Data Scientist - Case Study

Prepared by: Arman Didandeh

Submitted on: April 1, 2018

Table of Contents

1 Conceptual

1.1 Probability and Statistics

1.1.1 The Likelihood Function

Question 1

1. Through defining a logical likelihood function, and by assuming a normal distribution over the population along with a uniform prior, (Maximum Likelihood Estimation) MLE is capable of estimating the parameters such that the likelihood estimator[1] would be maximized (given that a maximum actually exists), using a small sample of the population. In doing so, MLE is consistent in convergence, and also efficient as the sample size grows.
2. MLE does not have a closed form and needs an optimization for computation. It also needs good guesses prior to using the estimator to maximize the likelihood, e.g., the likelihood function itself, whether or not θ are fixed parameters or random variables, the distribution of unobserved population, or the error distribution. All this makes MLE a highly bias-dependent approach and thus, less generic/robust. In addition, there is no implicit way of ranking the parameters in terms of being better or worth. All it does is maximizing the input function and provides one with the parameters.
   On the other hand, the parametric nature of MLE makes it a stronger statistical inference, because it allows a more complete representation of the population. It also elaborates for any type of data, e.g. experimental vs. observational, for which it guarantees the best one can get among all the testable hypotheses.
3. The first step in avoiding overfitting is to have a measure to capture whether or not we are in that direction. Likelihood Ratio Test (LRT) is one of the measures that is highly standard and popular. Once measured properly, adding penalties/regularization to the likelihood function is a way of avoiding overfitting. Also, one can stop the optimization process, say gradient descent, before a maximum is reached, to prevent local maxima. To do so, use of cross validation to calculate the gradient is a norm. An ensemble of early-stopped learned params might also be a better estimator of the population, i.e., a way to resolve the trade-off between bias (of the estimator) and variance (of the sample). Depending on the nature of the population, weak priors could also be introduced into the ensemble to further generalize the meta-model.

[1] In practice, its log, i.e., log-likelihood or even the average of log-likelihood is preferred.

1.1.2 Bayesian Analysis

Question 2

1. It is the probability distribution of a random variable (e.g., an event) that is calculated based on a prior knowledge of the population and a conditional aspect of the system, which is dependent on the observation, i.e., sample. For instance, if we assume that the adult human population is half man and half woman, i.e., 50-50, but then take a sample from a specific organization, only to realize that their employment policy implicitly hires more male data analysts, the posterior distribution would help us model their sexism discrimination more accurately.
2. Prior is basically our generic knowledge which is based on our mental model of the population space. Using the Bayes theorem, the posterior distribution can be defined as a function of the prior and the likelihood. If the prior is uninformative, such as the one in the example above, the posterior is highly determined by the likelihood, and hence, will be sample-driven.
3. Since we cannot compute the definite $\int$ in the interval (-10,10), we need to instead use the $\sum$ based on different values of $f(x_i)$ to estimate Area Under the Curve (AUC), using a large number of sampled $p_i = (x_i, y_i)$ data points. Without loss of generality, let us assume that we have decided to use N points to estimate the AUC. Let us assume $X = [x_i]$ with $i \in [1:N]$. In order to build X, we use the random number generator in the interval of (0,1) to uniformly build N values, each of which will be the x coordinate of our points p_i. We then transform this from (0,1) to (-10,10) using a simple scaling.
   The next step is find the best estimate for y_i for each point p_i. Using a binary policy, we start with the mean of the range of f(x) which is 1 for all points and make an inquiry whether $1 > f(x_i)$. If the answer is False, then we ask if if $1 < f(x_i)$. If the answer is once again false, we know that $f(x_i) = 1$. If either of the other two questions results in True, then we break the interval in two pieces and only search the remaining interval. For instance, if $f(x_i) < 1$, then we know that $0 < f(x_i) < 1$, so we set y_i to 0.5 and do another round of inquiry. We perform this act for a number of iterations M, until all points have a definite value (like 1 in the example above) or that the desired precision is met (e.g., 4f). This way, after M iterations, we have $P = [p_i]$ where $p_i = (x_i, y_i)$ for $i \in [1:N]$. With a big enough number N, we can estimate the AUC in a $\sum$.

1.2 Machine Learning

Question 3

1. In this case, removing the rows is not an option due to missing of a large portion of data. Therefore, after marking the missing/corrupt values in their corresponding features, one can take one of the following two approaches prior to making a decision:
    a. To remove the aforementioned feature;
    b. To impute them with either one of the methods that will follow;

   Upon validating these two approaches using the proper metric, a decision as to which would be the better option will be made.

   If imputation is the way to go forward, the following are the different options to choose from:
    - A distinct value different than all the current values in cases of categorical features.
    - The most frequent value other than Null/NaN.
    - In time-sensitive features, replacement with most recent values are also an option. However, this in combination with (b) might work a bit better in these cases.
    - The mean/median/mode of all the available values for that feature. Removing outliers are suggested prior to calculating the aforementioned values.
    - A value that is estimated by a separate model. This best fits when missing is not random. An example is using kNN when applicable.

   All these options account for single imputation. Multiple imputation is based on the same idea, while being applied several time, so that an aggregate value would converge.

   Another option is also to use an algorithm that can handle missing values properly, e.g. Random Decision Forests.
2. If by natural order and time dependency, the question is discussing a sample with features the value of which can be strongly determined as the time goes on, it translates into a time-series problem. Hence, certain methods that have proven themselves effective will be the first to be tested. Among them, Recurrent Neural Networks (RNN) using Long Short-Term Memory blocks (LSTMs) are the choice of confidence.
3. Most algorithms work much better using numeric values. Therefore, one needs to treat the categorical value to some degree or another. A starting point is to combine values using business logic and frequency. If the features represent some sort of implicit order, e.g. low, medium, high, then translating them into a numerical range of feature values will not decrease the generalizability of the feature. However, it is often the case that categorical features translate into a non-ordered set of features. In this case, a standard solution is to translate into a binary representation of the feature values, also known as

dummy coding or one-hot encoding. For instance, with 16 categorical values, one may break down the feature into 4 binary features, the collection of which representing a 4-bit value such as 0110, pointing to the 6th category.

4. This problem translates to the curse of dimensionality, an infamous concept in the domain of machine learning. Even though solving this problem is highly domain-specific, two methods have proven well to start testing with. One may decide to reduce the number of features drastically, using methods such as Principal Component Analysis (PCA), if and only if the new features (i.e., principal components which are the eigenvectors of the new feature space) do not lose a lot of the information that is translated through the original features. Another approach to attempt on creating more samples with the same (posterior) distribution by means of random sampling, e.g., Monte Carlo, or using Synthetic Minority Over-sampling TEchnique (SMOTE).

Question 4

1. This is a naturally imbalanced problem due to the nature of its domain being sales. Hence, ranking new customers with a probability of being a good lead can take two ways: (a) oversampling the good leads and balancing the training set, prior to feeding them to a stable ensemble model, such as a Random Forest (without a classifier sigmoid function so that we get likelihoods) or XGBoost (which introduces a regularization factor), which can result into a rank on which one can apply a threshold; (b) treating it as a one-class problem and filtering out the bad leads, and then feeding in the data into an SVM model, and to get a short-list of new customers who can be considered as "better" leads.
2. The AUC will help us see whether or not our method is separating good and bad leads in the training data properly. After this, log-loss will help one with the probabilities that one's model is producing. One may need to keep an eye on the window of probabilities as well: for instance, if most of the better leads fall in a ranking neighborhood of *60-65* percent, while rank of training data good leads are much higher, either the model or the measure would be subject to change.

2 Practical

Question 5

Based on the available data and its features, I would not apply a machine learning model to this question at all! This can simply translate into a basic statistical analysis, the data on basis of which is way less that dependable. However, the Figure 1 below shows that there has not been an effective difference when it comes to “day_of_week” being a strong indicator of conversion. Figure 2 brings in the month into the analysis as well, and the results follow the same pattern to a good degree.

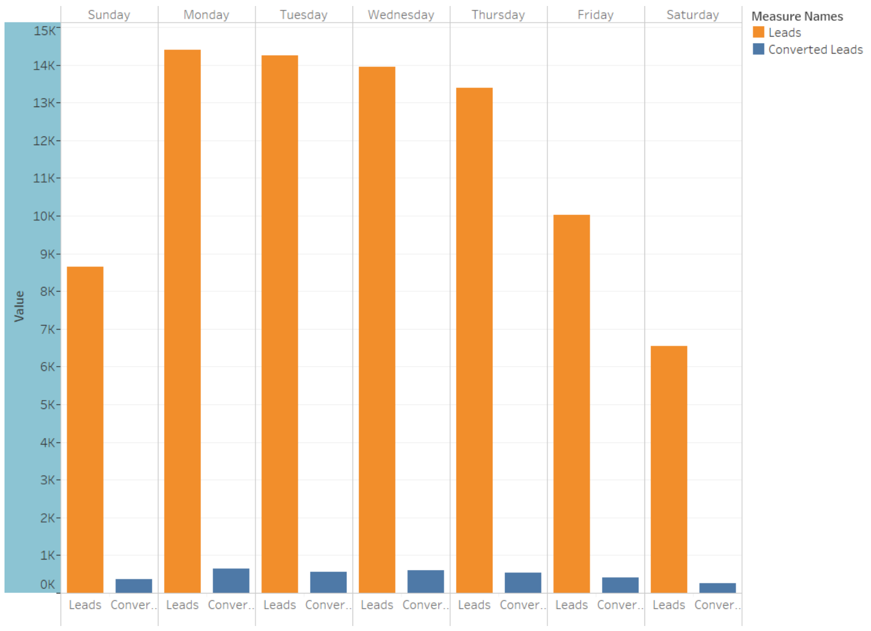

Figure 1. A comparison between different days of the week, in terms of leads and converted leads.

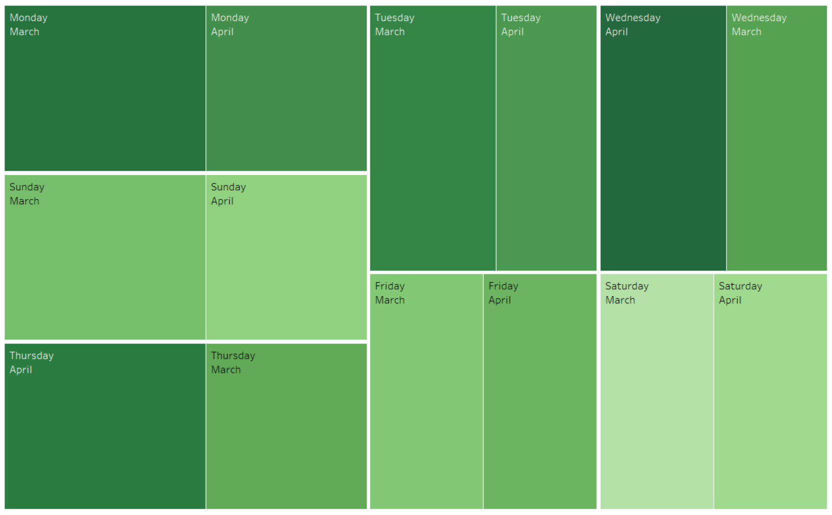

Figure 2 . A breakdown that includes the month into the analysis.

One would add to the discussion that a lack of knowledge of who/what category of probable leads were contacted on each day of the week, may also affect the analysis. In addition, we have no idea who the calling agents were, and what time of the day had more conversions. For instance, a hypothesis stating that “worse leads might be more probable to convert in the afternoons of the last Friday of each month and fairly close to getting paid” might be worth testing. Hence, my recommendation to the channel manager would be to try assigning random leads to random agents on random days of the week and gather information if she wants an informative decision; otherwise, this would be a “lucky guess” type of trial and error!

Question 6

1. This is by nature an imbalanced problem. Also, according to the available train data, the percentage of class 0 (i.e., the positive class) to 1 (i.e., the negative class) is 0.9485 to 0.0515, i.e., 37940 of the train samples in *dataset_2_train.csv* are classed as 0 and 2060 are classed as 1. Therefore, we will follow the footsteps described in **Question 4**. It is worth mentioning that the measures we report here are based on *dataset_2_train.csv*, since this is the dataset that contains labels. Let us now see how the 3 aforementioned models are built:

    - One-class SVM: We used scikit-learn to train this model. Firstly, we manually applied dummy coding to features *f17* and *f18* to transform categorical features to dichotomous features *f17_a*, *f17_b*,*f17_c*, *f18_a*, , *f18_b*, and , *f18_c*. To prepare a train set from *dataset_2_train.csv*, we built a model that is being learned to predict the majority class, i.e., class 0. Hence, we split the data in *dataset_2_train.csv* to the classed 0 and classes 1 and used 80% of the classed 0 data as our *train* set. The remaining 20% plus the classes 1 made our *test* set. In the first step, we learned 10 different models on the *train* set with a non-linear ‘rbf’ kernel function[2] over different values of *nu*[3] in the interval of [0.01,0.1] with a step of 0.01 and captured False Positive Rate (FPR) and True Positive Rate (TPR). We used the results to plot an ROC and found the best parameters to build on the final model (see Figure 3).

      The selected parameters are: *{nu=0.05, kernel=’rbf’, gamma=’auto’}* and the model is built on all the classed 0 data points, hence the name, one-class.
      We then applied this model onto *dataset_2_predict.csv* to make predictions of new labels. These two processes can be found as the following two cells in the file *one_class_svm.ipynb*[4], which is submitted along this report:
        - *param_selection*
        - *final_model*

[2] We tried different kernels and found out that the results of ‘rbf’ are more stable with our choices of *nu*.
[3] An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors [scikit-learn.org].
[4] It is important to notice that certain changes might be needed to run the same code on different machines.

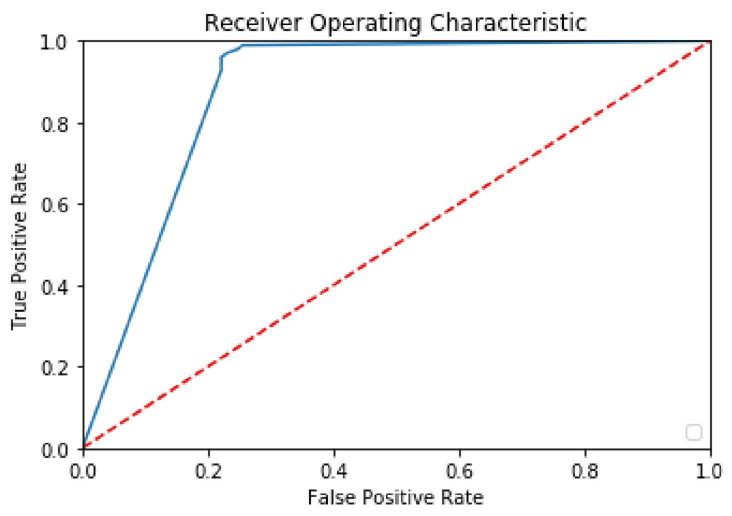

Figure 3 . ROC for 10 different one-class SVM models

- RandomForest: We used H2O to train this model and therefore, did not have to manually apply dummy coding, since it is automatically handled. We have learned a forest of trees with a depth of 9. The reason for choosing 9 features is that H2O allows one to calculate feature importance, which resulted in the number of features in Figure 4 below a threshold being selected as the depth of the trees. Using an ROC analysis (see Figure 5 below), we decided to learn a forest of 10 trees.
  The code for this model is submitted as *random_forest.ipynb* with the same cells as above.

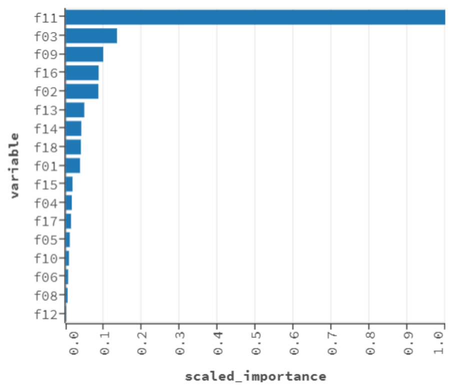

Figure 4 . Feature importance for all the features in *dataset_2_train.csv* for RandomForest

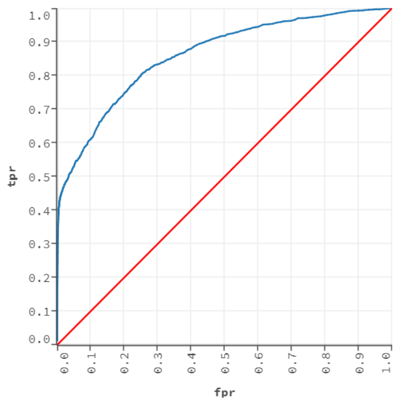

Figure 5 . ROC for the RandomForest model

- XGBoost: We used H2O to train this model as well. Once again, we used the process of feature importance computation (see Figure 6) and decided to go with parameters as *{ntree=10, max_depth=9}* to be able to compare it to

RandomForest. Figure 7 also shows the ROC prior to learning the final model. The code for this model is submitted as *xgboost.ipynb*.

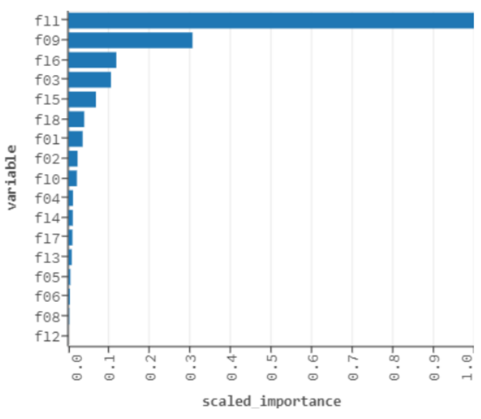

Figure 6 . Feature importance for all the features in *dataset_2_train.csv* for XGBoost

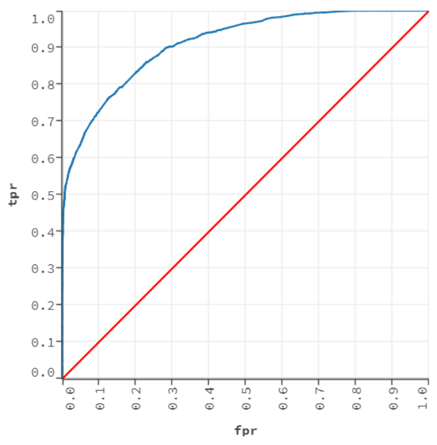

Figure 7 . ROC for the XGBoost model

2. We used the 3 aforementioned models to predict whether or not the users in *dataset_2_predict.csv* will churn. According to each model, the following are the frequency of churning among these users:
    - One-class SVM: 9439 out of 10000 rows will be classed 0 and the remaining 561 will be classed as 1.
    - RandomForest: 9771 out of 10000 rows will be classed 0 and the remaining 229 will be classed as 1.
    - XGBoost: 9708 out of 10000 rows will be classed 0 and the remaining 292 will be classed as 1.

    The results of our predictions can be explored in more details in the following files:
    - *Dataset_2_predict_one_class_svm.csv*: includes 1 new column '*churned_within_30_days*', which contains the estimated label.
    - *Dataset_2_predict_random_forest.csv*: includes 3 new columns '*churned_within_30_days*', 'p0', and 'p1', which contain the estimated label, the probability of class 0, and the probability of class 1.
    - *Dataset_2_predict_xgboost.csv*: includes 3 new columns '*churned_within_30_days*', 'p0', and 'p1', which contain the estimated label, the probability of class 0, and the probability of class 1.

3. Since this question needed ore research and due to time constraints, we will only provide 3 naive solutions in this document, which can act as starting points for this analysis. A fully-built model was not something that the author of this document was successful to achieve.

- Using the initial distribution of the data population in *dataset_2_train.csv* as prior probability of churning and not churning in the subsequent month, one may incorporate the calculated likelihoods of one of the models above, to estimate the posterior probability of churning in 2 months as:

  $$Posterior\_Pr(churn\ in\ 1\ month \mid data) = L(churn\ in\ 1\ month, data) * Prior\ Pr(churn\ in\ 1\ month))$$

  This can then be considered as the prior for the subsequent month, i.e., in 60 days, using the likelihood of churning, to calculate the next posterior, resulting in a posterior for the next 60 days. Following the same logic, one can estimate the probability of churning at every subsequent month for the next 12 months.
- The previous approach is a computationally cost-effective one. A more expensive but possibly more accurate approach is to use the posterior for the next 30 days—along with a threshold—to make predictions for the next 30 days on each validation data point (similar to those in *Dataset_2_predict_model.csv*). From there, one could calculate the new priors from the counts for classes 0 and 1, learn another model on top of this newly labelled data, get the likelihoods from this model, and calculate the posterior probabilities for the next 60 days. This approach will be followed for the next 90 days and so on until we get the results for the next 12 months. It is clear that this approach is more computationally expensive, but in practice, it is worth exploring. This can technically be translated into a non-homogeneous 1st-order Markov model, which is out of the range of this document's scope.
- A final approach that is worth investigating, at least for the sake of results analysis, starts with three simple assumptions:
  a. if a customer has churned at any point in time ***t***, that customer will retain the same status, unless contrary observations have been reported.
  b. if a customer has not churned at any point in time ***t***, chances of this customer churning in the subsequent time frame (***t+1***) would be less than any prior time frame leading to ***t***. In other words, the more a customer remains loyal, the less that customer may churn in general.

  Building on top of these assumptions, one can use the following formula to calculate an upper bound for the likelihoods:

  $$Pr(!churn\ in\ month\ t+1) = Pr(churn\ in\ month\ t) * (1+\psi)$$
  $$Where\ \psi\ is\ a\ regularization\ parameter\ and\ \psi \in (0, 0.00005]$$

  It is worth mentioning that the value of ψ is highly controversial and domain dependent, and therefore, it is subject to trial and error.