Machine Learning in Marketing: Predicting the Success Rate of Cold Calling

Capstone Project

Manuel Hurtado
Machine Learning Nanodegree
Udacity

September 2018

I Definition

1.1 Project Overview

By its definition, marketing is the "action or business of promoting and selling products or services, including market research and advertising."[1] As a corollary of this definition, the act of marketing implies the establishment of a relationship between two parties that facilitates the transactional *quid pro quo*: the promoter and provider of the product or service —traditionally, a company— and the customer. Theoretically, these transactions could be maximized by tailoring the product's promotions to match each customer's wants exactly. However, this process of tailoring a product's marketing to match the tastes of individual customers is resource-expensive in practice and consequently commercially inviable. Therefore, throughout history, marketing optimization has largely focused on fine-tuning a compromise between scalability and individuality to maximize the appeal of a product in a mass marketplace.

This optimization challenge has been enhanced by digital technologies. Research by Lim on the subject asserts that "data capabilities driven by affordable technology [have given] (...) institutions the ability to identify and reach customers in new ways."[2] The proliferation of the phone as an affordable household commodity, for instance, has facilitated B2C marketing contact. Despite the increasing efficacy of advertisement through techniques as telemarketing, or Direct Marketing —popularly referred to as 'cold calling'—, the compromise problem still renders fairly inefficient the process of randomly calling clients to offer them services/products and hoping they will buy them.

In this project, it will be attempted to devise a machine learning-driven solution to the inefficiency predicament of cold calling. To do so, a Multi-Layer Perceptron will be trained on the Bank Marketing Data Set found in the UCI Machine Learning Repository.[3] This dataset contains information of over 45,000 clients of a Portuguese financial institution and the corresponding outcome of the same marketing-purposed cold call for each. The resulting Classifier MLP will predict whether a call to a specific client will be successful or unsuccessful based on their data, making the process of cold-calling new clients less unpredictable and more efficient.

1.2 Problem Statement

Although the industry of telemarketing has advanced throughout the past decades, call centers remain largely inefficient in their operations, since a large fraction

[1] Marketing — Definition of marketing in English by Oxford Dictionaries. (n.d.). Retrieved August 10, 2018 from https://en.oxforddictionaries.com/definition/marketing

[2] Lim Chee Hiong, D. (2009). Building a Customer Centric Business for Service Excellence and Competitive Advantage in the Singapore Banking Industry. *University of Nottingham EPrints*. Retrieved August 10, 2018, from http://eprints.nottingham.ac.uk/22667/1/09MBAlixchdl1.pdf

[3] Moro, S., Cortez, P., & Rita, P. (2014). A Data-Driven Approach to Predict the Success of Bank Telemarketing. *Decision Support Systems, Elsevier*, 62, 22-31

of calls fail to make the sale and are therefore losses that the operating entity must assume. In fact, the Direct Marketing Association claims that the average success rate of cold calling in 2013 was 12.95% when using in-house lists[4](as would be the case for a financial institution such as the Portuguese bank in the Bank Marketing Data Set), and 8.21% when using lists provided by third parties[4]. The cost per lead (CPL) of cold calling for that year averaged USD \$190[4], which brings the inefficiency of Direct Marketing into perspective when one considers the negligible cost of a single phone call.

The goal of this project is to provide institutions on the supply-side of Direct Marketing with a preemptive probability —greater than 12.95%— of whether a call will be successful for each specific client being called. Statistically, this knowledge results in a more efficient allocation of the telemarketer's time and the institution's monetary resources: by increasing the probability of success of a single call above the benchmark figure of 13%, the institution can better allocate callers to value-bringing clients, increasing their overall success rate and lowering the CPL.

This classification problem will be addressed by creating a Classifier MLP trained on the UCI Bank Marketing Data Set. To do this, the network will be fed a set of the client's features and deliver a binary response to whether the sale will be successful. For a given number of features, n, and a set of two outcomes, sale, 1, or no sale, 0, the Neural Network acts as a function, f, where:

$$f : \vec{x} \longmapsto y$$
$$\vec{x} = (x_1, x_2 ... x_n)$$
$$y \in \{1,\ 0\}$$

The roadmap to be followed in this project is as follows:

1. Clean the data for use.
2. Engineer the data's features to enhance model performance.
3. Reduce the dimensionality of the features (PCA) to reduce computational complexity.
4. Train a benchmark model (Support Vector Classifier) on the data for comparing MLP performance later.
5. Design and train the MLP on the data, and fine-tune parameters for optimal performance.

1.3 Metrics

The model that will result from this project will not be 100% accurate, inevitably. This means some calls that would actually be successful will be labelled

[4]Direct Marketing Association. (2013, April 30). DMA 2013 'Statistical Fact Book'.

unsuccessful by the classifier (false negatives), and vice versa (false positives). In the context of the problem at hand, the former is more pressing: economically, the marginal cost of a single call is negligible, and the monetary value foregone by not making a successful call because the model determines it to be unsuccessful (false negative) is far greater than the value lost in making a call erroneously predicted to be positive (false positive). It is therefore in the interest of calling institutions to keep false negatives at a minimum. For this reason, the model should value recall more highly than precision. As a result of this analysis, an F-score of 2 will be used to evaluate the model's performance. This will assign greater weight on recall than precision, and ensure that the model penalizes false negatives more than false positives:

$$F_2 = \frac{(1 + 2^2) \cdot TP}{(1 + 2^2) \cdot TP + 2^2 \cdot FN + FP}$$

The inclination towards an F-score of 2 is also driven by an imbalance of the dataset's target variables (discussed more at length further on). Essentially, the data has a successful-to-unsuccessful target variable ratio of roughly 1:8. Because the model will be exposed towards 8 times more negative outcomes than positive ones during training, it may end up biased towards negative classifications. By penalizing false negatives more than false positives, the F-2 score combats this imbalance.

In addition, Cohen's Kappa will be used as a supplementary evaluation metric, since it evaluates "classification accuracy normalized by the imbalance of the classes in the data."[5]

[5]Brownlee, J. (2016, June 07). 8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset. Retrieved from https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/.

II Analysis

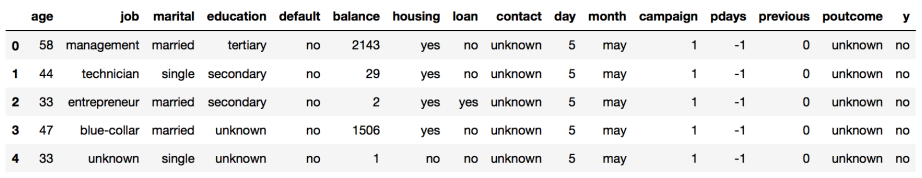
2.1 Data Exploration

The Bank Marketing Data Set of the UCI Machine Learning Repository contains client data for 45,211 calls. Each entry in the dataset corresponds to a single call made to a specific client, and details a set of the client's and call's (predictor) features along with the call outcome (target).

The dataset is composed of numeric and categorical values with 'int64' and 'object' data types, respectively. There are no missing values in any of the features of the dataset.

Dataset Description

The image below, containing the first 5 samples, helps to better illustrate the nature of this dataset:



	age	job	marital	education	default	balance	housing	loan	contact	day	month	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	1	-1	0	unknown	no

Figure 1: First five rows of the dataset

As it can be noted above, each client sample (row) has 15 predictor features and one target variable. For a better understanding of the data, a brief description of each has been included below, and they have been separated into groups for clarity:

1. Features containing client data:
   (a) Numeric:
      i. **Age**: Age at the time of call.
      ii. **Balance**: Balance in account at the time of last call.
   (b) Categorical:
      i. **Job**: Type of job.
      ii. **Marital**: Marital status.
      iii. **Education**: Highest level of education attained.
      iv. **Default**: Whether the client has a housing loan.
      v. **Housing**: Whether the client has a housing loan.
      vi. **Loan**: Whether the client has a personal loan with the bank.
2. Features containing information on the last contact during the current campaign (call that yielded the target variable):

   (a) Numeric:
      i. **Duration**: Duration of last call (in seconds).
      ii. **Day**: Day of the month in which the call was made.
   (b) Categorical:
      i. **Contact**: Type of channel used to contact the client.
      ii. **Month**: Month of the year in which the call was made.

3. Others:
   (a) Numeric:
      i. **Campaign**: Number of contacts to this client during this campaign (including last contact).
      ii. **Pdays**: Number of days that have elapsed since the client was last contacted.
      iii. **Previous**: Number of contacts to the client performed prior to this campaign.
   (b) Categorical:
      i. **Poutcome**: Outcome of the previous marketing campaign for that specific client.

4. Target:
   (a) Categorical
      i. **Y**: Whether the client subscribed to a term deposit after the call (call outcome).

Skewed Feature Distributions

The most significant and noteworthy attribute of this dataset is that the vast majority of its numeric features present positively skewed distributions, where a large portion of the feature values tend to be low but there are samples that contain very high ones. To illustrate, consider the variables *Age*, *Balance*, and *Campaign*, described in table 1.

As it can be noted, in all three features the maximum value for that feature lies at more than 5 standard deviations from the mean. What is more, for *Balance* and *Campaign*, this proportion climbs to approximately 20. This heavy skew of the features preemptively suggests that there will need to be a removal of some outlier samples prior to the models' training.

Target Variable Imbalance

The imbalance of feature values extends itself to categorical features as well. Most significantly, the target variable presents a greater quantity of 'no' outcomes than it does 'yes'. Specifically, there are 39,992 counts of calls that resulted in an unsuccessful sale, compared to a meager 5,289 successful calls.

Descriptive Statistics of Numeric Features

Statistic	**Age**	**Balance**	**Campaign**
Mean	40.94	1,362.27	2.76
Std	10.62	3,044.77	3.10
Min	18.00	-8,019.00	1.00
Lower Quartile	33.00	72.00	1.00
Median	39.00	448.00	2.00
Upper Quartile	48.00	1,428.00	3.00
Max	95.00	102,127.00	63.00

Table 1: Statistical Description of Age, Balance, and Campaign Features

This is approximately a ratio of 7.5 unsuccessful calls per successful call. Put into perspective, only 13.3% of the dataset corresponds to successful call outcomes. While this imbalance accurately resembles the benchmark figure of 13% success rate of cold calling in real life, it hampers the success of this project's goal. Therefore, it will be mitigated by using an F-2 Score as evaluation metric during the training of the models in order to more greatly penalize false negatives, as discussed previously.

It is important to note that the use of an F-2 Score is not the sole available remedy for this imbalance. Alternatively, it could be attempted to over-sample the data and create approximately 20k to 25k instances of positive outcomes. Otherwise, synthetic instances of the data could be created. The reason why the latter was ruled out is that it could result in the non-linear relationships between features not being preserved[6]. The former, on the other hand, would result in a less authentic data set than the one contemplated presently. In order to design and train a real-life predictive model to solve the inefficiency problem of cold calling, a reduction in authenticity is unacceptable. For these reasons, the F-2 Score alternative was preferred.

Feature Inspection

There are only two dataset-wide changes that must be executed prior to training the models. The first is changing the data type of the categorical variables from 'object' to 'categorical'. This not only saves space in memory but also makes future pre-processing steps such as one-hot encoding more straightforward computationally. The second is transforming and normalizing the numeric features to address their skewed distributions.

Otherwise, some specific features present issues that must be addressed. These are as follows:

1. **Duration**: As per its description, the *Duration* feature represents the length in seconds of the last call made to the client (i.e. the call that yielded the 'yes' or 'no' value of the target variable). In the context of

[6]Brownlee, Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset.

this project's real-life problem to be addressed, the length of the call yet-to-be-made is not known. Because the purpose of this project is to create a predictive model that is applicable to real-life scenarios, the predictive model must be trained and tested without this feature in the dataset. Therefore, it must be discarded.

2. **Campaign**: The *Campaign* feature represents the number of calls made to each specific client during the present marketing campaign. Yet because the purpose of the model to be designed and trained is to predict the outcome of this 'last call', it is necessary to deduct one call from all of the *Campaign* values in the dataset.

3. **Pdays**: The feature *Pdays* represents the number of days that have elapsed since the last call made to each specific client. Currently, clients that have not been contacted before have this feature filled with a value of -1. From a mathematical point of view, this value is too close to 0, which represents clients that were just called a day before. In the context of the problem, it is not logical to assimilate clients that have never been called before with clients that were called just 'yesterday'. Therefore, it is pertinent to change this value to a number that resembles clients that were called a long time ago, who perhaps forgot they had been called before and so represent somewhat of a 'blank slate'.

4. **Day**: The *Day* feature is naturally filled with numeric values ranging from 1 to 31, corresponding to the possible days of the month. However, although it is a variable with data in numerical form (int64), the actual value of these numbers is of no interest to a model. Insofar as statistics is concerned, a client could be called any day with equal probability, since any day of the week in a month could correspond to any number between 1 and 31. For this reason, its distribution does not resemble a normal distribution, like that of other numeric variables. It hence makes more sense to treat it as a categorical variable instead.

2.2 Exploratory Visualization

The presence of outliers in a dataset can be extremely harmful to a model's training process. Therefore, it is imperative to clearly understand how the numeric features in the dataset are distributed.

Figure 2 showcases the histograms of the five numeric features in the dataset (for the reasons explained above, *Day* is not included among them). The minimum and maximum values of each feature have been specified below the label on the X-axis to paint a clearer picture of the distribution ranges.

Histograms of the Numeric Features in the Dataset

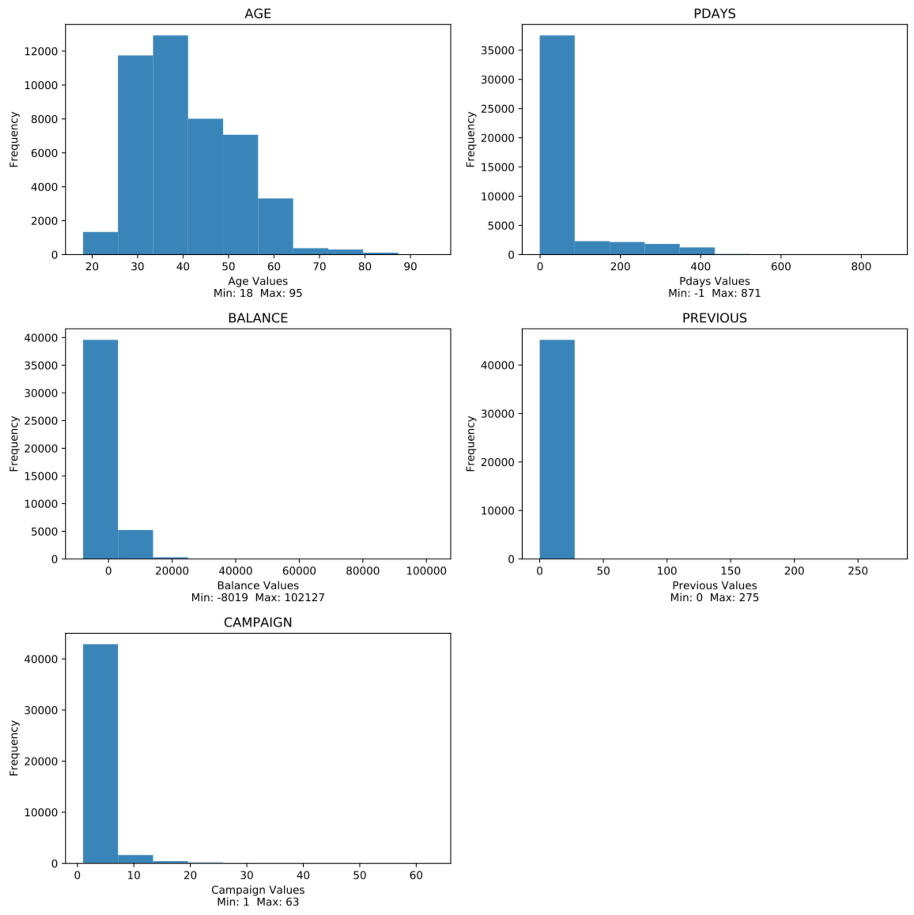


Figure 2: Distributions of Numeric Features

As it can be seen in the figure, all of the numeric features are positively skewed, with the median located around very low values. Particularly in the

Balance, *Previous*, and *Campaign* features, the negligible size of the tail in comparison to the body of the distribution highlights their imbalance.

This analysis is useful because it helps understand the large presence of outliers in the dataset, and consequently underlines the pertinence of removing them in order to center the distributions as much as possible. This is imperative to enhance the benchmark and final model performances.

Furthermore, visualizing the distributions of the numeric features permits the extraction of relevant information: most of the clients contacted by the bank are in their mid-to-late thirties, do not have a lot of money in their accounts and even owe the bank money (this could suggest that the product being offered is tailored for clients with low income), and were contacted very few times during the present campaign, amongst other things. From these statistics one can infer that by removing most of the outliers (noise) from the dataset and working only with most of the clustered samples, the model will be able to extract a more rich network of patterns.

2.3 Algorithms and Techniques

Multi-Layer Perceptron

The classifier to be used in attempting to devise a solution to cold calling's inefficiency problem is a Multi-Layer Perceptron, which will output a binary result (1 or 0) after being fed the features of a single client in the dataset. This output will correspond to the binary outcomes of the call to that client ('yes' or 'no').

To understand what MLPs are, it is imperative to first get a clear idea of what a perceptron is. In basic terms, a perceptron is composed of four different parts: input values, weights and bias, net sum, and an activation function.

In a perceptron, the input values are each multiplied by a corresponding weight and summed together (a dot product of weight and input vectors). To this summation is added a bias, which can be thought of as a constant input value that has a weight of 1.0. This net sum is then fed through an activation function, which produces a single output. Hence, the perceptron acts as a neuron, whereby multiple electrical signals (input values) are processed internally to emit a single electrical signal as output:

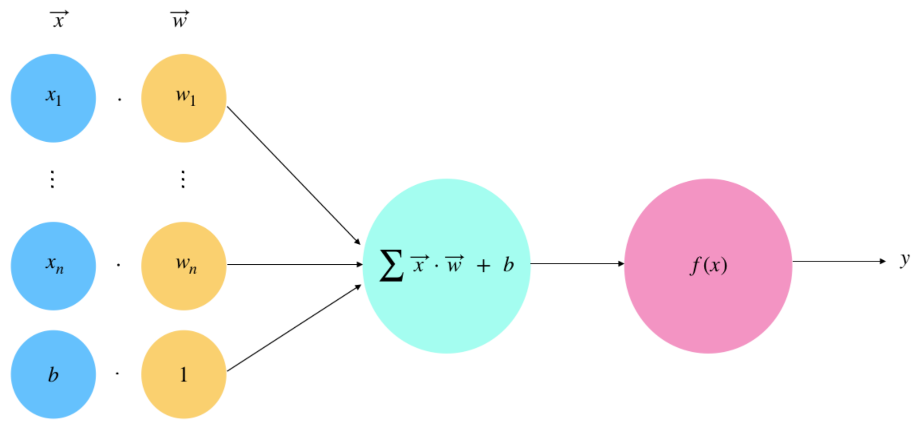


Figure 3: Perceptron

When arranged in multiple layers, the outputs of the perceptrons in one layer become the inputs of the perceptrons in the next layer. Because the output of each perceptron in the preceding layer becomes an input to each perceptron in the next layer, the layers become 'fully connected'. Because these networks of perceptrons essentially take an input vector and process it successively across layers to yield a final output, they are called 'Multi-Layer Perceptrons'. In essence, the MLP acts as a single perceptron and therefore becomes the function itself. Because the interconnected layers of perceptrons within a MLP resemble the way neurons are interconnected in the brain, MLPs are popularly referred to as 'Neural Networks'.

This project will use a MLP to devise a solution to the inefficiency problem of cold calling because while it is computationally more expensive to train, fine-tune, and evaluate than other algorithms, a Neural Network has a capacity of capturing non-linear relationships in data[7]that other simpler algorithms such as Linear Regression or Decision Trees cannot equate.

Principal Component Analysis

As part of the data pre-processing steps prior to training the benchmark and MLP models, the data will be reduced in dimensionality using Principal Component Analysis. In basic terms, Principal Component Analysis identifies non-cartesian axes within the data that capture the most variation. These are called *components*. After finding these principal components, it mathematically projects each of the data's features onto them. This process yields a set of n components, each with a weight corresponding to each of the m features of the dataset, where $n < m$.

[7]Somers, M. J., & Casal, J. C. (2009). Using Artificial Neural Networks to Model Nonlinearity. Organizational Research Methods, 12(3), 403-417. doi:10.1177/1094428107309326

PCA is useful in the context of this problem because the number of features in the dataset will increase dramatically after the categorical features are one-hot encoded, and it is effective because it can reduce the dimensionality of the data considerably while still capturing a satisfactory amount of variance in the dataset. This reduction dramatically reduces the computational costs of training, fine-tuning, and evaluating the models.

Grid Search Cross-Validation

To optimize a Neural Network, it is possible to tune many parameters simultaneously. Doing so manually is a complicated and mechanical endeavor, which is why Grid Search Cross-Validation can be used to automate this process.

In summary, Grid Search CV accepts an estimator model (MLP classifier, for our purposes) along with a dictionary of the parameters to be fine-tuned and the possible values of each of those parameters to be tested. Grid Search CV then proceeds to test all the possible combinations of those parameter values. To do so, it splits the data randomly into k folds (default is 3), trains the model on two of those folds, and evaluates the model's performance on the third one. It then repeats the process to test the model on the other two folds, and computes an average of the model's performance across all three validations for that combination of hyperparameters. After evaluating all combinations, it returns the one that yields the best performance.

In the case of a MLP, the following are some of the parameters that *can* be fine-tuned to optimize the Neural Network's performance:

1. Neural Network Architecture
   - (a) **Layers**: Number of hidden layers in the Neural Network.
   - (b) **Nodes**: Number of nodes in the hidden layers of the Neural Network.
   - (c) **Dropout**: Proportion of nodes (or perceptrons) that should be dropped out randomly in a layer during training to avoid overfitting.
   - (d) **Activation Function**: Function to be used in the layers of the dataset.
2. Training Parameters
   - (a) **Epochs**: Number of runs through the training data during training.
   - (b) **Batch Size**: Size of the batches of samples on which the model should train.
   - (c) **Optimizer**: The function that is used to update the weights and biases of the model to reduce the output error.
   - (d) **Loss**: The function used to calculate the model's loss, or error of the model's output compared to the target.

Because fine-tuning all these parameters at once is computationally expensive, a hybrid approach combining manual experimentation and Grid Search CV will be adopted.

2.4 Benchmark

The most straightforward measure of the proposed model's performance is the benchmark figure of 12.95% average success rate of cold calling, provided by the Direct Marketing Agency's 2013 Statistical Fact Book. If a model can predict with twice or three times as much certainty that a call will be successful, it could imply a reduction of the average CPL to approximately \$85 or \$63, respectively.

However, this figure is a secondary benchmark, because even a mediocre model can easily outperform an accuracy of 13%. To better compare and improve the MLP, a benchmark model will be designed, trained, and fine-tuned from scratch. This will be a Support Vector Machine Classifier with an Radial-Basis Function kernel, which essentially works by calculating differences in the feature values in many dimensions and creating ranges/multi-height boundaries to separate them, finding hyper-dimensional planes to split the feature values, and project the intersections of these planes and multi-height range differences onto the input space to create the classification boundaries in the data.

The choice for this type of benchmark model is simple: an SVM Classifier with an RBF kernel offers the closest performance with non-linear datasets to a MLP. This model is more computationally complex than a Logistic Regressor or Decision Tree Classifier, but its superior performance translates to a more demanding benchmark to compare the MLP to.

In order to fine-tune this model, Grid Search Cross-Validation will be used. Specifically, the values of `C` (the penalty by which the classification error is multiplied, influencing how separated a boundary should be from the points) and `Gamma` (controls how much influence each point has in determining the ranges/multi-height boundaries) will be optimized to obtain the best benchmark model possible.

III Methodology

3.1 Data Preprocessing

In accordance to the conclusions drawn in the Data Exploration step regarding the features, the following are the preprocessing steps that were taken to improve the overall performance of the model and address some particular issues in the data's features:

1. Feature-specific Preprocessing Steps

   (a) **Duration**: As it was noted earlier, the *Duration* feature conveys information that is not realistically accessible at the time of making a call. Therefore, it cannot be used by a model to predict the outcome of a call in real life. It was dropped from the dataset.

   (b) **Campaign**: Because the *Campaign* feature included the last call in the number of calls for each client, a value of 1 was subtracted from all the samples' values for this feature.

   (c) **Pdays**: Since the clients that had never been contacted before contained a *Pdays* value of -1, which was too close to 0 (for clients that had last been contacted a day prior), this value needed to be changed. Therefore, it was replaced with 1.5 times the maximum value of the *Pdays* feature after outliers had been removed, to let the model assimilate these clients' behavior to that of clients that were contacted a very long time ago and probably forgot about it.

   (d) **Day**: As it was argued in the 'Data Exploration' section, the *Day* feature conveys a rather ambiguous numeric information. To alleviate this, several solutions were possible.

   One solution could have been to convert the variable into a categorical type and couple the *Day* and *Month* features together. This would be more informative, but would result in 12 x 31 different possible categories after one-hot encoding, which is computationally unfavorable.

   A more pragmatic solution was to break down the days of the month (1 through 31) into groups that convey the most information in the context of this problem, and make each of those a category. The dataset at hand comes from a bank, and so it is fair to infer that the underlying mechanics of the services offered and accepted/rejected in each call are contingent on money. In most of the world, salaries are paid on a semi-monthly basis (on the 1st and 15th day of each month). Therefore, if the values from 1 to 31 were divided into 4 groups, each corresponding to a week of the month, one could theoretically draw a more latent pattern between weeks 1, 3 and 'yes' (salaries are paid during these weeks, and so a feeling of 'affluence' is prevalent), and weeks 2, 4 and 'no'. This second approach was chosen.

2. Dataset-wide Preprocessing Steps

   (a) **Identification and Removal of Outliers**: As it was made evident by the analysis of the numeric features' statistics and histogram distributions, the numeric features in the data are very much positively skewed. The maximum value of these features lies a significant number of standard deviations from the mean, and the upper tails of most of the distributions are barely (if at all) visible.

   To identify the outliers in the dataset the widely-accepted definition provided by John Tukey was used. According to this definition, an outlier, o, is a point in a dataset that satisfies[8]

$$o \in \,]\, Q_1 \; - \; k(Q_3 \; - \; Q_1), \; Q_3 \; + \; k(Q_3 \; - \; Q_1) \, [$$
$$Q_3 \; - \; Q_1 = IQR$$
$$k = 1.5$$

   Iterating over all the numeric features to identify samples that are outliers in any feature and dropping the duplicate rows from the resulting dataset, a total of 14,804 outliers are obtained from the data.

   Before proceeding to drop these samples from the dataset, it was necessary to examine how this removal would affect the target variable imbalance identified previously. The results indicated that dropping the 14,804 samples would increase the imbalance of 'yes' to 'no' targets from 1:7.5 to 1:10.3. This increase in the imbalance would naturally harm the model's performance.

   To address this issue, two alternatives were contemplated:

   i. *Calculate the exact number of outlier samples with target variable 'yes' needed to maintain the proportion at 1:7.5, and randomly select the necessary samples to be dropped to maintain this proportion.*

      The advantage of this first alternative was maintaining the proportion of targets. However, randomly selecting elements from the outliers list could mean that outliers in both ends of Tukey's range are conserved. Because this could distort the model's predictions, this option was ruled out in favour of a more systematic solution.

   ii. *Identify the outliers with target variable 'yes' of the whole dataset again, this time using a slightly greater value of k than was used to identify those with target variable 'no' (such as $k = 2$) so that the number of 'yes' outliers is slightly smaller.*

      This option was chosen. By making Tukey's range slightly more lenient to exclude less samples with 'yes' targets as outliers,

[8]Songwon, S. (2006). A Review and Comparison of Methods for Detecting Outliers in Univariate Data Sets. *University of Pittsburgh.*

slightly more samples with target variable 'yes' would be removed from the dataset and the imbalance could be preserved as much as possible.

After removing outliers with target variable 'no' from the dataset using Tukey's definition of outliers ($k = 1.5$) and a slightly more lenient range for samples with target variable 'yes' ($k = 2$), 12,199 outliers with target variable 'no' were identified compared to 1,235 with target variable 'yes'. After dropping them from the dataset, the remaining samples presented a target variable imbalance of 1 : 6.8, which improved upon the original ratio of 1 : 7.5.

(b) **Transformation of Numeric Features**: To transform the numeric features of the dataset, a Box-Cox approach was used[9]. Essentially, the data is transformed by elevating all the values in each feature to a specific value, λ. To determine the best value of λ one should elevate the data to, one must look at the standard deviation of the feature distributions after elevating them to the different values. Logically, the transformation that minimizes the standard deviation of each feature is the optimal one for that feature.

After iterating over the possible values -2.0, -0.5, 0.0, 0.5, 1.0, and 2.0, it was determined that the best λ for each feature was

```
age: -2.0, balance: -0.5, campaign: 0.5
       pdays: -2.0, previous: 0.5
```

The features were then transformed accordingly.

(c) **Normalization of Numeric Features**: The numeric features were then scaled using a MinMaxScaler, which mapped the feature values to a range between 0 and 1.

(d) **One-hot Encoding Categorical Features**: After converting their data type from 'object' to 'categorical' to save space in memory, the categorical features of the dataset were one-hot encoded to facilitate model training and improve performance. The number of features in the dataset increased to 53.

3.2 Implementation

To implement the algorithms and techniques examined previously, a systematic procedure was followed:

1. Data Treatment:

   (a) **Training & Testing Split**: Prior to reducing the dimensionality of the data through Principal Component Analysis, it was necessary to split the data into training and testing sets. To do so,

[9]Sherman, P. J. (n.d.). Tips for Recognizing and Transforming Non-normal Data. Retrieved from https://www.isixsigma.com/tools-templates/normality/tips-recognizing-and-transforming-non-normal-data/

`sklearn.model_selection.train_test_split` was used to split the dataset into 80% training data and 20% testing data.

   (b) **Principal Component Analysis**: To reduce the dimensionality of the data prior to training the benchmark and MLP models, PCA was applied to training and testing predictor features by creating a `sklearn.decomposition.PCA` object to capture 95% of the variance. The object was then fitted to the training predictor features, and the training and testing predictor features were subsequently transformed.

   The resulting training and testing predictor data had 27 features (almost 50% less than original dataset's 53 features).

2. Benchmark Model:

   (a) **Support Vector Machine Classifier**: To create a benchmark model, a Support Vector Machine Classifier with an RBF kernel was created using the class `sklearn.svm.SVC`. The model's hyperparameters, `C` and `gamma`, would be fine-tuned subsequently using Grid Search Cross-Validation.

   (b) **Grid Search Cross-Validation**: To fine-tune the SVM Classifier's hyperparameters, a `sklearn.model_selection.GridSearchCV` object was created. The object was passed the classifier created previously as estimator. A dictionary with both tunable parameters as keys and the corresponding list of possible values for each as values was passed as well (the possible values for `C` were 0.001, 0.01, 0.1, 1, and 10; the possible values for `gamma` were 0.001, 0.01, 0.1, and 1). Two scorers, an F-2 Scorer and Cohen's Kappa Scorer, were created using the `sklearn.metrics.make_scorer` class, and passed to the Grid Search object as a list of scores. The grid was instructed to evaluate the models' performance according to the F-2 Score.

   For comparison purposes, and to view the effect of PCA on a model's performance, the Grid Search Cross-Validation step above was conducted twice. The first time the non-PCA training data was passed for training and the best model was evaluated against the non-PCA testing data. The second time, the PCA training and testing data were used for training and testing, respectively. A discussion of the deltas in performance metrics between both models is discussed in the 'Refinement' section.

3. Multi-Layer Perceptron:

   Designing Neural Network architectures is a complicated and often ambiguous task, made considerably more difficult by the absence of any 'rulebook' or 'guideline' that outlines the best practices. There are more hyperparameters that can be tuned than is the case with a Support Vector Machine Classifier, and there are infinite different architectures (number of layers and types of layers) that can be designed and used.

To find the best possible model, therefore, it is not enough to simply iterate over different combinations of hyperparameters. That may yield the best combination of hyperparameters for a given architecture, but it obviates the other possible architectures that could be used instead.

For this reason, a hybrid approach was implemented to train and fine-tune the MLP to be used in this project: a manual Architecture Exploration was undertaken, evolving a simple model over 9 generations, until the two best model architectures after the 9th generation were attained[10]. These two best model architectures were then fine-tuned using Grid Search Cross-Validation.

It must be noted that this hybrid approach is innovative and perhaps more effective than simply fine-tuning one possible architecture. Although it is obviously limited in that only two architectures were fine-tuned out of an endless realm of possibilities, it is computationally expensive and practically unfeasible to attempt to devise tens or hundreds of architectures and fine-tune them all using Grid Search CV (which by itself is a computationally demanding exercise).

(a) **MLP Architectures**[11]:

   i. Model 1:
      - Input layer with 27 nodes (one for each of the features in the PCA-reduced data) and a ReLU activation.
      - Three densely-connected hidden layers with ReLU activation functions and L2 kernel regularizers (number of nodes and size of regularizer penalty to be determined using Grid Search CV).
      - Two dropout layers interleaved with the dense layers (dropout size to be determined using Grid Search CV).
      - Final single-output layer with a logistic Sigmoid function.
      - Compiled with RMSprop optimizer and Binary Crossentropy loss.

   ii. Model 2:
      - Input layer with 27 nodes (one for each of the features in the PCA-reduced data) and a ReLU activation.
      - Three densely-connected hidden layers with ReLU activation functions and L2 kernel regularizers (number of nodes and size of regularizer penalty to be determined using Grid Search CV).
      - Three dropout layers interleaved with the dense layers(dropout size to be determined using Grid Search CV).
      - Final single-output layer with a logistic Sigmoid function.

[10]This process is explained further in the 'Refinement' section.

[11]For greater detail, refer to the code for both architectures in Appendix I.

- Compiled with Adamax optimizer and Binary Crossentropy loss.

Both models were compiled with an F-2 Score as observatory evaluation metric (the training metric to be minimized, by default, is Validation Loss). Because Keras works with tensors, the `sklearn.metrics.fbeta_score` with `beta = 2` would not work. Thus, code of a Keras-compatible F-2 function written by Arseny Kravchenko and posted on GitHub[12] was borrowed kindly and used instead. Cohen's Kappa would not work either, and because it is an observatory metric, it was discarded for this stage.

(b) **Grid Search Cross-Validation**: Fine-tuning a MLP's hyperparameters manually is complex and mechanical, and can be automated by using Grid Search Cross-Validation through the `KerasClassifier` wrapper found in `keras.wrappers.scikit_learn`. To use this wrapper, a function that builds, compiles, and returns a Keras model is defined, and passed into the `KerasClassifier` wrapper as argument, which in turn returns a classifier compatible with Sci-kit Learn. This classifier can then be passed as the estimator of a Grid Search CV object, along with a dictionary of the parameters one desires to tune (and their possible values) and a list of scoring functions to determine the best performance.

3.3 Refinement

Benchmark Model

For the benchmark model, the refinement process was carried out by Grid Search Cross-Validation, explained in the Implementation.

For comparison purposes, two different Support Vector Machine Classifiers were trained and optimized using Grid Search Cross-Validation: one trained and tested using non-PCA (raw) data, and another using PCA-reduced data. It is noteworthy that both Grid Search CV procedures yielded the same ideal combination of hyperparameters: `C: 10` and `gamma: 0.1`.

The performance metrics of both models are summarized in table 2.

Performance of Support Vector Machine Classifiers

Data	**Stage**	**Cohen's Kappa**	**F-2 Score**	**Accuracy**
Raw	Training	0.46	0.52	0.92
	Testing	0.41	0.45	0.91
PCA	Training	0.37	0.44	0.91
	Testing	0.34	0.40	0.90

Table 2: Grid Search Cross-Validation of SVC

[12]Kravchenko, A. (n.d.). F-beta score for Keras — Kaggle. Retrieved August 28, 2018, from https://www.kaggle.com/arsenyinfo/f-beta-score-for-keras

The different SVM Classifiers show a delta in scores that is almost negligible: the model trained on the raw dataset shows a mere 11% increase in both F-2 Score and Cohen's Kappa. In terms of accuracy, both SVCs performed nearly identically on the training and testing sets, differing by only 1% in results. However, the training time for the raw data model was approximately twice as long.

From this analysis it follows that it is unproductive and unnecessary from a performance standpoint to train the MLP on the full training set, since the computational cost saved by training it on the PCA-reduced data exceeds the improvement in performance derived from training it on the raw data.

Multi-Layer Perceptron

As mentioned in the Implementation section, the process by which the final MLP was attained consisted of a hybrid approach between manual model tuning and Grid Search Cross-Validation.

The first step of this hybrid approach is denominated 'Architecture Exploration'. To begin, a simple model architecture was defined: three hidden layers with 10, 15, and 20 nodes each, respectively, ReLU activation functions, and no dropout layers. The model's architecture was then adapted through 9 generations, being tuned with at least 3 different combinations of hyperparameters during each generation. After each generation, the best model's performance was observed and its behavior during training and testing analyzed. This analysis combined the Testing Loss, F-2 Score, and Accuracy, as well as a plot of the Training and Validation Losses during the model's training. According to that performance and whether the model was underfitting or overfitting, the next generation's direction of evolution was determined. After the 9th generation, the two best-performing models were selected. These models were then fine-tuned using Grid Search Cross-Validation.

A summary of the different measures taken depending on the direction of evolution desired at each generation can be found below:

- *To Reduce Underfitting*:
  - Increase number of densely-connected layers.
  - Increase number of nodes in densely-connected layers.
- *To Reduce Overfitting*:
  - Add dropout layers.
  - Add kernel regularizers.
- *To Enhance Model Performance*:
  - Experiment with addition of Learning Rate-reducing callback.
  - Experiment with addition of kernel initializers.
  - Experiment with use of different optimizers.

– Experiment with use of different activation functions.

Table 3 summarizes the parameters tuned during the process of Grid Search Cross-Validation, explained in the Implementation section.

MLP Parameters Tuned

Parameter	**Values**
Nodes	10, 15, 20
L2 Penalty	0.01, 0.001
Dropout Size	0.2, 0.3
Batch Size	40, 80

Table 3: Grid Search CV for Multi-Layer Perceptron

Table 4 illustrates the results of this fine-tuning process.

Best Parameters

	Nodes	**L2 Penalty**	**Dropout Size**	**Batch Size**
Model 1	15	0.001	0.3	40
Model 2	10	0.001	0.2	40

Table 4: Result of Grid Search CV for MLP Architectures

In the interest of brevity, only three performance deltas of the two models will be reported to demonstrate the effect of the successive refinement steps outlined above on performance. The first comparison is strictly for Model 1, before and after Grid Search. The second comparison does the same for Model 2. Lastly, the third comparison contrasts the final versions (post-Grid Search) of Model 1 and Model 2. The performance metrics depict the model's performance on the testing set (20% of the dataset). These statistics are shown in table 5.

Comparison of Model Performance Throughout Development

		Loss	**F-2 Score**	**Accuracy**
Model 1	Before GS	0.291	0.396	0.903
	After GS	0.295	0.367	0.904
Performance Delta (%)		*-1.388*	*-7.208*	*0.052*
Model 2	Before GS	0.283	0.384	0.905
	After GS	0.288	0.381	0.905
Performance Delta (%)		*-1.669*	*-0.689*	*0.035*
Final Models	Model 1	0.295	0.367	0.904
	Model 2	0.288	0.381	0.905
Performance Delta (%)		*2.468*	*3.774*	*0.122*

Table 5: Evolution of Model Performance

It would be tempting to conclude, from the results above, that the final models performed better *before* Grid Search Cross-Validation than they did *after* it. However, this would stem from an erroneous line of reasoning: while the performance of the models *before* Grid Search was obtained by training and testing the models only once, the performance obtained *after* Grid Search CV is best of 72 different fittings. Perhaps the specific fitting that was observed and reported in the table above for pre-Grid Search performance was particularly good, but multiple tests would suggest otherwise. After all, the combination of hyperparameters the models used *before* this technique were inevitably tested during the technique itself 3 times, but they were found to be below-optimal.

In light of this, the table above helps further underline why it was imperative to conduct Grid Search Cross-Validation in tandem with the Architecture Exploration.

IV Results

4.1 Model Evaluation and Validation

After carrying out the hybrid process combining the Architecture Exploration and Grid Search Cross-Validation, a best model was obtained (Model 2). This model was chosen to be the best because it culminated a long procedure of manual testing of 35 different architectural combinations evolved through a process of methodical analysis, and a computationally-intense Grid Search Cross-Validation process through which the best resulting architecture was fine-tuned.

Selecting a best architecture for a Multi-Layer Perceptron is a complicated task that is filled with ambiguity, since there are a myriad possible modifications that could be made to try and enhance the network. Grid Search Cross-Validation is useful at finding the best combination of hyperparameters for a given architecture, but not at finding the best architecture out of endless possibilities. Therefore, although this hybrid approach is not without its obvious limitations, it is an improvement on either simply manually attempting to modify a given architecture blindly or feeding a random architecture to Grid Search Cross-Validation and neglecting the possibility of another architecture performing better.

In fact, the best model's performance is practically the same before and after Grid Search Cross-Validation, as it can be observed in table 5. The same is true for the runner-up. Besides the possibility that these extraordinary performances could be the product of serendipity, this anomaly could suggest that the Architecture Exploration phase of the hybrid approach converged on two of the best architectures possible. Furthermore, it could mean that the performance of both models is more contingent upon their architecture than it is upon the specific value of the hyperparameters studied during Grid Search Cross-Validation. In any case, a corollary of both insights is that the hybrid approach yielded better results than simply fine-tuning the model's hyperparameters through Grid Search Cross-Validation.

In light of this analysis and the explained benefits of carrying out this dual approach to a model's development, it is reasonable to conclude that the best model's architecture and parameters are appropriate —if not ideal— for the problem at hand. It is important to highlight that because the model obtained an accuracy of 90% during testing —data it had not seen before—, one can defend that the model has learned to generalize the trends in the data appropriately and that it can be trusted to perform adequately in real-life contexts[13].

To verify the robustness of the model, a simple experimentation process was followed: the dataset was replicated and altered distinctly several times. The alterations consisted of increasing and decreasing the values of the numeric features by one and two standard deviations (for a total of four alterations). All

[13]Of course, there are limitations to this capacity to generalize. The model was trained on a specific sample distribution, and its ability to perform may not be translatable to other distributions. For instance, the model would not be able to perform as well if the outliers had not been removed from the dataset, or some of the preprocessing steps had been omitted.

the other preprocessing steps were held constant. The best model was asked to predict on these full datasets, and its performance was evaluated to analyze its sensitivity.

Table 6 illustrates these results.

Final Model Performance on Full Dataset with Altered Numeric Features

Alteration Size (σ)	**Loss**	**F-2 Score**	**Accuracy**
1	0.591	0.182	0.721
2	0.697	0.162	0.675
-1	0.534	0.180	0.755
-2	0.516	0.189	0.776

Table 6: Sensitivity Analysis of Final Model

The results shown in the table suggest that the model does not perform as well on data that has been altered. The extent of this sensitivity is not very relevant in the context of the problem, however, for although the decrease in accuracy is of about 30%, the model's performance is still well above the benchmark figure of 13%. For a company, being able to predict with 70% accuracy the success of a cold call is still a significant advancement.

4.2 Justification

A comparison of the performance of the best model, the Model 2 MLP, and the benchmark model established earlier, a Support Vector Machine Classifier, is summarized in table 7. Cohen's Kappa was not included because it was not applicable to the Keras model.

Comparison of Benchmark and Final Model Performance

	F-2 Score	**Accuracy**
Benchmark	0.340	0.900
Final Model	0.381	0.905
Performance Delta (%)	*12.06*	*0.55*

Table 7: Comparison of Model Performance

Table 7 indicates that the MLP is more accurate, but by a very slight margin. On the other hand, the MLP's F-2 score is over 10% higher than that of the benchmark model.

From this analysis, we can conclude that the MLP is indeed better than the Support Vector Machine Classifier, even if marginally. Due to the simpler training and fine-tuning process, it would be tempting to state that the SVC is a better approach to solving this problem. However, it must be taken into account that the aim of this project was not only to create a model that could predict accurately whether a call would or would not be successful, but also one that emphasized the importance of avoiding false negatives to maximize the benefit offered to marketing institutions. Because the MLP does this better, it is more applicable to the real-world setting for which it was designed.

Beyond its comparison to the Benchmark model, the best model is a fitting solution to the broad problem targeted in this project. The Multi-Layer Perceptron was capable of identifying the non-linear relationships in the data and predict the success rate of cold calling with 90.5% accuracy and an F-2 Score of 0.38. This performance displays a marked improvement upon the benchmark success rate of cold calling, which stands at 13%.

As a matter of fact, in the context of Direct Marketing and the Compromise Problem faced by companies promoting goods/services, this model is a remarkable solution: it allows companies to exploit the wealth of information they have on clients to minimize sales costs and maximize efficiency in a way that product design or traditional marketing does not. Put differently, companies cannot tailor products specifically to each client's needs, nor can they hope to advertise a general product in a way that is specific to each client. However, by implementing such a model in their marketing practices, they can attempt to sell a general product to a specific client and know with 90% certainty that such an effort will bring value to the company.

This model is not a bullet-proof solution to the Compromise Problem, because it does not provide 100% accuracy and it is focused on a dataset that was cleansed of outliers prior to being processed. However, this project's findings lend weight to the assumption that if a model could predict with a high degree of certainty whether a cold call would be successful based on client samples from a specific dataset, accurately predicting the outcome of calls for clients that were discarded as outliers is only a matter of gathering more samples in those ranges of feature values and training a similar model on those distributions.

V Conclusion

5.1 Free-Form Visualization

Figure 4 synthesizes the stark contrast between a traditional and a MLP-driven Direct Marketing campaign (using data from the UCI Bank Marketing Data Set). The area of the bubbles represents the Cost Per Lead for each.

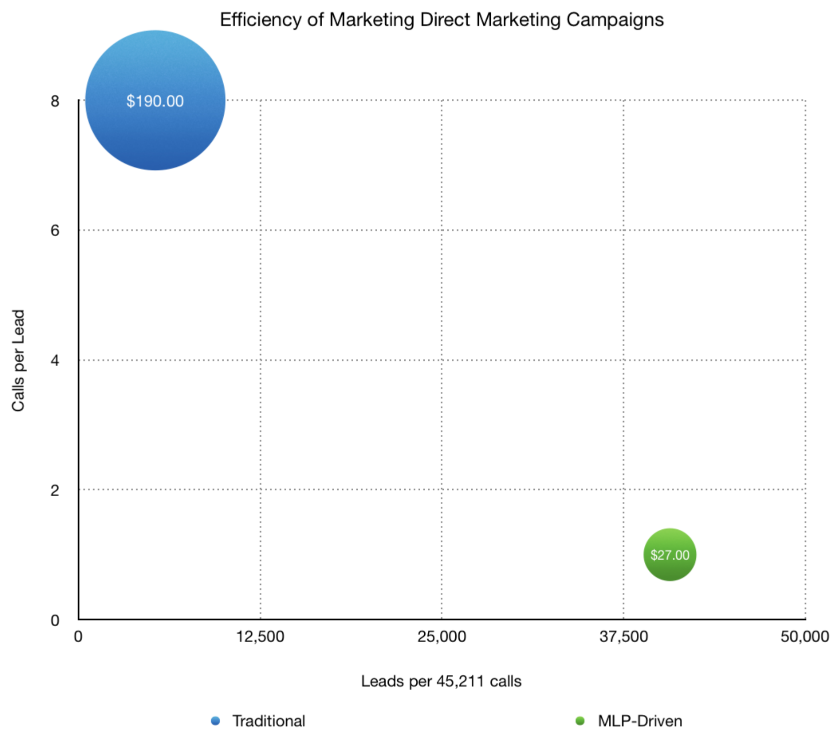


Figure 4: Effect of Data Analytics on Marketing Campaign Efficiency

The plot above highlights the inefficiencies of approaches to cold calling such as the one used in the Bank Marketing Data Set. While only 5,289 leads are obtained from 50,000 calls using traditional marketing techniques, MLP-driven approaches could increase this number to 40,690. Put into perspective, it takes approximately 8 calls for traditional marketing campaigns to obtain a single lead. Using a MLP-driven approach like the one devised in this project, this number could be reduced to approximately 1. Furthermore, at a CPL of $190, it would cost just over $1,000,000 to obtain 5,289 calls. In contrast, a MLP-driven campaign would require only $143,000.Direct Marketing

5.2 Reflection

Project Summary

In this project, a real-life predicament faced by institutions willing to maximize revenue by promoting a service/good to potential customers was identified. A potential solution involving a Multi-Layer Perceptron was proposed, and a dataset containing the Direct Marketing efforts of a financial institution was selected to model this problem. The dataset modeled the problem quite well, since it presented a successful-to-unsuccessful ratio equivalent to real-world statistics.

The dataset was cleaned, and a feature-by-feature inspection led to the elimination and transformation of several features to optimize the model's applicability to real-life scenarios. The data was then preprocessed —outliers were identified and discarded, the numerical features were transformed and normalized, and the categorical features were one-hot encoded— to enhance the model's performance. Principal Component Analysis was used to reduce the dimensionality of the data while retaining 95% of the variance, to facilitate computation.

A Support Vector Machine Classifier was then designed, fine-tuned using Grid Search Cross-Validation, and tested to set a benchmark for the MLP. The MLP was then designed, trained, and fine-tuned using a hybrid approach involving a manual, analytical exploration of MLP architectures and Grid Search Cross-Validation, and tested to observe performance. The MLP was tested for sensitivity to variations in the data, and compared to the benchmark model for evaluation purposes.

It was concluded that the MLP provided an adequate solution to the problem posed at the beginning of the project, and it is believed that it has the potential to profoundly ameliorate the marketing efforts of a commercial institution if it were to be applied in real-life.

Concluding Thoughts

This project provided an opportunity to design and test a Multi-Layer Perceptron from scratch in order to solve a real-world problem. It was extremely interesting to work with data that modeled the problem perfectly, and have the freedom to reflect on the data's features and come up with ways to make it optimal for a model's training. Furthermore, it was a pleasant challenge to slowly build the solution to the predicament, gradually evolving from one dataset to another, from one model to another, iterating through a cycle of observation, experimentation, and analysis, until a satisfactory result could be attained.

Despite being exciting, this project was nonetheless challenging to tackle. Particularly difficult was negotiating the infinite ambiguity involved in designing and training a MLP architecture from scratch. It was complicated to navigate the Architecture Exploration dimension of the hybrid approach because compromises had to be made inevitably, and no matter how many iterations or combinations were tested, there would always be a large *what if...* hovering over the process.

Notwithstanding these challenges, however, the final model attained fits the expectations had for this problem —it promises to improve the performance of call centers dramatically—, and could perfectly be used in real-life. Naturally, different datasets or ranges of values would demand different distributions to optimize performance. Yet this is only a matter of tweaking the process slightly.

5.3 Improvement

In the context of the overarching aim of this project, which is to enhance the practice of marketing, a beneficial improvement that could be implemented is attempting to generalize the predictive capabilities of the MLP to extend it beyond the financial industry.

Presently, the final model is capable of predicting the success rate of cold calling with 90% accuracy, but it requires a large number of client details that could only possibly be obtained by a financial institution. Examples of this are the features *Balance*, *Loan*, and *Default*. To extend this model's capacity to other industries, these features could be removed in favor of other more easily-accessible ones. Pertaining to the client, these could be *Gender*, *Nationality*, or even *Tax Bracket*. Other features that serve as proxies of the economy's state and consumers' purchasing power could be included as well, such as the current Federal Interest Rate, Customer Satisfaction Index, Average Stock Indices, etc. By shifting away from financial features that relate to a bank customer and move towards more general (and accessible) variables that communicate the state of the economy, one could attempt to model the behavior of the more general consumer. This way, the ground would be laid to predict the success rate of cold calling in broader marketing campaigns.

Of course, the accuracy obtained from these more general models are expected to be well below 90%. Despite the possibility of implementing more sophisticated or computationally complex techniques and algorithms —perhaps utilizing Early Stopping or modifying the Weight Decay of the model (techniques that could also be applied to this project's model, but lay beyond the knowledge of the author), or even applying Grid Search CV to tens or hundreds of different architectures to refine the Architecture Exploration process—, modeling so general a behavior gives room to ample error. Nevertheless, for such an endeavor to be successful it needs not achieve 90% accuracy - as long as a model duplicates, triplicates, or quadruplicates the average success rate of 13%, economic gains are there to be reaped by Direct Marketing agencies.

The symbolic conclusion of this project, then, is not that a MLP achieved an accuracy of 90% on predicting whether a call will end in a sale or not. If this model were to be used as a benchmark for further projects, more sophisticated algorithms, techniques, and practices could certainly yield better results. Rather, the bottom line is that some commercial practices are so outdated and lacking data analytics that there is a full spectrum of unidentified inefficiencies which, albeit challenging, can most certainly be solved through Machine Learning.

Works Cited

1. Marketing — Definition of marketing in English by Oxford Dictionaries. (n.d.). Retrieved August 10, 2018 from https://en.oxforddictionaries.com/definition/marketing

2. Lim Chee Hiong, D. (2009). Building a Customer Centric Business for Service Excellence and Competitive Advantage in the Singapore Banking Industry. *University of Nottingham EPrints.* Retrieved August 10, 2018, from http://eprints.nottingham.ac.uk/22667/1/09MBAlixchdl1.pdf

3. Moro, S., Cortez, P., & Rita, P. (2014). A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62, 22-31.

4. Direct Marketing Association. (2013, April 30). DMA 2013 'Statistical Fact Book'.

5. Brownlee, J. (2016, June 07). 8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset. Retrieved from https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

6. Somers, M. J., & Casal, J. C. (2009). Using Artificial Neural Networks to Model Nonlinearity. Organizational Research Methods, 12(3), 403-417. doi:10.1177/1094428107309326

7. Songwon, S. (2006). A Review and Comparison of Methods for Detecting Outliers in Univariate Data Sets. *University of Pittsburgh.*

8. Sherman, P. J. (n.d.). Tips for Recognizing and Transforming Non-normal Data. Retrieved from https://www.isixsigma.com/tools-templates/normality/tips-recognizing-and-transforming-non-normal-data/

9. Kravchenko, A. (n.d.). F-beta score for Keras — Kaggle. Retrieved August 28, 2018, from https://www.kaggle.com/arsenyinfo/f-beta-score-for-keras

Appendix I

Model 1 Architecture

```
# Model 1 architecture
def build_model_1(nodes, penalty, dropout):
    # Achitecture
    model = Sequential()

    model.add(Dense(units=X_train_red.shape[1],
                      input_dim=X_train_red.shape[1], activation='relu'))
    model.add(Dense(nodes, activation='relu',
                      kernel_regularizer=regularizers.l2(penalty)))
    model.add(Dropout(dropout))
    model.add(Dense(nodes + 5, activation='relu',
                      kernel_regularizer=regularizers.l2(penalty)))
    model.add(Dropout(dropout))
    model.add(Dense(nodes + 10, activation='relu',
                      kernel_regularizer=regularizers.l2(penalty)))
    model.add(Dense(1, activation='sigmoid'))

    # Compile
    model.compile(loss='binary_crossentropy',
                  optimizer='rmsprop',
                  metrics=[f_2, 'accuracy'])
    return model
```

Model 2 Architecture

```
# Model 2 architecture
def build_model_2(nodes, penalty, dropout):
    # Architecture
    model = Sequential()
    model.add(Dense(units=X_train_red.shape[1],
                      input_dim=X_train_red.shape[1], activation='relu'))
    model.add(Dense(nodes, activation='relu',
                      kernel_regularizer=regularizers.l2(penalty)))
    model.add(Dropout(dropout))
    model.add(Dense(nodes + 5, activation='relu',
                      kernel_regularizer=regularizers.l2(penalty)))
    model.add(Dropout(dropout))
    model.add(Dense(nodes + 10, activation='relu',
                      kernel_regularizer=regularizers.l2(penalty)))
    model.add(Dropout(dropout))
    model.add(Dense(1, activation='sigmoid'))
```

```
# Compile
model.compile(loss='binary_crossentropy',
              optimizer='adamax',
              metrics=[f_2, 'accuracy'])
return model
```