CERI - 8104 – HW #2

Machine Learning Seminar - Multivariate Regression

Reading: "Python Machine Learning", Raschka, Chapter 10

Submit a short report with figures that illustrate your results! Explain your observations and respond to all questions below!

For this assignment, you will use different machine learning algorithms to predict **continuous** target variable based on several input feature vectors. The problem-set is related to basic multivariate regression. However, instead of doing multivariate stats, we will use more complex machine learning algorithms with gradient descent and back propagation to adjust weights and biases so that a cost function is minimized. The cost function that needs to be minimized can be written as:

$$\text{Cost function: } J(w) = \frac{1}{2}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

which is the same cost function used for simple neural nets such as Adaptive Linear Neurons (ADALINE). The superscript, *i*, denotes the different data samples or instances.

Part I: Simple linear and non-linear multivariate regression with MLPs

The first component of the homework is to explore whether non-linear regression problems can be solved with artificial neural networks without any knowledge of the specific mapping function. You will construct a noisy dataset using a quadratic function and then fit the data with an MLP regression algorithm. We will start with the simple equation:

$$y = w_1 x_1 + w_2 {x_2}^2$$

Note that in the following all free parameters are highlighted in blue. You should introduce these parameters at the beginning of your code.

1. Create two random, independent feature vectors using np.random. You could us ath. like:

- `x1 = np.random.uniform( -2, 5, m)`
- `x2 = np.linspace( -1, 2, m) + np.random.randn( m)*stdDev +mu`

   Here, m is the total number of samples, and mu and stdDev are the mean and standard deviation of a random Gaussian distribution.

2. Create the feature matrix with x1, x2 and the target labels, y using the above quadratic equation. You can start with the following parameters:

$$w_1 = 5.1$$
$$w_2 = 1.5$$

3. Add a random noise term to y:
   $y = w_1 x_1 + w_2 {x_2}^2 + err$, where err is a random variable drawn from a normal distribution with zero-mean and standard deviation $\sigma = 1.2$.

4. Use the scikit machine learning toolkit to fit the complete dataset with a MLP-regressor. The syntax will be sth like:

```
from sklearn.neural_network import MLPRegressor
MLP = MLPRegressor( activation = 'logistic')
MLP.fit( X, y) # you may have to transpose X so that is is comprised of column vectors of input features
a_y_hat = MLP.predict( X)
```

   Run your entire code for a synthetic dataset with *m*=500 data instances. Evaluate the performance of the MLP using standard metrics (*R2, MSE* and *cost function*).
5. You can adjust your MLP architecture by using the optional keywork argument: hidden_layers_sizes. Adjust your neural network architecture to:
   a. 100 units and 1 hidden layer

   b. 30 units and 4 hidden layer

c. 10 units and 10 hidden layer

Describe your observation based on MSE and overall change in model performance!

6. Increase your error term σ to 1.5 and 2. What happens to your predictions?

7. Increase the non-linearity parameter w_2 to 3. How does that change the performance of your MLP regressor?

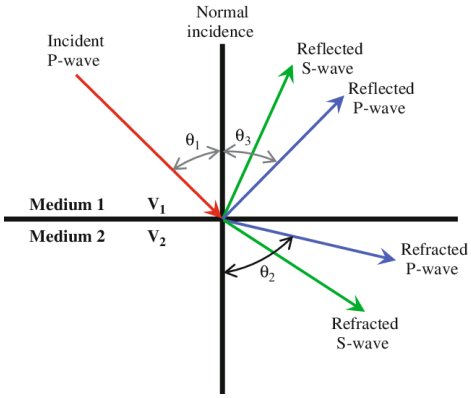
8. Try to tune your hyperparameters for the `MLPRegressor` to maximize the performance on the training set! List of hyper parameters include:
   hidden_layer_sizes=100, *activation='relu'*, *,*
   alpha=0.0001 (regularization term),
   learning_rate_init=0.001, (learning rate)

9. Explore the prediction performance for a testing data set outside of the range of your training data set.

10. Describe your observations! How easy or complicated is it to fit a multivariate, non-linear problem? What could be the potential underlying issues?

Part II: Well logs and seismic reflectivity estimates

The primary task will be to predict the seismic reflectivity (i.e. the reflection coefficient) based on the impedance contrast across rock interfaces.



Reflection, refraction and phase conversion of an incident P-wave

You will use several well-log measurements including seismic P and S wave velocities as well as rock density to determine reflectivity across layer boundaries. Seismic reflectivity describes the partitioning of seismic wave energy at an interface, typically a boundary between two different layers of rock. The reflection coefficient or reflectivity is the proportion of seismic wave amplitude reflected from an interface relative to the amplitude of the incoming wave. If 10% of the amplitude is returned, then the reflection coefficient is 0.10.

The data set for this exercise consists of 500 V_p, V_s, *density* and *reflectivity* measurements at different depths within a wellbore. Your task will be to predict the reflection coefficient for any new data set without knowing the incidence angle of the incoming wave and without any knowledge of the specific functional form that maps V_p, V_s and ϱ to reflectivity.

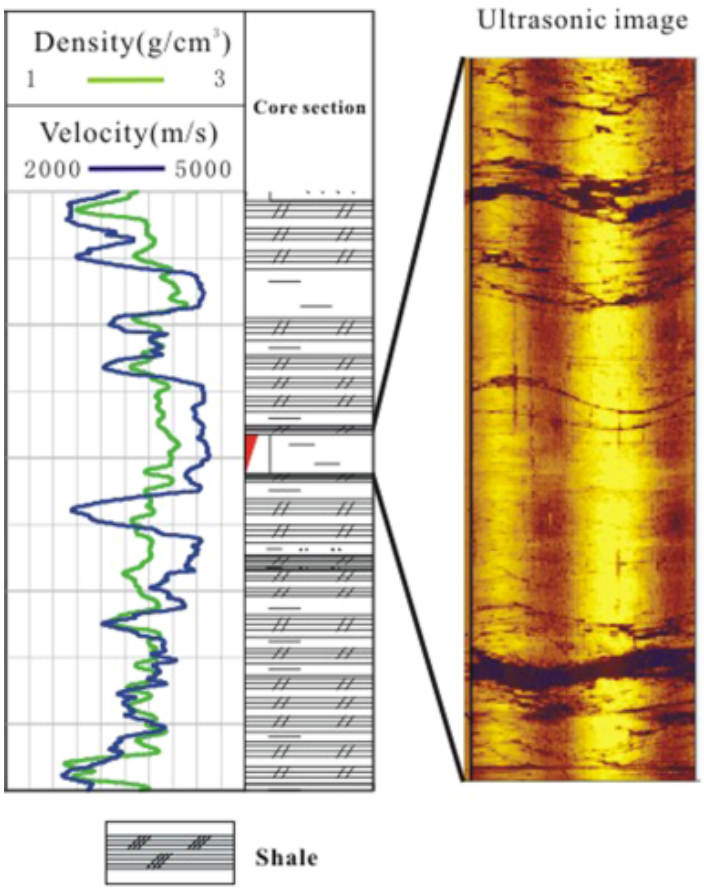


Figure 1: Drill core example and corresponding velocity and density measurements as a function of depth

A – Data Preparation and Screening

30 points

The first part of the assignment focuses on becoming more familiar with the data set and performing various data processing steps. Preparing the training data is a key component of using machine learning algorithms since every result will be as good (or bad) as the underlying data. More plainly said: junk in = junk out. Poor quality measurements will be reflected in poor results during machine learning algorithm training.

1. Load the data file (well_log.npy) using: np.load
2. How many rows and columns does the data matrix consist of? Print the length and range of each column in the data matrix to screen!

You may have already guessed that the last column contains the class labels, which here are seismic reflectivity. The first column is depth which will only be used for plotting purposes and the following columns are 'Vp' (P-wave velocity), 'Vs' (S-wave velocity) and 'rho' (rock density). You will use these three columns to predict the class labels in any new data set.

Data Visualization

3. Plot the continuous class labels and each attribute as a function of depth. You can use the subplot function to juxtapose all of the different panels.

Data screening with Pandas:

4. Create a Pandas dataframe object that contains all data features and class labels (not depth). Creating a dataframe from a numpy.array can be done in one line of code. You can do a web-search to see the exact syntax. Don't forget to add column labels.
5. Create a scatter plot and correlation matrix for all attributes. You can use:
   pandas.plotting.scatter_matrix([DataFrame])
   df.corr() where df is the dataframe object from above.
6. To plot the correlation matrix you can either use pcolor, imshow or the seaborn module which can be installed using the anaconda package manager:
   `>>conda install seaborn`
   see: seaborn.pydata.org/

Preprocessing steps and data splits.

You will now prepare the training and testing data files that will be used later in a separate python script to train the ML algorithm and to make predictions.

7. Create two separate arrays that contain the class labels (reflectivity) and the data features (V_p, V_s, and rho).
8. Use train_test_split() from scikit.model_selection To randomly separate training and testing data. Testing data should contain at least 10 to 30 percent of the original sample instances. The test data will later be used for a blind test after you fully-trained the machine learning algorithm. Ultimately, you would like to have sufficient data in your testing set to determine if your ML algorithm outperforms a simple base model.
9. Save the training and testing data as numpy binaries using np.save (e.g. train.npy and test.npy). You can save the data sets in two files or create two additional files that contain the target classes.

B - Training, Prediction and Performance Evaluation

50 points

1. Load the data created in part I.
2. Chose your regression algorithm, train the model and predict the target classes.
   You can use Gradient Descent regression (sklearn.linear_model.SGDRegressor) or a Neural Net (sklearn.neural_network.MLPRegressor).

Performance Evaluation:

1. We will use several methods to evaluate the model performance
   a. Plot the residuals (model - observation) for the test data and document the 5th and 95th percentile
   b. Compute the mean-squared-error:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2,$$

use sklearn.metrics.mean_squared_error()

c. Compute the coefficient of determination, R2:

$$R2 = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 \,/\, \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \mu_y)^2,$$

easy enough to write a function yourself or use sklearn.metrics.r2_score()

2. Describe the results from the performance metrics! Compare them to a simple base model for which you randomly assign class labels based on the class distributions in the training dataset.
3. Let's return to the training of the model and evaluate how well the algorithm is trained based on a cross-validation analysis. You will analyze the (i) overall consistency of the results, (ii) influence of sample size, and (iii) performance of training vs. validation data. Scikit-Learn has some nice functions that allows us to do all of these in two lines of code!
    a. Create a cross-validation object with a 10-fold data split and a test size of 10%:
    cv = ShuffleSplit(n_splits=10, test_size=0.1, random_state=12345)
    b. Compute the learning curve for training and validation data as a function of number of samples in the training data. You will use 10 equally spaced subsample sizes of the training data. You can evaluate the results using the R2 score:
    learning_curve(estimator=estimator,X=X_train,y=a_y_train,scoring='r2',train_sizes=np.linspace(0.1, 1.0, 10),cv=cv,n_jobs=-1)
    c. Plot the mean R2 score for training and validation data as a function of sample size! How well does your algorithm perform? Are you satisfied with the training outcome?
4. A nice way to quickly evaluate your model performance could be by plotting the observed and modeled class labels in the training data as well as observed and modeled class labels in the test data together with the base model prediction. You can add titles to each subplot that show the R2 values for training and testing. A third subplot showing the residuals (y_true – y_hat) may also be useful to spot potential outliers.

The last step is optimizing your model performance by minimizing the prediction errors. Here are a few ways to improve model performance:

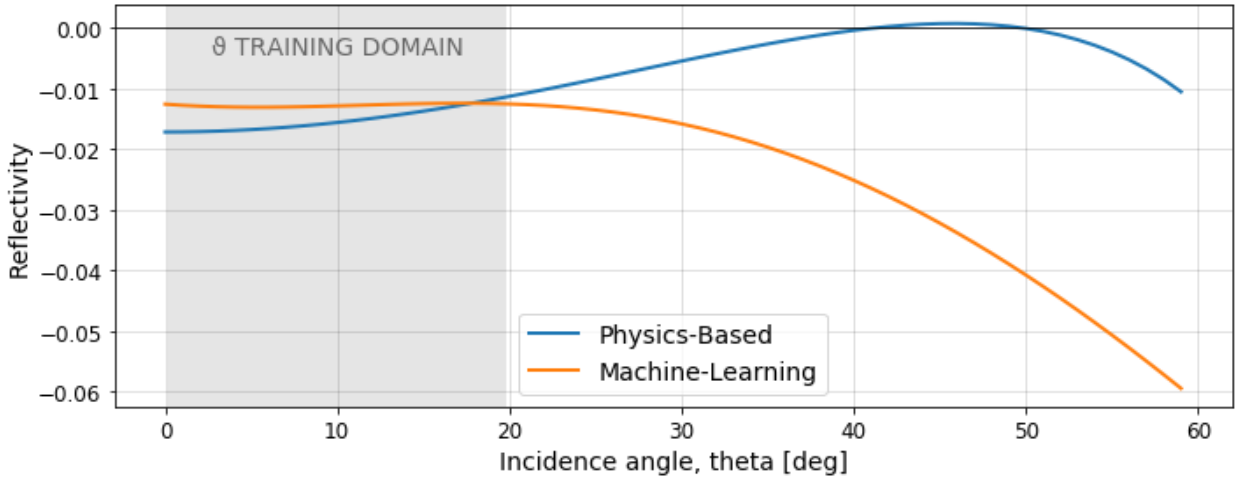
5. Standardize the feature vectors in your input data.

Since the reflection coefficient depends on impedance contrast between two layers a nice little trick could be to include the velocities and densities of every bottom layer:

6. Add three new data columns that corresponds to the elastic properties of the ensuing layer. Note that this will reduce your total number of samples by 1!
7. Experiment with different regression algorithms such linear regression, SGD regression and MLP regression.
8. What is the R2 score of your best performing model? What was most consequential for improving your model performance?

Is the model generalized enough for predictions beyond the data range represented by the available samples?

The final model should do an excellent job in predicting both training and testing data. However, if we were to expand the prediction range beyond the training data we would get something like this picture:



The above figure shows changes in amplitude based on reflectivity at different incidence angle. The blue line is a physics-based model and the orange curve is the prediction based on a trained machine learning algorithm.

9. Based on this figure, how generalizable are the machine learning results in this example? Would you say that the ML algorithm successfully learned the underlying physics? Explain your response!