Table of Contents

[Multicollinearity 1](#_Toc33595769)

[Controlling Multicollinearity 1](#_Toc33595770)

[Principal Components Analysis (PCA) 2](#_Toc33595771)

[Predictor Correlation 2](#_Toc33595772)

[Eigen Decomposition 3](#_Toc33595773)

[Eigenvectors 3](#_Toc33595774)

[Eigen Values 3](#_Toc33595775)

[Choosing Components 5](#_Toc33595776)

[Kaiser’s Stopping Rule 5](#_Toc33595777)

[Scree and Cumulative Variance Plots 5](#_Toc33595778)

[Logistic Regression with PCA 7](#_Toc33595779)

[PCA for Dimension Reduction 11](#_Toc33595780)

[Linear Regression with PCA 14](#_Toc33595781)

[Other Resources 19](#_Toc33595782)

## Multicollinearity

Multicollinearity occurs when one predictor is highly correlated with another. Moderate multicollinearity may not be problematic. If the data being modelled follows the same collinearity as the predictors then the model may be fine. If you notice wide fluctuations in predictor significance when adding and removing variables this is an indicator of multicollinearity. Volatile coefficients can lead to an unreliable model and higher prediction variance.

### Controlling Multicollinearity

Some simple ways to control multicollinearity are to:

* Avoid creating dummy variables for every category.
* Drop variables.
* Obtain more data.

Other ways to remedy multicollinearity include:

* Mean-centering the predictor variables.
* Standardizing the independent (predictor) variables.
* Using principal component analysis regression.

#### Variance Inflation Factor (VIF)

Multicollinearity can be detected using a variance inflation factor (VIF) score. The VIF score is calculated for a particular variable by a linear model based on all other independent variables. VIF increases with a higher goodness of fit.

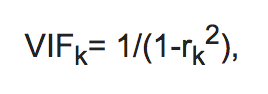


Table 1 suggests guidelines for interpreting VIF scores. These are rules of thumb and they are not absolute. As mentioned earlier, if the data being modelled follows the same collinearity pattern as the predictors then the model may be fine.

Table : Suggested Rules for Multicollinearity

|  |  |
| --- | --- |
| **VIF Score** | **Suggested Interpretation** |
| 1 | No multicollinearity |
| 1 to 5 | Moderate collinearity |
| 5+ | High collinearity |

## Principal Components Analysis (PCA)

For large feature data multicollinearity may be difficult to tame. However, principal component analysis (PCA) can reduce overfitting and multicollinearity through dimension reduction. Predictive models which use PCA are also usually faster due to the dimension reduction. PCA uses linear algebra which performs an orthogonal transformation to re-orient the coordinate system of the predictor variables. I will not cover the math behind PCA but this is a great article which helps to explain the theory and math.

<https://plot.ly/python/v3/ipython-notebooks/principal-component-analysis/>

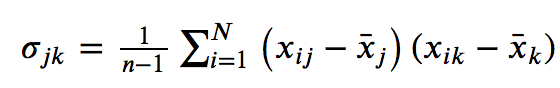
PCA does not require normally distributed data. However, usually before performing PCA, the data should be standardized on a -1 to 1 distribution which is centered at 0.

### Predictor Correlation

To understand the collinearity in the predictor variables covariance and correlation are examined.

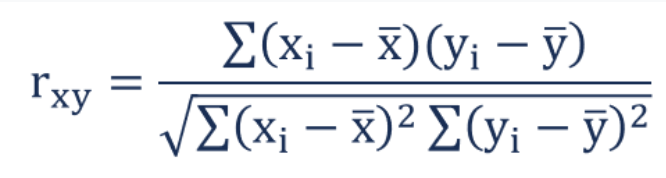
#### Covariance

Covariance measures how variables relate to each other. If two variables are independent their covariance is 0. The covariance between two features is calculated as follows:



#### Correlation

Correlation is the scaled version of covariance and correlation ranges between -1 and 1. When the covariance matrix is built with standardized data it is the same is the correlation matrix.



### Eigen Decomposition

To determine how to apply Principle Component Analysis, Eigen decomposition is then performed. Eigen decomposition involves generating Eigenvectors and Eigenvalues from the covariance matrix.

### Eigenvectors

Eigenvectors are vectors of direction weights. These vectors are used to transform the original feature coordinates. We will examine Eigenvectors more in Example 1.

### Eigen Values

An Eigenvalue tells us how much variance is explained by each Eigenvector. The Eigenvectors with the highest explanatory values are selected first.

Example : Eigenvectors and Eigenvalues

This example introduces Eigenvectors and Eigenvalues. For this case, Eigen decomposition is applied to understand the covariance and correlation between predictor variables in the Iris flower dataset.

|  |
| --- |
| from sklearn.preprocessing import StandardScaler  from sklearn import datasets  import numpy as np  # Load iris data set and apply standard scaler.  iris = datasets.load\_iris()  X = iris.data  featureNames = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']  y = iris.target  X\_std = StandardScaler().fit\_transform(X)  print(featureNames)  # Generate covariance matrix to show bivariate relationships.  cov\_mat = np.cov(np.transpose(X\_std))  print('\nCovariance matrix: \n%s' %cov\_mat)  # When data is standardized, the covariance matrix is same as the  # correlation matrix.  cor\_mat = np.corrcoef(np.transpose(X\_std))  print('\nCorrelation matrix: \n%s' %cor\_mat)  # Perform an Eigen decomposition on the covariance matrix:  eig\_vals, eig\_vecs = np.linalg.eig(cov\_mat)  print('\nEigenvectors \n%s' %eig\_vecs)  print('\nEigenvalues \n%s' %eig\_vals). |

The covariance matrix uses standardized data so the matrix values are the same as the correlation matrix. The covariance and correlation matrices also explain the relationships between predictor variables. For example, if you look at the output below you will notice that sepal length has a correlation of -0.1184 with sepal width.

The covariance matrix is used to generate the Eigenvector.

Try to think of the Eigenvectors as vectors of direction for generating the new feature coordinates.

Each Eigenvector is a principal component.

The Eigenvalue represents the magnitude (weighting) of the related Eigenvector.

|  |
| --- |
| Covariance matrix:  sepal l sepal w petal l petal w  sepal l [[ 1.00671141 -0.11835884 0.87760447 0.82343066]  sepal w [-0.11835884 1.00671141 -0.43131554 -0.36858315]  petal l [ 0.87760447 -0.43131554 1.00671141 0.96932762]  petal w [ 0.82343066 -0.36858315 0.96932762 1.00671141]]  Correlation matrix:  sepal l sepal w petal l petal w  sepal l [[ 1. -0.11756978 0.87175378 0.81794113]  sepal w [-0.11756978 1. -0.4284401 -0.36612593]  petal l [ 0.87175378 -0.4284401 1. 0.96286543]  petal w [ 0.81794113 -0.36612593 0.96286543 1. ]]  Eigenvectors  [[ 0.52106591 -0.37741762 -0.71956635 0.26128628]  [-0.26934744 -0.92329566 0.24438178 -0.12350962]  [ 0.5804131 -0.02449161 0.14212637 -0.80144925]  [ 0.56485654 -0.06694199 0.63427274 0.52359713]]  Eigenvalues  [2.93808505 0.9201649 0.14774182 0.02085386] |

Eigenvalues shown above weight the amount of variance between variables accounted for by each Eigenvector. The first Eigenvector (principal component) accounts for 73%of the variance between features.

= 0.72962%

Exercise (1 mark)

How much does sepal length correlate to petal length?

|  |
| --- |
| 0.871 |

Exercise (1 mark)

How does sepal length correlate to petal width?

|  |
| --- |
| 0.81794113 |

Exercise (2 marks)

How much variance does the second Eigenvector account for?

|  |
| --- |
| 0.9201649/ (2.93808505 + 0.9201649 + 0.14774182 + 0.02085386) = 0.23 |

Exercise (1 mark)

Together how much variance do the Eigenvectors attempt to describe? Are they 100% accurate?

|  |
| --- |
| All of the variance. No they are not 100% accurate. |

Exercise (1 mark)

According to the Eigenvalues, how much cumulative variance is accounted for by the first two Eigenvectors?

|  |
| --- |
| 0.96 |

Exercise (1 mark)

How much cumulative variance is accounted for by the first three Eigenvectors? In addition to showing your answer, please show your calculation or code used to generate the answer.

|  |
| --- |
| Values = [2.93808505 0.9201649 0.14774182 0.02085386]  sum = values[0] + values[1] + values[2]  total = values[0] + values[1] + values[2] + values[3]  Cumulative Variance = sum/total = 0.9948212914235801 |

## Choosing Components

When building a model with principal components, the method of component selection must be chosen. There is no perfect solution but it is important to select the first components which represent the highest amount of variance. Often, you might select components which explain the first 80% of the variance.

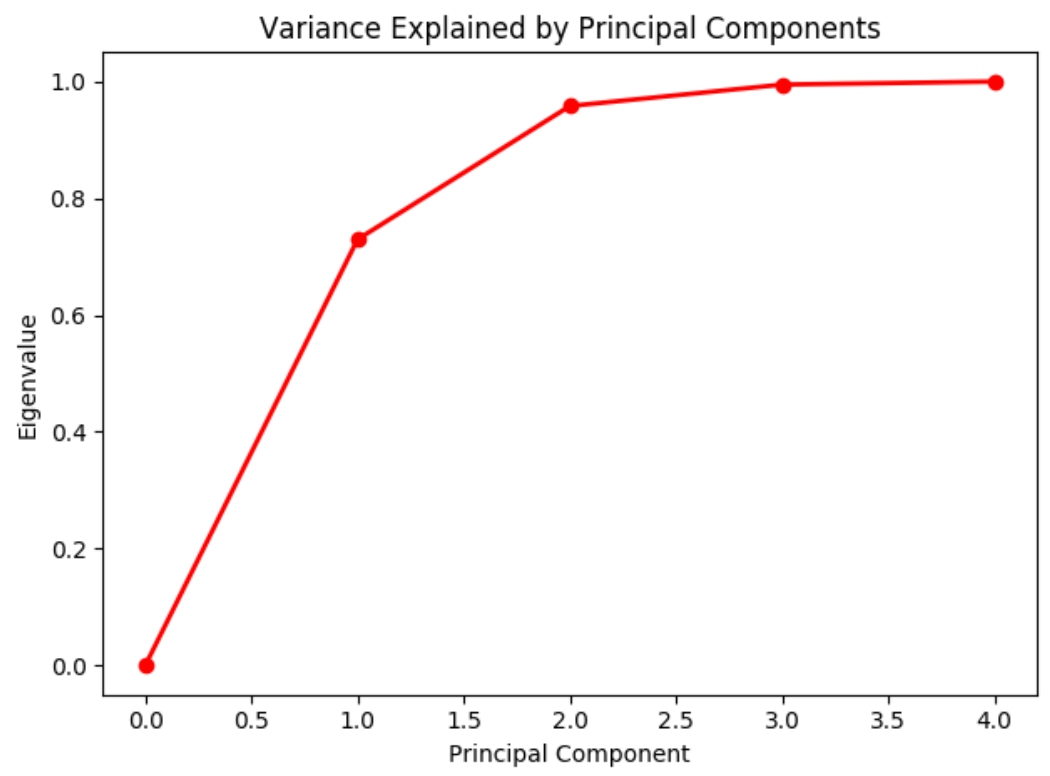
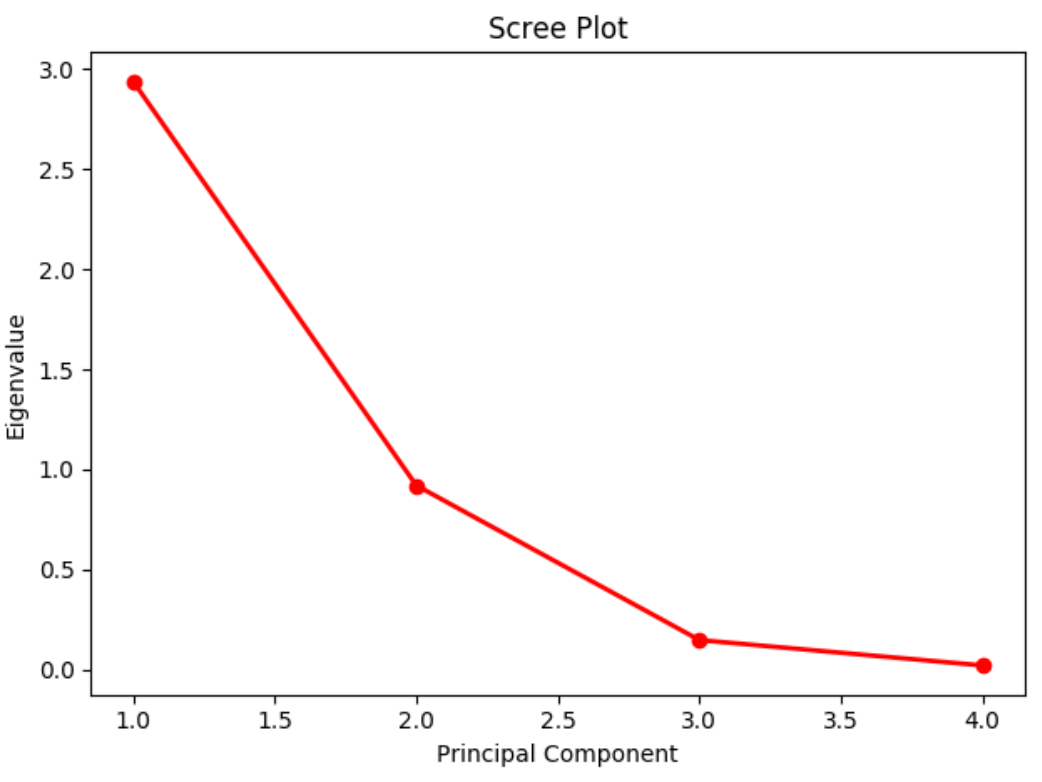
### Kaiser’s Stopping Rule

Kaiser’s Stopping rule suggests that we choose Eigenvalues scoring greater than 1.

### Scree and Cumulative Variance Plots

Scree and cumulative variance plots also help identify where gains from principal components diminish. (Scree means rock debris that has fallen down a mountain). Instead of using Kaiser’s Stopping rule, you may select the components which account for the most variance until gains flatten. See Figure 1.

Figure : Scree and Cumulative Variance Plots



Example : Drawing Scree and Cumulative Variance Plots

This code sample shows how to draw the curves in Figure 1. To build this example, start with the code in Example 1. Then add this code after.

|  |
| --- |
| import matplotlib.pyplot as plt  # Show the scree plot.  plt.plot([1,2,3,4], eig\_vals, 'ro-', linewidth=2)  plt.title('Scree Plot')  plt.xlabel('Principal Component')  plt.ylabel('Eigenvalue')  plt.show()  # Calculate cumulative values.  sumEigenvalues = eig\_vals.sum()  cumulativeValues = []  cumulativeSum = 0  for i in range(0,len(eig\_vals)+1):  cumulativeValues.append(cumulativeSum)  if(i<len(eig\_vals)):  cumulativeSum += eig\_vals[i] / sumEigenvalues  # Show cumulative variance plot.  import matplotlib.pyplot as plt  plt.plot([0,1,2,3,4], cumulativeValues, 'ro-', linewidth=2)  plt.title('Variance Explained by Principal Components')  plt.xlabel('Principal Component')  plt.ylabel('Eigenvalue')  plt.show() |

Exercise (1 mark)

Which principal components in Example 1 are recommended by Kaiser’s stopping rule and why?

|  |
| --- |
| Do not select Eigenvalues below 1. In this case component 1 should be selected. |

Exercise (1 mark)

Which components do the scree and cumulative variance plots in Figure 1 suggest using?

|  |
| --- |
| Scree: components 1 and 2 |

Exercise (1 mark)

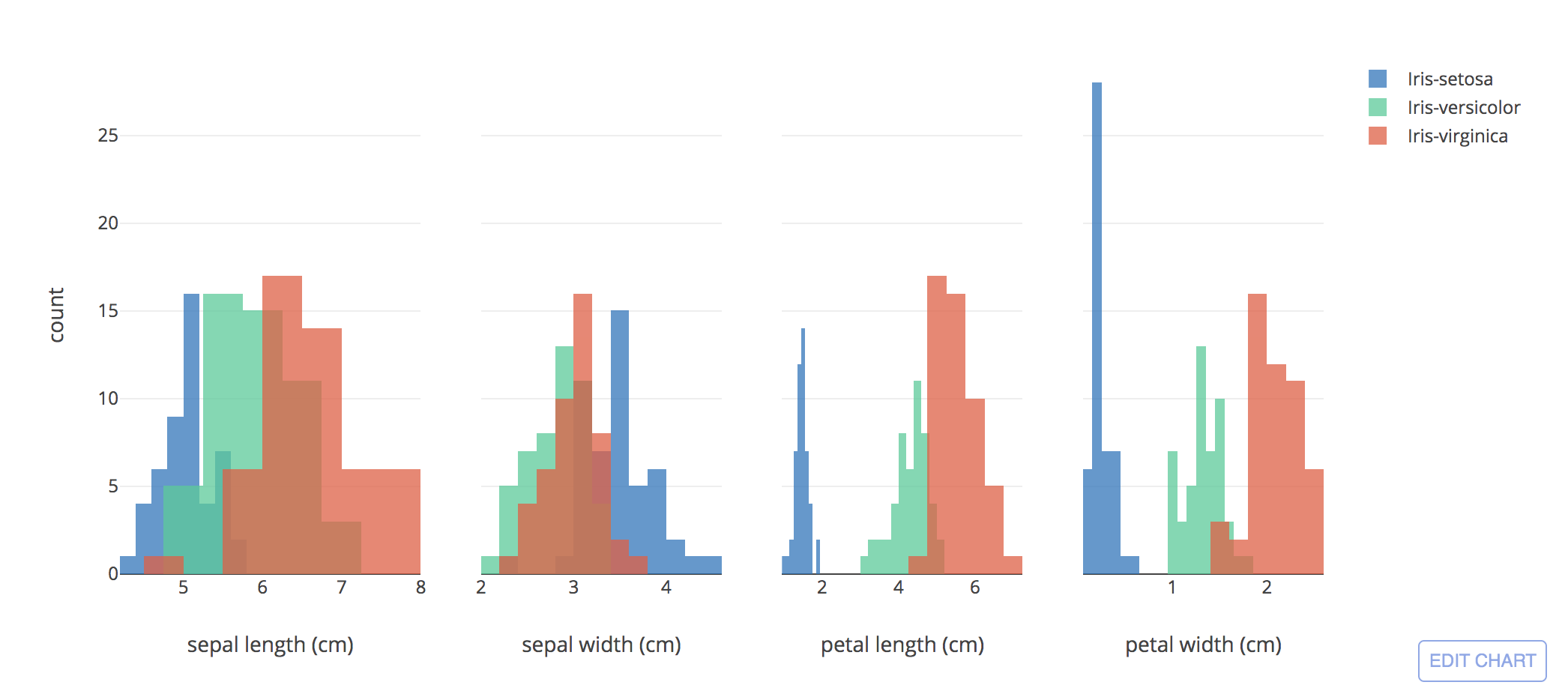
Imagine you have variance accounted for by five principal components where the Eigenvalues are: [3.3, 2.2, 0.5, 0.08, 0.05]

Which components do Kaiser’s Stopping rule recommend?

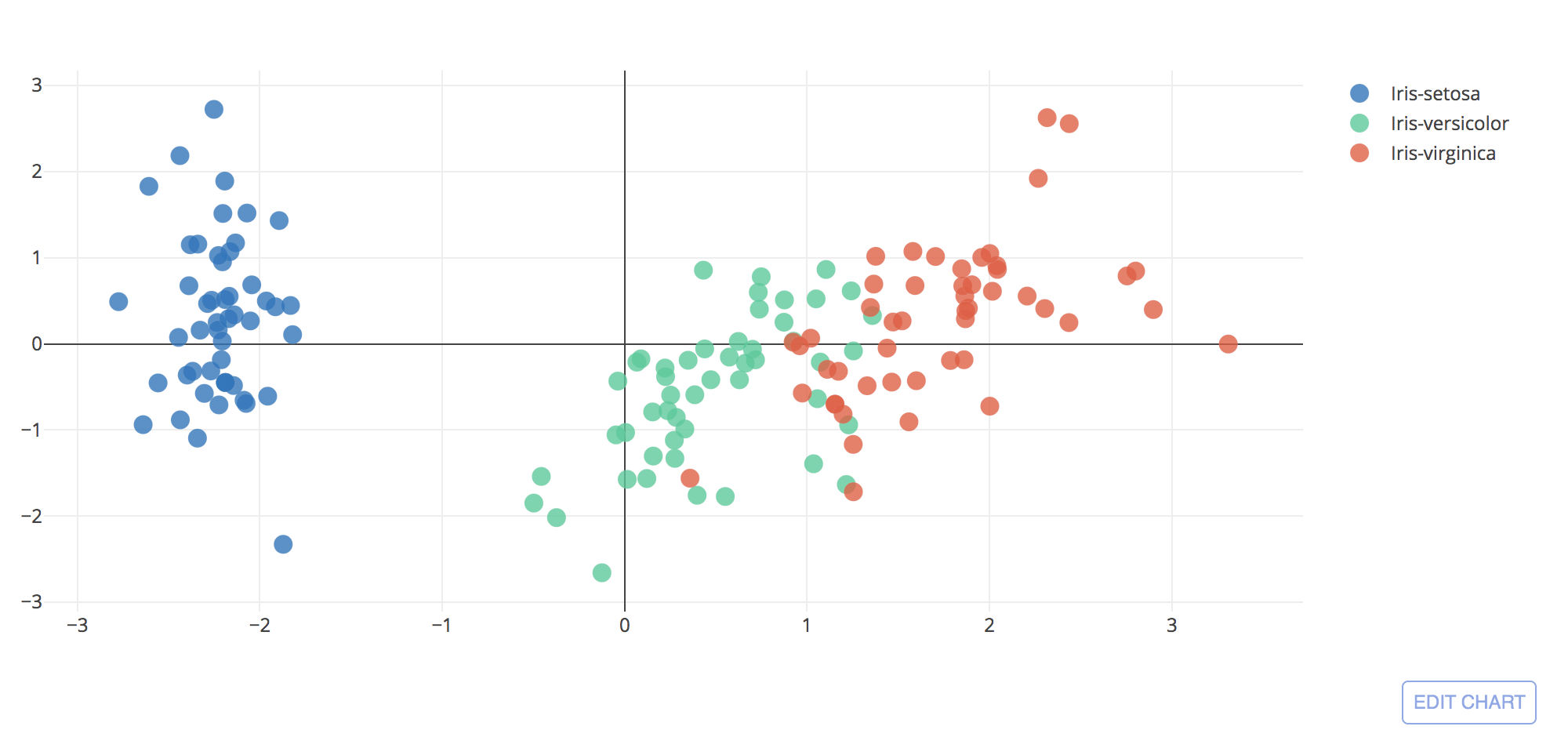
|  |
| --- |
| 3.3, and 2.2. |

## Logistic Regression with PCA

For logistic regression, principal components can be helpful since they can re-map coordinates to a new orientation that is more clearly separate by classification. Again, don’t think too hard about this. I do not plan to focus a lot here but very quickly imagine a data set where features overlap. Separating one class from the other is difficult.



A successful PCA transformation helps to separate classes by re-orienting the coordinates. Ideally PCA leads to a clear separation between classes. In display below a multi-dimensional feature set has been reduced to a two-dimensional set.



Example : PCA with Iris DataSet

This example shows how principal components can help perform logistic regression. For this case, the Iris dataset is used and the goal of the model is to classify flowers into Virginica, Setosa and Versicolor classes. For more detail see:

<https://plot.ly/python/v3/ipython-notebooks/principal-component-analysis/>

The Eigenvectors which we calculated in Example 1 are actually our principal components to be used in the dimension reduction. In this current example we will use the principal components to transform the standardized X values. We can then fit the logistic model on the scaled data.

As you can see from the output from our logistic regression we are able to generate our logistic model and the results of the logistic regression are quite decent.

|  |
| --- |
| \*\*\* Intercept:  [-1.58589278 -0.99583449 -2.01859188]  \*\*\* Model Coefficients:  [[-2.39890719 0.94546703]  [ 0.23032071 -1.08212721]  [ 2.30527623 0.0260823 ]]  \*\*\* Confusion Matrix  Predicted 0 1 2  Actual  0 13 0 0  1 0 10 6  2 0 0 9  \*\*\* Classification Report  precision recall f1-score support  0 1.00 1.00 1.00 13  1 1.00 0.62 0.77 16  2 0.60 1.00 0.75 9  accuracy 0.84 38  macro avg 0.87 0.88 0.84 38  weighted avg 0.91 0.84 0.84 38  \*\*\* Explained Variance  [3.0863862 0.98414798] |

Here is the code that generates the output above. You will also notice that the code also prints out the principal Eigenvectors and Eigenvalues which we generated in Example 1.

|  |
| --- |
| from sklearn.preprocessing import StandardScaler  from sklearn import datasets  import numpy as np  # Load iris data set and apply standard scaler.  iris = datasets.load\_iris()  X = iris.data  featureNames = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']  y = iris.target  X\_std = StandardScaler().fit\_transform(X)  print(featureNames)  # Generate covariance matrix to show bivariate relationships.  cov\_mat = np.cov(np.transpose(X\_std))  print('\nCovariance matrix: \n%s' %cov\_mat)  # When data is standardized, the covariance matrix is same as the  # correlation matrix.  cor\_mat = np.corrcoef(np.transpose(X\_std))  print('\nCorrelation matrix: \n%s' %cor\_mat)  # Perform an Eigen decomposition on the covariance matrix:  eig\_vals, eig\_vecs = np.linalg.eig(cov\_mat)  print('\nEigenvectors \n%s' %eig\_vecs)  print('\nEigenvalues \n%s' %eig\_vals)  #######  # Show the scree plot.  from sklearn.preprocessing import StandardScaler  from sklearn.decomposition import PCA as sklearnPCA  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn.metrics import classification\_report  from sklearn import datasets  import numpy as np  import pandas as pd  # Load iris data set and apply standard scaler.  iris = datasets.load\_iris()  X = iris.data  y = iris.target  X\_std = StandardScaler().fit\_transform(X)  # Split x and y into test and training.  X\_train,X\_test,y\_train,y\_test = train\_test\_split(  X\_std, y, test\_size=0.25,random\_state=0)  # Create principal components.  sklearn\_pca = sklearnPCA(n\_components=2)  # Transform the data.  X\_train = sklearn\_pca.fit\_transform(X\_train)  # Transform test data.  X\_test = sklearn\_pca.transform(X\_test)  # Perform logistic regression.  logisticModel = LogisticRegression(fit\_intercept=True, random\_state = 0,  solver='liblinear')  logisticModel.fit(X\_train, y\_train)  # Generate predictions.  y\_pred=logisticModel.predict(X\_test)  # Show model coefficients and intercept.  print("\n\*\*\* Intercept: ")  print(logisticModel.intercept\_)  print("\n\*\*\* Model Coefficients: ")  print(logisticModel.coef\_)  # Show confusion matrix and accuracy scores.  cm = pd.crosstab(y\_test, y\_pred, rownames=['Actual'],  colnames=['Predicted'])    print("\n\*\*\* Confusion Matrix")  print(cm)  print("\n\*\*\* Classification Report")  print(classification\_report(y\_test, y\_pred)) |

Example : PCA Transformation

To help understand the PCA transformation, we will look at the manual calculations for the PCA transform. After considering options from Example 1, two Eigenvectors (or principal components) have been selected. The left of Table 2 shows Iris feature values after a StandardScaler has been applied. The right shows the first two principal components that were generated in Example 1.

Table : StandardScaled X Values and Selected Components

|  |  |
| --- | --- |
| **StandardardScaled X Values (Sepal length, Sepal width, Petal length, Petal width)** | **Principal Components (Eigenvectors)** |
| [-9.0068e-01, 1.0190e+00, -1.3402e+00, -1.3154e+00], [-1.1430e+00, -1.3198e-01, -1.3402e+00, -1.3154e+00], [-1.3853e+00, 3.2841e-01, -1.3971e+00, -1.3154e+00], | [[ 0.52106591 ,-0.37741762],  [-0.26934744, -0.92329566],  [ 0.5804131, -0.02449161],  [ 0.56485654, -0.06694199]] |

Now we will use the two principal components to transform the original scaled data for the first row of X-values:

|  |  |  |
| --- | --- | --- |
| [-9.0068e-01, 1.0190e+00, -1.3402e+00, -1.3154e+00] **\*** | | [[ 0.52106591 ,-0.37741762],  [-0.26934744, -0.92329566],  [ 0.5804131, -0.02449161],  [ 0.56485654, -0.06694199]] |
| =  [-9.0068e-01 \* 0.52106591 +  1.0190e+00 \* -0.26934744 +  -1.3402e+00 \* 0.5804131 +  -1.3154e+00 \* 0.56485654 ] **+** | [-9.0068e-01 \* 0.37741762 +  1.0190e+00 \* -0.92329566 +  -1.3402e+00 \* -0.02449161 +  -1.3154e+00 \* -0.06694199 ] | |
| Our transformed X values are:  [-2.26470281, -0.48002659]  We will now use the PCA transformed variables to fit and test the logistic model. | | |

Exercise (1 mark)

Explain how the final result of [-2.26470281, -0.48002659] is a dimension reduction.

|  |
| --- |
| We went from four features to two features after the vector and matrix multiplication. |

Exercise (2 marks)

Using the selected two principal components above, perform a PCA transformation for this standardized feature set:

[-1.1430e+00, -1.3198e-01, -1.3402e+00, -1.3154e+00]

Show your calculations and result here (you can use Python for the calculation if you want but if you do please show your code and output):

|  |
| --- |
| import numpy as np matrix = [[0.52106591, -0.37741762],  [-0.26934744, -0.92329566],  [0.5804131, -0.02449161],  [0.56485654, -0.06694199]]  flower = [-1.1430e+00, -1.3198e-01, -1.3402e+00, -1.3154e+00] a = np.array(matrix) b = np.array(flower)  print(b.dot(a))  [-2.0809117893347997, 0.6741240502347999] |

Exercise (2 marks)

Using the selected two principal components above, perform a PCA transformation for this standardized feature set:

[-1.3853e+00, 3.2841e-01, -1.3971e+00, -1.3154e+00]

Show your calculations and result here (you can use Python for the calculation if you want but if you do please show your code and output):

|  |
| --- |
| import numpy as np matrix = [[0.52106591, -0.37741762],  [-0.26934744, -0.92329566],  [0.5804131, -0.02449161],  [0.56485654, -0.06694199]]  flower = [-1.1430e+00, -1.3198e-01, -1.3402e+00, -1.3154e+00] a = np.array(matrix) b = np.array(flower)  print(b.dot(a))  # Exercise 12 flower\_2 = [-1.3853e+00, 3.2841e-01, -1.3971e+00, -1.3154e+00] c = np.array(flower\_2) print(c.dot(a))  [-2.36419643 0.34188982] |

Exercise (4 marks)

To observe the multicollinearity scores of the principal components in our model, add this code to the end of the program in Example 3.

|  |
| --- |
| # For each X, calculate VIF and save in dataframe  from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  vif = pd.DataFrame()  vif["VIF Factor for Components"] = \  [variance\_inflation\_factor(X\_train, i) for i in range(X\_train.shape[1])]  print(vif) |

Show the output that appears after running the code.

|  |
| --- |
| ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']  Covariance matrix:  [[ 1.00671141 -0.11835884 0.87760447 0.82343066]  [-0.11835884 1.00671141 -0.43131554 -0.36858315]  [ 0.87760447 -0.43131554 1.00671141 0.96932762]  [ 0.82343066 -0.36858315 0.96932762 1.00671141]]  Correlation matrix:  [[ 1. -0.11756978 0.87175378 0.81794113]  [-0.11756978 1. -0.4284401 -0.36612593]  [ 0.87175378 -0.4284401 1. 0.96286543]  [ 0.81794113 -0.36612593 0.96286543 1. ]]  Eigenvectors  [[ 0.52106591 -0.37741762 -0.71956635 0.26128628]  [-0.26934744 -0.92329566 0.24438178 -0.12350962]  [ 0.5804131 -0.02449161 0.14212637 -0.80144925]  [ 0.56485654 -0.06694199 0.63427274 0.52359713]]  Eigenvalues  [2.93808505 0.9201649 0.14774182 0.02085386]  \*\*\* Intercept:  [-1.58589278 -0.99583449 -2.01859188]  \*\*\* Model Coefficients:  [[-2.39890719 0.94546703]  [ 0.23032071 -1.08212721]  [ 2.30527623 0.0260823 ]]  \*\*\* Confusion Matrix  Predicted 0 1 2  Actual  0 13 0 0  1 0 10 6  2 0 0 9  \*\*\* Classification Report  precision recall f1-score support  0 1.00 1.00 1.00 13  1 1.00 0.62 0.77 16  2 0.60 1.00 0.75 9  accuracy 0.84 38  macro avg 0.87 0.88 0.84 38  weighted avg 0.91 0.84 0.84 38  VIF Factor for Components  0 1.0  1 1.0 |

How can the VIF scores be interpreted?

|  |
| --- |
| Based on the VIF scores, we can see that a value of 1.0 indicates no collinearity for variable 1 and variable 2. |

Originally our feature set contained sepal length, sepal width, petal length, and petal width. How many features appear in the training data and why?

|  |
| --- |
| There are two features in the training data because we computed a dimension reduction through matrix multiplication. |

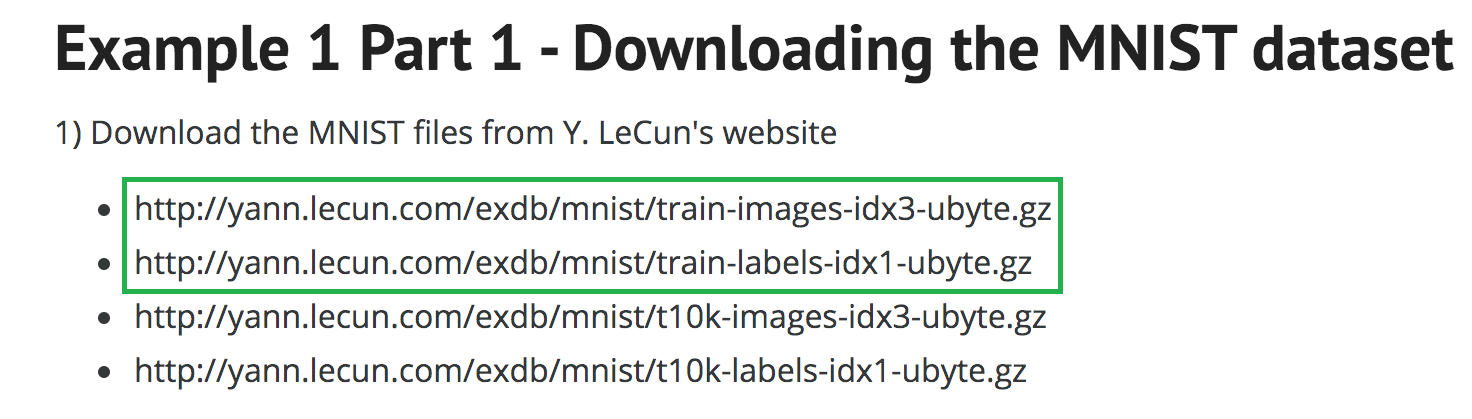
## PCA for Dimension Reduction

An important benefit of PCA is dimension reduction for data sets with large numbers of features. Not only does PCA eliminate complex multicollinearity issues, it also improves processing speed with the dimension reduction.

Example : MNIST

This example uses PCA to reduce the dataset to predict actual numbers with pixel input from images of hand-written digits. To obtain the data in a format that can be used easily in Python, go to <http://rasbt.github.io/mlxtend/user_guide/data/loadlocal_mnist/>

Download the train labels and train images.



The data set contains 28x28 gray scale images which are stored in arrays of integers. Each image array contains 784 elements. Data ranges from 0 to 256. Figure 2 displays the contents for a sample image on the left along with a rendering of the grey scale pixels on the right.

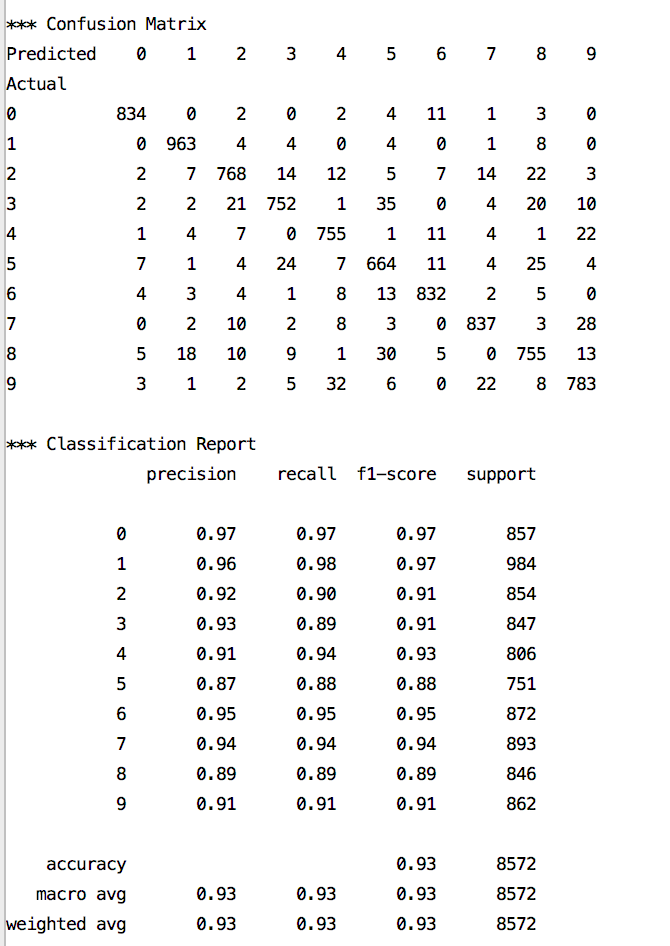
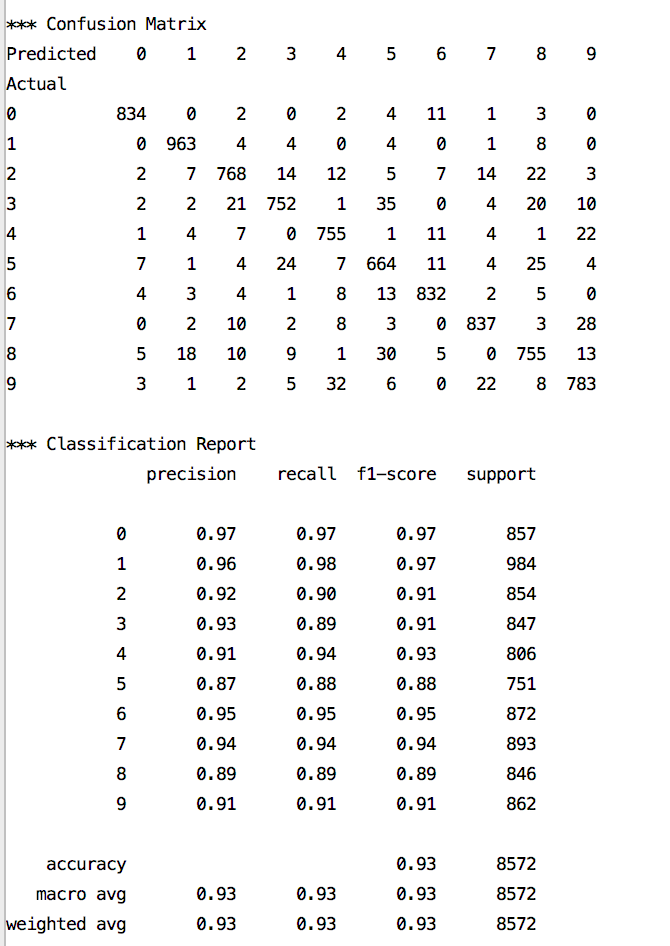
Figure : Numeric and Graphic Representation of Gray Scale Image

|  |  |
| --- | --- |
|  |  |

Before running the code, be sure to set the appropriate path the image files after extracting them.

|  |
| --- |
| from mlxtend.data import loadlocal\_mnist  from sklearn.metrics import classification\_report  X, y = loadlocal\_mnist(  images\_path='/Users/pm/Downloads/train-images-idx3-ubyte',  labels\_path='/Users/pm/Downloads/train-labels-idx1-ubyte')  # http://rasbt.github.io/mlxtend/user\_guide/data/loadlocal\_mnist/  print('Dimensions: %s x %s' % (X.shape[0], X.shape[1]))  print('\n1st row', X[0])  # Split the data.  from sklearn.model\_selection import train\_test\_split  # test\_size: what proportion of original data is used for test set  train\_img, test\_img, train\_lbl, test\_lbl = train\_test\_split( X,  y,  test\_size=1/7.0,  random\_state=0)  # Show image.  print("Image size: ")  print(train\_img[0].shape)  import matplotlib.pyplot as plt  import numpy as np  first\_image = train\_img[0]  first\_image = np.array(first\_image, dtype='float')  print(len(first\_image))  pixels = first\_image.reshape((28, 28))  plt.imshow(pixels, cmap='gray')  plt.show()  from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  # Fit on training set only.  scaler.fit(train\_img)  # Apply transform to both the training set and the test set.  train\_img = scaler.transform(train\_img)  test\_img = scaler.transform(test\_img)  from sklearn.decomposition import PCA  # Make an instance of the Model  pca = PCA(.95)  pca.fit(train\_img)  train\_img = pca.transform(train\_img)  test\_img = pca.transform(test\_img)  from sklearn.linear\_model import LogisticRegression  # all parameters not specified are set to their defaults  # default solver is incredibly slow which is why it was changed to 'lbfgs'  logisticRegr = LogisticRegression(solver = 'lbfgs', max\_iter=1000)  logisticRegr.fit(train\_img, train\_lbl)  y\_pred = logisticRegr.predict(test\_img)  score = logisticRegr.score(test\_img, test\_lbl)  print(score)  # Show confusion matrix and accuracy scores.  import pandas as pd  cm = pd.crosstab(test\_lbl, y\_pred, rownames=['Actual'],  colnames=['Predicted'])  print("\n\*\*\* Confusion Matrix")  print(cm)  print("\n\*\*\* Classification Report")  print(classification\_report(test\_lbl, y\_pred)) |

The output from running this code shows the result of an accurate predictive model.

Exercise (2 marks)

Read through the code and figure out how to show the second and third images in the training data. Show the images here.

|  |
| --- |
| second\_img = train\_img[1]  second\_img = np.array(second\_img, dtype= ‘float’)  print(len(second\_img))  pixels = second\_img.reshape((28,28))  plt.imshow(pixels, cmap=’gray’)  plt.show()  third\_img = train\_img[2]  third\_img = np.array(third\_img, dtype= ‘float’)  print(len(third\_img))  pixels = third\_img.reshape((28,28))  plt.imshow(pixels, cmap=’gray’)  plt.show() |

## Linear Regression with PCA

As you may have guessed, PCA can be used with linear regression. The selected components can be used as variables in the regression.

Example : Linear Regression with Principal Components

This example uses PCA to help perform linear regression to estimate the salary of a baseball hitters. For more detail, please see <http://www.science.smith.edu/~jcrouser/SDS293/labs/lab11-py.html> Table 1 shows preliminary statistics about the predictor and target variables.

Table : Predictor and Target Variable Statistics

|  |  |  |
| --- | --- | --- |
| # Column Non-Null Count Dtype  --- ------ -------------- -----  0 AtBat 263 non-null int64  1 Hits 263 non-null int64  2 HmRun 263 non-null int64  3 Runs 263 non-null int64  4 RBI 263 non-null int64  5 Walks 263 non-null int64  6 Years 263 non-null int64  7 CAtBat 263 non-null int64  8 CHits 263 non-null int64  9 CHmRun 263 non-null int64  10 CRuns 263 non-null int64  11 CRBI 263 non-null int64  12 CWalks 263 non-null int64  13 League 263 non-null object  14 Division 263 non-null object  15 PutOuts 263 non-null int64  16 Assists 263 non-null int64  17 Errors 263 non-null int64  18 Salary 263 non-null float64  19 NewLeague 263 non-null object |  | **Salary** **statistics**:  count 263.000000  mean 535.925882  std 451.118681  min 67.500000  25% 190.000000  50% 425.000000  75% 750.000000  max 2460.000000 |

To build this example download the Hitters.csv file from the Datasets folder at D2L. I am using a version which has been modified from the original. Here is the code:

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn import model\_selection  from sklearn.decomposition import PCA  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_squared\_error  from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  from sklearn.preprocessing import StandardScaler  PATH = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/"  CSV\_DATA = "Hitters.csv"  # Drop null values.  df = pd.read\_csv(PATH + CSV\_DATA).dropna()  df.info()  dummies = pd.get\_dummies(df[['League', 'Division', 'NewLeague']])  y = df.Salary  print("\nSalary stats: ")  print(y.describe())  # Drop the column with the independent variable (Salary),  # and columns for which we created dummy variables.  X\_ = df.drop(['Salary', 'League', 'Division', 'NewLeague'], axis=1).astype('float64')  # Define the feature set X.  X = pd.concat([X\_, dummies[['League\_N', 'Division\_W', 'NewLeague\_N']]], axis=1)  # Calculate and show VIF Scores for original data.  vif = pd.DataFrame()  vif["VIF Factor"] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]  vif["features"] = X.columns  print("\nOriginal VIF Scores")  print(vif)  # Standardize the data.  X\_scaled = StandardScaler().fit\_transform(X)  # Split into training and test sets  X\_train, X\_test , y\_train, y\_test = model\_selection.train\_test\_split(X\_scaled, y,  test\_size=0.25, random\_state=1)  # Transform the data using PCA for first 80% of variance.  pca = PCA(.8)  X\_reduced\_train = pca.fit\_transform(X\_train)  X\_reduced\_test = pca.transform(X\_test)[:,:7]  print("\nPrincipal Components")  print(pca.components\_)  print("\nExplained variance: ")  print(pca.explained\_variance\_)  # Train regression model on training data  model = LinearRegression()  model.fit(X\_reduced\_train[:,:7], y\_train)  # Prediction with test data  pred = model.predict(X\_reduced\_test)  print()  # Show stats about the regression.  mse = mean\_squared\_error(y\_test, pred)  RMSE = np.sqrt(mse)  print("\nRMSE: " + str(RMSE))  print("\nModel Coefficients")  print(model.coef\_)  print("\nModel Intercept")  print(model.intercept\_)  from sklearn.metrics import r2\_score  print("\nr2\_score",r2\_score(y\_test,pred))  # For each principal component, calculate the VIF and save in dataframe  vif = pd.DataFrame()  # Show the VIF score for the principal components.  print()  vif["VIF Factor"] = [variance\_inflation\_factor(X\_reduced\_train, i) for i in range(X\_reduced\_train.shape[1])]  print(vif) |

While not amazing, the output shows a reasonably decent goodness of fit. The correlation is = 0.6420 which is decent.

|  |
| --- |
| Explained variance:  [7.77597646 4.18461023 2.08996888 1.51498738]  RMSE: 362.53224118763626  Model Coefficients  [ 99.33549424 -38.1317022 27.85002734 -29.67678236]  Model Intercept  532.6254517766497  r2\_score 0.4122990272060483  VIF Factor  0 1.0  1 1.0  2 1.0  3 1.0 |

Exercise (2 marks)

Which features of the data set are transformed by the coordinates of each principal component?

|  |
| --- |
| 'Salary', 'League', 'Division', 'NewLeague' |

Exercise (4 marks)

Run the regression in Example 6 again but with enough principal components to account for 90% of the variance. How many principal components are used? Do the RMSE and scores improve much when compared to the original example output? Explain if you think the additional principal components are worthwhile.

|  |
| --- |
| 7 PCA components are used. RMSE improved marginally by a reducing of 6. R­2 improved by 8. I believe the additional components are not worthwhile because they add very little to R2 and our RMSE. It’s better to have a simpler model with fewer PCA components because in this example, the original example’s PCA components already cover a large amount of the variance. |

Exercise

Perform linear regression using PCA to estimate the sale price of a home using the USA Housing data set. Starter code is provided for you below.

NOTE:

This exercise is not intended to be an exhaustive exercise in any way. You do not need to create dummy or binned variables. You do not need to find the optimal variable combination for your model. Just build a crude model with the starter code provided. Also use **Example 6** as a guide to complete this section.

Starting with the following code, perform a linear regression using PCA. You do not need to create dummy variables, binned variables or any other variables

|  |
| --- |
| import pandas as pd  from sklearn import model\_selection  from sklearn.decomposition import PCA  from sklearn.linear\_model import LinearRegression  from sklearn.preprocessing import StandardScaler  from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  from sklearn.metrics import r2\_score, mean\_squared\_error  import numpy as np  import pandas as pd  PATH = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/"  CSV\_DATA = "USA\_Housing.csv"  df = pd.read\_csv(PATH + CSV\_DATA,  skiprows=1, # Don't include header row as part of data.  encoding = "ISO-8859-1", sep=',',  names=('Avg. Area Income','Avg. Area House Age',  'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', "Area Population", 'Price', "Address"))  # Show all columns.  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  df2 = df.\_get\_numeric\_data()  X = df2.copy()  X.drop(['Price'], inplace=True, axis=1)  y = df2.copy()  y = y[['Price']] |

* 1. What is the average housing price? The price standard deviation? The max price? The min price? (2 marks)

|  |
| --- |
| * Average Housing Price of Predicated Housing Prices * 1227056.901215787 * Standard Deviation of Housing Prices: 342136.65996049735 * Min House Price: * 156631.15584694454 * Max House Price * 2343432.6820526626   Can you explain how to do this with scaler?  You provided this code:  scaler =preprocessing.StandardScaler().fit(X\_train) |

* 1. What are the VIF scores of the original features? Do they suggest multi-collinearity? (2 marks)

|  |
| --- |
| * Original VIF Scores * VIF Factor features * 0 29.650899 Avg. Area Income * 1 27.447775 Avg. Area House Age * 2 45.257291 Avg. Area Number of Rooms * 3 14.537873 Avg. Area Number of Bedrooms * 4 12.825450 Area Population |

* 1. What dataset features are transformed and reduced by the principal components? (2 marks)

|  |
| --- |
| Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  Avg. Area Number of Bedrooms  Area Population |

* 1. What are the VIF scores of the principal components in the model? (1 mark)

|  |
| --- |
| * VIF Factor * 0 1.0 * 1 1.0 * 2 1.0 * 3 1.0 |

* 1. What is your value and your RMSE? (2 marks)

|  |
| --- |
| r2\_score 0.8829704142187813  RMSE: 124463.76078936488 |

* 1. Show the principal components here. You may use: (2 marks)

print("\nPrincipal Components")

print(pca.components\_)

|  |
| --- |
| Principal Components  [[ 2.14709858e-02 -4.05145160e-02 7.02714648e-01 7.09273371e-01  -3.19527656e-02]  [-6.83895055e-01 1.25132035e-01 7.75586361e-02 -1.68084239e-02  7.14375016e-01]  [ 1.93653790e-01 -9.17386541e-01 9.01422297e-04 -4.36171038e-02  3.44959244e-01]  [-7.01439565e-01 -3.74814876e-01 -5.43033441e-03 -2.20866449e-02  -6.05787947e-01]] |

* 1. Show your linear model here. You can obtain the model by summing model intercept and coefficients multiplied by the principal components. (5 marks)

|  |
| --- |
| * Avg. Area Income * Avg. Area House Age * Avg. Area Number of Rooms * Avg. Area Number of Bedrooms * Area Population   Housing Price = 1234012.18028601 + 225964.72315437(Avg. Area Income) + 161517.07941919(Avg. Area House Age) + 56686.92264235(Avg. Area Number of Rooms) + 66768.37957449(Avg. Area Number of Bedrooms) + 155028.8800317(Area Population) |

* 1. Show your completed code here (5 marks)

|  |
| --- |
| * import pandas as pd from sklearn import model\_selection from sklearn.decomposition import PCA from sklearn.linear\_model import LinearRegression from sklearn.preprocessing import StandardScaler from statsmodels.stats.outliers\_influence import variance\_inflation\_factor from sklearn.metrics import r2\_score, mean\_squared\_error import pandas as pd import numpy as np from sklearn import model\_selection from sklearn.decomposition import PCA from sklearn.linear\_model import LinearRegression from sklearn.metrics import mean\_squared\_error from statsmodels.stats.outliers\_influence import variance\_inflation\_factor from sklearn.preprocessing import StandardScaler from sklearn import preprocessing  import numpy as np import pandas as pd  PATH = "../dataset/" CSV\_DATA = "USA\_Housing.csv" df = pd.read\_csv(PATH + CSV\_DATA,  skiprows=1, # Don't include header row as part of data.  encoding = "ISO-8859-1", sep=',',  names=('Avg. Area Income','Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms',  "Area Population", 'Price', "Address")).dropna() # Show all columns. pd.set\_option('display.max\_columns', None) pd.set\_option('display.width', 1000) df2 = df.\_get\_numeric\_data()  X = df2.copy() X.drop(['Price'], inplace=True, axis=1) y = df2.copy() y = y[['Price']]  # Drop null values. df = pd.read\_csv(PATH + CSV\_DATA).dropna() df.info()   # Calculate and show VIF Scores for original data. vif = pd.DataFrame() vif["VIF Factor"] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])] vif["features"] = X.columns print("\nOriginal VIF Scores") print(vif)  # Standardize the data. X\_scaled = StandardScaler().fit\_transform(X)  # Split into training and test sets X\_train, X\_test , y\_train, y\_test = model\_selection.train\_test\_split(X\_scaled, y,  test\_size=0.25, random\_state=1)  scaler =preprocessing.StandardScaler().fit(X\_train) print("Mean")   # Transform the data using PCA for first 80% of variance. pca = PCA(.8) X\_reduced\_train = pca.fit\_transform(X\_train) X\_reduced\_test = pca.transform(X\_test)[:,:7]  print("\nPrincipal Components") print(pca.components\_)  print("\nExplained variance: ") print(pca.explained\_variance\_)  # Train regression model on training data model = LinearRegression() model.fit(X\_reduced\_train[:,:7], y\_train)  # Prediction with test data pred = model.predict(X\_reduced\_test) sum\_pred = sum(pred) pred\_avg = sum\_pred/len(pred) print("\nAverage Housing Price of Predicated Housing Prices") print(np.mean(pred)) print(f"\nStandard Deviation of Housing Prices: {np.std(pred)}") print("Min House Price:") print(pred.min()) print("Max House Price") print(pred.max())  # Show stats about the regression. mse = mean\_squared\_error(y\_test, pred) RMSE = np.sqrt(mse) print("\nRMSE: " + str(RMSE))  print("\nModel Coefficients") print(model.coef\_)  print("\nModel Intercept") print(model.intercept\_)  from sklearn.metrics import r2\_score print("\nr2\_score",r2\_score(y\_test,pred))  # For each principal component, calculate the VIF and save in dataframe vif = pd.DataFrame()  # Show the VIF score for the principal components. print() vif["VIF Factor"] = [variance\_inflation\_factor(X\_reduced\_train, i) for i in range(X\_reduced\_train.shape[1])] print(vif)  pca\_components = np.array(pca.components\_) coefficients = np.array(model.coef\_) transformation = coefficients.dot(pca\_components) print(f"Transformation {transformation}") |

## Other Resources

Here are two good articles on the math and theory behind the transforms:

<https://plot.ly/python/v3/ipython-notebooks/principal-component-analysis/>

<https://machinelearningmastery.com/calculate-principal-component-analysis-scratch-python/>