Why Data Normalization is necessary for Machine Learning models

Urvashi Jaitley
Oct 8, 2018 · 4 min read ★

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. For machine learning, every dataset does not require normalization. It is required only when features have different ranges.

For example, consider a data set containing two features, age(x1), and income(x2). Where age ranges from 0–100, while income ranges from 0–20,000 and higher. Income is about 1,000 times larger than age and ranges from 20,000–500,000. So, these two features are in very different ranges. When we do further analysis, like multivariate linear regression, for example, the attributed income will intrinsically influence the result more due to its larger value. But this doesn't necessarily mean it is more important as a predictor.

To explain further let's build two deep neural network models: one without using normalized data and another one with normalized data and at the end, I will compare the results of these 2 models and will show the effect of normalization on the accuracy of the models.

Elevation	Aspect	Slope	Horizontal_D	Vertical_Dist	Horizontal_D	Hillshade_9a	Hillshade_No	Hillshade_3p	Horizontal_Distance_To_Fire_Points
2596	51	3	258	0	510	221	232	148	6279
2590	56	2	212	-6	390	220	235	151	6225
2804	139	9	268	65	3180	234	238	135	6121
2785	155	18	242	118	3090	238	238	122	6211
2595	45	2	153	-1	391	220	234	150	6172
2579	132	6	300	-15	67	230	237	140	6031
2606	45	7	270	5	633	222	225	138	6256
2605	49	4	234	7	573	222	230	144	6228
2617	45	9	240	56	666	223	221	133	6244
2612	59	10	247	11	636	228	219	124	6230
2612	201	4	180	51	735	218	243	161	6222
2886	151	11	371	26	5253	234	240	136	4051
2742	134	22	150	69	3215	248	224	92	6091
2609	214	7	150	46	771	213	247	170	6211
2503	157	4	67	4	674	224	240	151	5600
2495	51	7	42	2	752	224	225	137	5576
2610	259	1	120	-1	607	216	239	161	6096
2517	72	7	85	6	595	228	227	133	5607

2504	0	4	95	5	691	214	232	156	5572

First Few Rows Of Original Data

Below is a Neural Network Model built using original unnormalized data:

```
'''Using covertype dataset from kaggle to predict forest cover
type'''

#Import pandas, tensorflow and keras

import pandas as pd
from sklearn.cross_validation import train_test_split
import tensorflow as tf
from tensorflow.python.data import Dataset
import keras
from keras.utils import to_categorical
from keras import models
from keras import layers

#Read the data from csv file

df = pd.read_csv('covtype.csv')

#Select predictors
x = df[df.columns[:54]]

#Target variable

y = df.Cover_Type

#Split data into train and test

x_train, x_test, y_train, y_test = train_test_split(x, y ,
train_size = 0.7, random_state =  90)

'''As y variable is multi class categorical variable, hence using
softmax as activation function and sparse-categorical cross entropy
as loss function.'''

model = keras.Sequential([
 keras.layers.Dense(64, activation=tf.nn.relu,
 input_shape=(x_train.shape[1],)),
 keras.layers.Dense(64, activation=tf.nn.relu),
 keras.layers.Dense(8, activation=  'softmax')
 ])

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history1 = model.fit(
 x_train, y_train,
```

```
 epochs= 26, batch_size = 60,
 validation_data = (x_test, y_test))
```

```
Output:
Epoch 1/26 406708/406708 [==============================] — 19s
47us/step — loss: 8.2614 — acc: 0.4874 — val_loss: 8.2531 — val_acc:
0.4880

Epoch 2/26 406708/406708 [==============================] — 18s
45us/step — loss: 8.2614 — acc: 0.4874 — val_loss: 8.2531 — val_acc:
0.4880

….............

Epoch 26/26 406708/406708 [==============================] — 17s
42us/step — loss: 8.2614 — acc: 0.4874 — val_loss: 8.2531 — val_acc:
0.4880
```

Validation accuracy of the above model is just 48.80%.

Now lets first normalize the data and then build a deep neural network model. There are different methods to normalize data. I will be normalizing features by removing the mean and scaling it to unit variance.

```
from sklearn import preprocessing

df = pd.read_csv('covtype.csv')

x = df[df.columns[:55]]
y = df.Cover_Type

x_train, x_test, y_train, y_test = train_test_split(x, y ,
train_size = 0.7, random_state =  90)

#Select numerical columns which needs to be normalized

train_norm = x_train[x_train.columns[0:10]]
test_norm = x_test[x_test.columns[0:10]]

# Normalize Training Data

std_scale = preprocessing.StandardScaler().fit(train_norm)
x_train_norm = std_scale.transform(train_norm)

#Converting numpy array to dataframe
training_norm_col = pd.DataFrame(x_train_norm,
index=train_norm.index, columns=train_norm.columns)
x_train.update(training_norm_col)
print (x_train.head())
```

```
# Normalize Testing Data by using mean and SD of training set

x_test_norm = std_scale.transform(test_norm)
testing_norm_col = pd.DataFrame(x_test_norm, index=test_norm.index,
columns=test_norm.columns)
x_test.update(testing_norm_col)
print (x_train.head())
```

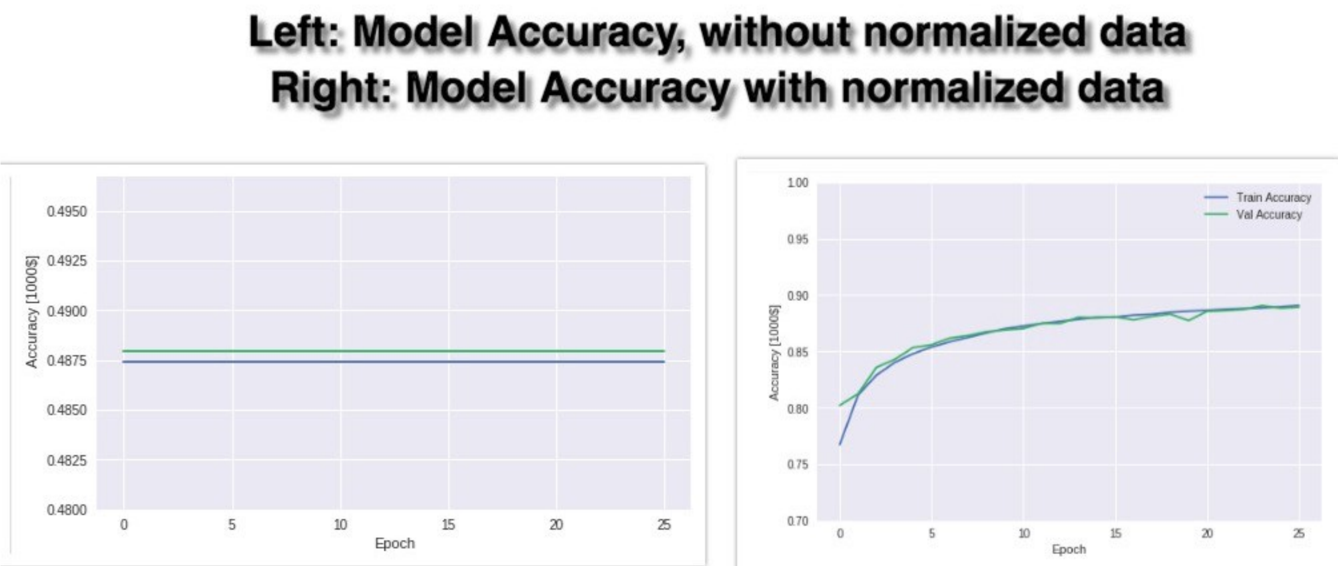
```
        Elevation    Aspect     Slope  Horizontal_Distance_To_Hydrology  \
152044   0.222366 -0.228639 -0.412503                          0.148486
363373   1.980490 -0.469989  0.255453                          3.018822
372733  -1.081933  0.271939  0.389044                         -0.867895
572846  -1.164122 -0.157128 -0.278912                         -1.267860
114145  -0.052787  0.861906  0.255453                         -0.279711

        Vertical_Distance_To_Hydrology  Horizontal_Distance_To_Roadways  \
152044                        0.149095                         1.336119
363373                        4.443372                         0.168073
372733                       -0.160093                        -0.241801
572846                       -0.795646                        -0.461170
114145                       -0.125739                         1.811419

        Hillshade_9am  Hillshade_Noon  Hillshade_3pm  \
152044       1.002687        0.539776      -0.510339
363373       1.227001       -0.270132      -1.190275
372733       0.292357        1.349684       0.378807
572846       0.965301        0.641014      -0.431885
114145      -1.090917        1.299065       1.581770

        Horizontal_Distance_To_Fire_Points     ...      Soil_Type32  \
152044                           -0.111226     ...                0
363373                           -0.703030     ...                0
372733                            0.038235     ...                0
572846                           -1.450334     ...                0
```

Output: Data after normalization

```
#Build neural network model with normalized data

model = keras.Sequential([
 keras.layers.Dense(64, activation=tf.nn.relu,
 input_shape=(x_train.shape[1],)),
 keras.layers.Dense(64, activation=tf.nn.relu),
 keras.layers.Dense(8, activation=  'softmax')
 ])

model.compile(optimizer=tf.train.AdamOptimizer(),
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])

history2 = model.fit(
 x_train, y_train,
 epochs= 26, batch_size = 60,
 validation_data = (x_test, y_test))
```

```
#Output:
Train on 464809 samples, validate on 116203 samples
Epoch 1/26 464809/464809 [==============================] - 16s
34us/step - loss: 0.5433 - acc: 0.7675 - val_loss: 0.4701 - val_acc:
0.8022
Epoch 2/26 464809/464809 [==============================] - 16s
34us/step - loss: 0.4436 - acc: 0.8113 - val_loss: 0.4410 - val_acc:
0.8124 Epoch 3/26
....................
Epoch 26/26 464809/464809 [==============================] - 16s
34us/step - loss: 0.2703 - acc: 0.8907 - val_loss: 0.2773 - val_acc:
0.8893
```

Validation accuracy of the model is 88.93%, which is pretty good.



The accuracy plot of the above 2 models

From the above graphs, we see that model 1(left side graph) have very low validation accuracy (48%) and a straight line for accuracy is coming in a graph for both test and train data. Straight line for accuracy means that accuracy is not changing with the number of epochs and even at epoch 26 accuracy remains the same (what it was at an epoch 1). The reason for straight accuracy line and low accuracy is that the **model is not able to learn** in 26 epochs. Because different features do not have similar ranges of values and hence **gradients may end up taking a long time and can oscillate back and forth and take a long time before it can finally find its way to the global/local minimum. To overcome the model learning problem, we normalize the data. We make sure that the different features take on similar ranges of values so that gradient descents can converge more quickly**. From the above right-hand side graph, we can see that after normalizing the data in model 2 accuracy is increasing with every epoch and at epoch 26, accuracy reached 88.93%.

Thanks. Happy Learning :)

Machine Learning Deep Learning Towards Data Science Data Processing Sklearn