arvato
FINANCIAL SOLUTIONS

Machine Learning Capstone

ARVATO FINANCIAL SERVICES CUSTOMER SEGMENTATION PROJECT

Jonathan Hasan | Machine Learning Engineer | 1/26/20

Definition

This project will start with defining the overview, problem statement and metrics. In this section of the report, we will be discussing details of the project, what needs to be solved and how it can be determined that the problem was solved.

PROJECT OVERVIEW

This project will attempt to use machine learning algorithms to generate a customer segmentation report which will allow us to get a birds eye view of which people have a propensity to become customers. Arvato Financial services is a subsidiary of Bertelsmann and had a dataset compiled from all the individuals who have become customers. A machine learning model needs to be made so that the right people can be targeted for a mail order campaign. In this project, four datasets are provided. These are azdias, customers, mailout_train and mailout_test. The azdias and customers dataset will be used for performing customer segmentation and clustering. The mailout_train set will be used for creating the unsupervised learning models. Finally, the mailout_test set will be used for validating the effectiveness of the models generated by the train set.

PROBLEM STATEMENT

The customers dataset and the azdias dataset need to be used to generate a customer segmentation report to decide which individuals are worth targeting. The analysis needs to cluster the azdias and customers into groups and see which clusters are worth putting more time into. Once this is done, a supervised learning model needs to be developed to successfully classify whether an individual becomes a customer based on a set of features.

METRICS

The machine learning model must be better than a random guess. In this case, the model needs to be able to beat 50% accuracy. In my initial proposal, I was way to enthusiastic in saying I would acquire a 90% accuracy. A more reasonable guess would be 60%. This was my initial goal for a metric. However, the classes are extremely imbalanced. Not everyone bothered to respond to the solicitation so the class 0 is far larger than class 1. In this case, a more appropriate metric would be ROC. More specifically I want to beat a .6 ROC.

Analysis

DATA EXPLORATION

There are two datsets we are preforming our analysis on. Azdias and customers. There are 366 columns for Azdias and 369 columns for customers. There were a lot of values that were missing in this data so I needed to impute those values. These features covered a whole gamut of different attributes of each customer and ranged from financial attributes to shopping habits, mindset, whether they had children and many others. The azdias dataset contains data from the german population as a whole and contains data on 891,211 people. The customers dataset has data on 191,652 people and contains the same attributes except there are an additional three attributes

EXPLORATORY VISUALIZATION

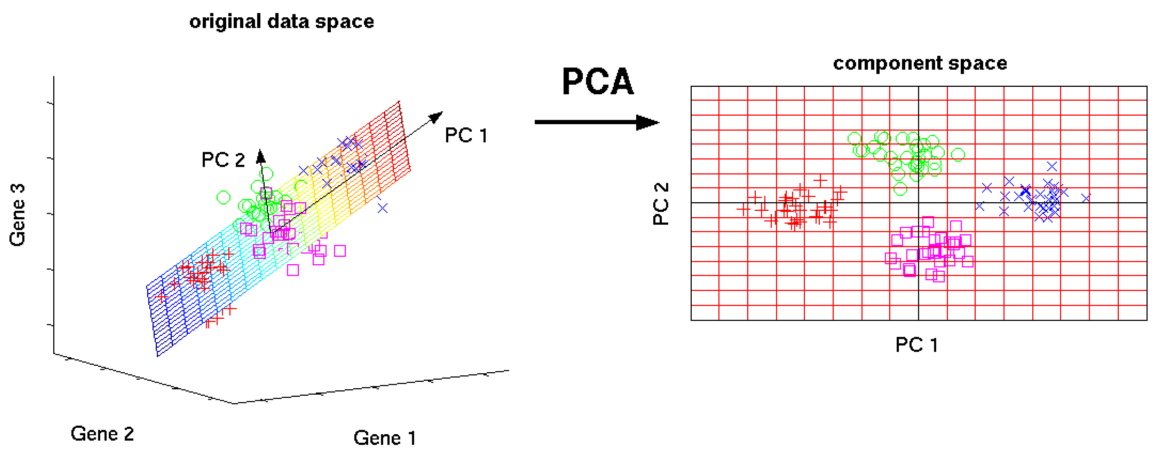
I performed a histogram analysis on all the variables but it seems to show that the distribution shared by the customers histogram is almost identical to the distribution shared by azdias for almost all the variables. There are a couple of variables like

AGER_TYPE that show some slight differences. Please reference the final cells found in part 0 to see the data histogram comparisons.

ALGORITHMS AND TECHNIQUES

The primary algorithms used for this project can be split up into two parts, the ones used for the customer segmentation and the ones used for the supervised learning model. The customer segmentation report used Principal component analysis and Kmeans clustering.

Principal component analysis is a way of finding which features capture the most information and discarding the ones that don't. It is a statistical technique that uses covariance matrices to create linear combinations of the features in the original dataset. The below picture represents visually what this technique does. It can reduce the dimensionality of a dataset allowing data points to be easily separable.



Source: http://www.nlpca.org/fig_pca_principal_component_analysis.png

Principal components are created that are a linear combination of current variables and capture the most useful information. These principal components will then be used to determine which direction in the dataset has the most variance. The features that are a part of those important components will be kept. These principal components will determine which features you should discard as the ones with less useful features will explain little to none of the variance.

Kmeans clustering is an algorithm that is used to group like datapoints together into clusters. It is one of the most popular unsupervised learning techniques in use today. The kmeans algorithms creates centroids, which are datapoints that are the central point of the defined clusters and goes through iterations that move the centroids towards their proper locations until they don't move anymore after calculating the Euclidean distances between the datapoints and their assigned clusters. These are then chosen as the center points of the clusters and the data points are color coded according to their respective cluster. This is how it normally works but since the dataset uses categorical features, the kmeans clustering analysis has to be done in conjunction with heatmaps to see which PCA components are correlated with what cluster. The component makeup can then be subsequently analyzed.

BENCHMARK

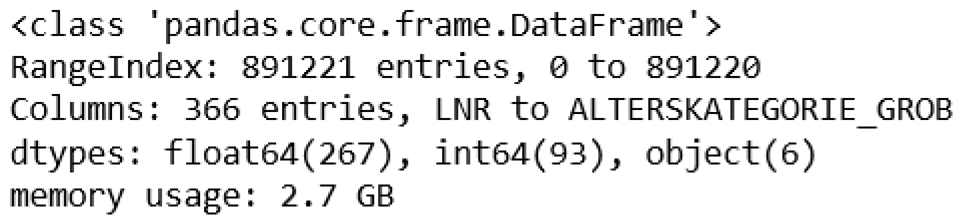
The benchmark I want to beat is .6 ROC_AUC. .5 ROC_AUC is the same as random guessing and any simple model can beat that. To beat .6 ROC_AUC, I need to choose the appropriate features and handle class imbalances to get above that metric.

Methodology

DATA PREPROCESSING

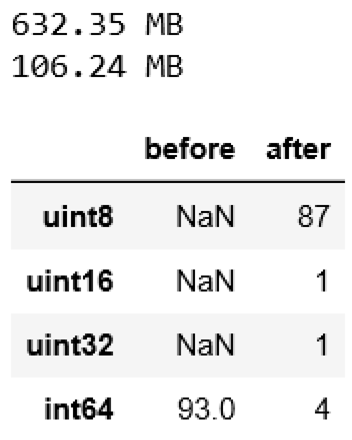
There was a lot of data processing that went into preparing the data for analysis. The first thing to do was to change the data types so that it would be easier to work with the data. There was a Dataquest article titled 'Tutorial: Using Pandas with Large Data Sets in Python' by

Josh Devlin that went into detail how to choose what type conversions should be chosen. I first used the pd.info method with memory_usage equal to deep. This shows how much memory is used to store all the variables. Using the info method on azdias shows the following.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891221 entries, 0 to 891220
Columns: 366 entries, LNR to ALTERSKATEGORIE_GROB
dtypes: float64(267), int64(93), object(6)
memory usage: 2.7 GB
```

So total memory usage was 2.7 GB, which is pretty high. It makes this data set sort of unwieldy when trying to do these preprocessing steps. Following the steps outlined in the article, I proceeded to apply variable downcasting and comparing how much the memory usage changes. There was a function provided in that article that gave the following result.

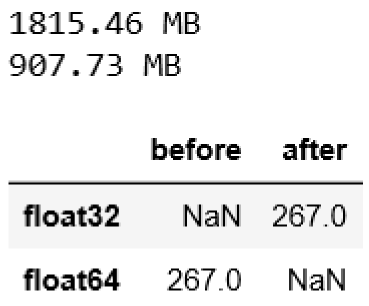


```
632.35 MB
106.24 MB
```

	before	after
uint8	NaN	87
uint16	NaN	1
uint32	NaN	1
int64	93.0	4

So before the conversion, all the integer variables were of type int 64 and had a memory usage of 632.25 MB. After the conversion, there are 87 variables of type uint8, 1 of uint16 and 4 of int64. This reduced memory usage from 632.35 MB to 106.24 MB.

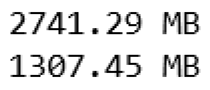
It is necessary to do the same for the float columns as well. Applying the same function to the float columns shows the following result.



```
1815.46 MB
907.73 MB
```

	before	after
float32	NaN	267.0
float64	267.0	NaN

Before the conversion, all columns were of float64. After the downcasting, the variables were turned into float32. This reduced the memory footprint by half. It is now of interest to see how the optimized azdias dataset compares to the original azdias in terms of memory footprint. Here is the result of applying the mem_usage function to both dataframes.



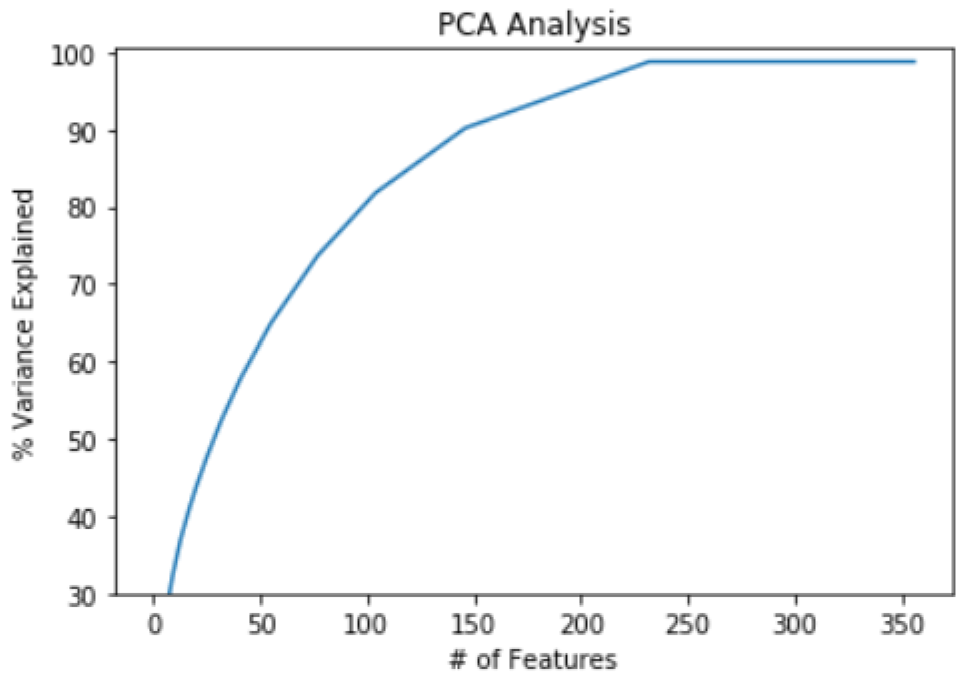
```
2741.29 MB
1307.45 MB
```

The original azdias dataset uses 2741.29 MB of memory whereas the optimized azdias dataset uses 1307.45 MB of memory. This will make it easier to manipulate the dataset now.

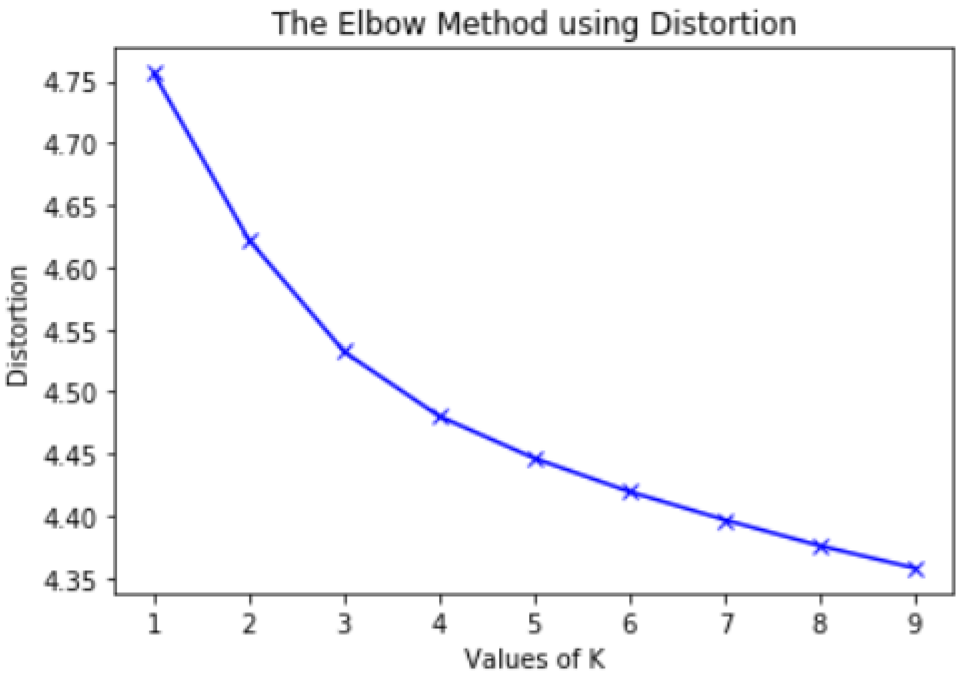
The next step after this was to apply a preprocessing function that I made on both datasets. This function coerced two columns ‘CAMEO_DEUG_2015' and 'CAMEO_INTL_2015' to become numeric columns and then split the data into int and float features to apply the fillna() method. I want to fill all the NA values with -1 since that represents an unknown value. The object columns don’t have any processing done. All the columns are then concatenated at the end.

IMPLEMENTATION

The algorithms used were Principal Component Analysis and Kmeans clustering to try and group people into people who would become a customers. Principal Component analysis is used to reduce the dimensionality of a dataset. I used the minmax scaler to normalize the features beween 0 and 1. I then used the PCA function in scikit learn on the dataset and then applied the fit transform method. The desired components represent most of the variance while discarding components that don't explain as much of the variance. I chose the first 180 principal components because they represented 98% of the variance. We were able to reduce the dimensionality of the dataset by almost half. This can be seen in the PCA analysis below. The code used to generate the following plot was found from https://etav.github.io/python/scikit_pca.html and was titled "Principle Component Analysis (PCA) with Scikit learn".

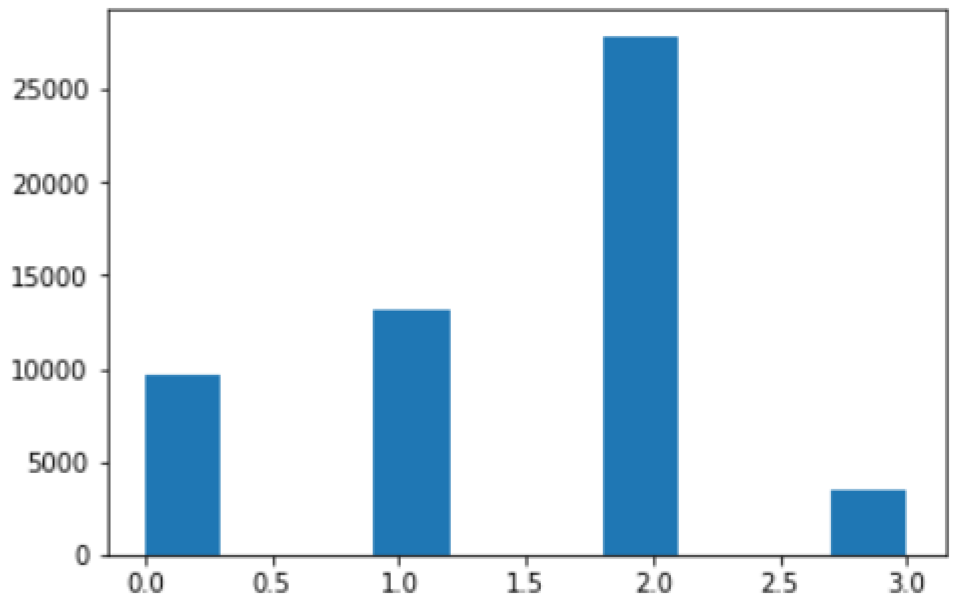


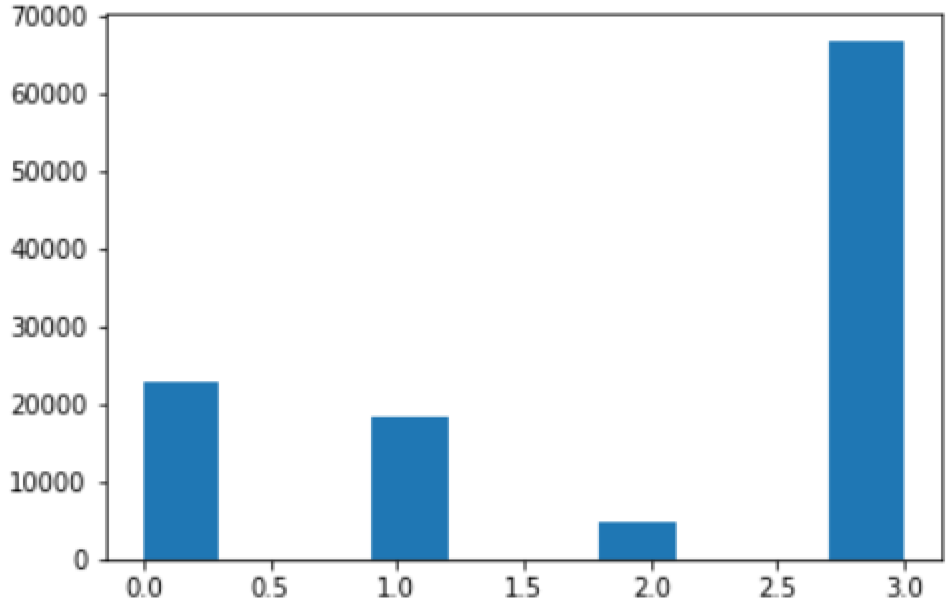
An issue I had was not knowing how many clusters to choose so that I could get a meaningful analysis. An article on the website geeksforgeeks.org called 'Elbow Method for optimal value of k in KMeans', showed code for performing an elbow analysis. The code calculates something called 'distortion' which is the squared distance from the data points to their assigned clusters. It is desired to choose the number K at which the distortion starts decreasing linearly. For this problem, K = 4 would be the desired number of clusters. The graph for the elbow analysis is shown below.



It looks like it starts decreasing linearly at k = 4 so I will choose that number for my cluster count.

Using this information, a KMeans model was used from the scikit learn model using k = 4. The results for the cluster breakdowns are shown below. The first graph is the cluster breakdown for azdias and the second graph is for customers.





It looks like most of the people in the customers dataset belong to cluster 3. Most of the people in the azdias dataset belong to cluster 2. It will probably pay off in the long run to focus on cluster 3 for the mail order campaign.

Heatmaps were generated and showed the components that were highly correlated with certain clusters. From the heatmaps, I found three components that were highly correlated with being a part of cluster 3. These were components 67, 133 and 245. The heatmaps are added as attachments.

REFINEMENT

For the customer segmentation, the Kmeans algorithm originally showed that the best number of clusters was 4. I was okay with this until I realized that this was caused by keeping the LNR feature which does not give any meaningful information whatsoever. Removing this feature reduced the number of clusters needed to 2. I then realized that I wasn't using the transformed dataset to do my analysis. I chose the first 180 features rather than the first 180 components. Doing this, led me back to k = 4 for my clusters. The heatmaps also changed and showed a lot more information on what components were correlated with what cluster. These heatmaps will be shown in the python notebook as well as in the attached pdf files.

For the unsupervised learning model I chose to find different features by using a correlation approach. I apply the corr function to the dataset to see which features correlated the most with the response variable. I chose the top 6 features that had the highest correlation. I decided to verify these features by applying SelectKBest and F_classif from the scikit learn library. These functions will allow me to calculate which features are the most significant. These features matched up with what I found when I applied the correlation method to the dataset. Using the top 4 features, my model jumped from 50% ROC which is no better than random guessing to 70% ROC when using the random forest classifier and decision tree classifier. In addition, the class imbalances were mostly solved using SMOTE. This sampling technique creates synthetic examples to make the decision boundary easier to find for the models. Increasing the feature size from the top 4 features to the top 6 features further increased my accuracy.

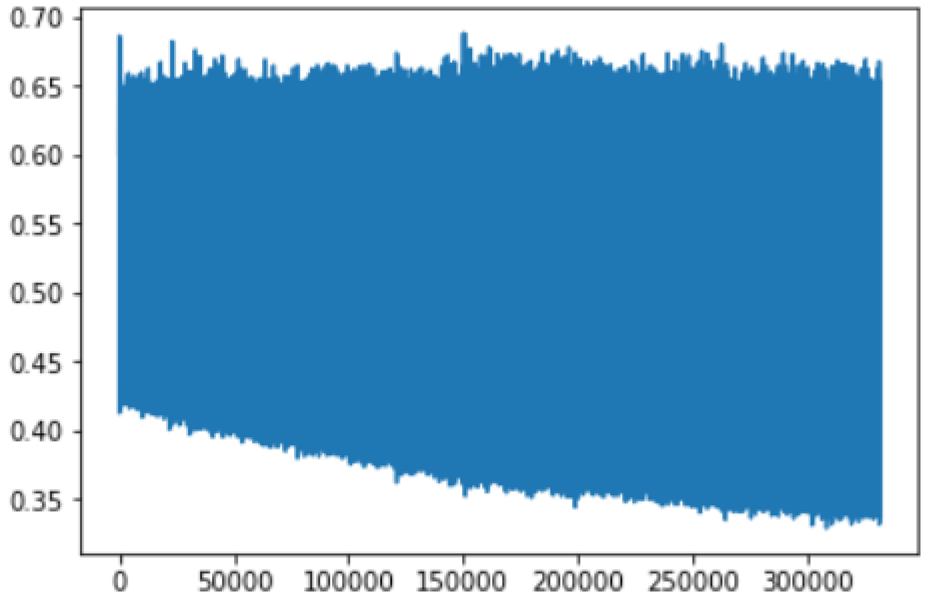
The models also underwent continuous changes. I originally tried to use a polynomial SVC kernel, XGboost tree and Decision tree for the sklearn models. The SVC kernel and XGboost performed quite badly. Further research seems to indicate these algorithms are bad at handling class imbalances. Instead I ended up using a decision tree and random forest classifiers. These performed well when I restricted the depth of the tree. The neural networks varied in architecture as well. The linear layers varied between two linear layers to six linear layers. Four linear layers seemed to yield the best results. The hidden layer size also was initially a sort of random guess. I just tried different values until I finally decided to keep the hidden layer size as a multiple of the input feature size. This means that if I have 6 features, I would have hidden layer sizes be a multiple of that input space. The neural network also had issues with class imbalances so I first tried random oversampling of class 1 so that I could have equal class balance but that increases the chance of overfitting. I instead opted to use the SMOTE algorithm to create synthetic examples that would allow the network to correctly classify better. My initial solutions were a decision tree model, a random forest and a pytorch neural network with 6 linear layers with a high hidden dim size. These models performed abysmally with the mean ROC_AUC on the test set being around .487, .499, and .495 respectfully. My final solutions after applying the SMOTE algorithm and correlation and F-score analysis bumped up the ROC_AUC to .734, .723 and .726 respectfully.

Results

MODEL EVALUATION AND VALIDATION

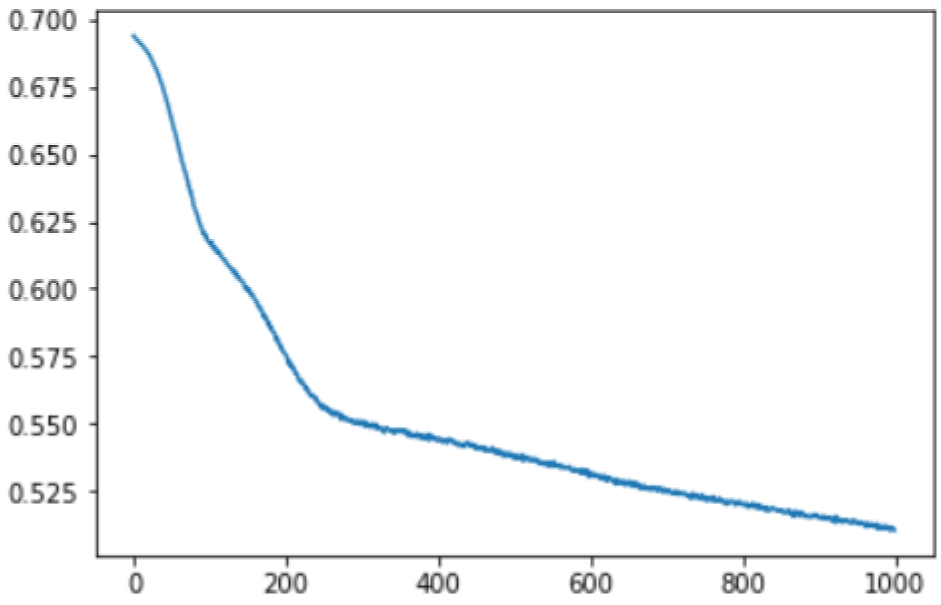
The final models I have made are two models from scikit learn and one model using pytorch. The two models I chose from scikit learn were a decision tree and random forest classifier with max depth reduced to 4 and 3 respectfully. The ROC_AUC for both was found to be .833. Restricting the depth of the decision tree and random forest classifiers to three or four reduced overfitting significantly. The train losses seemed to line up with the test losses more closely.

The second to last model for pytorch was a 5 layer network with one input layer, 3 hidden layers and one output layer. There is a dropout of .5 before the output layer as well. I used the adam optimizer and a learning rate of .001 as well as a BCELoss criterion. The loss graph is shown below. The reason why its outputting the wide range of losses was because I was batching the data in the train function and essentially performing multibatch gradient descent.



The final loss recorded for 2000 epochs was .35. The ROC_AUC on the train set was .767. The test set showed an ROC_AUC of .71.

Instead of batching, I also tried just feeding in the whole dataset and got a cleaner loss curve.



As a result, an ROC_AUC of .75 on the train set and an ROC_AUC .7268 on the test set was achieved. The model parameters were saved and manual adjustments were performed but training gets difficult at such small losses.

JUSTIFICATION

The benchmark that had been decided on was a .6 ROC_AUC. All three models beat this benchmark. The decision tree with max depth of 3 got a ROC-AUC of .736. The random forest with max depth of 4 got an ROC_AUC of .74158 and the neural network got an ROC_AUC of .7268.

Sources

1. http://www.hubga.com/wp-content/uploads/2017/09/arvato-logo-1.png
2. https://hackernoon.com/a-laymans-introduction-to-principal-components-2fca55c19fa0
3. https://www.bing.com/videos/search?q=smote&view=detail&mid=D94526DC0DCF9AB3AA7AD94526DC0DCF9AB3AA7A&FORM=VIRE
4. https://github.com/BVLC/caffe/issues/3801
5. https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1
6. https://www.dataquest.io/blog/pandas-big-data/
7. https://etav.github.io/python/scikit_pca.html
8. https://blogs.oracle.com/datascience/introduction-to-k-means-clustering