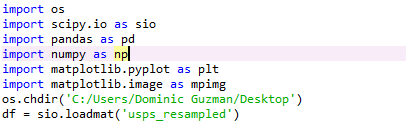
Dominic Guzman

4/25/2018

Dr. Mukesh Kumar

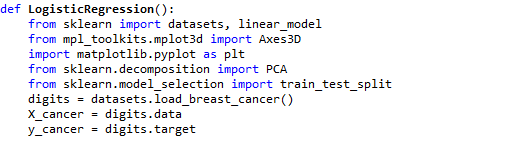
Final Report

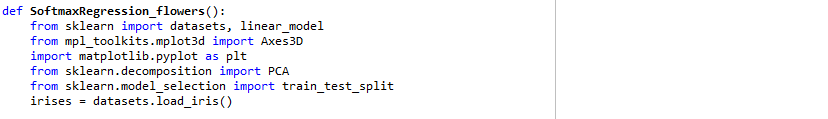
1. Processing the Data and Visualizing the Data

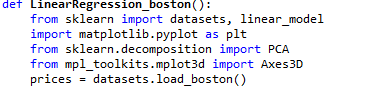


Before anything was implemented, the needed libraries had to be imported along with their packages so that the USPS handwritten data can be manipulated properly. The import os allowed the program to change directories. Scipy.io allowed the program to import the matlab data so that it may be usable for the algorithm, and Scipy allowed the necessary methods to be used for any linear algebra processes. Pandas allowed data frames to be created based on the keys received from each column on the data sets. Pandas also allowed the USPS to be organized into a much more manageable data frame. Numpy introduces mathematical methods to turn any necessary arrays into matrices so that it may be manipulated into the needed 16 x 16 matrix images. The above import statements are used for both the SVD project and the CNN project.

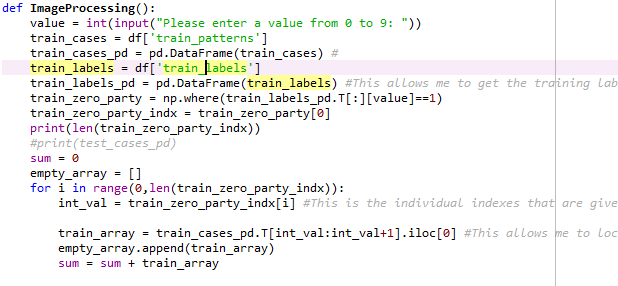
The only project in which these import statements weren’t used was for the second project. To test out linear and softmax, I imported built-in datasets from sklearn which were the cancer, iris, and boston datasets, respectively. For the breast cancer, the size of the dataset is 569 x 30. For the iris, the size of the dataset is 150 x 4. For the boston prices, the size of the dataset is 506 x 13. The only similarities





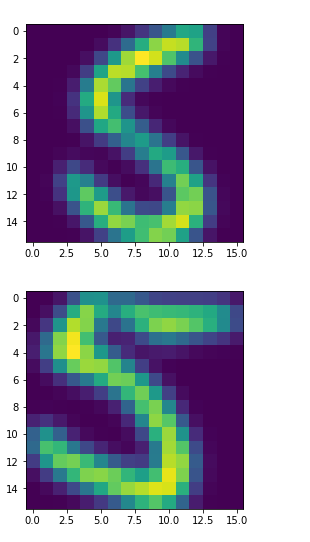
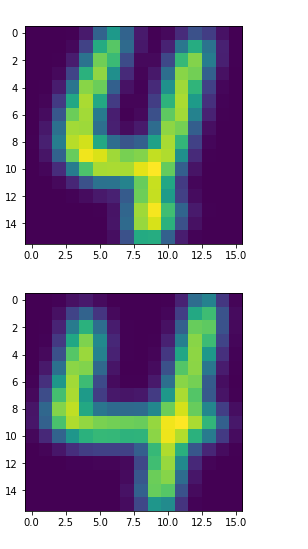
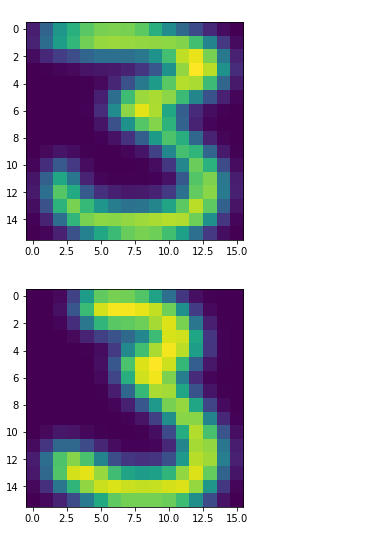


USPS visualization

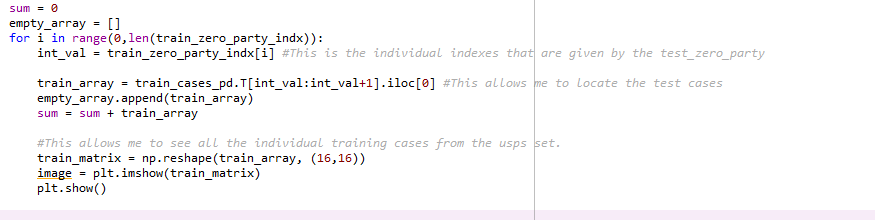


By utilizing the df, pd, and numpy libraries, the necessary data was able to be condensed into the necessary images by appending their columns into the empty\_array. Within this empty array, it contains all of the 255x1 vectors that represented the “value” images if they were reshaped to 16x16 matrix images. Df was used to find the data frames for both the ‘train\_patterns’ and the ‘train\_labels’.

The train\_zero\_party variable looks for the “value” within the label data set by looking for the columns that contain a positive 1. Once found, the columns are then saved into train\_zero\_party\_indx. By initializing sum and train\_array, the loops have variables to save the given integer index values into. Finally the train\_array value is appended into the empty\_array.

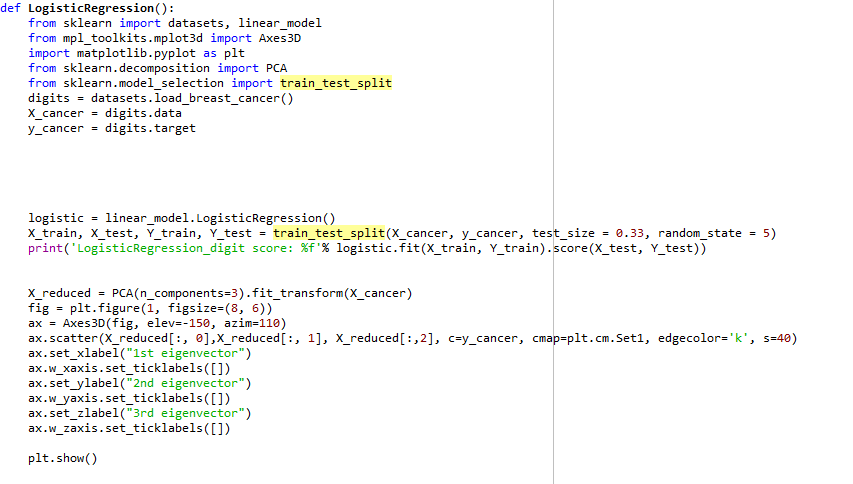


To be able to visualize the data, the sum was initialized and continuously added each train\_array to the sum. Once doing so, the array is reshaped into 16 x 16 matrix. The reshapes were done for each train\_zero\_party\_indx value. The above examples are printed out if the given “value” were either 3, 4, or 5. Once the reshapes were finished, each “train\_matrix” is used as an argument for the plt.imshow() method. Lastly the plt.show() method is called to visualize the following data.

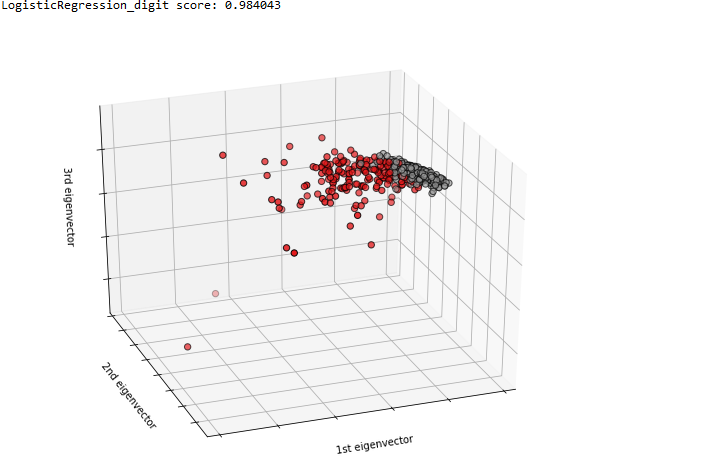


Logistic, Softmax, and Linear Regression

After the SVD classification, the next project was a combination of logistic, softmax, and linear regression models used on three different datasets. The first dataset used was logistic regression. With this regression method, I imported the breast dataset

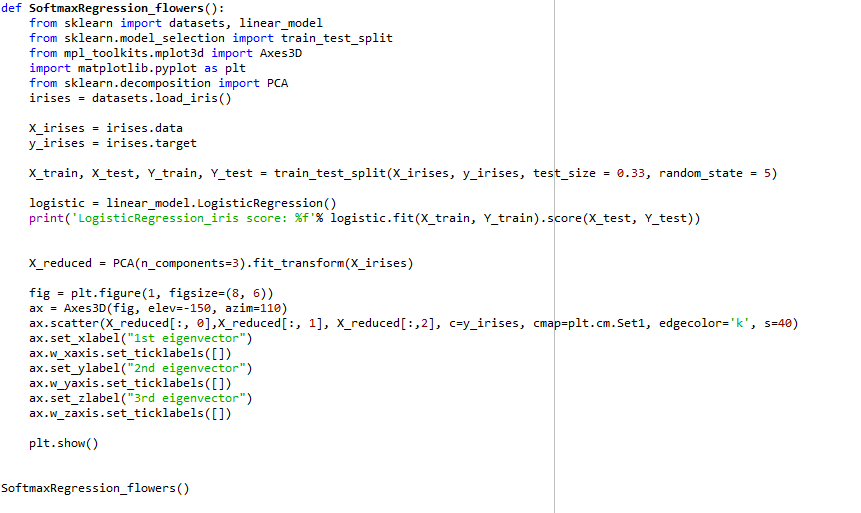
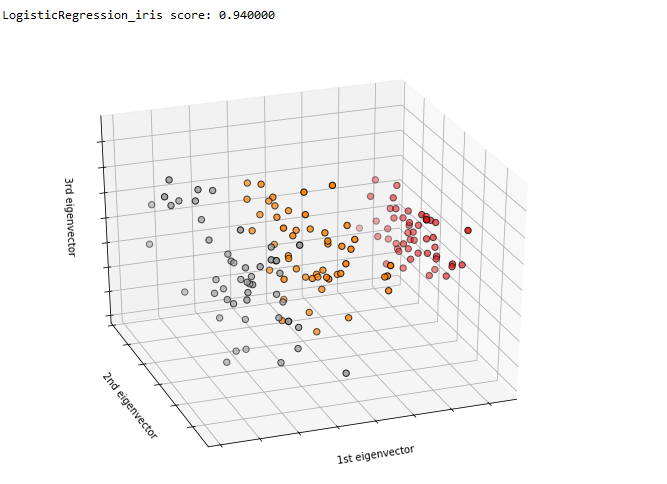


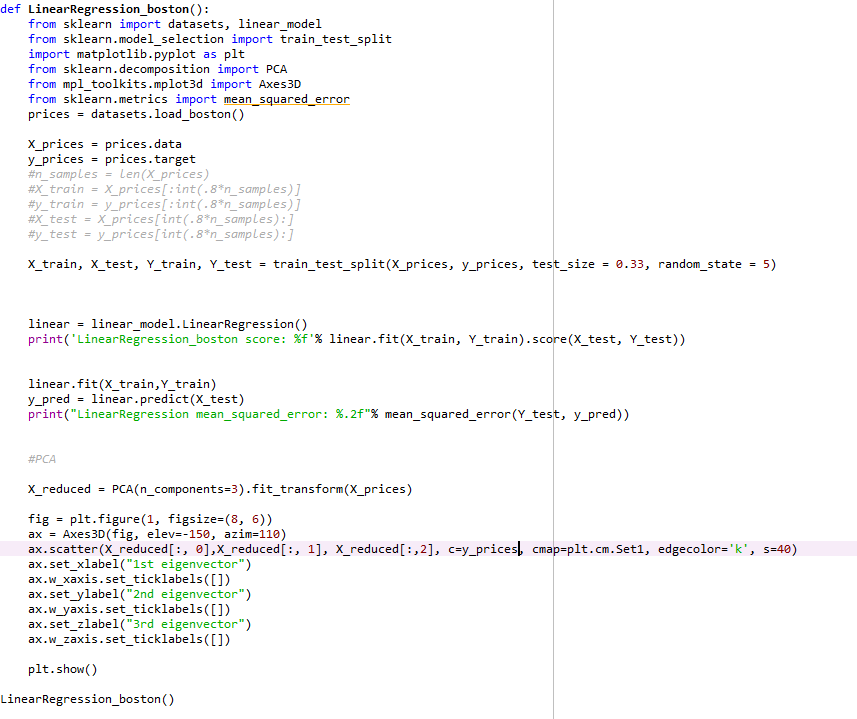
Before running any regression models on the data, I had to use the “train\_test\_split” function to properly split my data up. For the training data, I used 66% of the initial dataset, and for the test set, I used 33% of the initial dataset. Once I’ve decided on what sets to use for my training and testing, I took the respective values and initiated a “logistic.fit” function on both the training and testing sets. The logistic.fit method takes 2 arguments that are sets. The first argument is the X\_train and the second argument is the y\_train. The logistic.fit takes the X\_train and fits a model basing its classifications based off the y\_train. Once trained, the logistic.fit returns the mean accuracy from the logistic.fit(argument,argument).score(argument,argument). For the score function, it takes two more arguments that are sets. In my method, I use the X\_test and the y\_test for my score function.

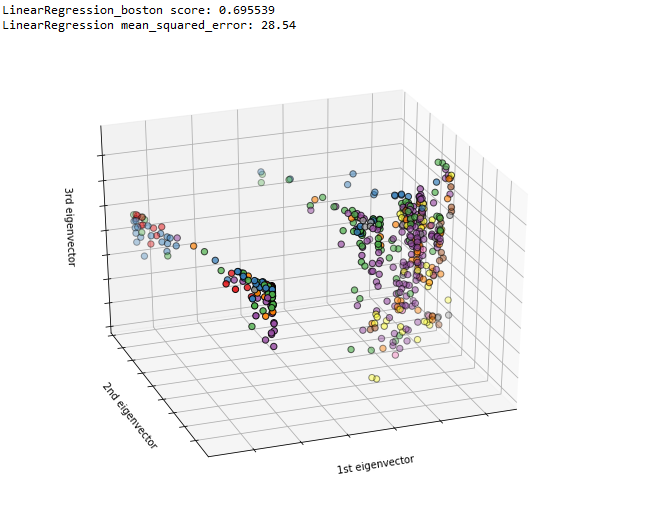


Since the dataset has a dimensionality of 64, there needed to be a way to visualize the clusters. After the score, the method condensed the X\_digits to three dimensions with a specific method. The PCA method accepts (n\_components = x) where “x” is an integer that condenses the data to “x” dimensions. In this case, the method made x = 3 which allowed a visual representation of the clusters. Once done, it can be seen that the 98% accuracy can be attributed to how well the clusters are divided from each other. The reds are clearly separated from the grays.

The following code is then used for both the iris dataset and the boston dataset with slight adjustments to correspond with each data set.



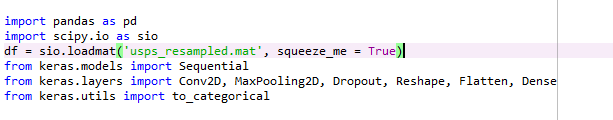
￼



￼

One of the major differences with the linear code and the rest of the other regression methods is the addition of the 3 lines of code following the “.fit” and “.score” methods. Since the method is dealing with continuous values, both the r2 score and the mean\_squared\_error had to be utilized for analysis. The mean\_squared\_error is the summation of the differences between the “y\_test” and the “y\_pred” values, while the r2 is the score of how well the linear function fits within the “X\_test” and the “y\_test”. For my initial measurement with the Boston dataset, the linear regression r2 score, noted by the “boston score” was negative. A negative r2 means that the linear regression model was fitted poorly. The mean\_squared\_error may be an explanation as to why the boston r2 score is negative. The train and test datasets are with a 66 to 33 split. By looking at the first figure, it can be stated that the plots aren’t as well scattered as compared to the logistic regression models. By having a badly scattered plot, the linear regression model is having a hard time fitting a model into the data set.

Convolutional Neural Networks



To start the convolutional neural networks, the necessary import statements be written. To start out the data preprocessing, both pandas and scipy.io must be imported to acquire the “usps\_resampled.mat”. It is similar to how the data was read in the first project. The most important import statements are the keras statements. Keras allows the method to take advantage of necessary methods when trying to implement a CNN. The first keras comes from the “keras.models”. It is the import statement for Sequential which is a linear stack of layers that the method continuously adds layers on to through the “.add” method. These layers are imported from the “keras.layers” method.

Conv2D - This layer adds a convolution kernel that is convolved with the layer input to produce a tensor of outputs.

MaxPooling2D - It is the method in which the CNN shrinks the image stack.

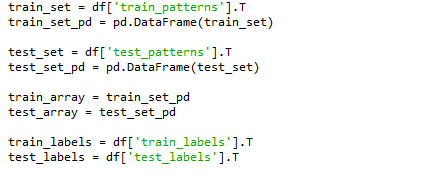
Dropout - The Dropout deactivates an arbitrary amount of nodes before each Conv2D activation.

Reshape - The Reshape takes an array input and reshapes it into a matrix with given value dimensions.

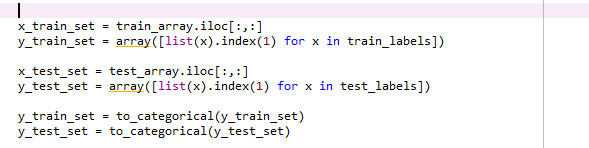
Flatten - It is the process that takes the 2D array and turns it in a 1D vector before putting it through the final part of the classification.

Dense - It is the part of the sequential model that classifies the flattened dataset that takes an integer input and an activation function.

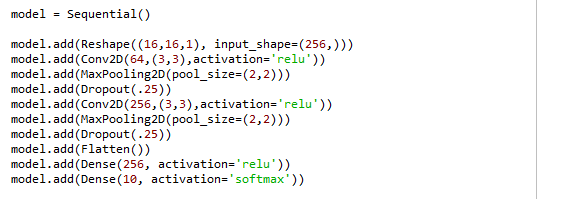
The next import statement was from the “keras.utils”. From that library, the method imported the “to\_categorical”. By providing a class vector as an input, the “to\_categorical” method allows the vector to become a binary class matrix.



Similar to the SVD data processing, the CNN method reads the data in the same fashion. It uses pandas to create a dataframe while also transposing the necessary data so that it may be spliced properly.



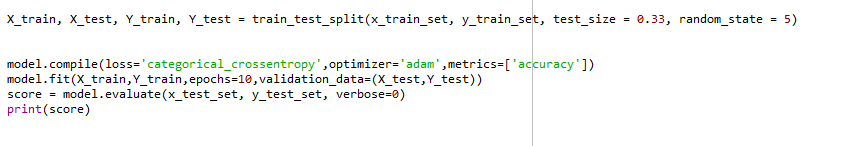
After loading in the data, the “x\_train\_set” and the “x\_test\_set” are loaded in from the “train\_array”. The “y\_train\_set” and the “y\_test\_set” are loaded into an array from the “test\_labels”.



Once all the necessary data is loaded in, the architecture is then set up for the classification. The first line loads in the sequential model so that method can start loading in the necessary layers for classification. The first layer is the reshape layer which takes an input vector of the size 256. The reason as to why the vector is size 256 is because the “usps\_resampled.mat” is loaded in as a collection of 256 length vectors that if reshaped into 16,16 matrices can form a 16x16 image matrix.

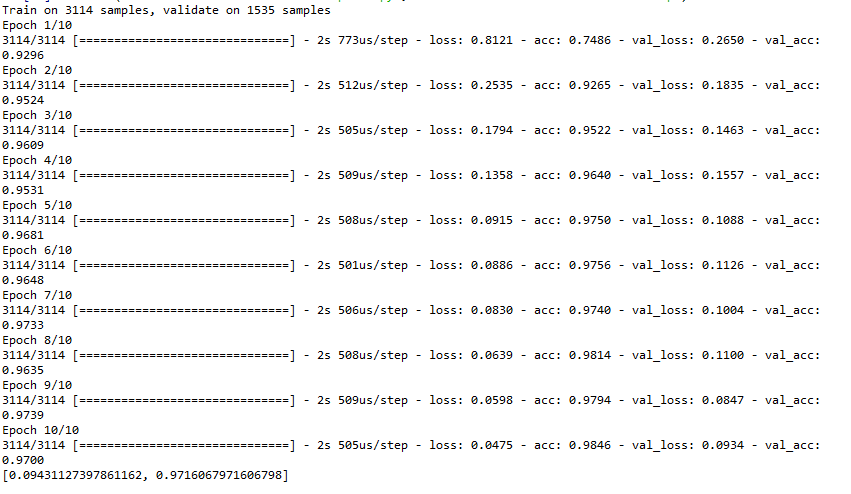
Once reshaped, the model adds the Conv2D layer to start out the convolutional process. Within the first convolutional layer, it has a 64 node architecture and has a stride of (3,3). This stride determines the window that filters through the input image. The activation function is set to the “relu” which is the rectified linear unit function. The next layer is the MaxPooling2D layer which starts shrinking the given input image with a (2,2) window size. Once pooled, the maximum value is taken from each pool.

Before moving onto the next Conv2D step, the next layer added is the Dropout layer which turns off, in this case, 25% of the nodes in the CNN. The purpose of the Dropout layer is to prevent overfitting and can create a much better generalization by shutting off 25% of the nodes. After the first run through the CNN, the process is repeated until the last dense layer where the amount of nodes is shortened to 10 nodes to represent the 0-9 digit classification. The activation function is changed to a softmax to represent the multi label classification.



The next lines is the final code segment that compiles and fits the data set through the neural network. The train\_test\_split takes a 66% to 33% division between the x\_train For the “model.compile”, the “loss” function uses the categorical\_crossentropy. The next argument for

the “model.compile” is the optimizer being used in which the loss function is implemented within. For the optimizer, the method uses the adam optimizer and measures the “accuracy” of the model. Lastly, once everything is processed and the necessary layers are added, the model then fits the training and the testing sets. They are tested over 10 iterations through the data with the “validation\_data” set to the given 33% from the “train\_test\_split”. Once the model is fit with the necessary data, model is tested against the “x\_test\_set” and the “y\_test\_set”, and the output score is the accuracy given.



These are the outputs after each epoch tested through the data set while the final value at the bottom is the given score from “model.evaluate” method. Within each iteration there is the “acc” which is the accuracy of the model as it is trained using the training data. For each iteration, both the “acc” and the “val\_acc” grow towards the > 90% percentages until both “acc” and “val\_acc” are 98% and 97% respectively. The score gives a 97% accuracy score, while the loss is 0.0943.