CS-345/M45 Lab Class 3

Release date: 26/10/2020
Due date: 09/11/2020

This lab is about utilizing Linear Regression for continuous value prediction. It also looks at Principal Component Analysis and Linear Discriminant Analysis for feature dimension reduction. We will be implementing these algorithms on some example training data, before then carrying out the methods on a testing dataset. The packages to use in this lab sheet are numpy, matplotlib and scikit-learn.

Again, you should be annotating your notebooks with Markdown cells.

☐ Task 3.1 – Linear Regression with sklearn

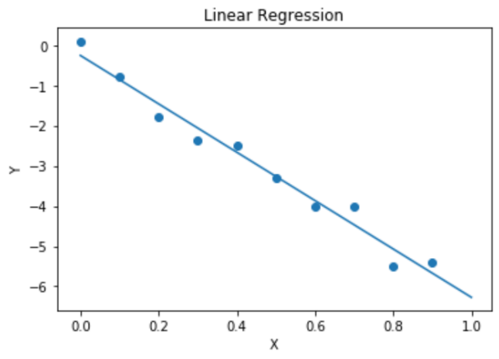
This first task involves using Linear Regression to fit a predictive regression model to a collection of observed datapoints. To do this you will need to utilise `LinearRegression` class from within `sklearn.linear_model`. You are provided with the data in the form of two numpy arrays, `x_values.npy` and `y_values.npy`, on Canvas. Download the files to your local directory.

`x_values.npy` corresponds to the regular variable, the magnitude along the x-axis. `y_values.npy` corresponds to the target variable, i.e. the measurement we want to try and predict along the y-axis. In our Linear Regression model we intend to provide `x_values.npy` and `y_values.npy` to train our model. Then, given a previously unseen value of x, we want to predict a corresponding y value.

To achieve this goal we will first load in our data and visualise x values against y values. We will then initialise a `LinearRegression` object and train the model on the observed data. As Linear Regression is a supervised approach we must provide targets, therefore when calling your `fit()` function you need to pass both the data (x) and the targets (y). We will then predict an output $\hat{y}$ value for an unseen value of x.

- Load `x_values.npy` and `y_values.npy`, and visualise the data with a scatter plot.
- Create a `LinearRegression` object from sklearn.
- Fit the `LinearRegression` object to the data, given the targets.
- Load `test_x_values.npy` and predict the $\hat{y}$ values for each of the x values within `test_x_values.npy`.
- Plot the training data x and y values using a scatter plot. On the same figure, plot the $test\_x$ and predicted $\hat{y}$ values using matplotlib's `plot` method (rather than scatter). Label the axes appropriately.
- Predict and print the $\hat{y}$ value for a value of $x = 0.48$.

You should end up with a plot like:

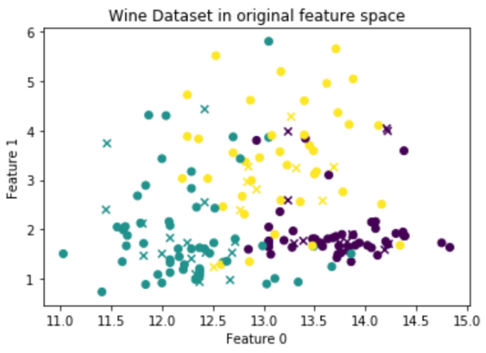


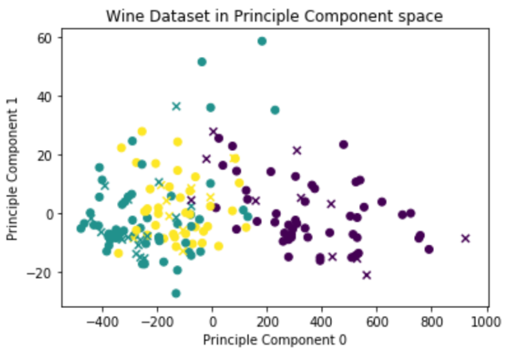
☐ Task 3.2 – Principle Component Analysis with sklearn

The second task is to use Principal Component Analysis to reduce the dimensionality of the Wine Dataset. Overall, we will: divide the dataset into a training and testing set, utilise a `sklearn.decomposition.PCA` object to reduce the dimensionality to two principle components, and then visualise the data in the new principle component space. You are provided with the data in the form of two numpy arrays, `wineData.npy` and `wineLabels.npy`, on Canvas. Download the files to your local directory.

- Load `wineData.npy` and `wineLabels.npy`, inspect the data, and visualise it.
- Divide the data and labels into two sets: training and testing. Use slicing here to select some proportion of the data and assign them into two separate variables `train_data` `test_data`. Use the same indices to slice your labels and create `train_labels` `test_labels`, you need to make sure that the data and labels still match. A common split in literature is 80% training to 20% testing.
- Produce a scatter plot which shows your train:test split. For example, plot the training data with circle markers and the testing data with triangle markers. Remember to colour them based on their respective labels. Plot feature 0 against feature 1.
- Initialise a `PCA` object, with an input argument to the constructor which will only keep the first 2 principle components. This will reduce dimensionality from 13 features, to 2.
- Fit the `PCA` model to the **training data**.
- Apply the dimensionality reduction transform to the training data. This takes in our 13-dimensional data and reduces it down to 2 dimensions. Hint: Check API for `transform()`.
- Apply the dimensionality reduction transform to the testing data.
- Visualise the reduced-dimensionality training and testing data, using a scatter plot. Don't forget to use different markers for the two sets, and to colour the markers with the labels.

You should end up with plots like:

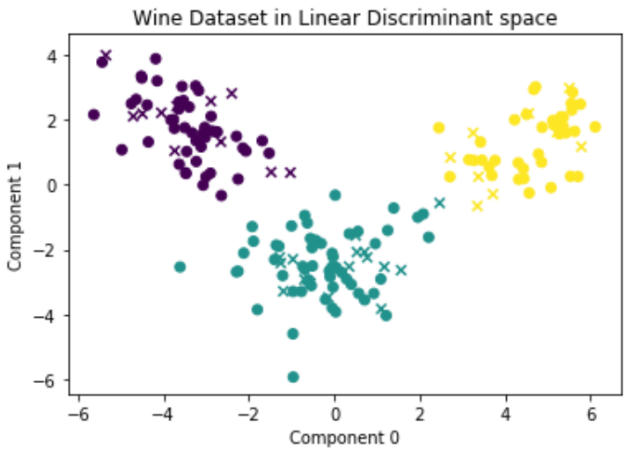




☐ Task 3.3 – Linear Discriminant Analysis with sklearn

The third task is to use Linear Discriminant Analysis to reduce the dimensionality of the Wine Dataset. This time we will be using a supervised technique to reduce our dimensionality. In this task you will use the same train:test split you have identified in task 3.2, i.e. train_data, test_data, train_labels, and test_labels. LDA can also be used for class prediction, and has a `predict` method, but we will be using it for dimensionality reduction in this task by calling the `transform` method.

- Create a `sklearn.discriminant_analysis.LinearDiscriminantAnalysis` object. Again we want to provide an argument to the constructor which will only keep the first 2 components.
- Fit the model to the training data and training labels. Notice that unlike with PCA, we now need to provide the class labels for our training data (why?).
- Apply the dimensionality reduction transform to the training data.
- Apply the dimensionality reduction transform to the testing data.
- Visualise the reduced-dimensionality training and testing data, using a scatter plot. Don't forget to use different markers for the two sets, and to colour the markers based on the labels.



You should end up with a plot like:

☐ Task 3.4 – Principal Component Analysis by hand

The fourth task is to implement Principal Component Analysis by hand. We will perform the Singular Value Decomposition and project the data into the new principal component space, before visualising the data in the reduced dimensionality space.

In this task you will use the same train:test split you have identified in task 3.2, i.e. train_data, test_data, train_labels, and test_labels

- Mean-centre the training data. To do this, you should identify the mean vector of the training data, and subtract that vector from the samples in the training data. You should save this mean vector, you will need it later for centring the test data.
- Calculate the Singular Value Decomposition of the mean-centred training data. You can do this with `numpy.linalg.svd()`, which returns the variables u, s and vh. The values returned by numpy's `svd` are analogous to U, S and V^T on slide 22 of the Dimensionality Reduction lecture. They form the linear system:

  $$X = USV^T \tag{1}$$

  where X is our mean-centred data, U is the left-singular vectors, S the singular values, and V the right-singular vectors.
- Project your data into a 2-dimensional Principle Component space. To do this you will need to select the first few significant eigenvectors of V^T (see slide 23 for reasoning) and *project* our training data through this matrix with a matrix multiplication. You should be doing something along the lines of:

  ```
  projection_matrix = vh[slice indices]
  projected_train_data = centred_train_data @ np.transpose(projection_matrix)
  ```

  The slicing into vh selects our top significant eigenvectors, but why do we need the transpose? How can we tell that our dimensionality has been reduced?
- Now that you have projected your training data into the principle component space, we will project the testing data. To do this we need to mean-centre the test data using the mean vector from the training data (why?), and we need to use the same projection matrix from before (why?).
- Visualise your projected training and testing data with a scatter plot. Don't forget to use different markers for the two sets, and to colour the markers based on the labels.

☐ Challenge Task 3.5

Some questions to consider:

1. Why do we not compute a projection matrix and mean value for the testing sets?
2. Why does LDA give us nice distinct clusters for our Wine Dataset when PCA does not?
3. What benefit does dimensionality reduction provide? What are the drawbacks?
4. How could you use LDA to predict class labels?