Contents

Abstract

It is known that….

This work addressed the problem from the perspective of…

It could be shown that…

Henceforth, it should be considered that…

1 Introduction

A U.S. American veterans organization regularly conducts direct marketing campaigns, asking their members for donations (gifts) for the treatment of veterans with spinal injuries. The goal for the organisation is to maximize the net profit from their campaigns.

Only a small proportion of the members give a gift in reply to a campaign, while each letter sent out has a unit cost of 0.68 $US. In order to maximize profit, it is therefore desirable to only mail members who are likely to donate.

The members are grouped, among other criteria, by the recency of their last gift. Of these groups, the so-called *lapsed* donors are of particular interest. These are members who made their last gift to the organisation 13 to 24 months prior to a given campaign. This group is important for two reasons: Firstly, the likelihood of a member donating decreases with the time the member has not donated. Enticing these lapsed donors to give again therefore maintains an active member base. Secondly, the organisation has found that there is a negative correlation between the dollar amount donated and the likelihood to respond to a campaign. This means it is important to include the most unlikely donors in future mailings because if they donate, the net revenue is particularly large. If these unlikely donors would be suppressed from future campaigns, the gains from additional *small dollar* lapsed donors would not offset the losses from the potential *high dollar* donors.

The data at hand was distributed for the purpose of the KDD-CUP of the year 1998[1]. The cup was until recently held yearly under the aegis of the special interest group on Knowledge Discovery and Data Mining (SIGKDD), which itself is part of the Association for Computing Machinery[2] (ACM).

At the time of the competition, 20 years ago, machine learning was in it's second boom phase. Since then, there was one more period of slump before the current boom that started in the mid-2000's (see Goodfellow, Bengio, and Courville (2016)). Both computing power and tools available have undergone a substantial transformation since 1998, when the frontrunners in the KDD-CUP competition were all big players in the field, showcasing their proprietary software. This thesis thus also demonstrates the possibilities that are currently at hand, from powerful open-source toolkits like scikit-learn to cloud computing.

1.1 Task

The task for this thesis was to perform a complete data analysis (data preprocessing, model evaluation, model selction, prediction).

A requirement set by the supervisor of this thesis was that the solution be demonstrated using Python (see Rossum (1995)) as a programming environment.

[1] For an archive of past cups, see SIGKDD - KDD Cup

[2] https://acm.org

1.2 Goal

The ultimate goal was to beat the winner of the original cup in terms of the predicted net revenue for the campaign.

Furthermore, the thesis should support future work on the data set by providing a solid basis especially on the preprocessing of the data.

For reference, the exact problem description from the cup announcement is given in verbatim:

> […], the objective of the analysis will be to maximize the net revenue generated from this mailing – a censored regression or estimation problem. The response variable is, thus, continuous (for the lack of a better common term.) Although we are releasing both the binary and the continuous versions of the target variable (TARGET_B and TARGET_D respectively), the program committee will use the predicted value of the donation (dollar) amount (for the target variable TARGET_D) in evaluating the results. So, returning the predicted value of the binary target variable TARGET_B and its associated probability/strength will not be sufficient. The typical outcome of predictive modeling in database marketing is an estimate of the expected response/return per customer in the database. A marketer will mail to a customer so long as the expected return from an order exceeds the cost invested in generating the order, i.e., the cost of promotion. For our purpose, the package cost (including the mail cost) is $0.68 per piece mailed. KDD-CUP committee will evaluate the results based solely on the net revenue generated on the hold-out or validation sample. The measure we will use is: Sum (the actual donation amount - $0.68) over all records for which the expected revenue (or predicted value of the donation) is over $0.68. This is a direct measure of profit. The winner will be the participant with the highest actual sum. The results will be rounded to the nearest 10 dollars.

1.3 Glossary

In the scope of this thesis, the following naming conventions are used.

Table 1.1: Naming conventions.

Example	One observation of the data, a row in the matrix.
Feature	One variable like *address*, *age* or *time since failure*, a column in the matrix
Generalization Error	Also called out-of-sample error. The error rate on new observations.
Gift	A donation made by a member to the veterans organisation
Hyperparameter	A parameter controlling the learning algorithm. Generally, large (regularization) hyperparameters almost guarantee not overfitting the model at the cost of good performance.
Promotion	A solicitation to a member of the veterans organisation asking for a donation
Regularization	Constraints on the learning algorithm defined by *hyperparameters*. These parameters are of the learning algorithm, not the resulting model.
Target	The solution, or target variable, in the learning dataset. Used in supervised learning to train the algorithm.

2 Experimental Setup and Methods

The general workflow with which the problem was solved is shown in Figure 2.1.

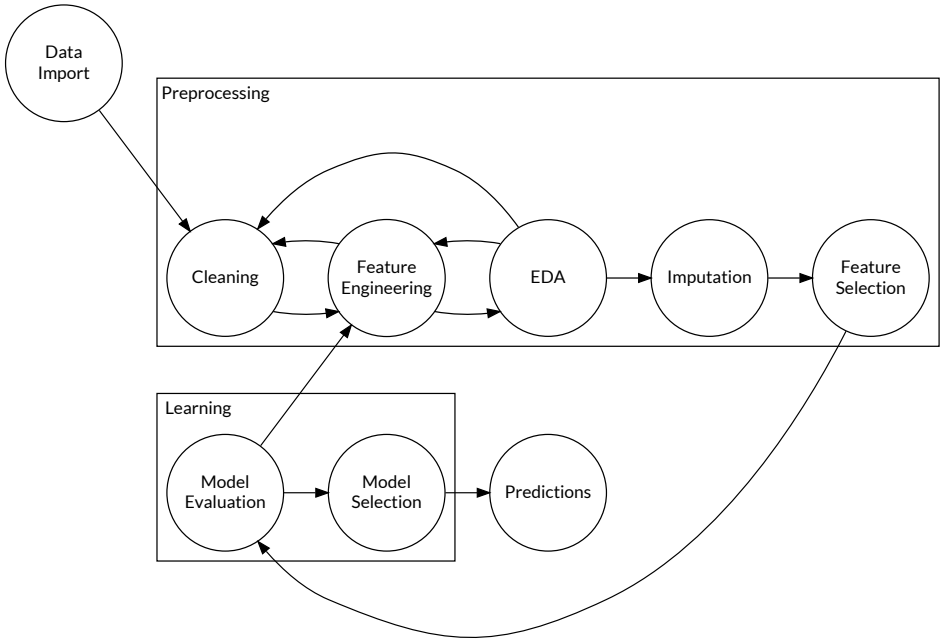


Figure 2.1: Implemented workflow. During import, data was type-cast to usable types. During preprocessing, input errors were corrected, binary features recoded, missing values correclty coded for each feature and categorical data processed manually where necessary. During feature engineering, redundant features were dropped, date features converted to time differences and geographical information acquired to visualize locations of examples.

2.1 Tools Used

The problem itself was solved using the python language with several established data-science packages. Except for the development of a helper packge, most programming was performed in interactive notebooks. The report was written in rmarkdown (see Allaire et al. (2016)) and rendered into several different output formats. All work was tracked in version control.

Some key tools are given below.

2.1.1 Jupyter Notebook

Following the principle of literate programming first proposed by Knuth (1984), Jupyter by Kluyver et al. (2016) provides a client-server solution that enables interactive programming sessions in the browser. Blocks of code producing output including interactive plots can be mixed with formatted text. These notebooks can easily be shared with others or exported in various formats. The individual steps in the machine learning process outlined in Figure 2.1 were developed in several notebooks which are available online[1], enabling the reproduction of the process.

For this thesis, jupyter notebooks were run in the cloud. The results were then exported and integrated in the report.

2.1.2 Pandas

The python package pandas by McKinney (2010) facilitates data loading, inspection and -manipulation. The package is very fast at transforming, filtering and selecting in big data sets.

2.1.3 Scikit-learn

The package scikit-learn by Pedregosa et al. (2011) is a self-contained toolset for machine learning purposes in python. Individual tasks in preprocessing, feature engineering and model training can be combined in pipelines. The pipelines ensure that the same transformation steps are applied to all data sets. The pipelines are first trained (fit) on the learning data and then applied (transform / predict) to the test data. A wide range of contribution and 3rd party packages also implement scikit-learn's API so that they can be integrated.

2.1.4 Rmarkdown

The report was written in Rmarkdown. Analogous to a jupyter notebook, formatted text and code *junks* can be integrated into one document. The R packages `knitr`, Xie (2015) and `bookdown`, Xie (2016) combined with a latex installation can then be employed to generate high-quality pdf documents and/or html versions.

2.2 Data Handling

The complete data is distributed pre-split into a learning and validation data set.

Preprocessing was performed on the learning data set. After preprocessing, the learning data set was split 80/20 into training and validation sets (see Figure 2.2). The training set was used to train different models while the validation set served to tune hyperparameters. The split was performed using a stratified sampling algorithm to preserve the target class frequencies.

[1] see

The validation set was kept back until after model selection. The same preprocessing steps that were applied to the learning data set were then applied to the validation data set and the final prediction performed.

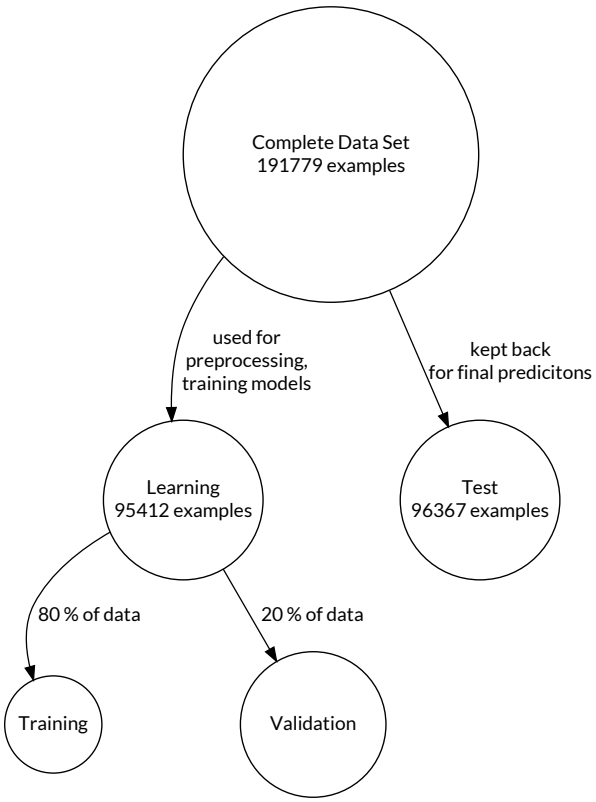


Figure 2.2: Data set use for training and predictions.

2.3 Data Preprocessing

To make data usable for learning algorithms, it generally has to be preprocessed. Preprocessing may encompass fixing input errors, coercing data to correct types, encoding categorical (string) data and dealing with missing values through imputation or removal. The result of this process is an all-numeric data set.

The necessary transformations were determined interactively in jupyter notebooks. Once finalized, the tranformations were implemented in the python package `kdd98`[2]. The package can be used to download and read in raw data and apply all transformations. Each transformation is enclosed in a class that implements scikit-learn's API for `BaseEstimator` and `TransformerMixin`, which allows for the transformations to be applied within a pipeline. Furthermore, the transformers can be trained and then persisted on disk for later application on other data.

The data set can be obtained at the following intermediate steps from `kdd98.data_handler.KDD98DataProvider`:

[2] see

- **raw**, as imported from csv using `pandas.read_csv()`
- **preprocessed**, input errors removed, correct data types for all features, missing at random (MAR) imputations applied
- **numeric**, after feature engineering (encoded categories, date and zip code transformations)
- **imputed**, with NaN-values replaced by modelled values
- **all-relevant**, filtered down to a set of relevant features

2.3.1 Cleaning

The cleaning stage of preprocessing encompassed the following transformations:

- Removing 'noise': Input errors, inconsistent encoding of binary / categorical features
- Dropping constant and sparse (i.e. those where only few examples have a value set) features
- Imputation of values missing at random (MAR)

MAR values in the sense of Rubin (1976) are missing conditionally on other features in the data. For example, there are three related features from the promotion and giving history: *ADATE*, the date of mailing a promotion, *RDATE*, the date of receiving a donation in response to the promotion and *RAMOUNT*, the amount received. For missing *RAMOUNT* values, we can check if *RDATE* is non-missing. If *RDATE* is missing, then the example most likely has not donated and we can set *RAMOUNT* to zero. If, on the other hand, both date features have a value, *RAMOUNT* is truly missing.

The transformations applied can be studied in the jupyter notebook *1_Preprocessing.ipynb*[3].

2.3.2 Date features

There were 53 date features specified in the documentation. The specified format for these features is 'yymm', the two-digit year followed by the two-digit month.

Several values in the feature DOB (date of birth) are only three digits long. Comparing these values with the corresponding example's values for feature AGE shows that, considering the reference date of June 1997, it is very likely that the 3-digit DOB's are lacking a leading zero (see Figure 2.3).

All these values were therefore prepended by a zero.

2.3.3 Further sanitizations

Several features needed recoding and sanitizing. Information in the data set documentation was used to determine necessary transformations. The transformations are shown in 2.1.

[3] see

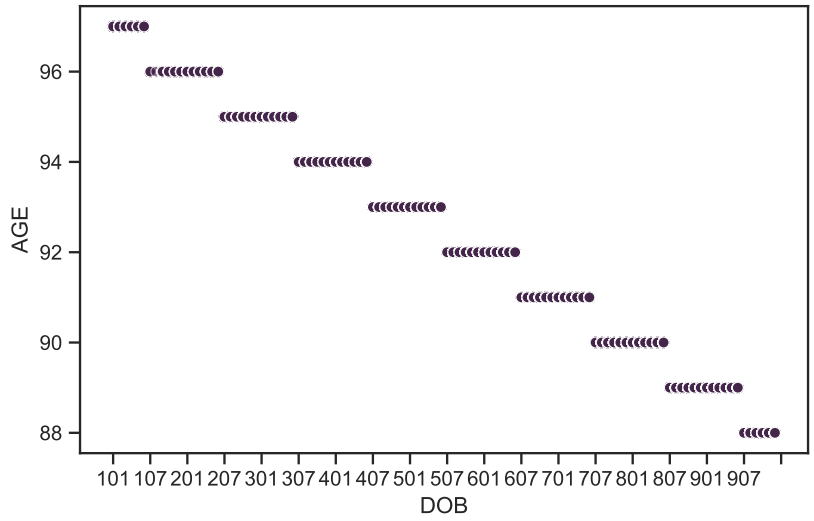


Figure 2.3: Values for date of birth (DOB) with one missing digit versus age. We observe jumps in July of each year, this is because the reference date for age is 1997-06-01. By looking at the ages, it is evident that these values indeed lack a leading zero to put them in the correct decade.

2.3.4 Binary Features

There are several codings for the different binary features present in the data set. All were recoded to *1* / *0* with missing data coded as not a number (NaN). The recoding was done through a custom `Transformer` class (see Appendix 6.2.2) derived from scikit-learn's `TransformerMixin`.

Table (2.2) shows the original coding of the 29 binary features in the raw data set. There are two classes of binary features that have blanks as the *False* value. Even though the data set documentation states that blanks should be treated as missing values, in these two classes blanks were interpreted as *False*. For the remaining binary features, blanks were interpreted as missing and coded as `NaN`.

The feature *NOEXCH* had both `X` and `1` for True in the data. This was fixed by replacing all occurrences of `X` by `1`.

2.3.5 Multi-byte Categorical Features

Several features combine related information into bytewise codes. These codes were split into individual features. A custom child class of `scikit-learn.TransformerMixin` was written for this purpose (see Appendix 6.2.2).

The donation history contains recency / frequency / amount (RFA) features for each previous mailing. The dataset already contained one mailing (number 2) where this code was split into individual features. The remaining RFA codes were split also.

Table 2.1: Overview of trivial data transformations applied.

	Feature explanation	Operation	Reason
ZIP	U.S. zip code	Remove trailing hyphen	A hyphen after a 5-digit zip most likely stems from incomplete 'ZIP+4' codes^[see (USPS: ZIP+4 Code)[https://about.usps.com/publications/p
MDMAUD_*	Major donor matrix features	Replace X with NaN	Encode NaN as specified by the data set documentation.
TARGET_B	Binary target indicating whether an example has donated or not.	Set to 1 when TARGET_D is $ ew$ 0	When a dollar amount (TARGET_D) was donated, the binary indicator target has to be 1.
RFA_*	Multi-byte categorical. Recency, Frequency, Amount.	Set to NaN if length is $ eq$ 3	The values in these features have to be of length 3, each byte is one categorical.
NOEXCH	NOEXCH: Do not exchange address flag	Recode X to 1	Both X and 1 mean 'True'
RAMNT_*	RAMNT_*: Amount donated for a certain campaign.	Set to NaN if corresponding RDATE_* (date of donation reception) is not missing, else set to zero.	Avoid NaN values if possible.

Table 2.2: Original coding of binary features.

True	False	Features
X	blank	PEPSTRFL, MAJOR, RECINHSE, RECP3, RECPGVG, RECSWEEP
Y	N	COLLECT1, VETERANS, BIBLE, CATLG, HOMEE, PETS, CDPLAY, STEREO, PCOWNERS, PHOTO, CRAFTS, FISHER, GARDENIN, BOATS, WALKER, KIDSTUFF, CARDS, PLATES
1	0	NOEXCH, HPHONE_D, TARGET_B
E	I	AGEFLAG
H	U	HOMEOWNR
B	blank	MAILCODE

2.3.5.1 Ordinal Features

Ordinal features were recoded using a custom child classes of `scikit-learn.TransformerMixin` (see Appendix 6.2.2).

2.3.6 Feature Engineering

During feature engineering, all non-numeric (i.e. categorical) features were encoded into numeric values. Also, several features were transformed to better usable representations. Care was taken to keep the dimensionality of the data set as low as possible.

The result of this transformation step was an all-numeric data set usable for downstream learning. The transformations applied in feature engineering are described in detail in the jupyter notebook *2_Feature Engineering.ipynb*[4].

2.3.6.1 Dates

All date features were transformed into time differences against a reference date according to Table 2.3.

Table 2.3: Transformation of dates to time differences

	Feature Explanation	Reference date	Unit
DOB	Date of birth	1997-06-01 (date of most recent campaign)	Years
RDATE	Month when donation was received	ADATE (sending date of the corresponding campaign)	Months
LASTDATE	Most recent donation prior to last campaign	1997-06-01	Months
MINRDATE	Date of smallest donation	1997-06-01	Months
MAXRDATE	Date of highest donation	1997-06-01	Months
MAXADATE	Date of the most recent promotion received	1997-06-01	Months

2.3.6.2 Zip Codes

U.S. zip codes were transformed into coordinates (latitude, longitude of the centroid for a given zip) using zip code tabulation data from United States Census Bureau (2018). Several of the examples are army members, identifiable through their values in feature 'STATE'. For these zip codes, no geographical data is available. These example's latitude and longitude were set to the coordinates of the pentagon, the U.S. department of defense. The 2018 zip code tabulation data was missing several zip codes that existed in 1997, at the time of the campaign. These missing zip codes were looked up on the fly using a web service[5]

[4] see
[5] HERE geocoding

2.3.6.3 Categorical Encoding

Categorical data was encoded using three different methods. For nominal features, the encoding method was chosen with respect to the number of levels in the categories. Nominal features with less than 10 levels were one-hot encoded. Those with more levels were binary-encoded. Ordinal features were consistently encoded to integer values.

A one-hot encoded feature with n levels is transformed into n new binary features, each feature representing one of the original levels. For each example, there can be at most one `true` value in these new features (denoting which category the example had in the original feature). If the original value was missing, all new features are set to missing.

Binary encoding first assigns an ordinal value to each category. These integer values are then binary encoded. For each binary digit, one new feature is created. Compared to one-hot encoding, the dimensionality increases less with this method.

As an example, the US states are represented by a categorical variable with 52 levels. While one-hot encoding would result in 52 new features, we can encode 52 values with only 5 binary digits, thus adding 5 instead of 52 new features.

2.3.7 Imputation

As required by the cup documentation, missing values were imputed. Instead of choosing a simple approach like replacing with the feature's mean or median, a modelling approach was chosen. Package `fancyimpute` provides an iterative imputation algorithm for this purpose. Features are ordered by the fraction of missing values [FINISH EXPLANATION]

Missing values were imputed using a k-Neaerest Neighbors (kNN) algorithm provided in the python package `missing_py` Troyanskaya et al. (2001). For each example with a missing value in a specific feature, $k = 3$ nearest neighbors that have a value for the feature are searched for and the missing value imputed with the metric *distance*.

2.3.8 Feature Selection

One of the biggest caveats in machine learning is the infamous "Curse of Dimensionality" coined by Bellman, Corporation, and Collection (1957). The curse comes from the fact that with an increasing number of dimensions, the required number of examples grows exponentially. In the area of machine learning, high dimensionality frequently leads to an overfitting of the training data, meaning that the generalisation error is unacceptably big (see Goodfellow, Bengio, and Courville (2016)).

It is therefore beneficial to reducde the data set dimensionality while preserving as much relevant information as possible. A method to deal with the problem is called boruta, introduced by Kursa, Rudnicki, and others (2010). The algorithm was found to perform very well regarding selection of relevant features in B. Kursa and Rudnicki (2011). It works sequentially and removes features found to be less relevant at each iteration. By doing so, it solves the so-called all-relevant feature problem. The algorithm is actually a wrapper function around a random forest classifier. A random forest classifier is fast, can usually be run without parameters and returns an importance measure for each feature.

In short, the alogrithm works as follows:

1. The input matrix **X** of dimension n x p is extended with p so-called *shadow features*. The shadow features are permuted copies of the features in **X**. They are therefore decorrelated with the target.
2. On the resulting matrix $\mathbf{X}^*$, a random forest classifier is trained and the Z-scores ($\frac{loss}{sd}$) for each of the $2p$ features calculated.
3. The highest Z-score among the shadow features $MZSA$ is determined.
4. All original features are compared against $MZSA$ and those features with a higher score selected as important.
5. With the remaining features, a two-sided test for equality of the Z-scores with $MZSA$ is performed and all features with significantly lower score are deemed unimportant.
6. All shadow copies are removed, go to step 1.

The algorithm terminates when all attributes are marked as either important or not important or when the maximum number of iterations is reached.

For this thesis, a python implementation[6] was used. In effect, it is a port of the original R package by Kursa, Rudnicki, and others (2010) which plugs into scikit-learn.

2.4 Model Evaluation

During model evaluation, several algorithms were trained and their performance compared.

Randomized grid search with 10-fold cross validation was used to optimize hyperparameters for the algorithms.

The best parameter combination was determined using *recall* as the performance metric (see 2.4.2).

2.4.1 Considered Algorithms

The algorithms were chosen so as to cover a wide range of approaches to learning. A short introduction of each algorithm, along with the hyperparameters used for regularization is given below.

2.4.1.1 Random Forest

2.4.1.2 Gradient Boosting Machine

Boosting is a method that can be applied to any learning algorithm. It was introduced by Freund and Schapire (1997) in the form of the algorithm AdaBoost.M1, intended for classification problems. The main idea behind boosting is to train a set of weak classifiers which are only slightly better than a random decision. The predictions of the individual weak classifiers are then combined into a majority vote.

Gradient boosting extends on this idea. New trees are added in the direction of the gradient of the loss function.

[6] see scikit-learn-contrib/boruta_py

For this thesis, the package `XGBoost` by Chen and Guestrin (2016) was used. The general principle of gradient descent is given below.

Assume we have a data set with n examples and m features: $D = \{\{\mathbf{x_i}, y_i\}\}(|D| = n, \mathbf{x_i} \in \mathbb{R}^m, y_i \in \mathbb{R})$. The implementation uses a tree ensemble using K additive functions (regression trees) to predict the outcome for an example in the data.

$$\hat{y}_i = \phi(\mathbf{x_i}) = \sum_{k=1}^{K} f_k(\mathbf{x_i}), f_k \in F \tag{2.1}$$

where $F = \{f(\mathbf{x}) = w_{q(x)}\}(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T)$ is the space of regression trees. T is the number of leaves in a tree, q is the structure of each tree, mapping an example to the corresponding leaf index. Each tree f_k has an independent structure q and weights w at the terminal leafs. An example is classified on each tree in F and the weights of the corresponding leafs are summed up to calculate the final prediciton.

For learning the functions in F, the following loss function is minimized:

$$L(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y_i}) + \sum_{k}^{K} \Omega(f_k) \tag{2.2}$$

Here, l is a differentiable, convex loss function that measures the difference between predictions and true values. Since l is convex, we are guaranteed to find a global minimum. $\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2$ is a penalty on the complexity of the trees to counter over-fitting.

For learning the model, an additive approach is used. At each iteration t, the tree f_t that improves the model most is added. For this, we add $f_t(\mathbf{x_i})$ to the predictions at $t-1$.

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{t-1} + f_t(\mathbf{x_i})) + \Omega(f_t) \tag{2.3}$$

To find the best f_t to add, the gradients finally come into play. With g_i and h_i the first- and second-order gradient statistics of l, the loss function becomes:

$$\tilde{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{t-1} + g_i f_t(\mathbf{x_i}) + h_i f_t^2(\mathbf{x_i})) + \Omega(f_t) \tag{2.4}$$

Regularization

Apart from the penalization of complicated trees that is implicitly applied in Eq. (2.2), regularization was controlled with the following hyperparameters:

- Shrinkage: A learning rate < 1.0 decreases step size for the gradient descent, helping convergence. The number of estimators f_k has to be increased for small learning rates in order for the algorithm to converge.
- Early stopping: Based on an evaluation set, learning stops when no improvement on the performance metric is made for a fixed number of steps.

- Column Subsampling: A random sample from the m features of size $s, s < m$ is drawn for each f_k, countering overfitting and speeding up learning.

2.4.1.3 GLMnet

The GLMnet is an implementation of a generalized linear model (GLM) with penalized maximum likelihood by Hastie and Qian (2014). For the binary target in the data, a logistic regression was chosen. Regularization is achieved through L^2 and L^1 penalties (i.e. ridge and the lasso or their combination known as elastic net).

The logistic regression model for a two-class response $G = \{0, 1\}$ with target $y_i = I(g_i = 1)$:

$P(G = 1|X = x) = \frac{e^{\beta_0+\beta^T x}}{1+e^{\beta_0+\beta^T x}}$ or, in the log-odds transformation: $log\frac{P(G=1|X=x)}{P(G=0|X=x)} = \beta_0 + \beta^T x$.

The loss function is:

$$\min_{(\beta_0,\beta)\in\mathbb{R}^{m+1}} - \left[\frac{1}{n}\sum_{i=1}^{n} y_i(\beta_0 + x_i^T\beta) - log(1 + e^{\beta_0+x_i^T\beta})\right] + \lambda\left[\frac{(1-\alpha)}{2}||\beta||_2^2 + \alpha||\beta||_1\right] \quad (2.5)$$

It's first term is the negative log-likelihood for the examples in the data. The second term is the elastic-net penalty. λ controls the amount of penalization. Parameter $\alpha, 0 < \alpha < 1$ controls the elastic-net penalty. For $\alpha = 0$, it is pure ridge, for $\alpha = 1$ pure lasso.

For learning, glmnet tries many different λ values for a given α. Each λ is then evaluated through cross validation.

** Regularization **

- The parameter α parametrizes the elastic net.
- The scoring method for cross validation can be chosen (log-loss, classification error, accuracy, precision, recall, average precision, roc-auc)

2.4.1.4 Multi Layer Perceptron (Neural Network)

2.4.1.5 Support Vector Machine

2.4.2 Performance Metrics

For classification problems, the confusion matrix (see Table 2.4) helps in constructing several performance measures. If we predict an event correctly, it is a *true positive* (TP), predicting an event if there is none, it is a *false positive* (FP). A *false negative* (FN) occures when predicting no event when there was one and a *true negative* (TN) occurs when no event was predicted correctly.

From the confusion matrix, several metrics can be deduced. The definitions of some often-used metrics are given below:

Table 2.4: Definition of the confusion matrix

		True value	
		Event	No Event
Predicted	**Event**	TP	FP
	No Event	FN	TN

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$
$$\text{Sensitivity} = \frac{TP}{TP + FN}$$
$$\text{Specificity} = \frac{TN}{TN + FP}$$
$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{F1 score} = \frac{2TP}{2TP + FP + FN}$$

In literature the following synonyms are found:

- Precision: Positive predictive value (PPV)
- Sensitivity: Recall, True positive rate (TPR), hit rate
- Specificity: Selectivity, True negative rate (TNR)

It is evident that sensitivity measures the proportion of the predicted events from all events present.

Analogous, specificity measures the proportion of correctly predicted no events from all no events present.

3 Data

The data set, which is freely available online[1] contains information on a subset of the turnout of a direct mailing addressed to 3.5 million members of a US American veterans organisation for the purpose of a fundraising campaign that was conducted in 1997. The data set contains all donors with a *lapsed* donation status, meaning their last donation was made between 13 and 24 months prior to the 1997 campaign.

The data is provided in two sets, of which one is intended for learning, the other for validation. The features are identical between the two except for the target that has been stripped from the validation set.

In this section, the *learning* data set will be characterized.

3.1 General Structure

The dimension of the input data **D** is $n = 95412$ examples by $m = 481$ features. The target y is of dimension n x 2. Of the features, one was used as the index, resulting in $\mathbf{X_{learn}}$ with $m^* = 478$ explanatory features.

There are four blocks of related information in the data:

- Member database: Personal particulars, member status features, 85 features
- US census 1990, 286 features
- Promotion history: History of past mailings sent to a member and response patterns, 97 features
- Giving history: History of past donations from a member (summary statistics), 13 features

3.2 Exploratory Data Analysis

The detailed analysis can be studied online in the corresponding jupyter notebook[2]. The findings are shown here.

[1] See UCI Machine Learning Repository: KDD Cup 1998 Data
[2] KDD-CUP98: EDA notebook

3.2.1 Data types

An analysis of the dataset dictionary (see 6.4) reveals the following data types:

- Index: CONTROLN, unique record identifier
- Dates: 48 features in yymm format.
- Binary: 30 features
- Categorical: 90 features
- Numeric: 286

The data set structure after import into a `pandas.DataFrame` object is shown in Table 3.1. Since these numbers do not match with the documented data types, it is evident that several features will need to be transformed.

Table 3.1: Data types after import of raw csv data

	Data content	Number of features
Integer	Discrete features, no missing values	297
Float	Continuous features and discrete features with missing values	49
Categorical	Nominal and ordinal features	24
Object	Features with alphanumeric values	110
Total		480

3.2.2 Targets

Of the two targets, one is binary (TARGET_B), the other discrete (TARGET_D). The former indicates whether an example has donated in response to the current promotion. The latter represents the dollar amount donated in response to the current promotion.

The distribution of *TARGET_D*, excluding non-donors, is shown in Figure 3.1. Evidently, most donations are small. The range of donations is between 1 and 200 $ with the 50-percentile at 13 $. The most frequent donation amount is 10 $. There are a few outliers for donations above 100 $.

As can be seen in Figure 3.2, the target is imbalanced. Only about 5 % of examples have donated. This poses a challenge in model training because there is a high risk of overfitting.

3.2.3 Skewness

Most features are skewed, although several are relatively symmetric.

Figure 3.2.3 gives an abstract idea of the skewness of (numerical) features. The confidence bound gives the $\alpha = 5\%$ bound for a normal distribution. Evidently, no feature was found to be strictly normally distributed, although several features are relatively symmetric.

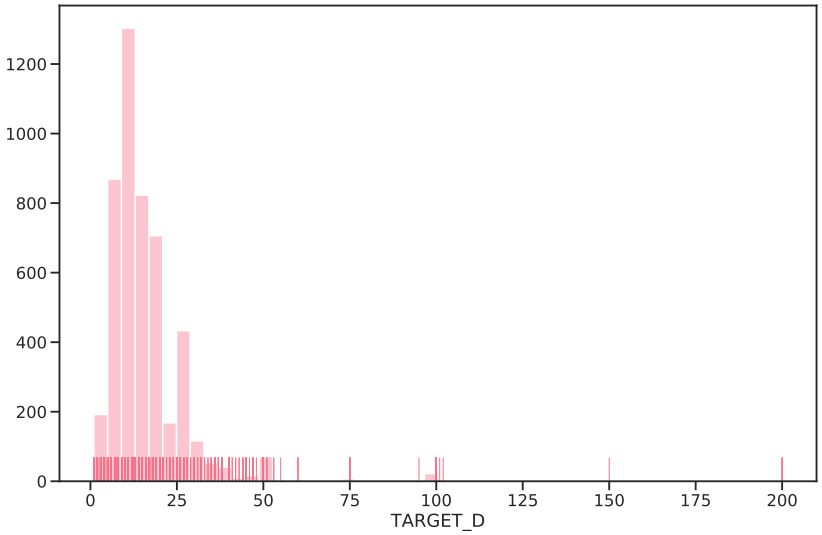


Figure 3.1: Distribution of TARGETD. The donation amount in US dollar has a discrete distribution. Most donations are below 20 dollar, peaks are visible at 50, 75 and 100 dollar, while the maximum donation amount is 200 dollar.

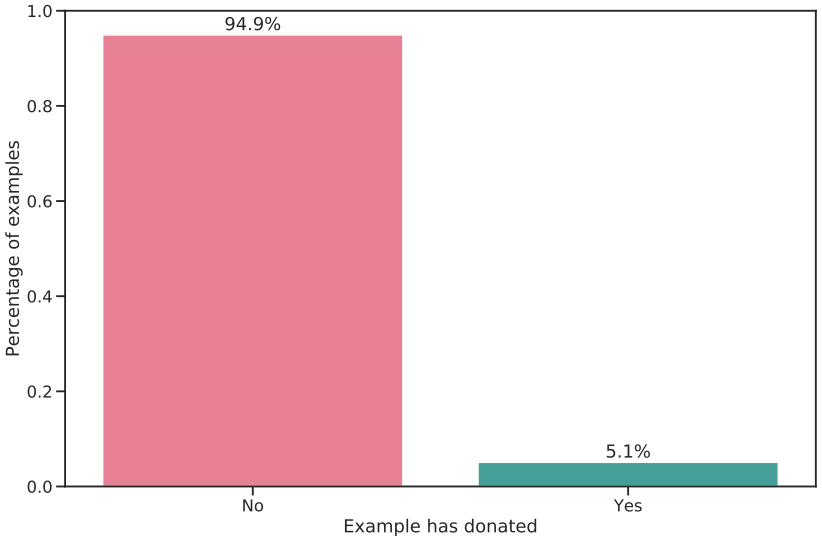
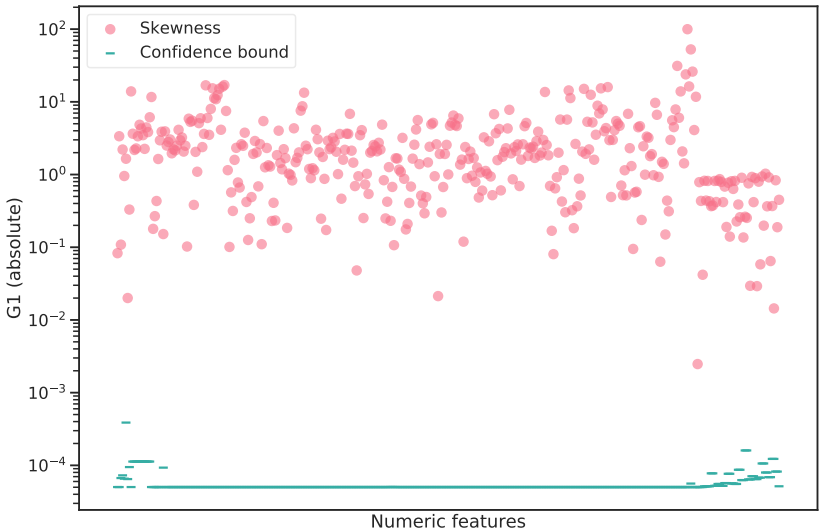


Figure 3.2: Ratio of donating examples. Less than 6 percent have made a donation.

Skewness was measured with `pandas.skew()`, which uses the Fisher-Pearson standardized moment coefficient $G_1 = \frac{\sqrt{n(n-1)}}{n-2} \frac{1}{n} \frac{\sum_{i=1}^{n}(x_i - \bar{x})^3}{s^3}$. Here, the term in the denominator is the sample standard deviation.

\begin{figure}



{

}

\caption{Absolute values of the Fisher-Pearson standardized moment coefficient (G1) for all numeric features contained in the dataset. The confidence bound indicates the $\alpha = 5$ % bound for the skewness of a normal distribution for any given feature. It is evident that no feature passes.} \end{figure}

Looking at the 9 least skewed features (Figure 3.3), we find distributions that resemble normal or uniform distributions, or balanced binary features.

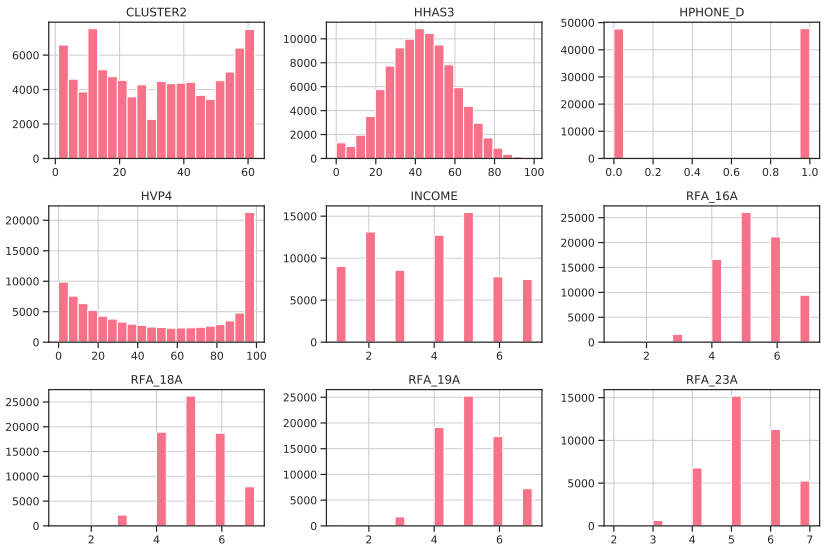


Figure 3.3: The 9 least skewed features. Skewness metric: adjusted Fisher-Pearson standardized moment coefficient.

The 9 most skewed features (Figure 3.4) show heavily right-skewed distributions which are the result of outliers.

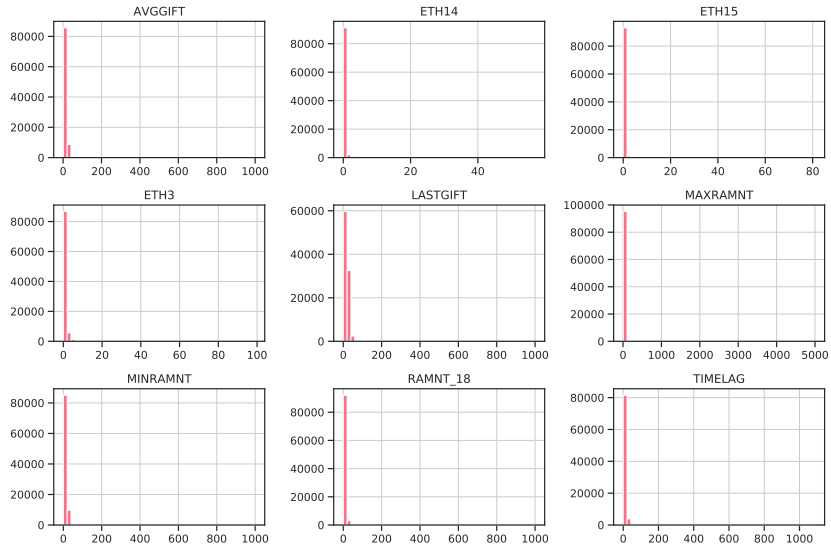


Figure 3.4: The 9 most skewed features. Skewness metric: adjusted Fisher-Pearson standardized moment coefficient.

3.3 Preprocessing

The dataset contains input errors (noisyness), features with datatypes that are impractical to work with (dates, categorical features) and redundant information. Furhtermore, many features contain missing values.

Noisy and categorical features were processed by the author manually before furhter evaluation of the data. Handling of missing values, zero variance and sparse features was carried out through scikit-learn preprocessors.

3.3.1 Noisy data

There were two types of noisyness that were treated manually:

- Binary features with a mixture of 0, 1 and other codes Binary features were all recoded to $\{\text{True}, \text{False}\}$, preserving missing values. 1 was always set to True and 0 was coded False. Specific mappings for True/False values given in the data dictionary per binary field were respected accordingly. As per the data dictionary, ' ' was interpreted as False for some features. For all other features, ' ' was interpreted as missing.
- Dates are expected in *mmyy* as per the dataset dictionary. For several date features, one digit was missing [EXPLAIN HOW FIXED].
- Zip codes: Input errors, inconsistent data (alphanumeric values)

3.3.2 Constant features

- Per the cup's documentation, features with zero variance have to be excluded.

3.3.3 Missing values / sparse features

- Character features: ' ',"
- Numeric features: ' ','','.'
- Missing values are to be kept in the dataset for learning. Approriate methods for imputation are to be employed (median, mean, mode, modeled, …)
- Exception: Features with more than 99.5 % missing values are to be dropped
- Features with a sparse distribution are to be dropped [DEFINITION???]

3.3.4 Categorical features

Several variables are aggregated into byte-wise codes (referred to as *symbolic* fields in the data dictionary) that need to be spread out across separate variables.

Most machine-learning methods require strictly numerical data [REFERENCE]. Several methods exist to transform categorical (string-) values into a usable format. These include:

- One-hot tranformation: Creating dummy variables for each category level. This approach greatly increases dimensionality, which is both more resource-intensive and prone to overfitting.
- Ordinal encoding: The categorical targets are transformed to ordinal values (integer numbers). This preserves dimensionality, but the algorithm chosen to assign the ordinal levels can introduce unwanted effects (an implied similarity based on closeness of the ordinal variables) Feature hashing: The individual values are hashed into a value of fixed length.
- Hashing: [DEFINITION]

3.3.5 Feature Engineering

3.3.6 Feature Selection

-> Selecting the most useful features

3.3.7 Feature Extraction

-> Trying to group correlated features into one (dimensionality reduction). Unsupervised learning.

4 Model Learning

Workflow:

Split training dataset 80/20 into training* and test

1. Train on training* dataset
2. Adjust hyperparameters on validation set
3. Run on test dataset to get generalisation error estimate

5 Results and Discussion

6 Conclusions

6.1 Comparison with Cup winners

The KDD-CUP committee evaluated the results based on the net revenue generated on the validation sample. The measure used was the sum (the actual donation amount - $0.68) over all records for which the expected revenue (or predicted value of the donation) is over $0.68. This measure is simple, objective and a direct measure of profit. Table 2 depicts the results. The participants are listed based on the last column.

Table 6.1: Top five of the KDD-CUP participants. N* denotes the number for which the predicted donation amount is > 0.68. $Sum is the total profit, meaning the donation minus 0.68$ for each example.

		Amount, $				
	N*	Min	Mean	Std	Max	Sum
GainSmarts	56330	-0.68	0.26	5.57	499.32	14712
SAS	55838	-0.68	0.26	5.64	499.32	14662
Quadstone	57836	-0.68	0.24	5.66	499.32	13954
CARRL	55650	-0.68	0.25	5.61	499.32	13825
Amdocs	51906	-0.68	0.27	5.69	499.32	13794

References

Allaire, J, Joe Cheng, Yihui Xie, Jonathan McPherson, Winston Chang, Jeff Allen, Hadley Wickham, Aron Atkins, Rob Hyndman, and Ruben Arslan. 2016. *Rmarkdown: Dynamic Documents for R. R Package Version*. Vol. 1.

Bellman, R., Rand Corporation, and Karreman Mathematics Research Collection. 1957. *Dynamic Programming*. Rand Corporation Research Study. Princeton University Press. https://books.google.de/books?id=wdtoPwAACAAJ.

B. Kursa, Miron, and Witold Rudnicki. 2011. "The All Relevant Feature Selection Using Random Forest," June.

Chen, Tianqi, and Carlos Guestrin. 2016. "Xgboost: A Scalable Tree Boosting System." In *Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, 785–94. ACM.

Freund, Yoav, and Robert E Schapire. 1997. "A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting." *Journal of Computer and System Sciences* 55 (1): 119–39.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Hastie, Trevor, and Junyang Qian. 2014. "Glmnet Vignette." *Retrieve from Http://Www. Web. Stanford. Edu/~ Hastie/Papers/Glmnet_Vignette. Pdf. Accessed September* 20: 2016.

Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, et al. 2016. "Jupyter Notebooks – a Publishing Format for Reproducible Computational Workflows." Edited by F. Loizides and B. Schmidt. IOS Press.

Knuth, Donald Ervin. 1984. "Literate Programming." *The Computer Journal* 27 (2): 97–111.

Kursa, Miron B, Witold R Rudnicki, and others. 2010. "Feature Selection with the Boruta Package." *J Stat Softw* 36 (11): 1–13.

McKinney, Wes. 2010. "Data Structures for Statistical Computing in Python." In *Proceedings of the 9th Python in Science Conference*, edited by Stéfan van der Walt and Jarrod Millman, 51–56.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research* 12: 2825–30.

Rossum, G. van. 1995. "Python Tutorial." CS-R9526. Amsterdam: Centrum voor Wiskunde en Informatica (CWI).

Rubin, Donald B. 1976. "Inference and Missing Data." *Biometrika* 63 (3): 581–92.

Troyanskaya, Olga, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B Altman. 2001. "Missing Value Estimation Methods for Dna Microarrays." *Bioinformatics* 17 (6): 520–25.

United States Census Bureau. 2018. “2018 U.s. Gazetteer Files.” 2018. https://www.census.gov/geographies/reference-files/2018/geo/gazetter-file.html.

Xie, Yihui. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC.

———. 2016. *Bookdown: Authoring Books and Technical Documents with R Markdown*. Boca Raton, Florida: Chapman; Hall/CRC.

Appendix

6.2 Code

All code is also available online at github.com/datarian/master-thesis-code

6.2.1 Preprocessing

6.2.1.1 Date parser

```
def dateparser(date_features):

    def fix_format(d):
        if not pd.isna(d):
            if len(d) == 3:
                d = '0'+d
        else:
            d = pd.NaT
        return d

    def fix_century(d):
        ref_date = App.config("reference_date")
        if not pd.isna(d):
            try:
                if d.year > ref_date.year:
                    d = d.replace(year=(d.year-100))
            except Exception as err:
                logger.warning("Failed to fix century for date {}, reason: {}".format(d, err))
                d = pd.NaT
```

```
        else:
            d = pd.NaT
        return d

    try:
        date_features = [fix_century(pd.to_datetime(fix_format(d),
                                     format="%y%m", errors="coerce"))
                         for d in date_features]
    except Exception as e:
        logger.warn("Failed to parse date array {}.\nReason: {}".format(date_features, e))
    return date_features
```

6.2.2 Transformers

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 24 10:18:44 2018
@author: Florian Hochstrasser
"""

import copy
import datetime
import hashlib
import logging
import sys

import numpy as np
import pandas as pd
from dateutil import relativedelta
from dateutil.rrule import MONTHLY, YEARLY, rrule
from sklearn.base import BaseEstimator, TransformerMixin
```

```
from category_encoders import OrdinalEncoder

# Set up the logger
logging.basicConfig(filename=__name__+'.log', level=logging.ERROR)
logger = logging.getLogger(__name__)


__all__ = ['DropSparseLowVar',
           'BinaryFeatureRecode',
           'MultiByteExtract',
           'RecodeUrbanSocioEconomic',
           'DeltaTime',
           'MonthsToDonation']


class DropSparseLowVar(BaseEstimator, TransformerMixin):
    """ Transformer to drop:

    * low variance
    * sparse (abundance of NaN)
    features.

    Parameters:
    ___________

    * var_threshold float
        Defines the threshold for the variance below which columns
        are interpreted as constant.
    * sparse_threshold loat
        Defines the threshold (percentage) of NaN's in a column, anything
        having greater than this percentage NaN's will be discarded.
    """
```

```
def __init__(self, var_threshold=1e-5, sparse_threshold=0.1,
             keep_anyways=[]):
    """
    Removes features with either a low variance or
    those that contain only very few non-NAN's.

    Parameters:
    ___________

    var_threshold: float. Anything lower than this
                   is considered constant and dropped.
    sparse_threshold: Minimum percentage of non-NaN's needed to keep
                      a feature
    keep_anyways: List of regex patterns for features to keep regardless of
        variance / sparsity.
    """
    self.thresh_var = var_threshold
    self.thresh_sparse = sparse_threshold
    self.feature_names = []
    self.drop_names = []
    self. is_transformed = False
    self.keep_anyways = keep_anyways

def fit(self, X, y=None):
    assert isinstance(X, pd.DataFrame)
    nrow = X.shape[0]

    keep_names = set()
    for search in self.keep_anyways:
        print(X.filter(regex=search).columns)
        keep_names.update(X.filter(regex=search).columns)
```

```
        sparse_names = set(
            [c for c in X if X[c].count() / nrow >= self.thresh_sparse]) - keep_names

        low_var_names = set([c for c in X.select_dtypes(
            include="number") if X[c].var() <= self.thresh_var]) - keep_names

        self.drop_names = list(sparse_names.union(low_var_names))
        print("Constant features: " + str(low_var_names))
        print("Sparse features: " + str(sparse_names))
        print("Keep anyways features: " + str(keep_names))
        print(self.drop_names.sort())
        return self

    def transform(self, X, y=None):
        X_trans = X.copy()
        X_trans = X_trans.drop(columns=self.drop_names)
        self.feature_names = X_trans.columns
        self.is_transformed = True
        return X_trans

    def get_feature_names(self):
        if not isinstance(self.is_transformed, list):
            raise ValueError("Must be transformed first.")
        return self.feature_names


class MultiByteExtract(BaseEstimator, TransformerMixin):
    """
    This is a transformer for multibyte features. Each byte in such
    a multibyte feature is actually a categorical feature.

    The bytes are spread into separate categoricals.
```

```
Params:
_______

field_names: A list with the new names for each byte
                that is to be spread

impute: False means missing / malformatted entries will be coded NaN
        If a value is passed, fields will be filled with that instead.

drop_orig: Whether to drop the original multibyte feature or not.
"""

def __init__(self, field_names, impute=False, drop_orig=True):
    self.field_names = field_names
    # determines how many bytes to extract
    self.sigbytes = len(self.field_names)
    self.impute = impute
    self.drop_orig = drop_orig
    self.feature_names = []
    self.is_transformed = False

def fit(self, X, y=None):
    assert isinstance(X, pd.DataFrame)
    self.feature_names = list(X.columns)
    return self

def _fill_missing(self):
    if not self.impute:
        return [np.nan]*self.sigbytes
    else:
        return [self.impute]*self.sigbytes
```

```
def _spread(self, feature, index_name):
    """ Fills the byte dataset for each record
    Params:
    _______
    feature: A pandas series
    """
    # Dict to hold the split bytes
    spread_field = {}
    # Iterate over all rows, fill dict
    for row in pd.DataFrame(feature).itertuples(name=None):
        # row[0] is the index, row[1] the content of the cell
        if not row[1] is np.nan:
            if len(row[1]) == self.sigbytes:
                spread_field[row[0]] = list(row[1])
            else:
                # The field is invalid
                spread_field[row[0]] = self._fill_missing()
        else:
            # handle missing values
            spread_field[row[0]] = self._fill_missing()

    # Create the dataframe, orient=index means
    # we interprete the dict's contents as rows (defaults to columns)
    temp_df = pd.DataFrame.from_dict(
        data=spread_field, orient="index")
    temp_df.columns = ["".join([feature.name, f])
                       for f in self.field_names]
    temp_df.index.name = index_name
    # make sure all fields are categorical
```

```
            temp_df = temp_df.astype("category")
            return temp_df

        def transform(self, X, y=None):
            assert isinstance(X, pd.DataFrame)
            X_trans = pd.DataFrame(index=X.index)
            for f in X.columns:
                new_df = self._spread(X[f], X.index.name)
                X_trans = X_trans.merge(new_df, on=X.index.name, copy=False)
            self.feature_names = list(X_trans.columns)
            self.is_transformed = True
            if not self.drop_orig:
                return X.merge(X_trans, on=X.index.name)
            else:
                return X_trans

        def get_feature_names(self):
            if self.is_transformed:
                return self.feature_names

    class RecodeUrbanSocioEconomic(BaseEstimator, TransformerMixin):
        def __init__(self):
            self.feature_names = None

        def fit(self, X, y=None):
            self.feature_names = ["DOMAINUrbanicity", "DOMAINSocioEconomic"]
            return self

        def transform(self, X, y=None):
            urb_dict = {'1': '1', '2': '2', '3': '2', '4': '3'}
            X_trans = pd.DataFrame(X, columns=self.feature_names).astype('category')
```

```
        X_trans.loc[X_trans.DOMAINUrbanicity == 'U', 'DOMAINSocioEconomic'] = X_trans.loc[X_trans
        X_trans.DOMAINSocioEconomic = X_trans.DOMAINSocioEconomic.cat.remove_unused_categories()
        return X_trans

    def get_feature_names(self):
        if isinstance(self.feature_names, list):
            return self.feature_names

class BinaryFeatureRecode(BaseEstimator, TransformerMixin):
    """
    Recodes one or more boolean feature(s), imputing missing values.
    The feature is recoded to float64, 1.0 = True, 0.0 = False,
    NaN for missing (by default). This can be changed through correct_noisy

    Params:
    _______
    correct_noisy: boolean or dict.
                   dict: custom map of nan -> bool mapping
                   True: maps {'1': True, '0': False, '□': False} + value_map
                   False: maps {'1': True, '0': False, '□': np.nan} + value_map
    value_map: dict describing which values are mapped to True/False
    """

    def __init__(self, correct_noisy=True, value_map=None):
        self.correct_noisy = correct_noisy
        self.value_map = value_map
        self.is_fit = False

    def fit(self, X, y=None):
        assert isinstance(X, pd.DataFrame)
        self.feature_names = list(X.columns)
```

```
        self.is_fit = True
        return self

    def transform(self, X, y=None):
        assert isinstance(X, pd.DataFrame)
        # Dict holding the mapping for data -> boolean
        if self.correct_noisy:
            if type(self.correct_noisy) is dict:
                vmap = self.correct_noisy
            else:
                vmap = {'1': 1.0, '0': 1.0, '□': 0.0}
        else:
            vmap = {'1': 1.0, '0': 0.0, '□': np.nan}
        vmap[self.value_map.get('true')] = 1.0
        vmap[self.value_map.get('false')] = 0.0
        temp_df = X.copy()
        try:
            for feature in X:
                # Map values in data to True/False.
                # NA values are propagated.
                temp_df[feature] = temp_df[feature].astype('object').map(
                    vmap, na_action='ignore').astype('float64')
        except Exception as exc:
            logger.exception(exc)
            raise
        else:
            return temp_df

    def get_feature_names(self):
        if self.is_fit:
```

```
            return self.feature_names


class DeltaTime(BaseEstimator, TransformerMixin):
    """Computes the duration between a date and a reference date in months.

    Parameters:
    ___________

    reference_date: either a single datetimelike or a series of datetimelike
        For series, the same length as the passed dataframe is expected.
    unit: ['months', 'years']
    """
    def __init__(self, reference_date=pd.datetime(1997, 6, 1), unit='months',suffix=True):
        self.reference_date = reference_date
        if suffix:
            self.feature_suffix = "_DELTA_"+unit.upper()
        else:
            self.feature_suffix = ""
        self.unit = unit
        self.suffix = suffix
        self.feature_names = None

    def get_duration(self, date_pair):
        if not pd.isna(date_pair.target) and not pd.isna(date_pair.ref):
            delta = relativedelta.relativedelta(date_pair.ref, date_pair.target)
            if self.unit.lower() == 'months':
                duration = (delta.years * 12) + delta.months
            elif self.unit.lower() == 'years':
                duration = delta.years + 1
        else:
```

```
            logger.info("Failed to calculate time delta. Dates: {} and {}.".format(date_pair.targ
            duration = np.nan
        return duration

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        assert isinstance(X, pd.DataFrame)

        # We need to ensure we have datetime objects.
        # The dateparser has to return int64 to work with sklearn, so
        # we need to recast here.
        X_trans = pd.DataFrame().astype('str')

        for f in X.columns:
            X_temp = pd.DataFrame(columns=['target', 'ref'])
            X_temp['target'] = X[f]
            if isinstance(self.reference_date, pd.Series):
                # we have a series of reference dates
                feature_name = f+"_"+str(self.reference_date.name)+self.feature_suffix
            else:
                feature_name = f+self.feature_suffix

            X_temp['ref'] = self.reference_date
            X_trans[feature_name] = X_temp.apply(self.get_duration, axis=1)

        self.feature_names = X_trans.columns

        return X_trans

    def get_feature_names(self):
```

```
        return self.feature_names


class MonthsToDonation(BaseEstimator, TransformerMixin):

    def __init__(self):
        self.feature_names = list()
        self.is_transformed = False

    def fit(self, X, y=None):
        return self

    def calc_diff(self, row):
        ref = row[0]
        target = row[1]
        if not pd.isna(ref) and not pd.isna(target):
            try:
                duration = relativedelta.relativedelta(ref, target).years * 12
                duration += relativedelta.relativedelta(ref, target).months
            except TypeError as err:
                logger.error("Failed to calculate time delta. " +
                             "Dates: {} and {}\nMessage: {}".format(row[0], row[1],err))
                duration = np.nan
        else:
            duration = np.nan
        return duration

    def transform(self, X, y=None):
        assert isinstance(X, pd.DataFrame)

        X_trans = pd.DataFrame(index=X.index)
        for i in range(3, 25):
```

```
            feat_name = "MONTHS_TO_DONATION_"+str(i)
            mailing = X[["ADATE_"+str(i), "RDATE_"+str(i)]]
            diffs = mailing.agg(self.calc_diff, axis=1)
            X_trans = X_trans.merge(pd.DataFrame(
                diffs, columns=[feat_name]), on=X_trans.index.name)
            self.feature_names.extend([feat_name])
            self.is_transformed = True
        return X_trans.astype("float64")

    def get_feature_names(self):
        if self.is_transformed:
            return self.feature_names
```

6.3 Python Environment

Python was installed through the Anaconda distribution

Most of the programming was done interactively in jupyter notebooks.

A python package for handling the loading of datasets was developed separately and the called from the notebooks.

6.4 Dataset Dictionary

The original dictionary file can be found at
https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1998+Data.

```
======================================================================
EPSILON CONFIDENTIAL      EPSILON CONFIDENTIAL    EPSILON CONFIDENTIAL

    INFORMATION LISTED BELOW IS AVAILABLE UNDER THE TERMS OF THE
                      CONFIDENTIALITY AGREEMENT

EPSILON CONFIDENTIAL      EPSILON CONFIDENTIAL    EPSILON CONFIDENTIAL
======================================================================

+--------------------------------------------------------------------+
|                PARALYZED VETERANS OF AMERICA (PVA)                 |
|                  DATA DICTIONARY TO ACCOMPANY                      |
```

```
|                                                                      |
|                          KDD-CUP-98                                  |
|                                                                      |
|          The Second International Knowledge Discovery and           |
|                 Data Mining Tools Competition                        |
|                                                                      |
|                Held in Conjunction with KDD-98                       |
|                                                                      |
|          The Fourth International Conference on Knowledge            |
|                   Discovery and Data Mining                          |
|                    [www.kdnuggets.com] or                            |
|                  [www-aig.jpl.nasa.gov/kdd98] or                     |
|                 [www.aaai.org/Conferences/KDD/1998]                  |
|                                                                      |
|                        Sponsored by the                              |
|                                                                      |
|      American Association for Artificial Intelligence (AAAI)         |
|                  Epsilon Data Mining Laboratory                      |
|                Paralyzed Veterans of America (PVA)                   |
+----------------------------------------------------------------------+
|                                                                      |
| Created:     7/20/98                                                 |
| Last update: 7/20/98                                                 |
| file name:   cup98DIC.txt                                            |
|                                                                      |
+----------------------------------------------------------------------+

Variable                    Description
--------------------------  ----------------------------------------
ODATEDW                     Origin Date. Date of donor's first gift
                            to PVA YYMM format (Year/Month).

OSOURCE                     Origin Source
                            - (Only 1rst 3 bytes are used)
                            - Defaulted to 00000 for conversion
                            - Code indicating which mailing list the
                              donor was originally acquired from
                            - A nominal or symbolic field.

TCODE                       Donor title code
                            000    = _
                            001    = MR.
                            001001 = MESSRS.
                            001002 = MR. & MRS.
                            002    = MRS.
                            002002 = MESDAMES
                            003    = MISS
```

```
003003 = MISSES
004    = DR.
004002 = DR. & MRS.
004004 = DOCTORS
005    = MADAME
006    = SERGEANT
009    = RABBI
010    = PROFESSOR
010002 = PROFESSOR & MRS.
010010 = PROFESSORS
011    = ADMIRAL
011002 = ADMIRAL & MRS.
012    = GENERAL
012002 = GENERAL & MRS.
013    = COLONEL
013002 = COLONEL & MRS.
014    = CAPTAIN
014002 = CAPTAIN & MRS.
015    = COMMANDER
015002 = COMMANDER & MRS.
016    = DEAN
017    = JUDGE
017002 = JUDGE & MRS.
018    = MAJOR
018002 = MAJOR & MRS.
019    = SENATOR
020    = GOVERNOR
021002 = SERGEANT & MRS.
022002 = COLNEL & MRS.
024    = LIEUTENANT
026    = MONSIGNOR
027    = REVEREND
028    = MS.
028028 = MSS.
029    = BISHOP
031    = AMBASSADOR
031002 = AMBASSADOR & MRS.
033    = CANTOR
036    = BROTHER
037    = SIR
038    = COMMODORE
040    = FATHER
042    = SISTER
043    = PRESIDENT
044    = MASTER
046    = MOTHER
047    = CHAPLAIN
```

```
048    = CORPORAL
050    = ELDER
056    = MAYOR
059002 = LIEUTENANT & MRS.
062    = LORD
063    = CARDINAL
064    = FRIEND
065    = FRIENDS
068    = ARCHDEACON
069    = CANON
070    = BISHOP
072002 = REVEREND & MRS.
073    = PASTOR
075    = ARCHBISHOP
085    = SPECIALIST
087    = PRIVATE
089    = SEAMAN
090    = AIRMAN
091    = JUSTICE
092    = MR. JUSTICE
100    = M.
103    = MLLE.
104    = CHANCELLOR
106    = REPRESENTATIVE
107    = SECRETARY
108    = LT. GOVERNOR
109    = LIC.
111    = SA.
114    = DA.
116    = SR.
117    = SRA.
118    = SRTA.
120    = YOUR MAJESTY
122    = HIS HIGHNESS
123    = HER HIGHNESS
124    = COUNT
125    = LADY
126    = PRINCE
127    = PRINCESS
128    = CHIEF
129    = BARON
130    = SHEIK
131    = PRINCE AND PRINCESS
132    = YOUR IMPERIAL MAJEST
135    = M. ET MME.
210    = PROF.
```

STATE	State abbreviation (a nominal/symbolic field)
ZIP	Zipcode (a nominal/symbolic field)
MAILCODE	Mail Code ” ”= Address is OK B = Bad Address
PVASTATE	EPVA State or PVA State Indicates whether the donor lives in a state served by the organization's EPVA chapter P = PVA State E = EPVA State (Northeastern US)
DOB	Date of birth (YYMM, Year/Month format.)
NOEXCH	Do Not Exchange Flag (For list rental) _ = can be exchanged X = do not exchange
RECINHSE	In House File Flag _ = Not an In House Record X = Donor has given to PVA's In House program
RECP3	P3 File Flag _ = Not a P3 Record X = Donor has given to PVA's P3 program
RECPGVG	Planned Giving File Flag _ = Not a Planned Giving Record X = Planned Giving Record
RECSWEEP	Sweepstakes file flag _ = Not a Sweepstakes Record X = Sweepstakes Record
MDMAUD	The Major Donor Matrix code The codes describe frequency and amount of giving for donors who have given a $100+ gift at any time in their giving history. An RFA (recency/frequency/monetary) field. The (current) concatenated version is a nominal or symbolic field. The individual bytes could separately be used as fields and refer to the following: First byte: Recency of Giving C=Current Donor L=Lapsed Donor I=Inactive Donor

	D=Dormant Donor 2nd byte: Frequency of Giving 1=One gift in the period of recency 2=Two-Four gifts in the period of recency 5=Five+ gifts in the period of recency 3rd byte: Amount of Giving L=Less than $100(Low Dollar) C=$100-499(Core) M=$500-999(Major) T=$1,000+(Top) 4th byte: Blank/meaningless/filler 'X' indicates that the donor is not a major donor. For more information regarding the RFA codes, see the promotion history field definitions.
DOMAIN	DOMAIN/Cluster code. A nominal or symbolic field. could be broken down by bytes as explained below. 1st byte = Urbanicity level of the donor's neighborhood U=Urban C=City S=Suburban T=Town R=Rural 2nd byte = Socio-Economic status of the neighborhood 1 = Highest SES 2 = Average SES 3 = Lowest SES (except for Urban communities, where 1 = Highest SES, 2 = Above average SES, 3 = Below average SES, 4 = Lowest SES.)
CLUSTER	CLUSTER Code indicating which cluster group the donor falls into. Each cluster is unique in terms of socio-economic status, urbanicty, ethnicity and a variety of other demographic characteristics. A nominal or symbolic field.
AGE	Overlay Age 0 = missing
AGEFLAG	Age Flag

	E = Exact I = Inferred from Date of Birth Field
HOMEOWNR	Home Owner Flag H = Home owner U = Unknown
CHILD03	Presence of Children age 0-3 B = Both, F = Female, M = Male
CHILD07	Presence of Childern age 4-7
CHILD12	Presence of Childern age 8-12
CHILD18	Presence of Childern age 13-18
NUMCHLD	NUMBER OF CHILDREN
INCOME	HOUSEHOLD INCOME
GENDER	Gender M = Male F = Female U = Unknown J = Joint Account, unknown gender
WEALTH1	Wealth Rating
HIT	MOR Flag # HIT (Mail Order Response) Indicates total number of known times the donor has responded to a mail order offer other than PVA's.
---	---
	The following variables indicate the number of known times the donor has responded to other types of mail order offers.
MBCRAFT	Buy Craft Hobby
MBGARDEN	Buy Gardening
MBBOOKS	Buy Books
MBCOLECT	Buy Collectables
MAGFAML	Buy General Family Mags
MAGFEM	Buy Female Mags
MAGMALE	Buy Sports Mags
PUBGARDN	Gardening Pubs
PUBCULIN	Culinary Pubs
PUBHLTH	Health Pubs
PUBDOITY	Do It Yourself Pubs
PUBNEWFN	News / Finance Pubs
PUBPHOTO	Photography Pubs
PUBOPP	Opportunity Seekers Pubs

DATASRCE	Source of Overlay Data Indicates which third-party data source the donor matched against 1 = MetroMail 2 = Polk 3 = Both
MALEMILI	% Males active in the Military
MALEVET	% Males Veterans
VIETVETS	% Vietnam Vets
WWIIVETS	% WWII Vets
LOCALGOV	% Employed by Local Gov
STATEGOV	% Employed by State Gov
FEDGOV	% Employed by Fed Gov
SOLP3	SOLICIT LIMITATION CODE P3 = can be mailed (Default) 00 = Do Not Solicit or Mail 01 = one solicitation per year 02 = two solicitations per year 03 = three solicitations per year 04 = four solicitations per year 05 = five solicitations per year 06 = six solicitations per year 12 = twelve solicitations per year
SOLIH	SOLICITATION LIMIT CODE IN HOUSE = can be mailed (Default) 00 = Do Not Solicit 01 = one solicitation per year 02 = two solicitations per year 03 = three solicitations per year 04 = four solicitations per year 05 = five solicitations per year 06 = six solicitations per year 12 = twelve solicitations per year
MAJOR	Major ($$) Donor Flag _ = Not a Major Donor X = Major Donor
WEALTH2	Wealth Rating Wealth rating uses median family income and population statistics from each area to index relative wealth within each state

	The segments are denoted 0-9, with 9 being the highest income group and zero being the lowest. Each rating has a different meaning within each state.
GEOCODE	Geo Cluster Code indicating the level geography at which a record matches the census data. A nominal or symbolic field. Blank=No code has been assigned or did not match at any level.
---	---
	The following variables reflect donor interests, as collected from third-party data sources
COLLECT1	COLLECTABLE (Y/N)
VETERANS	VETERANS (Y/N)
BIBLE	BIBLE READING (Y/N)
CATLG	SHOP BY CATALOG (Y/N)
HOMEE	WORK FROM HOME (Y/N)
PETS	HOUSEHOLD PETS (Y/N)
CDPLAY	CD PLAYER OWNERS (Y/N)
STEREO	STEREO/RECORDS/TAPES/CD (Y/N)
PCOWNERS	HOME PC OWNERS/USERS
PHOTO	PHOTOGRAPHY (Y/N)
CRAFTS	CRAFTS (Y/N)
FISHER	FISHING (Y/N)
GARDENIN	GARDENING (Y/N)
BOATS	POWER BOATING (Y/N)
WALKER	WALK FOR HEALTH (Y/N)
KIDSTUFF	BUYS CHILDREN'S PRODUCTS (Y/N)
CARDS	STATIONARY/CARDS BUYER (Y/N)
PLATES	PLATE COLLECTOR (Y/N)
LIFESRC	LIFE STYLE DATA SOURCE Indicates source of the lifestyle variables listed above 1 = MATCHED ON METRO MAIL ONLY 2 = MATCHED ON POLK ONLY 3 = MATCHED BOTH MM AND POLK
---	---
PEPSTRFL	Indicates PEP Star RFA Status blank = Not considered to be a PEP Star 'X' = Has PEP Star RFA Status

	The following variables reflect characteristics of the donors neighborhood, as collected from the 1990 US Census.
POP901	Number of Persons
POP902	Number of Families
POP903	Number of Households
POP90C1	Percent Population in Urbanized Area
POP90C2	Percent Population Outside Urbanized Area
POP90C3	Percent Population Inside Rural Area
POP90C4	Percent Male
POP90C5	Percent Female
ETH1	Percent White
ETH2	Percent Black
ETH3	Percent Native American
ETH4	Percent Pacific Islander/Asian
ETH5	Percent Hispanic
ETH6	Percent Asian Indian
ETH7	Percent Japanese
ETH8	Percent Chinese
ETH9	Percent Philipino
ETH10	Percent Korean
ETH11	Percent Vietnamese
ETH12	Percent Hawaiian
ETH13	Percent Mexican
ETH14	Percent Puerto Rican
ETH15	Percent Cuban
ETH16	Percent Other Hispanic
AGE901	Median Age of Population
AGE902	Median Age of Adults 18 or Older
AGE903	Median Age of Adults 25 or Older
AGE904	Average Age of Population
AGE905	Average Age of Adults >= 18
AGE906	Average Age of Adults >= 25
AGE907	Percent Population Under Age 18
CHIL1	Percent Children Under Age 7
CHIL2	Percent Children Age 7 - 13
CHIL3	Percent Children Age 14-17
AGEC1	Percent Adults Age18-24
AGEC2	Percent Adults Age 25-34
AGEC3	Percent Adults Age 35-44
AGEC4	Percent Adults Age 45-54
AGEC5	Percent Adults Age 55-64
AGEC6	Percent Adults Age 65-74
AGEC7	Percent Adults Age >= 75

CHILC1	Percent Children Age <=2
CHILC2	Percent Children Age 3-5
CHILC3	Percent Children Age 6-11
CHILC4	Percent Children Age 12-15
CHILC5	Percent Children Age 16-18
HHAGE1	Percent Households w/ Person 65+
HHAGE2	Percent Households w/ Person 65+ Living Alone
HHAGE3	Percent Households Headed by an Elderly Person Age 65+
HHN1	Percent 1 Person Households
HHN2	Percent 2 Person Households
HHN3	Percent 3 or More Person Households
HHN4	Percent 4 or More Person Households
HHN5	Percent 5 or More Person Households
HHN6	Percent 6 Person Households
MARR1	Percent Married
MARR2	Percent Separated or Divorced
MARR3	Percent Widowed
MARR4	Percent Never Married
HHP1	Median Person Per Household
HHP2	Average Person Per Household
DW1	Percent Single Unit Structure
DW2	Percent Detached Single Unit Structure
DW3	Percent Duplex Structure
DW4	Percent Multi (2+) Unit Structures
DW5	Percent 3+ Unit Structures
DW6	Percent Housing Units in 5+ Unit Structure
DW7	Percent Group Quarters
DW8	Percent Institutional Group Quarters
DW9	Non-Institutional Group Quarters
HV1	Median Home Value in hundreds
HV2	Average Home Value in hundreds
HV3	Median Contract Rent in hundreds
HV4	Average Contract Rent in hundreds
HU1	Percent Owner Occupied Housing Units
HU2	Percent Renter Occupied Housing Units
HU3	Percent Occupied Housing Units
HU4	Percent Vacant Housing Units
HU5	Percent Seasonal/Recreational Vacant Units
HHD1	Percent Households w/ Related Children
HHD2	Percent Households w/ Families
HHD3	Percent Married Couple Families
HHD4	Percent Married Couples w/ Related Children
HHD5	Percent Persons in Family Household
HHD6	Percent Persons in Non-Family Household
HHD7	Percent Single Parent Households
HHD8	Percent Male Householder w/ Child
HHD9	Percent Female Householder w/ Child

HHD10	Percent Single Male Householder
HHD11	Percent Single Female Householder
HHD12	Percent Households w/ Non-Family Living Arrangements
ETHC1	Percent White < Age 15
ETHC2	Percent White Age 15 - 59
ETHC3	Percent White Age 60+
ETHC4	Percent Black < Age 15
ETHC5	Percent Black Age 15 - 59
ETHC6	Percent Black Age 60+
HVP1	Percent Home Value >= $200,000
HVP2	Percent Home Value >= $150,000
HVP3	Percent Home Value >= $100,000
HVP4	Percent Home Value >= $75,000
HVP5	Percent Home Value >= $50,000
HVP6	Percent Home Value >= $300,000
HUR1	$ 1 or 2 Room Housing Units
HUR2	Percent >= 6 Room Housing Units
RHP1	Median Number of Rooms per Housing Unit
RHP2	Average Number of Rooms per Housing Unit
RHP3	Median Number of Persons per Housing Unit
RHP4	Average Number of Persons per Room
HUPA1	Percent Housing Units w/ 2 thru 9 Units at the Address
HUPA2	Percent Housing Units w/ >= 10 Units at the Address
HUPA3	Percent Mobile Homes or Trailers
HUPA4	Percent Renter Occupied Single Unit Structure
HUPA5	Percent Renter Occupied, 2 - 4 Units
HUPA6	Percent Renter Occupied, 5+ Units
HUPA7	Percent Renter Occupied Mobile Homes or Trailers
RP1	Percent Renters Paying >= $500 per Month
RP2	Percent Renters Paying >= $400 per Month
RP3	Percent Renters Paying >= $300 per Month
RP4	Percent Renters Paying >= $200 per Month
MSA	MSA Code
ADI	ADI Code
DMA	DMA Code
IC1	Median Household Income in hundreds
IC2	Median Family Income in hundreds
IC3	Average Household Income in hundreds
IC4	Average Family Income in hundreds
IC5	Per Capita Income
IC6	Percent Households w/ Income < $15,000
IC7	Percent Households w/ Income $15,000 - $24,999
IC8	Percent Households w/ Income $25,000 - $34,999
IC9	Percent Households w/ Income $35,000 - $49,999
IC10	Percent Households w/ Income $50,000 - $74,999
IC11	Percent Households w/ Income $75,000 - $99,999
IC12	Percent Households w/ Income $100,000 - $124,999

IC13	Percent Households w/ Income $125,000 - $149,999
IC14	Percent Households w/ Income >= $150,000
IC15	Percent Families w/ Income < $15,000
IC16	Percent Families w/ Income $15,000 - $24,999
IC17	Percent Families w/ Income $25,000 - 34,999
IC18	Percent Families w/ Income $35,000 - $49,999
IC19	Percent Families w/ Income $50,000 - $74,999
IC20	Percent Families w/ Income $75,000 - $99,999
IC21	Percent Families w/ Income $100,000 - $124,999
IC22	Percent Families w/ Income $125,000 - $149,999
IC23	Percent Families w/ Income >= $150,000
HHAS1	Percent Households on Social Security
HHAS2	Percent Households on Public Assistance
HHAS3	Percent Households w/ Interest, Rental or Dividend Income
HHAS4	Percent Persons Below Poverty Level
MC1	Percent Persons Move in Since 1985
MC2	Percent Persons in Same House in 1985
MC3	Percent Persons in Different State/Country in 1985
TPE1	Percent Driving to Work Alone Car/Truck/Van
TPE2	Percent Carpooling Car/Truck/Van)
TPE3	Percent Using Public Transportation
TPE4	Percent Using Bus/Trolley
TPE5	Percent Using Railways
TPE6	Percent Using Taxi/Ferry
TPE7	Percent Using Motorcycles
TPE8	Percent Using Other Transportation
TPE9	Percent Working at Home/No Transportation
PEC1	Percent Working Outside State of Residence
PEC2	Percent Working Outside County of Residence in State
TPE10	Median Travel Time to Work in minutes
TPE11	Mean Travel Time to Work in minutes
TPE12	Percent Traveling 60+ Minutes to Work
TPE13	Percent Traveling 15 - 59 Minutes to Work
LFC1	Percent Adults in Labor Force
LFC2	Percent Adult Males in Labor Force
LFC3	Percent Females in Labor Force
LFC4	Percent Adult Males Employed
LFC5	Percent Adult Females Employed
LFC6	Percent Mothers Employed Married and Single
LFC7	Percent 2 Parent Earner Families
LFC8	Percent Single Mother w/ Child in Labor Force
LFC9	Percent Single Father w/ Child in Labor Force
LFC10	Percent Families w/ Child w/ no Workers
OCC1	Percent Professional
OCC2	Percent Managerial
OCC3	Percent Technical
OCC4	Percent Sales

OCC5	Percent Clerical/Administrative Support
OCC6	Percent Private Household Service Occ.
OCC7	Percent Protective Service Occ.
OCC8	Percent Other Service Occ.
OCC9	Percent Farmers
OCC10	Percent Craftsmen, Precision, Repair
OCC11	Percent Operatives, Machine
OCC12	Percent Transportation
OCC13	Percent Laborers, Handlers, Helpers
EIC1	Percent Employed in Agriculture
EIC2	Percent Employed in Mining
EIC3	Percent Employed in Construction
EIC4	Percent Employed in Manufacturing
EIC5	Percent Employed in Transportation
EIC6	Percent Employed in Communications
EIC7	Percent Employed in Wholesale Trade
EIC8	Percent Employed in Retail Industry
EIC9	Percent Employed in Finance, Insurance, Real Estate
EIC10	Percent Employed in Business and Repair
EIC11	Percent Employed in Personnal Services
EIC12	Percent Employed in Entertainment and Recreation
EIC13	Percent Employed in Health Services
EIC14	Percent Employed in Educational Services
EIC15	Percent Employed in Other Professional Services
EIC16	Percent Employed in Public Administration
OEDC1	Percent Employed by Local Government
OEDC2	Percent Employed by State Government
OEDC3	Percent Employed by Federal Government
OEDC4	Percent Self Employed
OEDC5	Percent Private Profit Wage or Salaried Worker
OEDC6	Percent Private Non-Profit Wage or Salaried Worker
OEDC7	Percent Unpaid Family Workers
EC1	Median Years of School Completed by Adults 25+
EC2	Percent Adults 25+ Grades 0-8
EC3	Percent Adults 25+ w/ some High School
EC4	Percent Adults 25+ Completed High School or Equivalency
EC5	Percent Adults 25+ w/ some College
EC6	Percent Adults 25+ w/ Associates Degree
EC7	Percent Adults 25+ w/ Bachelors Degree
EC8	Percent Adults 25+ Graduate Degree
SEC1	Percent Persons Enrolled in Private Schools
SEC2	Percent Persons Enrolled in Public Schools
SEC3	Percent Persons Enrolled in Preschool
SEC4	Percent Persons Enrolled in Elementary or High School
SEC5	Percent Persons in College
AFC1	Percent Adults in Active Military Service
AFC2	Percent Males in Active Military Service

AFC3	Percent Females in Active Military Service
AFC4	Percent Adult Veterans Age 16+
AFC5	Percent Male Veterans Age 16+
AFC6	Percent Female Veterans Age 16+
VC1	Percent Vietnam Veterans Age 16+
VC2	Percent Korean Veterans Age 16+
VC3	Percent WW2 Veterans Age 16+
VC4	Percent Veterans Serving After May 1975 Only
ANC1	Percent Dutch Ancestry
ANC2	Percent English Ancestry
ANC3	Percent French Ancestry
ANC4	Percent German Ancestry
ANC5	Percent Greek Ancestry
ANC6	Percent Hungarian Ancestry
ANC7	Percent Irish Ancestry
ANC8	Percent Italian Ancestry
ANC9	Percent Norwegian Ancestry
ANC10	Percent Polish Ancestry
ANC11	Percent Portuguese Ancestry
ANC12	Percent Russian Ancestry
ANC13	Percent Scottish Ancestry
ANC14	Percent Swedish Ancestry
ANC15	Percent Ukranian Ancestry
POBC1	Percent Foreign Born
POBC2	Percent Born in State of Residence
LSC1	Percent English Only Speaking
LSC2	Percent Spanish Speaking
LSC3	Percent Asian Speaking
LSC4	Percent Other Language Speaking
VOC1	Percent Households w/ 1+ Vehicles
VOC2	Percent Households w/ 2+ Vehicles
VOC3	Percent Households w/ 3+ Vehicles
HC1	Percent Median Length of Residence
HC2	Percent Median Age of Occupied Dwellings in years
HC3	Percent Owner Occupied Structures Built Since 1989
HC4	Percent Owner Occupied Structures Built Since 1985
HC5	Percent Owner Occupied Structures Built Since 1980
HC6	Percent Owner Occupied Structures Built Since 1970
HC7	Percent Owner Occupied Structures Built Since 1960
HC8	Percent Owner Occupied Structures Built Prior to 1960
HC9	Percent Owner Occupied Condominiums
HC10	Percent Renter Occupied Condominiums
HC11	Percent Occupied Housing Units Heated by Utility Gas
HC12	Percent Occupied Housing Units Heated by Bottled, Tank or LP
HC13	Percent Occupied Housing Units Heated by Electricity
HC14	Percent Occupied Housing Units Heated by Fuel Oil
HC15	Percent Occupied Housing Units Heated by Solar Energy

```
HC16                        Percent Occupied Housing Units Heated by Coal, Wood, Other
HC17                        Percent Housing Units w/ Public Water Source
HC18                        Percent Housing Units w/ Well Water Source
HC19                        Percent Housing Units w/ Public Sewer Source
HC20                        Percent Housing Units w/ Complete Plumbing Facilities
HC21                        Percent Housing Units w/ Telephones
MHUC1                       Median Homeowner Cost w/ Mortgage per Month dollars
MHUC2                       Median Homeowner Cost w/out Mortgage per Month dollars
AC1                         Percent Adults Age 55-59
AC2                         Percent Adults Age 60-64

-------------------------  ------------------------------------------------

                          The fields listed below are from the promotion history file.

                            PROMOTION CODES:
                            ----------------

                            The following lists the promotion codes and their
                            respective field names (where XXXX refers to ADATE, RFA,
                            RDATE and RAMNT.)

                            '97NK' ==> xxxx_2 (mailing was used to construct
                                               the target fields)
                            '96NK' ==> xxxx_3
                            '96TK' ==> xxxx_4
                            '96SK' ==> xxxx_5
                            '96LL' ==> xxxx_6
                            '96G1' ==> xxxx_7
                            '96GK' ==> xxxx_8
                            '96CC' ==> xxxx_9
                            '96WL' ==> xxxx_10
                            '96X1' ==> xxxx_11
                            '96XK' ==> xxxx_12
                            '95FS' ==> xxxx_13
                            '95NK' ==> xxxx_14
                            '95TK' ==> xxxx_15
                            '95LL' ==> xxxx_16
                            '95G1' ==> xxxx_17
                            '95GK' ==> xxxx_18
                            '95CC' ==> xxxx_19
                            '95WL' ==> xxxx_20
                            '95X1' ==> xxxx_21
                            '95XK' ==> xxxx_22
                            '94FS' ==> xxxx_23
                            '94NK' ==> xxxx_24
```

```
1st 2 bytes of the code refers to the year of the
mailing while 3rd and 4th bytes refer to the
following promotion codes/types:

LL mailings had labels only
WL mailings had labels only
CC mailings are calendars with stickers but do
   not have labels
FS mailings are blank cards that fold into
   thirds with labels
NK mailings are blank cards with labels
SK mailings are blank cards with labels
TK mailings have thank you printed on the
   outside with labels
GK mailings are general greeting cards (an
   assortment of birthday, sympathy, blank, & get
   well) with labels
XK mailings are Christmas cards with labels
X1 mailings have labels and a notepad
G1 mailings have labels and a notepad

This information could certainly be used to calculate
several summary variables that count the number of
occurrences of various types of promotions received
in the most recent 12-36 months, etc.

RFA (RECENCY/FREQUENCY/AMOUNT)
------------------------------

The RFA (recency/frequency/amount) status of the
donors (as of the promotion dates) is included in the
RFA fields.

The (current) concatenated version is a nominal
or symbolic field. The individual bytes could
separately be used as fields and refer to the
following:

First Byte of code is concerned with RECENCY
based on Date of the last Gift

F=FIRST TIME DONOR Anyone who has made their
  first donation in the last 6 months and has
  made just one donation.

N=NEW DONOR Anyone who has made their first
  donation in the last 12 months and is not a
```

	First time donor. This is everyone who made their first donation 7-12 months ago, or people who made their first donation between 0-6 months ago and have made 2 or more donations.
	A=ACTIVE DONOR Anyone who made their first donation more than 12 months ago and has made a donation in the last 12 months.
	L=LAPSING DONOR A previous donor who made their last donation between 13-24 months ago.
	I=INACTIVE DONOR A previous donor who has not made a donation in the last 24 months. It is people who made a donation 25+ months ago.
	S=STAR DONOR STAR Donors are individuals who have given to 3 consecutive card mailings.
	Second Byte of code is concerned with FREQUENCY based on the period of recency. The period of recency for all groups except L and I is the last 12 months. For L it is 13-24 months ago, and for I it is 25-36 months ago. There are four valid frequency codes.
	1=One gift in the period of recency 2=Two gift in the period of recency 3=Three gifts in the period of recency 4=Four or more gifts in the period of recency
	Third byte of the code is the Amount of the last gift.
	A=$0.01 - $1.99 B=$2.00 - $2.99 C=$3.00 - $4.99 D=$5.00 - $9.99 E=$10.00 - $14.99 F=$15.00 - $24.99 G=$25.00 and above
ADATE_2	Date the 97NK promotion was mailed
ADATE_3	Date the 96NK promotion was mailed

ADATE_4	Date the 96TK promotion was mailed
ADATE_5	Date the 96SK promotion was mailed
ADATE_6	Date the 96LL promotion was mailed
ADATE_7	Date the 96G1 promotion was mailed
ADATE_8	Date the 96GK promotion was mailed
ADATE_9	Date the 96CC promotion was mailed
ADATE_10	Date the 96WL promotion was mailed
ADATE_11	Date the 96X1 promotion was mailed
ADATE_12	Date the 96XK promotion was mailed
ADATE_13	Date the 95FS promotion was mailed
ADATE_14	Date the 95NK promotion was mailed
ADATE_15	Date the 95TK promotion was mailed
ADATE_16	Date the 95LL promotion was mailed
ADATE_17	Date the 95G1 promotion was mailed
ADATE_18	Date the 95GK promotion was mailed
ADATE_19	Date the 95CC promotion was mailed
ADATE_20	Date the 95WL promotion was mailed
ADATE_21	Date the 95X1 promotion was mailed
ADATE_22	Date the 95XK promotion was mailed
ADATE_23	Date the 94FS promotion was mailed
ADATE_24	Date the 94NK promotion was mailed
RFA_2	Donor's RFA status as of 97NK promotion date
RFA_3	Donor's RFA status as of 96NK promotion date
RFA_4	Donor's RFA status as of 96TK promotion date
RFA_5	Donor's RFA status as of 96SK promotion date
RFA_6	Donor's RFA status as of 96LL promotion date
RFA_7	Donor's RFA status as of 96G1 promotion date
RFA_8	Donor's RFA status as of 96GK promotion date
RFA_9	Donor's RFA status as of 96CC promotion date
RFA_10	Donor's RFA status as of 96WL promotion date
RFA_11	Donor's RFA status as of 96X1 promotion date
RFA_12	Donor's RFA status as of 96XK promotion date
RFA_13	Donor's RFA status as of 95FS promotion date
RFA_14	Donor's RFA status as of 95NK promotion date
RFA_15	Donor's RFA status as of 95TK promotion date
RFA_16	Donor's RFA status as of 95LL promotion date
RFA_17	Donor's RFA status as of 95G1 promotion date
RFA_18	Donor's RFA status as of 95GK promotion date
RFA_19	Donor's RFA status as of 95CC promotion date
RFA_20	Donor's RFA status as of 95WL promotion date
RFA_21	Donor's RFA status as of 95X1 promotion date
RFA_22	Donor's RFA status as of 95XK promotion date
RFA_23	Donor's RFA status as of 94FS promotion date
RFA_24	Donor's RFA status as of 94NK promotion date

	The following fields are summary variables from the promotion history file.
CARDPROM	Lifetime number of card promotions received to date. Card promotions are promotion type FS, GK, TK, SK, NK, XK, UF, UU.
MAXADATE	Date of the most recent promotion received (in YYMM, Year/Month format)
NUMPROM	Lifetime number of promotions received to date
CARDPM12	Number of card promotions received in the last 12 months (in terms of calendar months translates into 9603-9702)
NUMPRM12	Number of promotions received in the last 12 months (in terms of calendar months translates into 9603-9702)
---	---
	The following fields are from the giving history file.
RDATE_3	Date the gift was received for 96NK
RDATE_4	Date the gift was received for 96TK
RDATE_5	Date the gift was received for 96SK
RDATE_6	Date the gift was received for 96LL
RDATE_7	Date the gift was received for 96G1
RDATE_8	Date the gift was received for 96GK
RDATE_9	Date the gift was received for 96CC
RDATE_10	Date the gift was received for 96WL
RDATE_11	Date the gift was received for 96X1
RDATE_12	Date the gift was received for 96XK
RDATE_13	Date the gift was received for 95FS
RDATE_14	Date the gift was received for 95NK
RDATE_15	Date the gift was received for 95TK
RDATE_16	Date the gift was received for 95LL
RDATE_17	Date the gift was received for 95G1
RDATE_18	Date the gift was received for 95GK
RDATE_19	Date the gift was received for 95CC
RDATE_20	Date the gift was received for 95WL
RDATE_21	Date the gift was received for 95X1
RDATE_22	Date the gift was received for 95XK
RDATE_23	Date the gift was received for 94FS
RDATE_24	Date the gift was received for 94NK
RAMNT_3	Dollar amount of the gift for 96NK
RAMNT_4	Dollar amount of the gift for 96TK
RAMNT_5	Dollar amount of the gift for 96SK
RAMNT_6	Dollar amount of the gift for 96LL

RAMNT_7	Dollar amount of the gift for 96G1
RAMNT_8	Dollar amount of the gift for 96GK
RAMNT_9	Dollar amount of the gift for 96CC
RAMNT_10	Dollar amount of the gift for 96WL
RAMNT_11	Dollar amount of the gift for 96X1
RAMNT_12	Dollar amount of the gift for 96XK
RAMNT_13	Dollar amount of the gift for 95FS
RAMNT_14	Dollar amount of the gift for 95NK
RAMNT_15	Dollar amount of the gift for 95TK
RAMNT_16	Dollar amount of the gift for 95LL
RAMNT_17	Dollar amount of the gift for 95G1
RAMNT_18	Dollar amount of the gift for 95GK
RAMNT_19	Dollar amount of the gift for 95CC
RAMNT_20	Dollar amount of the gift for 95WL
RAMNT_21	Dollar amount of the gift for 95X1
RAMNT_22	Dollar amount of the gift for 95XK
RAMNT_23	Dollar amount of the gift for 94FS
RAMNT_24	Dollar amount of the gift for 94NK
---	---
	The following fields are summary variables from the giving history file.
RAMNTALL	Dollar amount of lifetime gifts to date
NGIFTALL	Number of lifetime gifts to date
CARDGIFT	Number of lifetime gifts to card promotions to date
MINRAMNT	Dollar amount of smallest gift to date
MINRDATE	Date associated with the smallest gift to date
MAXRAMNT	Dollar amount of largest gift to date
MAXRDATE	Date associated with the largest gift to date
LASTGIFT	Dollar amount of most recent gift
LASTDATE	Date associated with the most recent gift
FISTDATE	Date of first gift
NEXTDATE	Date of second gift
TIMELAG	Number of months between first and second gift
AVGGIFT	Average dollar amount of gifts to date
---	---
CONTROLN	Control number (unique record identifier)
TARGET_B	Target Variable: Binary Indicator for Response to 97NK Mailing
TARGET_D	Target Variable: Donation Amount (in $) associated with the Response to 97NK Mailing
HPHONE_D	Indicator for presence of a published home phone number

```
--------------------------  ------------------------------------------------
                            (See the section on RFA for the meaning of the
                            codes)

RFA_2R                      Recency code for RFA_2
RFA_2F                      Frequency code for RFA_2
RFA_2A                      Donation Amount code for RFA_2
MDMAUD_R                    Recency code for MDMAUD
MDMAUD_F                    Frequecy code for MDMAUD
MDMAUD_A                    Donation Amount code for MDMAUD

--------------------------  ------------------------------------------------

CLUSTER2                    Classic Cluster Code (a nominal symbolic field)
GEOCODE2                    County Size Code

=====================================================================
EPSILON CONFIDENTIAL      EPSILON CONFIDENTIAL    EPSILON CONFIDENTIAL

    INFORMATION LISTED BELOW IS AVAILABLE UNDER THE TERMS OF THE
                      CONFIDENTIALITY AGREEMENT

EPSILON CONFIDENTIAL      EPSILON CONFIDENTIAL    EPSILON CONFIDENTIAL
=====================================================================
```