1. Partitioning the Data

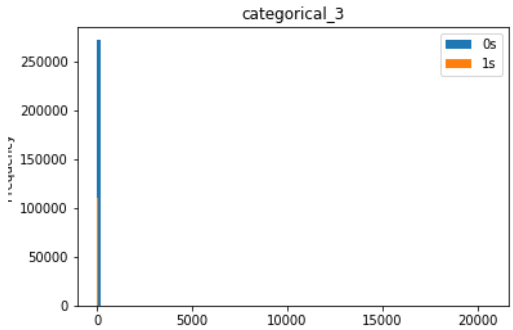
The training file contained 45 840 618 rows of data. This was much too large to read directly into memory. The solution was to find 2 Million indices which we wanted to extract. This was done by taking a 2 Million random numbers between 0 and 45 840 617 (0 indexed). These were the 2 million random integers that would be used throughout the assignment.

These 2 Million random integers were further randomly divided into 3 sets. 1 Million for training, 750k for testing, and 250k for validation. First, 1 Million integers within the 2 Million were randomly selected. Then 250k random numbers from the remaining 1 Million were selected for the validation set. The remaining 750k were taken to be the testing set. These three sets of indices were then saved to csv files for future use.

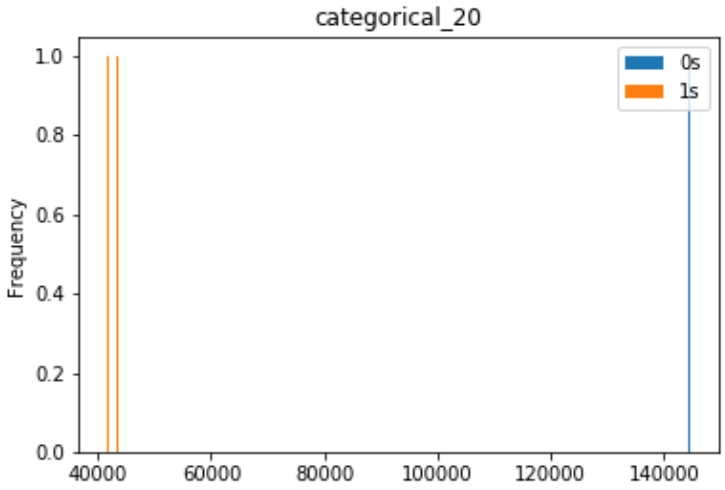
For each of the three sets of indices, a data set was produced from the original training file. For the training indices, this was done by reading in 4 million rows of data from the training file at a time and copying the ones that were present in training index set into a separate pandas dataframe object to keep track of them. The 4 million rows were then discarded and another 4 million were loaded. This was done until the end of the training file was reached. The training indeces partition was called train1M (this data set was saved to a csv file in order to not have to complete this long operation all the time although it is possible to do all the data partitioning in 30 mins on a relatively powerful computer without the need to save, which is done in the assignment submission). The same procedure was done for the validation set (called validation250k) and for the testing set (called test750k). These three datasets were returned as arrays when they were all computed.

2. Summary Statistics

Histograms were computed for each of the features. All of the integer features were kept and a list of categorical features that should be kept was generated based on the histograms generated. Generally speaking, any histogram that looked like the one here were not kept as they would not enable good separation of data (the orange and blue lines representing different labels, can't be well separated).

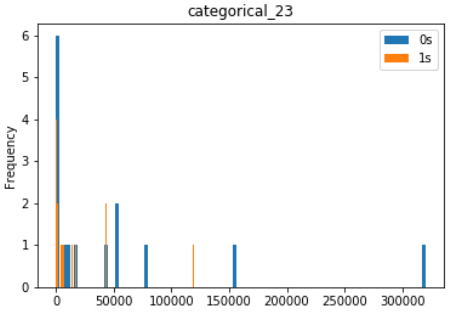


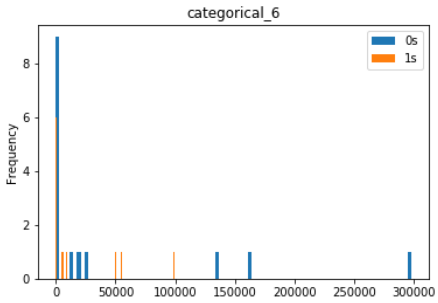
This is because, even with the data separated into 100 bins, the histogram shows only 1 bin that is highly utilized by both labels and so the separation of data on categorical_3 would not be as good as categorical_20.



Categorical_20 is a good feature to separate data on as it will create 3 bins that are utilized. Two of them are utilized for label 1 and the third is utilized for label 0. Categorical_20 is better than categorical_3, however it is not a very good category either as it has a very low frequency count on any given value. A total of 3 items were found to have a categorical_20 value.

A much better feature to keep would be one that has both high count and better separation between the labels. For example, categorical_23 or categorical_6:





Both these histograms can be considered better than categorical_20 as they have more separation of data (the labels can be more clearly separated) and a greater count of values overall.

For the Integer features several statistics were measured (count, mean, standard deviation, min, 25th, 50th and 75th percentiles as well as max). The table below is a resume of those statistics:

	integer_1	integer_2	integer_3	integer_4	integer_5	integer_6	integer_7	integer_8	integer_9	integer_10	integer_11	integer_12	integer_13
count	547351.000000	1000000.000000	785154.000000	782791.000000	9.745330e+05	777146.000000	956983.000000	999512.000000	956983.000000	547351.000000	956983.000000	235572.000000	782791.000000
mean	3.506584	106.377807	27.065321	7.315527	1.864183e+04	115.039034	16.442098	12.512986	106.179319	0.616889	2.732943	0.988131	8.199752
std	9.752981	392.962915	402.800192	8.781313	7.002784e+04	315.677901	77.506505	16.608903	220.433666	0.683757	5.195889	4.929478	15.480945
min	0.000000	-2.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	2.000000	2.000000	3.250000e+02	8.000000	1.000000	2.000000	10.000000	0.000000	1.000000	0.000000	2.000000
50%	1.000000	3.000000	6.000000	4.000000	2.812000e+03	32.000000	3.000000	7.000000	38.000000	1.000000	1.000000	0.000000	4.000000
75%	3.000000	35.000000	18.000000	10.000000	1.011600e+04	102.000000	11.000000	19.000000	110.000000	1.000000	3.000000	1.000000	10.000000
max	2344.000000	27673.000000	65535.000000	683.000000	2.513122e+06	45918.000000	34536.000000	4985.000000	17556.000000	8.000000	148.000000	293.000000	4375.000000

3. Normalization and Feature Engineering

When the statistics and histograms were created, the normalization and feature engineering part could be performed. The first pre-processing step was to remove any empty fields by filling them with 0 (if integer) or 'Dummy' if non-integer. The -1 integers were also replaced with 0s. Before any categorical feature selection, the integer features were normalized (mean-centered and unit variance). This was done using sklearn's preprocessing standard scaler fitted to the training data's integer columns. The scaler was then dumped to an object file for future retrieval. When it was time to scale the training or testing integers (preparing the data), the standard scaler was loaded from file and used to transform the integer data. The scaler was saved to file in order to not create a global variable.

Then, the features that were to be removed were identified. As shown in the previous section, the features whose histograms were single values were removed. This was because they did not change regardless of label and other features. This made the dropping of the following categorical features justified: "categorical_1", "categorical_3", "categorical_4", "categorical_5", "categorical_8", "categorical_10", "categorical_12", "categorical_16", "categorical_19", "categorical_21", "categorical_24" and "categorical_26". The other categorical data columns were kept because they gave a histogram which clearly separated the data set.

When the features were dropped, a one-hot encoding was performed on all the remaining categorical features. This one hot encoding was done on the 30 categories within a feature that were most present in the feature (greatest count), 30 was based on the histograms of the data which were analyzed. Ideally, the mean count would have been taken and only those features with more hits than the mean would have been used for one hot encoding. However, due to resource (both memory and processing time) constraints, 30 was chosen as the best number of features to take overall. The on hot encoding was done using pandas get_dummies() function on the DataFrame which applies one hot encoding based on a given category array. The top-30 category array was computed for each feature when the training data was loaded. The operations were done using sparse DataFrame for both computational and memory efficiency. The sparse DataFrame was returned as a sparse matrix after all the one hot encodings were done. Returning a sparse matrix was also a design choice with respect to the end goal of this feature selection. In general, machine learning algorithms are more easily trained on sparse matrices and so giving the categorical features as a sparse matrix (and combining the integers as a sparse matrix in some future pre-processing) would allow the machine learning algorithm, whichever one is chosen, to work more efficiently than if the data was dense.