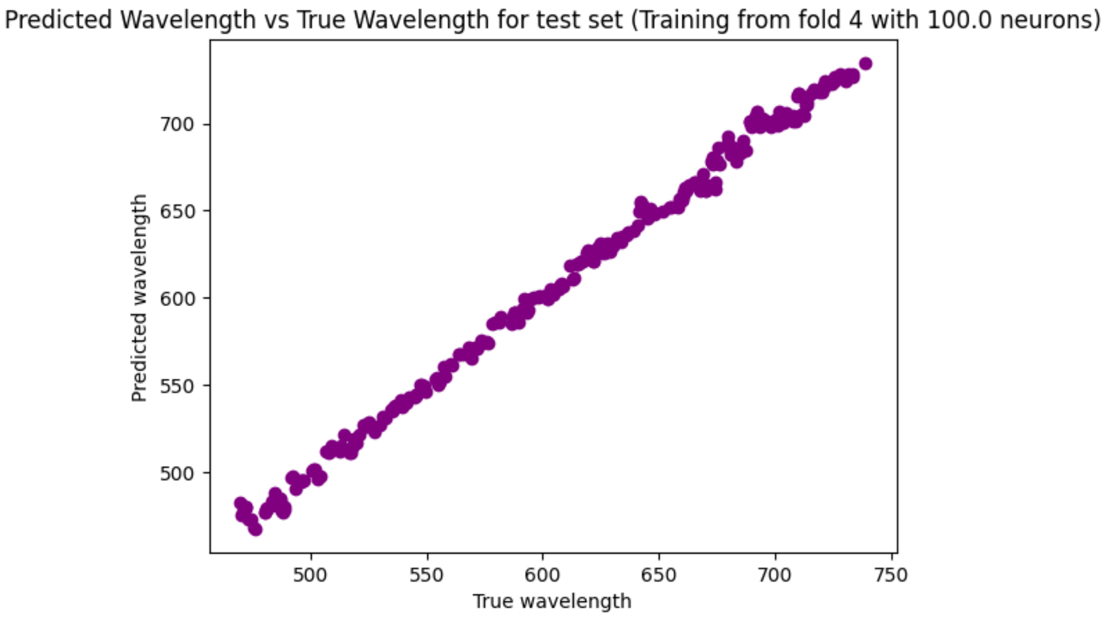
[20]:
```
# Define network architecture
initial_neuron = 20
final_neuron = 100
num_steps = 5
num_neurons = np.linspace(initial_neuron, final_neuron, num_steps)

# for storing the errors corresponding to each neuron number
mlp_val_errors = []
mlp_train_errors = []

# Create a figure with subplots
```

```
fig, axs = plt.subplots(nrows=num_steps, ncols=Num_Folds, figsize=(24, 12))

#KFold
for fold_num, (train_index, val_index) in enumerate(kf.split(X_train_scaled)):
    X_train, X_val = X_train_scaled[train_index], X_train_scaled[val_index]
    y_train, y_val = y_trainNVal[train_index], y_trainNVal[val_index]

    # MLP
    for i, neuron_count in enumerate(num_neurons):
        # compute the size of the hidden layer
        hidden_layer_sizes = int(neuron_count)

        # create MLP Regressor model
        model = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,␣
 ↪activation='relu', solver='sgd', max_iter=2500, random_state=seed)

        # fit the model on trainig data
        model.fit(X_train, y_train)

        # make predictions on the training data
        y_train_pred = model.predict(X_train)

        # make predictions on the validation data
        y_val_pred = model.predict(X_val)

        # calculate MSE for this fold and this number of neurons on the train␣
 ↪and validation data and append it to the val_errors and train_errors
        mse_train = mean_squared_error(y_train, y_train_pred)
        mse_val = mean_squared_error(y_val, y_val_pred)
        mlp_train_errors.append(mse_train)
        mlp_val_errors.append(mse_val)


        # if this has the min error, calculate for test set
        if mse_val == min(mlp_val_errors):
            best_fold = fold_num
            best_neurons = neuron_count
            y_test_pred = model.predict(X_test_scaled)

        # Add the figure to the plot
        axs[i, fold_num].scatter(y_val, y_val_pred, color=colours[fold_num])
        axs[i, fold_num].set_xlabel('True Wavelength')
        axs[i, fold_num].set_ylabel('Predicted Wavelength')
        axs[i, fold_num].set_title(f'Fold: {fold_num}, neurons: {neuron_count},␣
 ↪loss:{round(mse_val, 4)}')

plt.suptitle('Predicted Wavelength vs True Wavelength')
```

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

# compute MSE for the training, validation and test sets
mean_mse_train = np.mean(mlp_train_errors)
mean_mse_val = np.mean(mlp_val_errors)
test_mse = mean_squared_error(y_test, y_test_pred)
print('Mean MSE in training set:', mean_mse_train)
print('Mean MSE in validation set:', mean_mse_val)
print('MSE for test data:', test_mse)

plt.scatter(y_test, y_test_pred, color='purple')
plt.xlabel('True wavelength')
plt.ylabel('Predicted wavelength')
plt.title(f'Predicted Wavelength vs True Wavelength for test set (Training from␣
 ↪fold {fold_num} with {best_neurons} neurons)')
plt.show()
```



```
Mean MSE in training set: 1870.454115672679
Mean MSE in validation set: 1887.850186707077
MSE for test data: 22.806603779074738
```


Predicted Wavelength vs True Wavelength for test set (Training from fold 4 with 100.0 neurons)
Predicted wavelength
True wavelength
500
550
600
650
700
750