

**Module: EE5907 Pattern Recognition**

**Assignment: CA2**

**Student: ROGATIYA Mohmad Aspak Arif**

**Matriculation ID: A0179741U**

**Email:** [**e0269760@u.nus.edu**](mailto:e0269760@u.nus.edu)

Contents

[Introduction 3](#_Toc39051342)

[**a)** **The MNIST Database** 3](#_Toc39051343)

[**Load MNIST datasets** 3](#_Toc39051344)

[**b)** **Calculating Eigenvalues and Eigenvectors for Features Extraction** 4](#_Toc39051345)

[**(1)** **First Row: Display selected images from original training set.** 5](#_Toc39051346)

[**(2)** **Second Row: Display reconstructed images from the test set.** 5](#_Toc39051347)

[**c)** **Using Eigen-digits to Classify each digit image** 6](#_Toc39051348)

[**(1)** **Linear Regression** 7](#_Toc39051349)

[**(2)** **Polynomial Regression** 10](#_Toc39051350)

[**d)** **Possible Techniques to Improve the Test Accuracy** 11](#_Toc39051351)

# Introduction

This report is written as part of EE5907 assignment CA2. The assignment is to implement linear and polynomial regression to classify images of handwritten digits 0 to 9. The digit images are obtained from popular MNIST database <http://yann.lecun.com/exdb/mnist/>. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimum time and efforts on collecting, preprocessing and formatting training and test data.

## **The MNIST Database**

Four files are available on MNIST database.

[train-images-idx3-ubyte.gz](http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz): training set images (9912422 bytes)  
[train-labels-idx1-ubyte.gz](http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz):  training set labels (28881 bytes)  
[t10k-images-idx3-ubyte.gz](http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz):   test set images (1648877 bytes)  
[t10k-labels-idx1-ubyte.gz](http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz):   test set labels (4542 bytes)

The training and test dataset contains 60000 and 10000 handwritten digit images respectively. Each image has 784 pixels (28x28). In other words, the feature data length for each test or training sampleis 784. Each pixel is an 8-bits value, i.e. 0 to 255; where 0 means white and 255 means black.

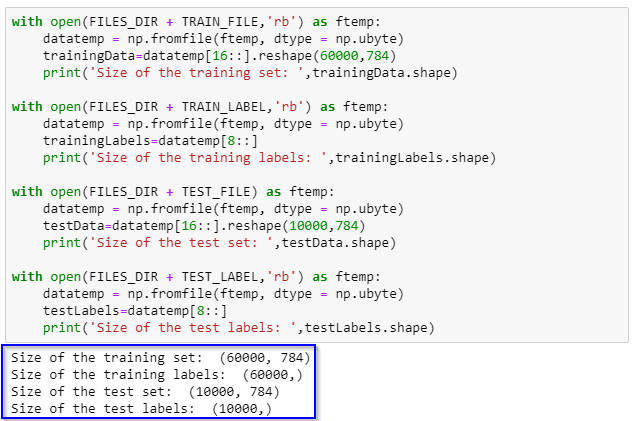
### **Load MNIST datasets**

The downloaded datasets are in compressed \*.gz format. I have uncompressed these files and put under MNIST\_Data folder.

Following constants are defined for the files and folder names.



The Python’s numpy module provides np.fromfile to read from \*ubyte file as shown below. The datasets “shapes” are highlighted.



## **Calculating Eigenvalues and Eigenvectors for Features Extraction**

The necessary steps to visualize the topmost 10 eigenvectors are as below.

**Step-1: Calculate the mean of training data.**

The formula to calculate the **mean** **Ψ** is as shown below.



Where, m = number of training samples, 𐌲i = training vector

**Implementation**

* The StandardScalermethod from the preprocessingclass of the sklearnmodule is used to standardize the data by removing the mean and scaling to unit variance.
* The mean method from numpy library is used to calculate the mean matrix. The mean matrix shape is 784x1 (same as training samples).



**Step-2: Normalize the matrix and calculate covariance matrix**

Normalized matrix can be calculated by subtracting mean from each training sample.

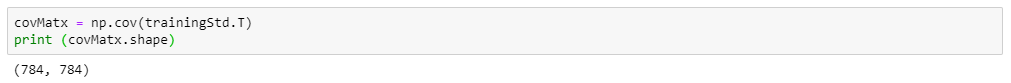


The StandardScalermethod from the preprocessingclass of the sklearnmodule is used to normalize the data by removing the mean and scaling to unit variance to generate trainingStd matrix.



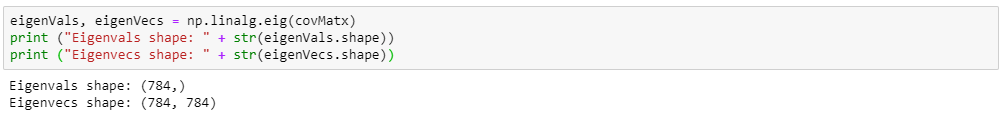


We can use numpy module’s “cov” method on the normalized trainingStd matrix to calculate the covariance matrix. The shape of covariance matrix is 784x784 (N2xN2).



**Step-3: Calculate eigenvalues and eigenvectors.**

I have used the linalg.eigfrom numpylibrary to calculate eigenvalues and eigenvectors.

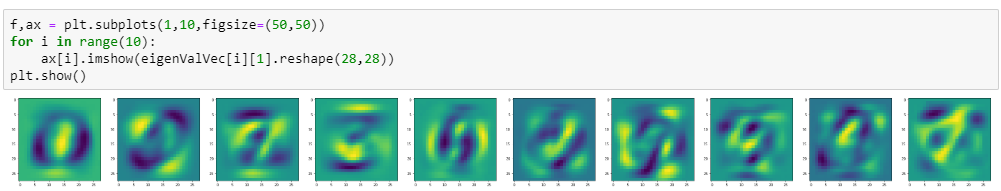


**Step-4: Visualize eigenvectors corresponding to topmost 10 eigenvalues.**

First, I have to sort the eigenvectors based on eigenvalues.



Then, I used “imshow” method from matplotlib library to display top 10 eigenvectors.

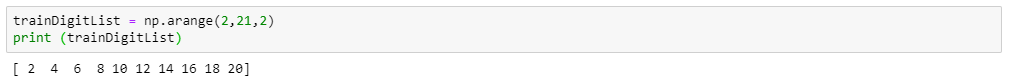


**Image Reconstruction:**

This section describes the method of displaying selected images from test and training data. It also describes the method to reconstruct the test images using the eigenvectors calculated in the previous section.

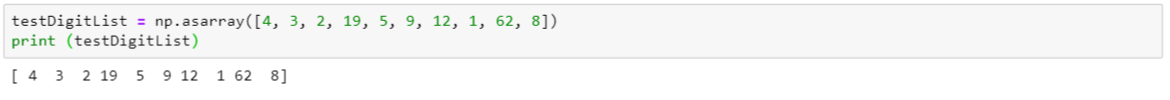
### **First Row: Display selected images from original training set.**

I have displayed following columns of the training data: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 in the first row of the plot. I have prepared a list of these indices.



### **Second Row: Display reconstructed images from the test set.**

I have displayed reconstructed images corresponding to the following columns of the test data: 4, 3, 2, 19, 5, 9, 12, 1, 62, 8 in second row.



**Step-1:** The λi of the test sample is calculated using the following formula**.**

****

**Step-2:** The reconstructed test digits was then calculated by the sum of the product between the λi and the eigenvectors using following formula.



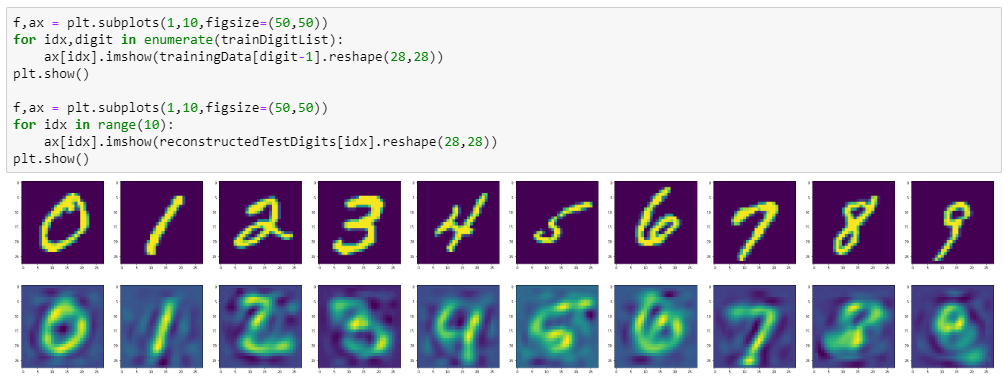
**Implementation:**

Two steps listed above are implemented as following.



**Display First and Second Row:**

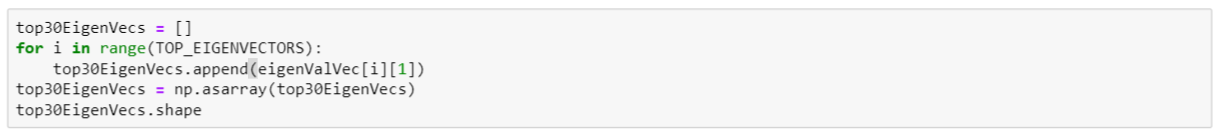
The selected training digits (first row) and reconstructed selected test digits are plotted using “imshow” method from matplotlib library.



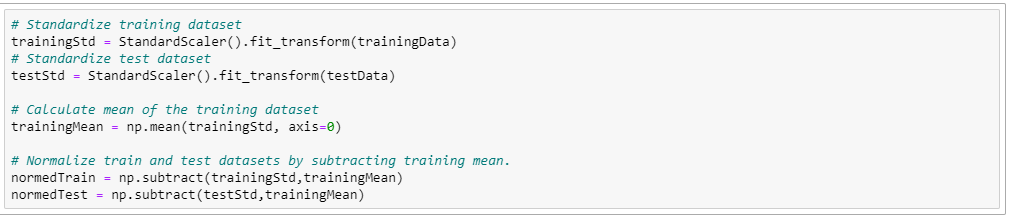
## **Using Eigen-digits to Classify each digit image**

**Project data to 30-dimensional space**

* Step-1: Prepare an array of top 30 eigenvectors. This will be used in next steps.



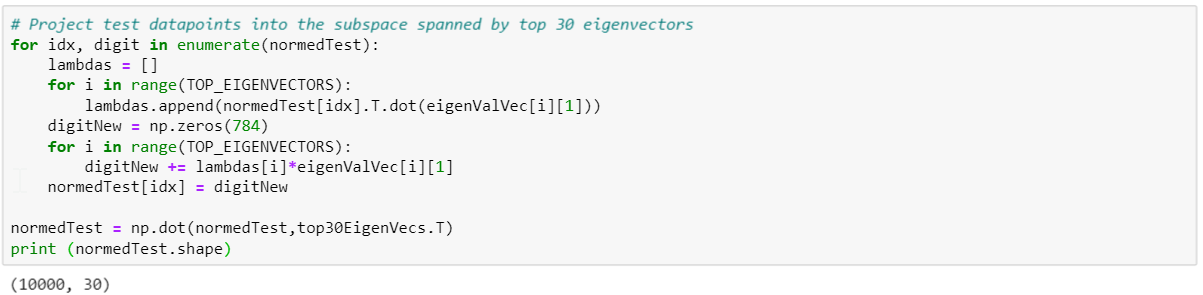
* Step-2: Normalize training and test datasets by subtracting training data mean from each sample.



* Step-3: Project each training datapoints into the subspace spanned by top 30 eigenvectors.

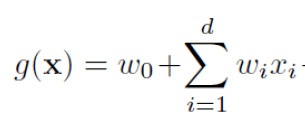


* Step-4: Project each test datapoints into the subspace spanned by top 30 eigenvectors.



### **Linear Regression**

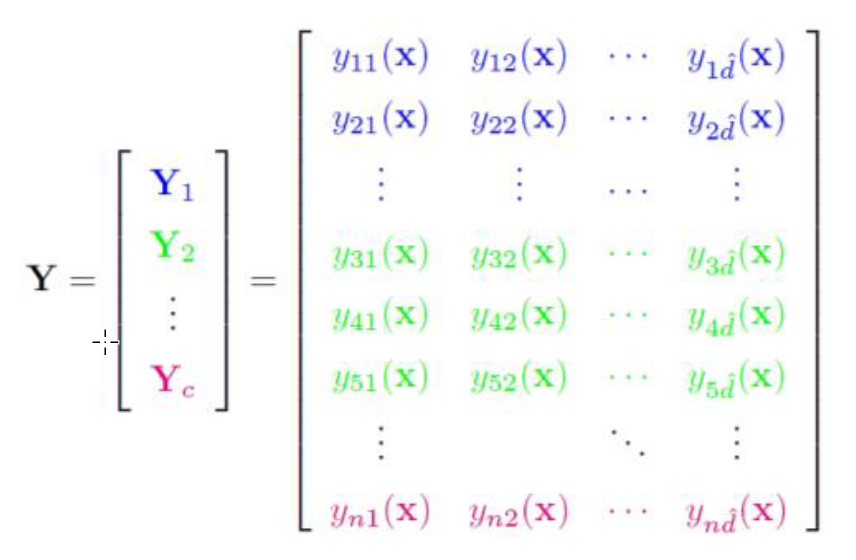
We will now apply the Linear Regression on training and test datasets and measure accuracy. Linear Regression classifier is implemented by following formula.



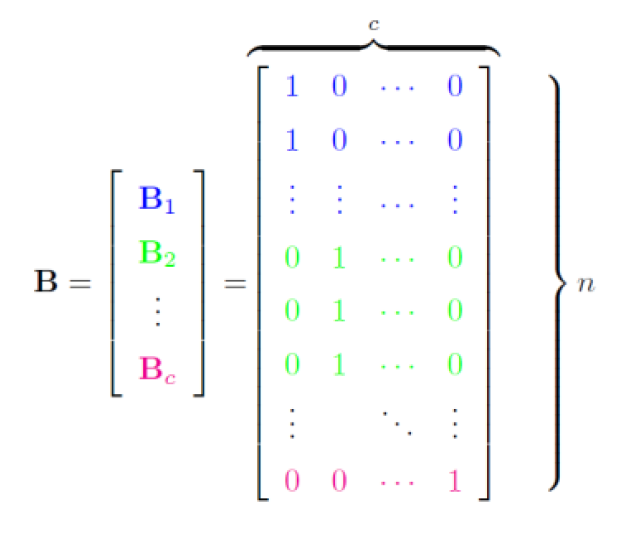
We will first generate a weight matrix A of sized (𝑑 + 1) × 𝑐, where d is feature dimension and c is number of classes. A matrix is calculated by following formula.



Where, Y matrix looks like below.



B is a one-hot encoded matrix as shown below.



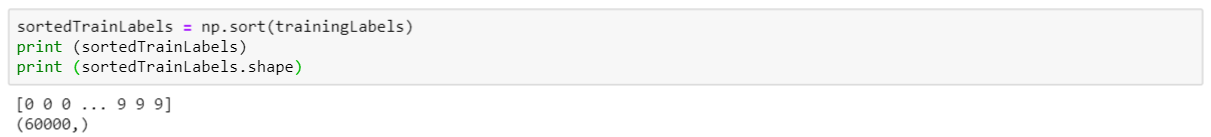
Linear Regression steps can be summarized as below.

**Step-1: Calculate Y matrix.**

1. We have to sort the training datasets in the order of training labels.



1. We will sort the training labels as well.

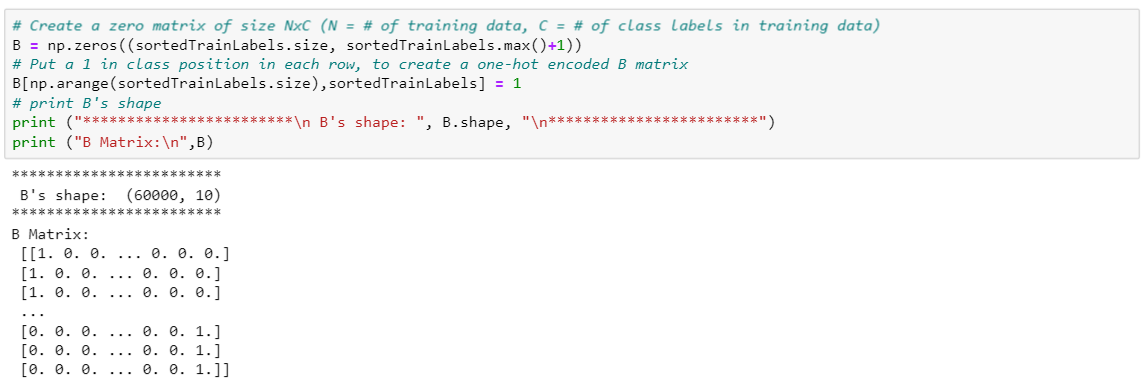


1. Append a bias column at the end of the sorted normalized training data to generate Y matrix.



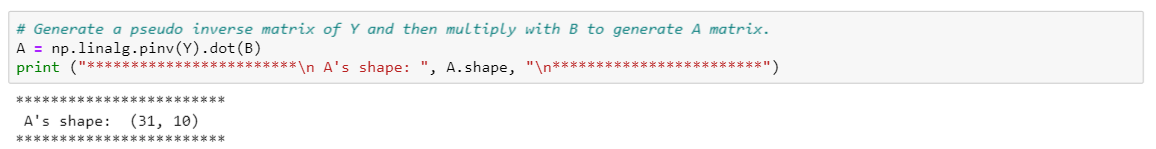
**Step-2: Generate B matrix.**

1. Generate a zero matrix of size NxC.
2. Put a 1 in each row’s class “label” position to get one-hot encoded B matrix.



**Step-3: Generate A matrix.**

1. Generate a pseudo inverse of Y matrix using “pinv” method from numpy library.
2. Do a dot multiplication with B to generate A matrix.

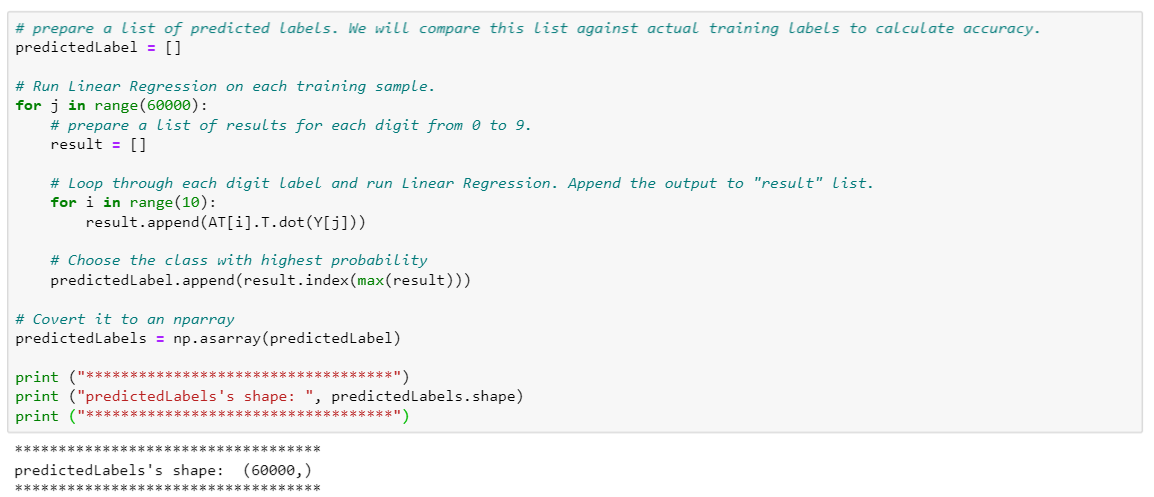


**Step-4: Transpose the A matrix.**

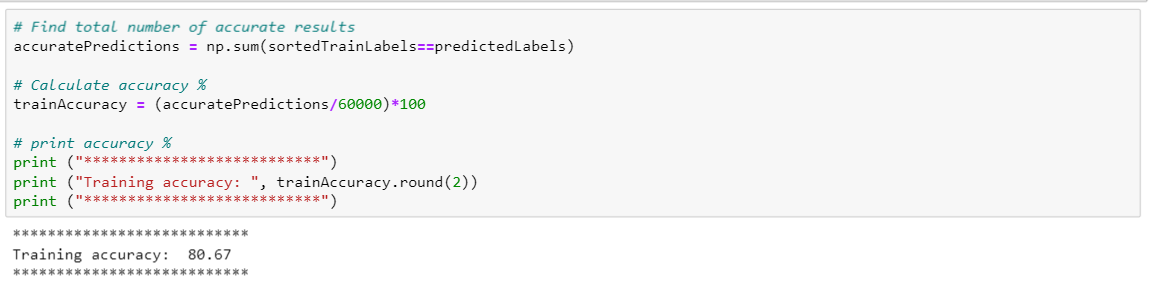
****

**Predict training labels using linear regression**

**Step-1:** Run Linear Regression classifier on each training sample for each label and select the class label with highest probability. Prepare a list of predicted labels. We will compare this list against actual training labels to calculate accuracy.

****

**Step-2:** Find total number of accurate results and then calculate accuracy percentage.

****

**Predict test labels using linear regression**

**Step-1:** Generate a Y matrix for test data.

* Similar to Y matrix described for training data.

**Step-2:** Run Linear Regression classifier on each test sample for each label and select the class label with highest probability. Prepare a list of predicted labels. We will compare this list against actual test labels to calculate accuracy.

* Similar to Linear Regression on training data.

**Step-3:** Find total number of accurate results and then calculate accuracy percentage.

* Similar to accuracy calculation for training data.

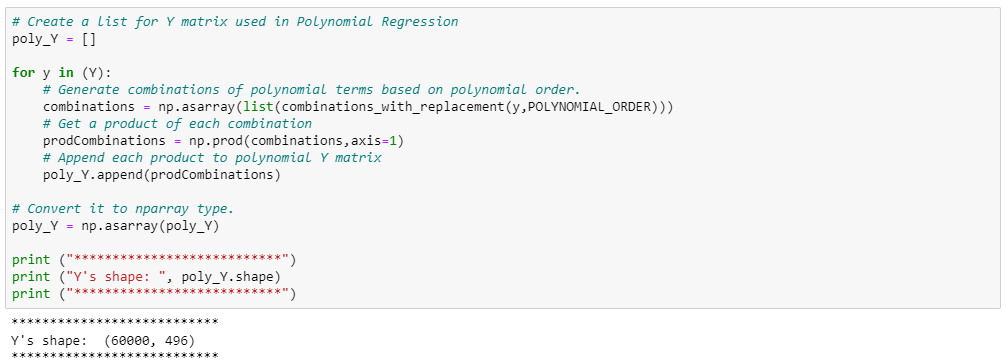
### **Polynomial Regression**

We will now apply the Polynomial Regression on training and test datasets and measure accuracy. The Polynomial Regression classifier is implemented by following formula.



**Step-1: Generate Y matrix.**

Each row of Y matrix has all the terms in the polynomial discriminant function. We can get all these terms (combinations) using combinations\_with\_replacement method in the itertools module.



**Step-2: Generate B matrix.**

Similar to B matrix described in Linear Regression.

**Step-3: Generate A matrix.**

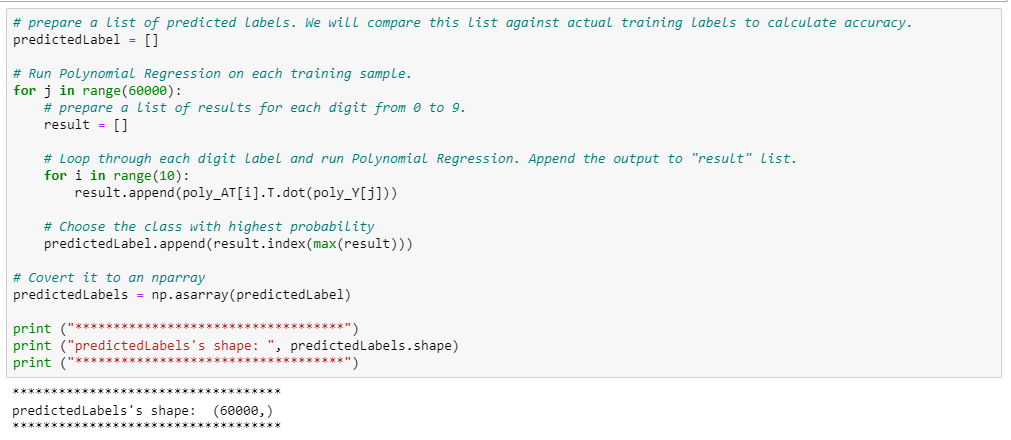
Similar to A matrix described in Linear Regression.

**Step-4: Transpose A matrix.**

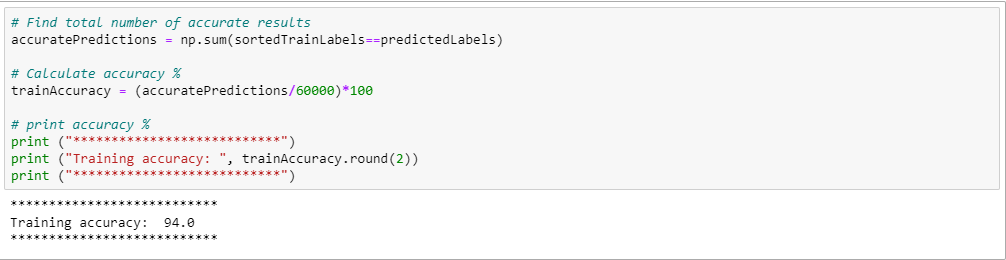
Similar to A matrix transpose described in Linear Regression.

**Predict training labels using Polynomial Regression**

**Step-1:** Run Polynomial Regression classifier on each training sample for each label and select the class label with highest probability. Prepare a list of predicted labels. We will compare this list against actual training labels to calculate accuracy.

****

**Step-2:** Find the total number of accurate results and then calculate accuracy percentage.

****

**Predict test labels using Polynomial Regression**

**Step-1:** Generate a Y matrix for test data.

* Similar to Y matrix we described for training data

**Step-2:** Run Polynomial Regression classifier on each test sample for each label and select the class label with highest probability. Prepare a list of predicted labels. We will compare this list against actual test labels to calculate accuracy.

* Similar to Polynomial Regression on training data.

**Step-3:** Find total number of accurate results and then calculate accuracy percentage.

* Similar to accuracy calculations on training data.

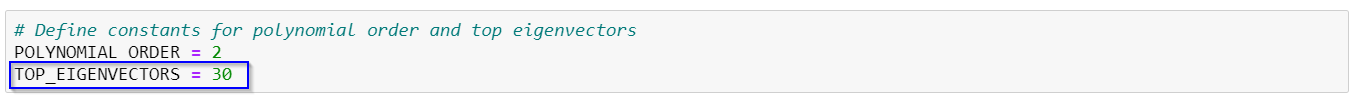
## **Possible Techniques to Improve the Test Accuracy**

In general, Polynomial regression has better accuracy and higher execution time compared to Linear Regression.

We can also increase the test accuracy using following techniques.

1. **Increase number of Eigenvectors**

We can simply modify TOP\_EIGENVECTORS constant to modify the number of top eigenvectors.



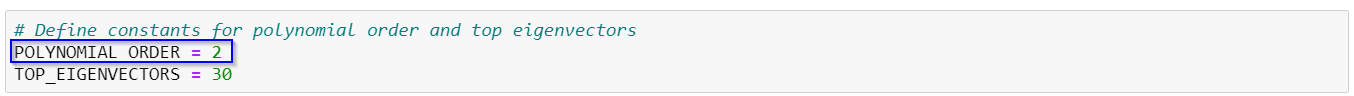
The accuracy improves as we use more than 30 eigenvectors. Following table shows the test accuracy with different eigenvectors. The execution time also increases drastically with increasing number of eigen vectors. Hence, we have to find a balance between accuracy and execution time.



1. **Increase Polynomial Order**

We can increase the test accuracy by increasing the Polynomial Order. The execution time also increases dramatically with increasing Polynomial Order. This is because the number of combinations of polynomial terms in the discriminant function increases exponentially with increasing polynomial order. Hence, we have to find a balance between test accuracy and execution time.

We can modify the polynomial order by simply modifying POLYNOMIAL\_ORDER constant.



Following table shows the test accuracy with different Polynomial orders.



1. **Including Bias term**

We can include or remove bias term by simply modifying **INCLUDE\_BIAS** constant.



It was observed that including Bias term improves the test accuracy slightly.

