Fraudulent Credit & Efficient or Deficient Energy

Christopher Lee

I. Introduction

While in this course, many different types of machine learning algorithms and methods have been covered both in regards to regression and classification type problems. This paper will cover how I have chosen to demonstrate what I have learned using three different models for each type of machine learning problem using ULB's credit card fraud detection data set and Data Science Dojo's Energy Efficiency Data set. With the credit card fraud detection data set I use Random Forests, a SVC, and a Dense Neural Network, while attempting to overcome the fact the data set is imbalanced. Then with the energy efficiency data set I use Linear Regression, Polynomial Regression and Ridge Regression. Both have their successes and failures to speak of, but over all acceptable goals were reached such as near 100% accuracy albeit with an f1 score as high as 85%, and mean squared error as low as 1.02 with an mean absolute error of 0.66 for heating load, but also at the same time with an mean squared error as high as 7.70, with and mean absolute error of 1.95 for cooling load.

II. Classification: Detecting Credit Card Fraud

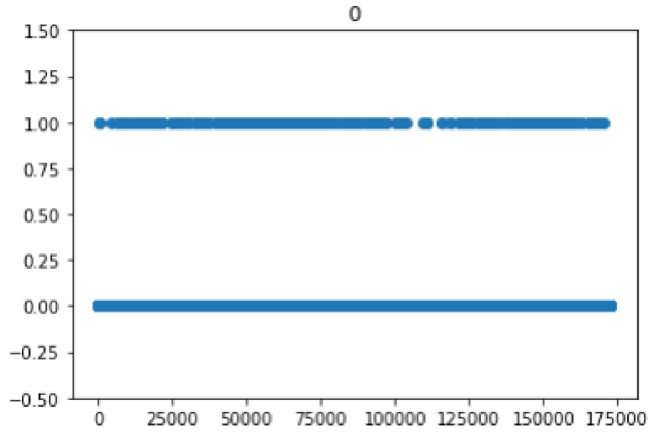
A. The Data set

For the classification set, we have the ”Credit Card Fraud Detection” data set, which was collected and analyzed by a research collaboration between Worldline and the Machine Learning Group of ULB (Université Libre de Bruxelle). This an extremely large data set bolstering 284,807 samples with 31 features. The 31st column is ”Class” which is meant to be contain either ”1” or ”0” value wise and is meant to dictate whether the sample is a fraud or not. Unfortunately the data set is heavily imbalanced having only 492 fraudulent samples of the 284,315 samples meaning only 0.1727% of the samples are actual fraud. This would result in a very inaccurate and imprecise as unless splitting is done in a ratio of at least 10,000:17 would guarantee the training set having at least 1 fraudulent sample. So the problem we face is the imbalanced data set, which I attempt to solve using the imbalanced-learn library.

B. Data Exploration

The first thing to do is to observe the features seeing is there are thirty not including the ”Class” column. By observing the features via scatter plotting with ”Class” as the other axis, I hoped to find a feature that seemed to not be as important or irrelevant. I found one such data feature, ”Time”. (Below) As anyone could see there is

Fig. 1. Time vs Class



only a minimal gap in time of times for when a fraud has been recorded to have occurred. Those gaps however are not dependable as they are not the min and max of the time of when a fraudulent credit card usage can occur. With that being the only difference between the two I sought to then remove it and leaving me with twenty-nine remaining features to train with.

C. Data Pre-Processing

The Data set being as unbalanced as it was clearly needed to be either over sampled or under sampled and having experience in neither I attempted both. I started with using train_test_split() and divided the data set using a 7:3 ratio meaning the 70% of the samples became the training set and 30% of them became the data set. Then I initialized and refit the training data sets using the RandomOverSampler and RandomUnderSampler methods of imbalanced-learn and also the fit_resample() command on both to create over sampled data sets and under sampled data sets. Afterwards I then split both types of data sets again using train_test_split() with an 8:2 ratio to create validation and smaller training sets for training the neural network later on, while also being care to not overwrite the original training sets as not everything needs a validation set.. Finally after all those steps i initialised a StandardScaler object for both and scaled the respective full, smaller, and validation training sets as well as the scaled a training set for both over sampled and under sampled data sets. In the end, for both types of sampling, the testing sets had the same shape of 85,443 samples by 29 features, however, training sets held the major differences. The over sampled set's smaller training and validation sets had 318,436 and 79,610 samples respectively, meaning together the full training set had 398046 samples. However, the under sampled set's smaller and validation sets had 545 and 137 samples respectively, which makes the whole training set altogether have 682 samples. This large difference does not seem like much, but with the testing set sizes being the same, I encountered a large problem that I will go over during the analysis portion of this section

D. *Training and Validation*

I decided to use a **S**upport **V**ector **C**lassifier, Random Forest, and a Dense Neural Network. The SVC was easy and simple to train, but still took a considerable amount of time considering I was training two models via an over sampled and under sampled training set. The Random Forest was a bit more complicated as I used scikit_learn's *GirdSearchCV* command. the parameters I chose to start training with were the same and at first were just criterion and max_leaf_nodes. As i went through I found issues that helped add or remove parameters and I ended with having criterion, n_estimators, and max_features as the parameters to look out for with the over sampled training set and additionally max_leaf_nodes for the under sampled training set. I did not set a random seed before so I had ended up repeatedly running the grid search commands until I found the best result out of 5 runs in regards to recall and precision. Moreover, as a side note I did limit my over sampled training set to 4,000 samples for the sake of time as i felt 4,000 samples would still give a good basis to judge for the best parameters. In the end the best parameters for the over sampled training set were 'gini', 'sqrt', and 100 for the criterion, max_features, and n_estimators. For the under sampled training set, the best parameters were 'gini', 'sqrt', 18, and 100 for criterion, max_features, max_leaf_nodes, and n_estimators. Finally, there is the last model, the neural network. For both models I started with a 5-layer network with an input_dim of 29, 15 neurons each layer, and relu as the activation function for all the layers except the output which was sigmoid. For the over sampled training set this seemed to be enough, albeit it mainly due to the sheer size of the sample set being around roughly 400,000 samples requiring a large batch size and epoch count in order to avoid quality degradation as a low batch size and epoch count wouldn't be particular efficient, low batch size and high epoch count would likely lead to overfitting, and high batch size and low epoch count likely to underfitting. As such I chose a batch size of 600 and epoch count of 530 resulting in 530 batches of 531 samples. As for under sampled training set that was more difficult as set of 682 samples could not handle a high batch size and epoch count and for reasons described before high over low of either in either way were the worse decisions so I settled on a batch size of 5 and epoch count of 30, i.e., 30 batches of 109 samples. As one would guess recall was simple to get to a high percentage but precision not so much when the training set was roughly 550 samples- approximately 130 were used as validation - to predict roughly 85,000 samples. In the end, I used a couple of methods and raised the precision to above 5% by adding 14 more neurons to each layer, adding another 4th hidden layer, changing the activation for hidden layers to softmax, and added a kernel for weight/bias to every even hidden layer. This resulted in the under sampled neural network model being different from the over sampled model by having 6 layers, 29 neurons in each layer save the output, two bias/weight neurons added, and the hidden layers working on softmax activation.

E. *Model Comparison*

Of the three models, the best was objectively the Random Forest in both cases of over sampled and under sampled sets. The random forest models bolstered very high accuracy in both cases and the best f1 scores among the three.

model	accuracy(%)	precision (%)	recall(%)	f1(%)
SVC	99.94	85.71	80.77	83.17
RF	99.95	94.70	78.13	85.62
DNN	99.90	69.09	75.50	72.15

TABLE I
OVER SAMPLED RESULTS

model	accuracy(%)	precision (%)	recall(%)	f1(%)
SVC	97.70	6.83	91.67	12.71
RF	98.51	7.87	89.38	14.48
DNN	96.40	4.22	89.40	8.07

TABLE II
UNDER SAMPLED RESULTS

As we can tell for both accuracy as one can tell does not mean much, but when looking at the precision, recall, and f1 scores we see the true and major differences in quality between models with the DNN models being the worst in both cases of sampling while the Random Forests have the best with approximately 86% and approximately 14.5%.

F. Analysis

There are a good number of topics to analyze from the models. The first thing to take note of is of course the idea of over sampling vs under sampling. From the results one can say that under sampling was not best choice at all and that over sampling was the best. While objectively true in this case, I will admit it was more lack of samples than the failure of under sampling. Under sampling can work or else it would never have been recommended, but in this case of my splitting the data sets into training and testing sets first rather than later, under sampling is simply not viable. Maybe it could have been if the under sampler had been used before splitting the data set into training and data sets, but with the limited time I do not have the time to truly test it. This fact I believe is best shown by the lack of precision in each of the models' results. for each model the recall and accuracy is strong or at the very least decent but in all three models the precision score is in the single digits meaning a high number of false negatives, yet low number of false positives. This fact is shown by all three when looking at the confusion matrix values of all three.

model	confusion matrix	
SVC	83335	1952
	13	143
RF	83611	1672
	17	143
DNN	82231	3061
	16	135

TABLE III
UNDER SAMPLED MATRICES

Regardless, all the models were still very accurate, only some were not only accurate but very good at what they were designed to do. Random Forests may have come out on top, but that was also due to the fact I used scikit_learn's *GridSearchCV* to find the best parameters to use when initializing the model, which I did not do with SVC and could not reasonably do with neural networks without a significant amount of time.

III. Regression: Decline or Incline in Energy Efficiency

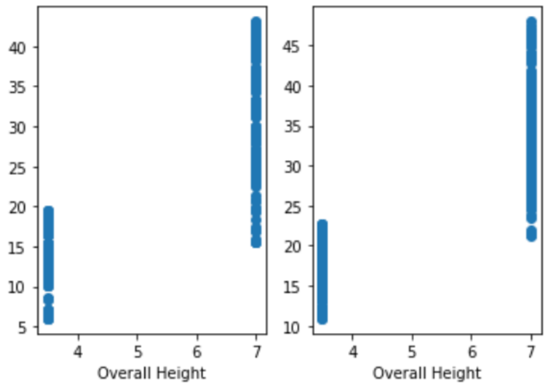
A. The Data set

For the Regression data set, there is the Energy Efficiency Data Set, which has 10 columns of data, and 768 samples. The data set was for observing the heating load and cooling load requirements for different buildings, 12 to be exact. So, the two target data values to predict are heating and cooling loads which are 2 of the 10 columns of data leaving 8 features to train with over 768 samples of data. This however will likely lead to an issue as 12 different buildings for over 768 is bound to lack some variety meaning regularization will be needed.

B. Data Exploration

To begin observing and exploring the data, I did as before and graphed the different features over the target values which instead of "0" and "1" are two different sets of values and as such 2 different graphs are made to observe the data. Additionally since non of the values are simply 0 or 1, I also graphed the features using the heating or cooling values as the y axis like in the classification set and the features to the x axis. From the graphing I can confirm that not much variance exists in some features and in others it seams to be decent at most as most samples are limited to similar values.

Fig. 2. Case: feature with Low Variance



Moreover, I also notice and acknowledge that orientation by using a graph appears to be irrelevant. There are a total of 4 orientations and for each one there seems to be a similar if not exact same min and max in regards to both.

Fig. 3. Case: feature with Decent But Not High Variance

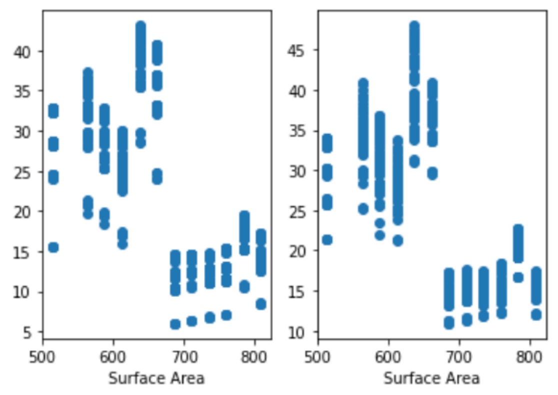
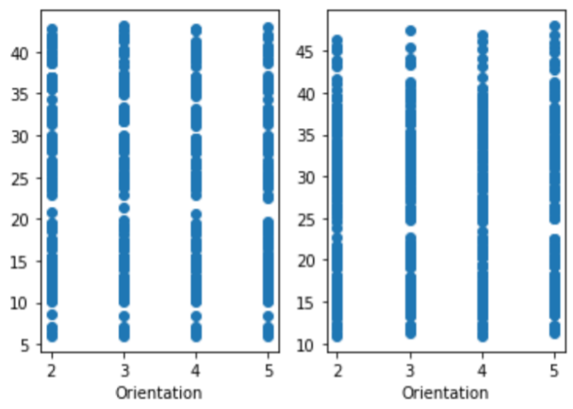


Fig. 4. Orientation vs Heating Load and Cooling Load



This entire feature does not seem to be relevant, but in lieu of there being only 8 determining features I kept them in just in case they could be helpful or that the combination of those samples and their other values may make the data have more variance than the feature graph dictates.

C. Data Pre-Processing

So for my models I chose simple Linear Regression, Ridge Regression, and Polynomial Regression. I decided to separate the data set using a 8:2 ration. This resulted in a training set containing 614 samples and a test set with 154 samples. Then, I pre-processed all the data by scaling it using the scikit_learn's *StandardScaler*'s *fit_transform* function for the training features and *transform* function for the testing features. Furthermore the only other notable pre-processing, I did that could be counted as such is initializing alpha an alphas for two of my Ridge Regression models.

D. Linear Regression

The Linear model was obviously the most simple to implement and as such it was the most easy to create, test, and run. After training both the scores are as follows:

Target Model	MSE	MAE	R2(%)
Heating Load	2.94	2.12	91.48
Cooling Load	3.25	2.30	88.52

TABLE IV
LINEAR REGRESSION SCORING

From the results we see that the linear regression model came out excellent for the heating load given the R2 score of 91.5% not to say the cooling load's model did not work just as well albeit a little lower by about 3%. Regardless the mean squared error, or mean squared error, for the heating and cooling load were low and so were their mean absolute error, or mean absolute error. Both models have their mean squared error and mean absolute error sit comfortably at approximately 3 for mean squared error and approximately 2 for mean absolute error. More on the significance later in analysis.

E. Ridge Regression

Like Linear Regression, Ridge Regression straight forward. I fit models for heating and cooling loads like with Linear Regression, but I was unsure about which type or Ridge Regression I should work with and so I decide on using both a standard normal Ridge Regression model and one with cross validation implemented. With the normal Ridge model, I started with an alpha of 0.1 and kept decreasing by a factor of 10^{-x} until I was satisfied, which ended up being simply 0.01. For the Ridge Regression with cross validation I was able to put in multiple alphas and so I chose to input an array of eight integers of which were the powers of ten from -3 to 5 excluding 4. At the moment of writing this I am not exactly sure why I had excluded 10^4, but regardless I doubt a change would have been noticed if I did on account of the model probably doing better with the alpha set lower. The resulting scores are as follows:

Target Model	MSE	MAE	R2(%)
Normal Ridge			
Heating Load	8.65	2.12	91.48
Cooling Load	10.55	2.30	88.52
RidgeCV			
Heating Load	8.68	2.10	91.45
Cooling Load	10.57	2.30	88.50

TABLE V
RIDGE REGRESSION SCORING

Unlike Linear Regression, the mean squared error is larger by a good margin of at least 6-7 in comparison to Linear Regression model scoring in both load cases, but the mean absolute error seems to have stayed at approximately 2.

F. Polynomial Regression

For the Polynomial Regression like previous cases, I made separate models for the heating load and cooling load, but like Ridge Regression I made 4 models in total to train. For Each load, I made a model set to a degree of 2 and one set to a degree of 3. The scoring is as follows:

Target Model	MSE	MAE	R2(%)
2-degree			
Heating Load	0.67	0.63	99.34
Cooling Load	3.92	1.31	96.74
3-degree			
Heating Load	0.61	0.56	99.40
Cooling Load	14.75	2.36	86.95

TABLE VI
POLYNOMIAL REGRESSION SCORING

Finally, with Polynomial Regression we see a significant change as the mean squared error and mean absolute error for both heating load models are below 1. Moreover, for cooling load while the 2-degree model still keeps an mean squared error of approximately 3 like the other models the mean absolute error is now 1.31 instead of being slightly above 2. Beyond that, however, we can still tell that the polynomial is by far the best model by the r2 scores focusing on the 2-degree of course as the 3-degree is when the model degrades from high degrees shown by cooling load's r2 score being 86.95% compared to the 2-degree model having one of 96.74.

G. Model Comparison

To Compare all the models let us recap all of the scoring for all the different models. The best model for

Target Model	MSE	MAE	R2(%)
Linear Regression			
Heating Load	2.94	2.12	91.48
Cooling Load	3.25	2.30	88.52
Normal Ridge			
Heating Load	8.65	2.12	91.48
Cooling Load	10.55	2.30	88.52
RidgeCV			
Heating Load	8.68	2.10	91.45
Cooling Load	10.57	2.30	88.50
2-degree			
Heating Load	0.67	0.63	99.34
Cooling Load	3.92	1.31	96.74
3-degree			
Heating Load	0.61	0.56	99.40
Cooling Load	14.75	2.36	86.95

TABLE VII
ALL SCORING

predicting heating load is the 3-degree polynomial and for cooling load, it is the 2-degree polynomial regression. In regards to heating load prediction, only the polynomial regression models have Mean Squared Error and Mean Absolute Error scores that are below 1. Between the two the better is clearly the 3-degree as while both share the same R2 score of approximately 99%, the 3-degree has a better mean squared error and mean absolute error, albeit by mere hundredths, but better nonetheless. Next in regards to cooling, the model with the best mean squared error is the 2-degree polynomial model sitting at 3.92 which way better than the 3-degree model's 14.75, and then the 2-degree polynomial model's mean absolute value of 1.31 is also better and while not be a large margin is still by 1.05 regardless.

H. Analysis

As mentioned in the comparison, the best model yielded low mean squared error and mean absolute error. In fact, to be exact the 2-degree polynomial model yielded 0.67 and 0.63 for the mean squared error and mean absolute error for the heating load and then 3.92 and 1.31 for the cooling load. The significance of these however are not explicitly easy to tell whether either are good. Objectively, regardless the lower the score or rating is always better, but in the case of cooling load, 3.92 mean squared error and 1.31 mean absolute error there might be a small issue. Moreover, since the r2 score is 99.34% and 99.74% for heating and cooling respectively

FUTURE WORKS

If I had more time I would do a number of things for both. For the classification, of the things to fix I would first redo the sampling so that I over and under sampled before splitting the data set into training and data sets to really see a difference between under and over sampling. Then afterwards I would add use *GridSearchCV* on the SVC and work more on the neural network for both and work on raising the f1 scores seeing as while the over sampled results were definitely better than their under sampled results the score was still at 72.15% meaning the model could use some work. As for the Regression, there are not many things I would want to do, but of the few things I would attempt to remake all the models after removing the orientation feature. Moreover, I would have tried to fix and alter the cooling model to make it work better and lower the mean squared error and absolute mean error on all the models to attempt to make the results on par with the heating load models.