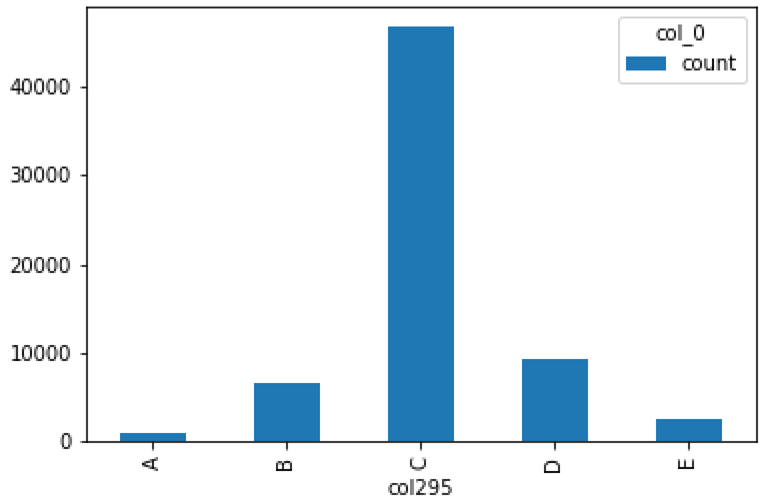
Project Solution : Darko Yovanoski

1. Make some initial data analysis.

Note: As we are not having header and for consistency of my understanding I just create header and applied (so all columns are 'col' + number of the column)

Based on the data provided it is classification task of 5 classes A, B, C , D ,E. As we have class imbalance where :

A	867
B	6602
C	46882
D	9279
E	2507

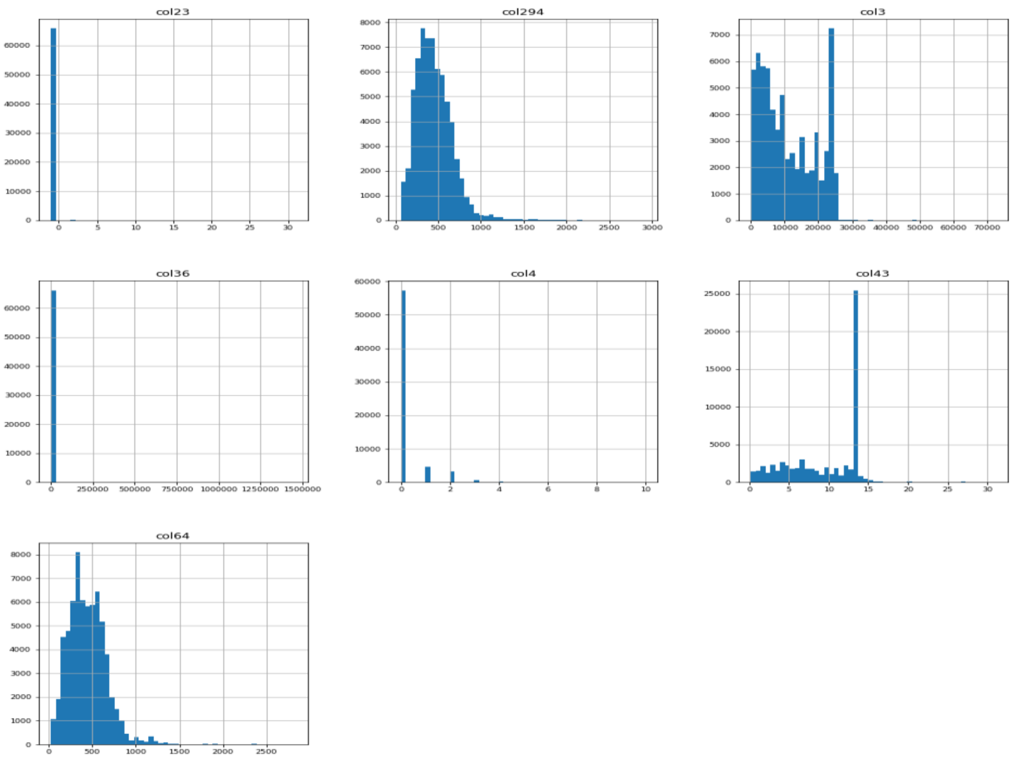


Based on the data I found that all columns was integer and float basically all integers as unique values was 0 and 1 so my opinion is this maybe were previously some categorical value so here in this database they are encoded as 0 and 1, also 9 columns have only one unique value. (As I mention before I think all of this column are created from some orginial categorical data columns , maybe im wrong). So down in the plot is the distribution of all float dtype columns .

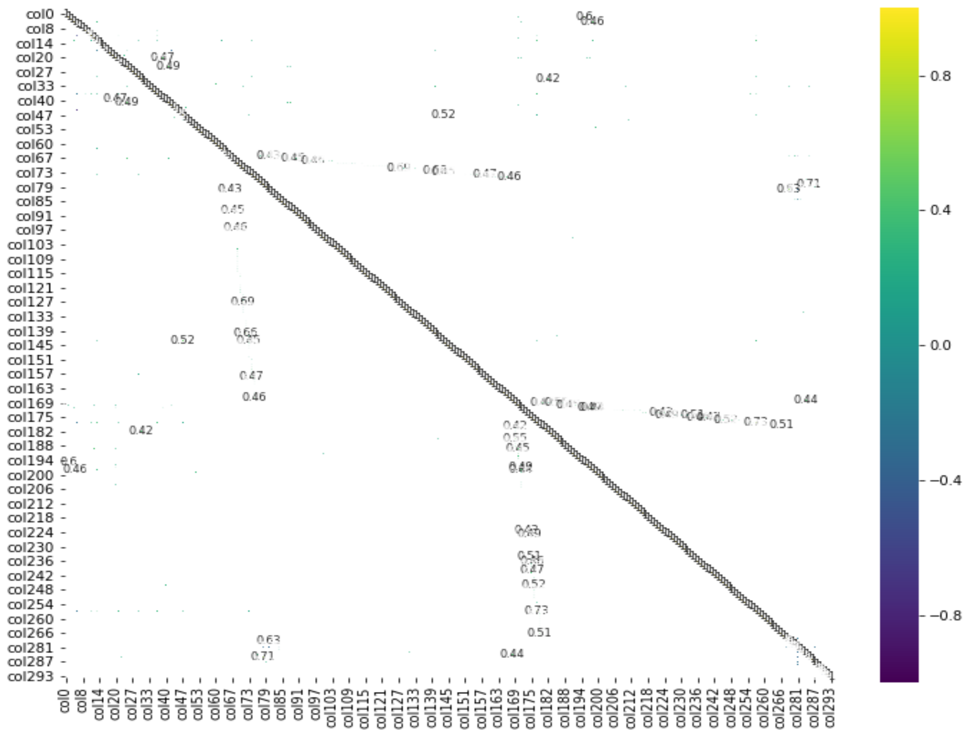
So having only the float columns i analyze correlation between them not with the target :

- There is 1 strongly correlated values with col3: col43 0.584448
- There is 0 strongly correlated values with col4
- There is 0 strongly correlated values with col23
- There is 0 strongly correlated values with col36
- There is 1 strongly correlated values with col43: col3 0.584448
- There is 1 strongly correlated values with col64: col294 0.748347

- There is 1 strongly correlated values with col294:0 col64   0.748347



Correlation between columns with 0 and 1 values:



2. Fit some ML model(s).

As we are having imbalanced dataset (plot above) I decided to use Deep Belief Nets – Restricted Boltzman Machines.

The code is written using Tensorflow Framework simple implementation of DBN using low level Tensorflow API.

Note about why I choose Restricted Boltzman Machines :

- Use unsupervised training to teach a single layer of hidden neurons to reflectively reproduce the raw data inputs without regard to the desired outputs. If there are fewer hidden neurons than inputs, this forces the layer to learn patterns rather than rote memorization to spit back.
- For each training case, use the activations of the trained hidden layer as inputs to another hidden layer. Teach this second hidden layer to reflectively reproduce its inputs.
- A closely related property is that deep belief nets are shockingly robust against overfitting

The Model can be executed running train_validate.py (I have default value defined so it can be runned as my hyperparameters are defined).

This solution is based on Tensorflow version 1.10 basically im using docker image from nvidia cloud : nvcr.io/nvidia/tensorflow:18.09-py3 and also please pip3 install requriments.txt as some modules were not included so they are added.

3. Show with some X-validation the power of your model and comment the results.

Based on my trained model I succeed to achieve accuracy around 0.71, which for sure it is not even closer to good I think I need some data engineering but due to time constrained I can not perform many tests.

4. Present us the results and the steps you have taken.

The Code Implementation is done in 4 scripts as advised using Classes and functions :

Note: I try writing all the class and method to be self explanatory, I hope It will make sense when you reading the code.

1. rbn.py - Implementation of RBN Layer where is defined pretrain method of Restricted Boltzman Machines
2. dbn.py - Implementation of DeepBeliefNets (defining the graph, loss function , defining the layer – rbn_layer and dense_layer using not builded layer class from Tensorflow I derived manual implementation of dense layer with initializing the weights and biases , applying nonlinear function.
3. prepare_dataset.py - class Dataset for preprocessing of the dataset, defining the number of inputs and output size, reading the csv file , applying normalization of the numerical values to be in range 0 to 1 (required by DBN) , converting to one hot encoding the labels
4. trainer.py - here is applied whole pipeline build on the hyperparameters and csv file, buidling the graph, loop through the dataset, training the optimizer and also calculating the validation

accuracy and loss. As we have small data which can be fitted in to memory I did not use batches on the validation data.

5. train_validate.py - Tensorflow app for running the model where can be defined all the hyperparameters used by the model, saving directory from the graph .