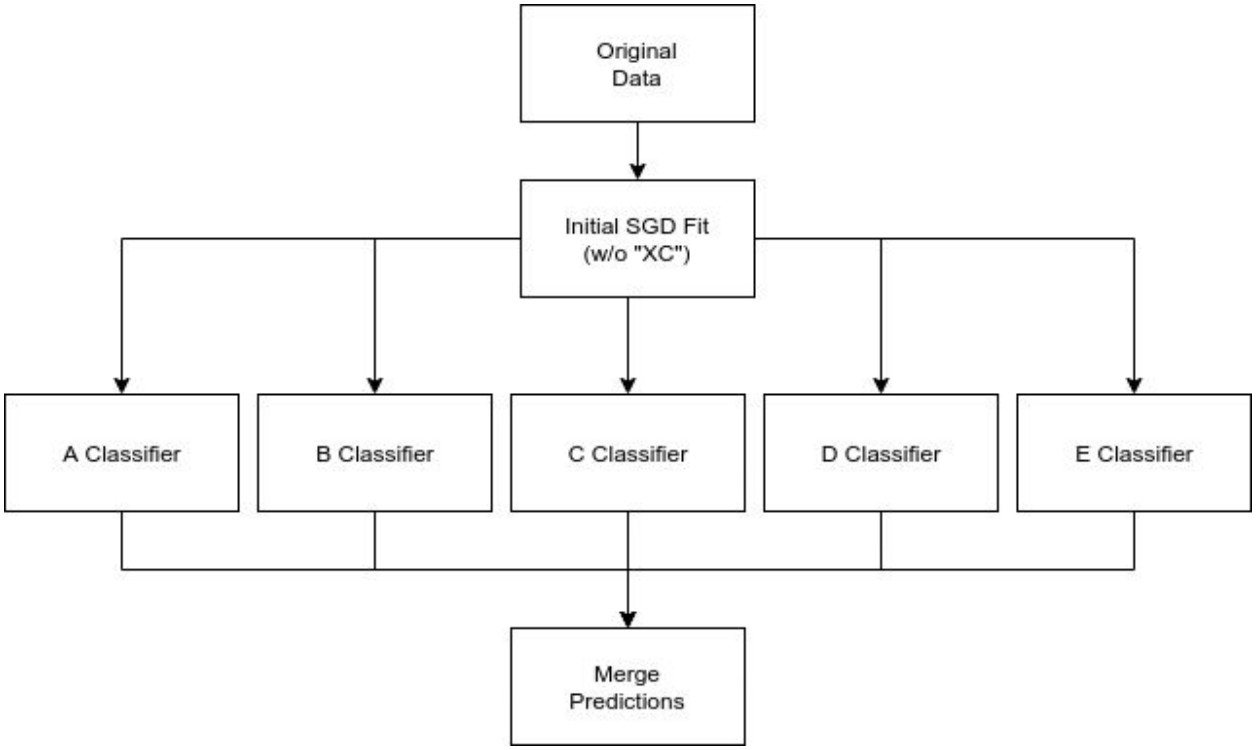
The process of my learning algorithm went in a few stages. At first, I was confused by the last column of the dataset, the "XC" column since it wasn't a numerical value. And although I briefly considered applying a numerical value to the column (0.2 for A, 0.4 for B and so on), I wasn't confident in applying a numerical relationship between the letters for data I didn't know about. For example, what if A and E were similar classes? Without knowing anything about the data, I decided to train the model by completely ignoring the "XC" column.

I was surprised when in the first few runs of my model, splitting up the training set into a train and test set, I was able to get approximately 85% accuracy on the test set splits. However, I didn't want to completely ignore the "XC" column, and came up with this solution: to split the data into 5 separate sets, one for each value in the "XC" column. It seemed promising since as I skimmed the training set, the labels in the "XC" column were approximately even (~600 labels each) and had a variety of 0s and 1s as the y values. After training each set separately (after the initial training over the whole set), I was able to get approximately 95% accuracy, which is a notable increase just by incorporating one column of data. With this good rate, I decided to submit the model as it was.

The code attached was written in a Jupyter notebook, using the Pandas and Numpy packages as data structures, and the SGDClassifier method from sklearn. All ideas and code are my own, although some thinking may have been inspired from Andrew Ng's Machine Learning course from Coursera. The link is here: https://www.coursera.org/learn/machine-learning.

Tue Do
tuedo2@illinois.edu

Learning model



I have never come across someone using separate classifiers for one specific feature, but I'm sure I'm not the first to do so.

References

Géron, Aurélien. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow*.