

Mean Normalization

In machine learning we use large amounts of data to train our models. Some machine learning algorithms may require that the data is *normalized* in order to work correctly. The idea of normalization, also known as *feature scaling*, is to ensure that all the data is on a similar scale, *i.e.* that all the data takes on a similar range of values. For example, we might have a dataset that has values between 0 and 5,000. By normalizing the data we can make the range of values be between 0 and 1.

In this lab, you will be performing a different kind of feature scaling known as *mean normalization*. Mean normalization will scale the data, but instead of making the values be between 0 and 1, it will distribute the values evenly in some small interval around zero. For example, if we have a dataset that has values between 0 and 5,000, after mean normalization the range of values will be distributed in some small range around 0, for example between -3 to 3. Because the range of values are distributed evenly around zero, this guarantees that the average (mean) of all elements will be zero. Therefore, when you perform *mean normalization* your data will not only be scaled but it will also have an average of zero.

To Do:

You will start by importing NumPy and creating a rank 2 ndarray of random integers between 0 and 5,000 (inclusive) with 1000 rows and 20 columns. This array will simulate a dataset with a wide range of values. Fill in the code below

```
In [ ]: # import NumPy into Python

# Create a 1000 x 20 ndarray with random integers in the half-open interval [0,
5001).
X =

# print the shape of X
```

Now that you created the array we will mean normalize it. We will perform mean normalization using the following equation:

$$i = \frac{i - \mu_i}{\sigma_i}$$

where i is the i th column of X , μ_i is average of the values in the i th column of X , and σ_i is the standard deviation of the values in the i th column of X . In other words, mean normalization is performed by subtracting from each column of X the average of its values, and then by dividing by the standard deviation of its values. In the space below, you will first calculate the average and standard deviation of each column of X .

```
In [ ]: # Average of the values in each column of X
ave_cols =

# Standard Deviation of the values in each column of X
std_cols =
```

If you have done the above calculations correctly, then `ave_cols` and `std_cols`, should both be vectors with shape `(20,)` since `X` has 20 columns. You can verify this by filling the code below:

```
In [ ]: # Print the shape of ave_cols

# Print the shape of std_cols
```

You can now take advantage of Broadcasting to calculate the mean normalized version of `X` in just one line of code using the equation above. Fill in the code below

```
In [ ]: # Mean normalize X
X_norm =
```

If you have performed the mean normalization correctly, then the average of all the elements in `X` should be close to zero, and they should be evenly distributed in some small interval around zero. You can verify this by filling the code below:

```
In [ ]: # Print the average of all the values of X_norm

# Print the average of the minimum value in each column of X_norm

# Print the average of the maximum value in each column of X_norm
```

You should note that since `X` was created using random integers, the above values will vary.

Data Separation

After the data has been mean normalized, it is customary in machine learning to split our dataset into three sets:

1. A Training Set
2. A Cross Validation Set
3. A Test Set

The dataset is usually divided such that the Training Set contains 60% of the data, the Cross Validation Set contains 20% of the data, and the Test Set contains 20% of the data.

In this part of the lab you will separate `X_norm` into a Training Set, Cross Validation Set, and a Test Set. Each data set will contain rows of `X_norm` chosen at random, making sure that we don't pick the same row twice. This will guarantee that all the rows of `X_norm` are chosen and randomly distributed among the three new sets.

You will start by creating a rank 1 ndarray that contains a random permutation of the row indices of `X_norm`. You can do this by using the `np.random.permutation()` function. The `np.random.permutation(N)` function creates a random permutation of integers from 0 to `N - 1`. Let's see an example:

```
In [ ]: # We create a random permutation of integers 0 to 4  
np.random.permutation(5)
```

To Do

In the space below create a rank 1 ndarray that contains a random permutation of the row indices of `X_norm`. You can do this in one line of code by extracting the number of rows of `X_norm` using the `shape` attribute and then passing it to the `np.random.permutation()` function. Remember the `shape` attribute returns a tuple with two numbers in the form `(rows, columns)`.

```
In [ ]: # Create a rank 1 ndarray that contains a random permutation of the row indices  
of `X_norm`  
row_indices =
```

Now you can create the three datasets using the `row_indices` ndarray to select the rows that will go into each dataset. Remember that the Training Set contains 60% of the data, the Cross Validation Set contains 20% of the data, and the Test Set contains 20% of the data. Each set requires just one line of code to create. Fill in the code below

```
In [ ]: # Make any necessary calculations.  
# You can save your calculations into variables to use later.  
  
# Create a Training Set  
X_train =  
  
# Create a Cross Validation Set  
X_crossVal =  
  
# Create a Test Set  
X_test =
```

If you performed the above calculations correctly, then `X_train` should have 600 rows and 20 columns, `X_crossVal` should have 200 rows and 20 columns, and `X_test` should have 200 rows and 20 columns. You can verify this by filling the code below:

```
In [ ]: # Print the shape of X_train  
  
# Print the shape of X_crossVal  
  
# Print the shape of X_test
```

```
In [ ]:
```