Lab 9

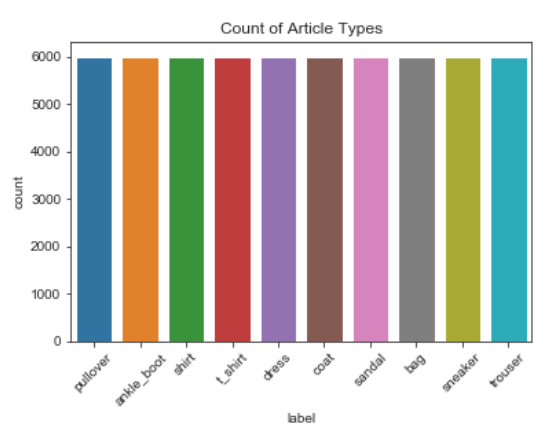
# Hypothesis:

Through the OSEMIN method, we can determine whether one machine learning algorithm and compute methodology can be employed to accurately classify 10 types of clothing from the fashion MNIST data set.

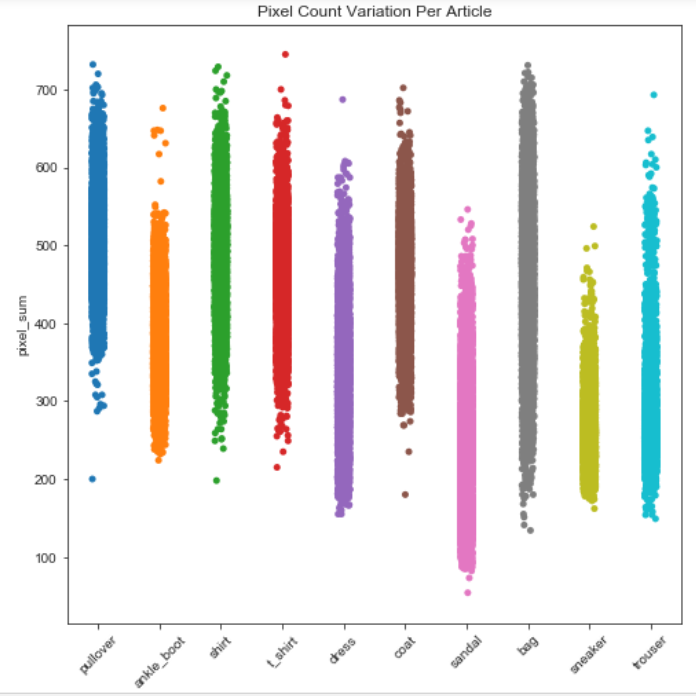
# Methodology:

## Visualization:

The fashion MNIST data set was available as a training and test set in \*.csv files through www.kaggle.com. Using Keras on top of TensorFlow in Python 3.6, the fashion MNIST data set was imported. Once imported, name labels were added to the label category and the data was examined to view the distribution:



Additionally, each category was examined to see if there was a pattern between articles of clothing and the number of pixels each article generally covered:



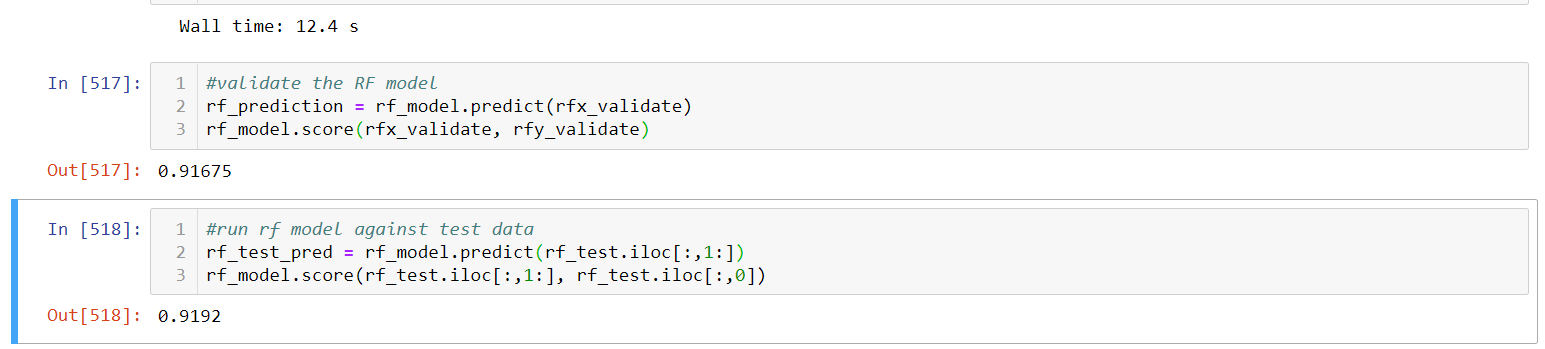
To assist the random forest algorithm in identifying the articles of clothing, a library was constructed that included the range of pixels each piece of clothing covered. According to the fashion MNIST Github site, the following labels match with each article:

|  |  |
| --- | --- |
| **Article** |  |
| t-shirt |
| Trouser |
| Pullover |
| Dress |
| Coat |
| Sandal |
| Shirt |
| Sneaker |
| Bag |
| Ankle boot |

## Modeling:

Two models were used to attempt to find the best fit for identifying clothing from the fashion MNIST data set; random forest (RF) and a convolution neural network (CNN). These two models were chosen based on their well-known accuracy when predicting the original MNIST data set. In addition to their known performance, I really wanted to try building a successful version of both models.

To set up the random forest model, the training data was broken into training and validation sets that included the minimum and maximum pixels per article of clothing. Using this additional data while tuning the models maximum leaf nodes to 100 and number of estimators to 10 helped to raise the model’s training validation accuracy to about 91.6% and test accuracy to about 92%. Without the additional features added, the model predicted with about 10% less accuracy. This model has a lot of variables to tune to figure out the best fitting model. However, it was the better fitting model for this data set according to the prediction outcome. This model took 12.4 seconds to run as tuned.



For the convolution neural network model, the training data was again broken into training and validation sets that only included the pixels and labels. The model was tuned by including a learning rate of 0.01. Tuning this model proved to be a bit more complex and only provided an output of between 89% and 91% accuracy for both the training and test sets. This model incorporates lots of very specific library functions. There are less tuning parameters but it’s more difficult to understand how to get the model set up. This model took about 10 minutes for the initial TensorFlow setup. However, after the initial run, the model only took 40.1 seconds to run.

