

# CutoffPredictor v2.0 – Description and User Guide

Theodore Bohn

January 14, 2020

## Summary

This document describes CutoffPredictor v2.0, a tool for water utilities to predict customer cutoff risk from customer metadata and histories of payments and water usage. This document describes v2.0 and denotes what has changed since v1.0. A number of improvements were implemented in v2.0, including more correct algorithms and a lower potential for overfitting.

The best-performing model in version 1.0 was random forest, using all customer metadata features, late payment fraction, and all 18 volume anomaly features, with a sample window length of 18 months. The AUC score achieved this way (on the test subset) was 0.92. But there was no constraint on the maximum node depth, and the large number of collinear features and the likely incorrect handling of non-ordinal categorical features likely led to over-fitting.

In version 2.0, logistic regression consistently outperforms random forest. In all cases considered, only 2 volume anomaly features are ever used: the fraction of zeros and the fraction of months whose anomaly  $> 3$  (measured in one of 2 ways; details below). Thus, far fewer features are used, and these all have very little correlation with each other. The maximum node depth in the random forest model is generally limited to 3 nodes. The best value of sample window length  $N$  is 6-8 months when customer metadata are included as features, and 12 months when these are excluded. AUC scores on the test subset are lower than for version 1.0 (AUC ranging from 0.7 to 0.8) but these AUC scores are very noisy compared to AUC scores from the cross-validation set, likely due to the small number of cutoffs available to work with. A larger train-test dataset could give a fairer assessment of model performance.

## Table of Contents

1. Introduction .....	3
1.1. Original Project (v1.0) .....	3
1.2. Improvements in v2.0 .....	4
2. Data .....	6
3. Exploratory Data Analysis and Feature Engineering.....	7
3.1. Data Cleaning/Preparation.....	7
3.2. Feature Engineering.....	7
3.2.1. Window Clipping and Labeling .....	8
3.2.2. Features.....	11
3.3.3. Distribution of Cutoffs as a Function of Window Length .....	20
3.3.4. Cutoff Rates as Empirical Functions of 2 Features.....	22
3.3.5. Major Feature Combinations Explored in this Study .....	28
4. Models and Parameters.....	29
5. Tool/Dashboard.....	34
5.1. Requirements .....	34
5.2. CutoffPredictor Installation and Setup .....	34
5.3. CutoffPredictor Process Flow and Usage .....	35
5.3.1. Components and Process Flow .....	35
5.3.2. Usage.....	37
5.3.3. Dashboard .....	40
6. Recommendations for Future Releases.....	41

# 1. Introduction

Water and other utilities face challenges arising from the substantial time lags inherent to their business operations: customers are billed for resource usage at monthly intervals, with the monthly bill arriving typically 1-2 weeks after the end of the monitoring period (i.e., during the next period) and often due 1-2 weeks thereafter (i.e., at the end of the next period). Customers who fail to pay are typically given another month to pay with a late fee, in addition to a new bill for their usage during the subsequent month. Customers who fail to pay after that 2<sup>nd</sup> month (at which point they may have used another months' worth of water) are at risk of having their service terminated (a cutoff), at which point a crew must visit the customer's property to shut off the service. Service can later be resumed if a new customer takes over, or if the existing customer pays their outstanding balance plus any penalties, again requiring a crew to visit the property. Thus, for a utility, a cutoff represents up to 3 months' lost revenue and water (or other resource), plus the cost of sending a crew to the property twice.

By law, utilities are required to serve any customers residing in their service area. Thus, it would be beneficial for a utility to be able to identify customers who are at risk of an impending cutoff and work with them to find a way to avoid the cutoff, either through advice on how to reduce consumption, or checking for leaks, or implementation of a payment plan, etc.

CutoffPredictor is a data product that enables a water utility to predict the risk of cutoff for all customers, and to monitor these risks via a dashboard.

## 1.1. Original Project (v1.0)

CutoffPredictor began in June 2019 as a consulting project by the Seattle office of Insight Data Science, for Valor Water Analytics in San Francisco. The author/consultant (Theodore Bohn) was a fellow at Insight, and the client contact was Bahman Roostaei, the lead Data Scientist at Valor.

The project goal was to create a machine learning model to predict cutoff risk from the available data in a given utility database. An initial model had already been developed at Valor, relating the customers' payment histories to their risk of cutoff. For this project, the client requested that the model additionally use customers' water usage histories and customer metadata (location, customer type, meter size, etc) to explore whether these additional features could improve predictive power.

The project was required to be a stand-alone tool that would query the utility database, perform any cleaning and processing of the data to prepare features, train and test models to evaluate performance, and display the results in a dashboard. The Insight program allowed the fellows 4 weeks to complete their projects.

My final product, CutoffPredictor v1.0, satisfied all of these requirements. The model related customer payment and usage behavior within a window of N months prior to the prediction date, as well as customer metadata (location, customer type, meter size), to cutoff risk. Two different models were explored: logistic regression and random forest. The best model and optimum value of N were

determined during the model training/validation process. The final, best-performing model (AUC = 0.92) turned out to be random forest, with N = 18 months.

Payment behavior was characterized by the fraction of the N months in the window for which a late payment fee was assessed. Usage behavior was characterized by 18 different statistics of usage, many of which were highly collinear. The reason for this was that there was not enough time to find a smaller set of “best” features by the delivery deadline. I hoped that using L2 regularization would address this collinearity in the logistic regression model, and I assumed that the random forest model would not be sensitive to the collinearity due to its giving all collinear features low individual feature importance.

To train and test these models, Valor gave me access to a small subset (2400 customers) of a water utility’s database. Unfortunately, this subset did not sample across customer type codes evenly, resulting in severe underrepresentation of some types (including the residential class, which should have been one of the most populous types). In addition, the dataset contained only 73 cutoffs, leading to substantial imbalance in the labels. I rebalanced the labels by oversampling the cutoff customers during the training phase. However, the assumptions made when oversampling creates synthetic records may introduce artifacts into the training data that hinder model performance on exogenous data.

However, the issues I had encountered with collinearity (and potential overfitting), underrepresentation, and imbalance bothered me. In addition, I was concerned that the feature engineering process did not truly predict the risk of imminent cutoff, but rather simply classified customers as high- or low-risk without any indication of when the cutoff might occur. During subsequent months, as I learned more about methods and available packages, and prepared for interviews, I thought of ways to improve the approach. Therefore, as an exercise, I updated the EDA, models, and tool with several improvements.

## 1.2. Improvements in v2.0

Changes from version 1.0 to 2.0 include:

- EDA and model training:
  - Replacement of 3 realizations of randomly-placed sample windows, a single window per customer, with a single realization consisting of as many consecutive windows as possible throughout each customer’s time series during the training period
  - Use of a trinary label to explore differences in behavior between windows immediately before cutoffs and windows earlier in the record (this label is later reverted to a binary label for model training)
  - Reduction of the number of collinear volume anomaly features
  - Better understanding of underrepresentation of some customer types
- Model training and validation:
  - Switch from SciKit-Learn to H2O for model implementation
  - Better handling of standardization and regularization

- Grid search over random forest max\_depth and logistic regression regularization lambda coefficient
- Better handling of non-ordinal categorical features
- Label imbalance: Replaced oversampling (via SMOTE) with weights column

I describe these improvements in detail in the sections 'Exploratory Data Analysis and Feature Engineering' and 'Models and Parameters'.







The best-performing model in version 1.0 was random forest, using all customer metadata features, late payment fraction, and all 18 volume anomaly features, with a sample window length of 18 months. The AUC score achieved this way (on the test subset) was 0.92. But there was no constraint on the maximum node depth, and the large number of collinear features and the likely incorrect handling of non-ordinal categorical features likely led to over-fitting.

In version 2.0, logistic regression consistently outperforms random forest. In all cases considered, only 2 volume anomaly features are ever used: the fraction of zeros and the fraction of months whose anomaly > 3 (measured in one of 2 ways; details below). Thus, far fewer features are used, and these all have very little correlation with each other. The maximum node depth in the random forest model is generally limited to 3 nodes. The best value of sample window length N is 6-8 months when customer metadata are included as features, and 12 months when these are excluded. AUC scores on the test subset are lower than for version 1.0 (AUC ranging from 0.7 to 0.8) but these AUC scores are very noisy compared to AUC scores from the cross-validation set, likely due to the small number of cutoffs available to work with. A larger train-test dataset could give a fairer assessment of model performance.

## 2. Data

For my model development efforts, I was given a view into one utility's database (a utility that had recorded cutoff dates, which not all utilities do). This view only allowed me to see a small subset of the database, with only 2400 customers (likely less than 10% of the total customer base), very few of which were residential or multi-family (which I would expect should be the most numerous categories). The subset spanned from the beginning of 2012 through March of 2019. Only 73 cutoffs occurred within this subset, resulting in extreme imbalance in the cutoff label.

The important tables and columns from the database are depicted in Fig. 1. At each location, the occupants and meters change over time asynchronously. For a given location, the occupants who have lived at that location are indexed in chronological order by the `cis_occupant_id` key, which is simply an integer with initial value of 0. `cis_occupant_id` is intended to be combined with `location_id` to uniquely identify customers. Billing records (**charge** table) are keyed by `location_id`, usage records (**volume** table) are keyed by `meter_id`, and the **cutoffs** table is keyed by location/occupant.

		
<b>Occupant</b>	<b>Meter_Location</b>	<b>Meter</b>
<ul style="list-style-type: none"><li>• Location ID</li><li>• CIS Occupant ID</li><li>• Move-in/out dates</li></ul>	<ul style="list-style-type: none"><li>• Location ID</li><li>• Meter Address</li><li>• Municipality Code</li></ul>	<ul style="list-style-type: none"><li>• Meter ID</li><li>• Location ID</li><li>• Meter Size</li><li>• Customer Type Code</li></ul>
		
<b>Charge</b>	<b>Volume</b>	<b>Cutoffs</b>
<ul style="list-style-type: none"><li>• Location ID</li><li>• Bill Date</li><li>• Total Charge</li><li>• Late Charge</li></ul>	<ul style="list-style-type: none"><li>• Meter ID</li><li>• Read Date</li><li>• Volume_kgals</li></ul>	<ul style="list-style-type: none"><li>• Location ID</li><li>• CIS Occupant ID</li><li>• Cutoff Date</li></ul>

**Fig. 1.** Major tables and columns from the pilot test utility database.

## 3. Exploratory Data Analysis and Feature Engineering

### 3.1. Data Cleaning/Preparation

The tool's first step in preparing the data is to create an `occupant_id` string, consisting of the `location_id` and the `cis_occupant_id`, joined by a hyphen. I added this `occupant_id` string to all tables via multiple joins as necessary so that I could construct a single time series for each occupant for each billing and usage variable.

The next step is to align the dates in the **charge**, **volume**, and **cutoffs** tables, by setting the day of the month to 1 for all records and summing any instances of multiple records in the same month to yield a single record per month. My original reason for this was so that I could compare across customers (as if all meters were read on the same day of the month), for example to enable detecting anomalous behavior of a given customer relative to their customer type or geographic neighborhood. But I never ended up needing to do this. Another advantage of this approach is to facilitate the insertion of any missing records (when records are known to occur on the 1<sup>st</sup> day of each month, it becomes easier to detect and fill missing records). On the other hand, throwing away the day of the month reduces the tool's precision to the month level, which may be disadvantageous later on. A better solution would be to adjust the logic in the tool to handle these differing days of the month correctly.

The next step is to replace bad values (negative, extremely large, nan) of billing and usage records with 0s. The only nans that I found were those inserted to fill gaps, so values of 0 were appropriate in those cases.

Finally, the tool calls geocoding functions to convert the location addresses to geographic coordinates (latitude, longitude) for plotting on the dashboard. This also could be useful later on if one wanted to implement a spatial clustering algorithm.

### 3.2. Feature Engineering

After initial data preparation, the tool prepares a table of features and labels. For time-varying variables (i.e., billing and usage variables), features are generated by the statistics of these variables within sample windows of length  $N$  (with  $N$  being a parameter to optimize) clipped out of the time series. The general process flow is to loop over `occupant_ids`; for each `occupant_id`: (1) construct the time series of all billing (**charge**) and usage (**volume**) variables corresponding to the occupant; (2) for occupants who appear in the **cutoffs** table, divide the time series into segments preceding each of the occupant's cutoffs plus a final segment following the final cutoff; (3) for occupants not in the cutoff table, assign their entire time series to a single segment; and (4) divide each segment into consecutive sample windows of length  $N$ , working backwards from the end of the training period. The details of window clipping, labeling, and feature computation are described below.

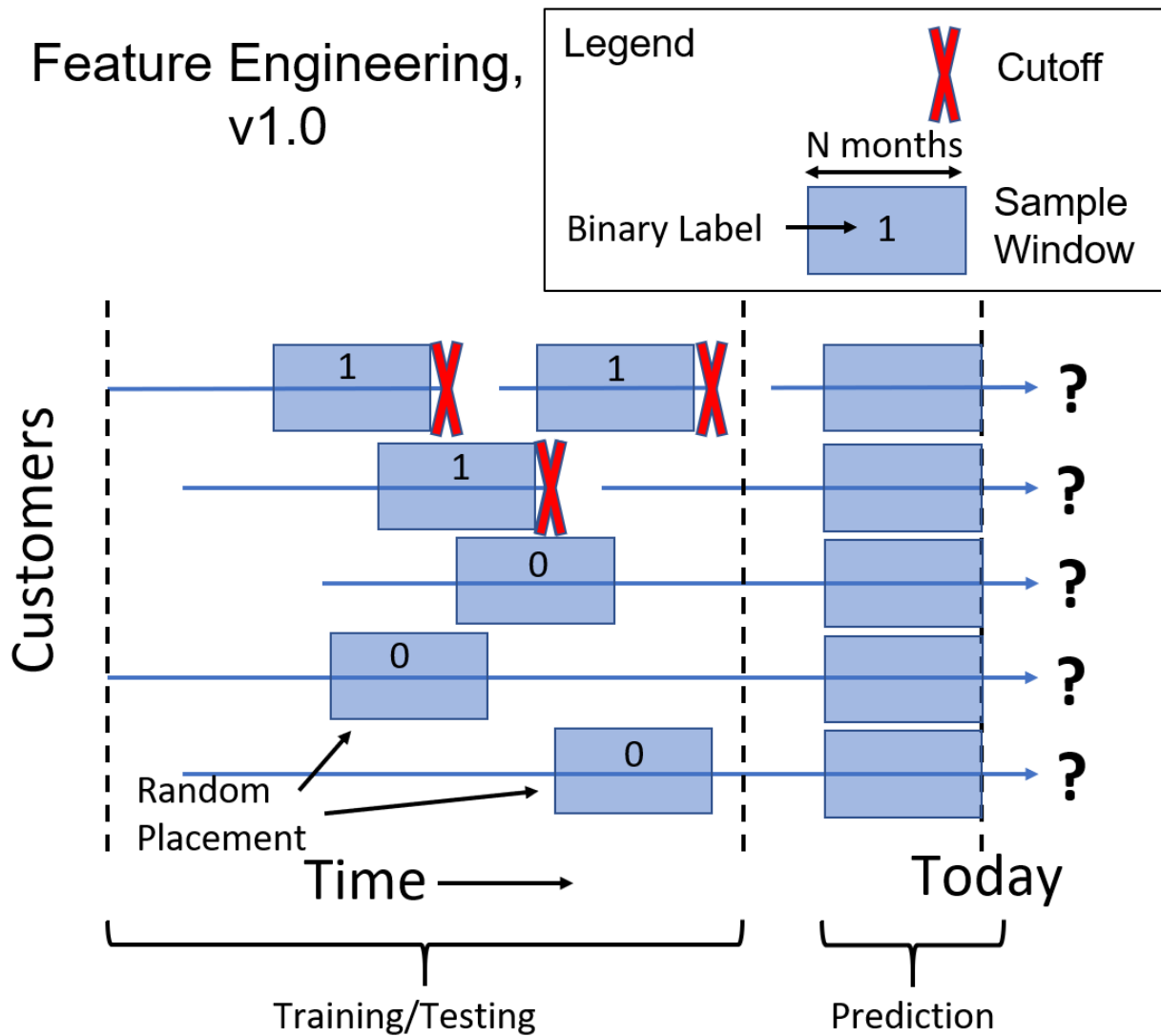
### 3.2.1. Window Clipping and Labeling

In version 1.0, for each non-cutoff customer (customer with no cutoffs), I analyzed a single N-month-long sample window, randomly placed within the customer's time series within the training period (Fig. 2). For each cutoff customer, I analyzed 1 sample window per cutoff (some customers had multiple cutoffs), beginning N months before the cutoff, so that conditions immediately prior to the cutoff were analyzed. To lessen the impacts of random window placement for non-cutoff customers, I performed the sampling 3 times ('realizations') with different random window placements, training models on each realization, and averaging performance metrics across the 3 realizations. The model parameters from the best-performing realization were then used in subsequent applications. Windows immediately before a cutoff were labeled with 1, while windows from non-cutoff customers were labeled with 0.

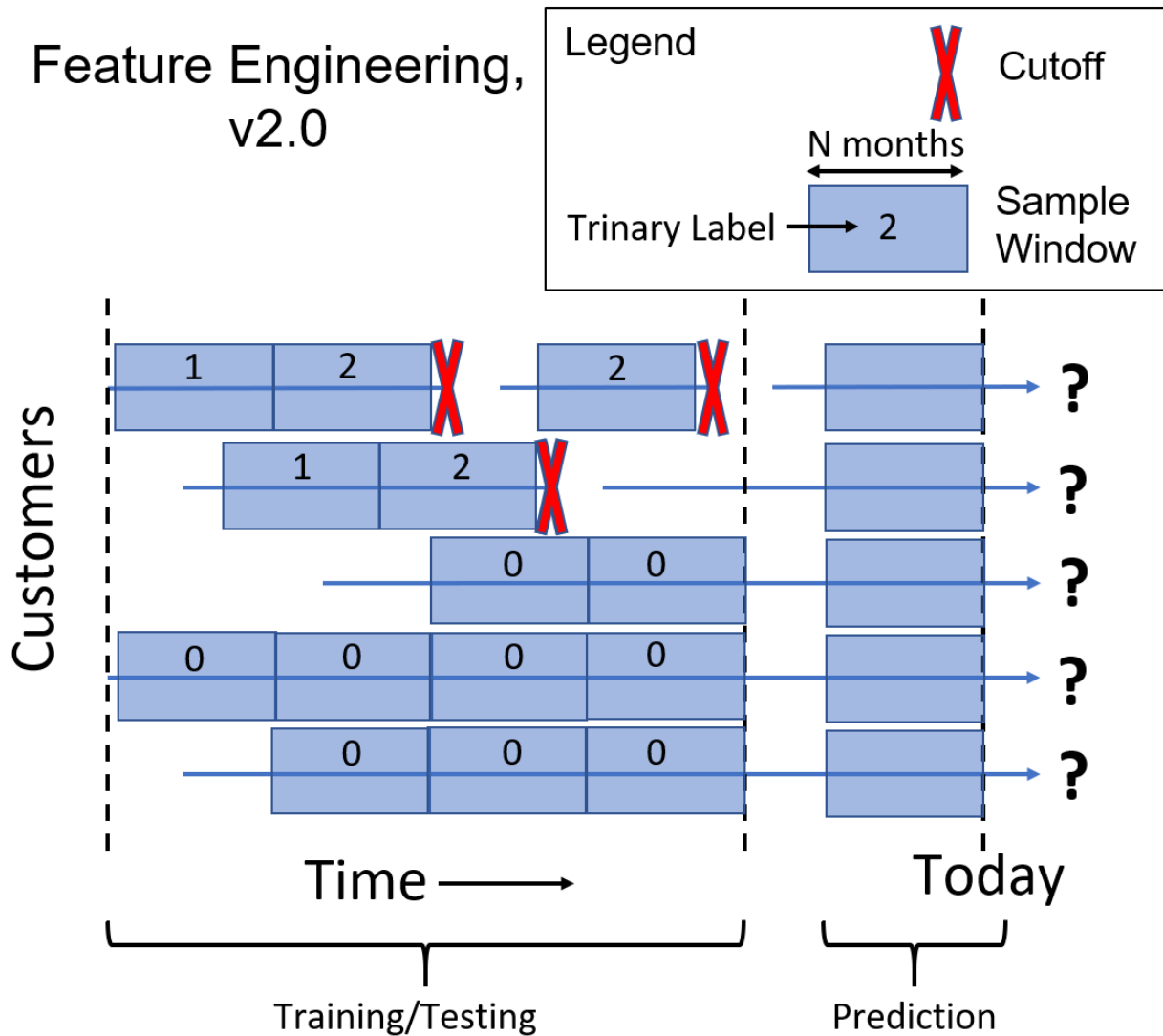
One disadvantage of this approach was the cumbersome logic used to handle 3 realizations of the features and models. Another disadvantage was that, for cutoff customers, I never knew if time series characteristics more than N points before a cutoff were predictive of a cutoff. Did earlier windows in a cutoff customer's time series look like those immediately before a cutoff, or did they look like windows from non-cutoff customers? If the former, then the models would be incapable of telling the user whether a cutoff is imminent; rather, the models would simply identify customers who are always "risky". This would have the disadvantage of identifying a larger number of customers, perhaps few of whom would ever encounter a cutoff (i.e., higher false positive rate), and spreading resources too thinly (not to mention alienating customers who might be offended at the suggestion). If the latter, then the models would be predicting the odds of each customer having a cutoff within the next N months, which would isolate the highest-priority customers and give the users a timeline to work with. But I couldn't check this without sampling earlier windows in the cutoff customers' time series.

In version 2.0, I changed the window placement to allow multiple windows per customer (Fig. 3). In fact, I placed sample windows consecutively throughout the entire timeseries of each customer. This allowed me to (1) check for differences between earlier and later windows of cutoff customers, and (2) eliminate the logic dealing with multiple realizations.





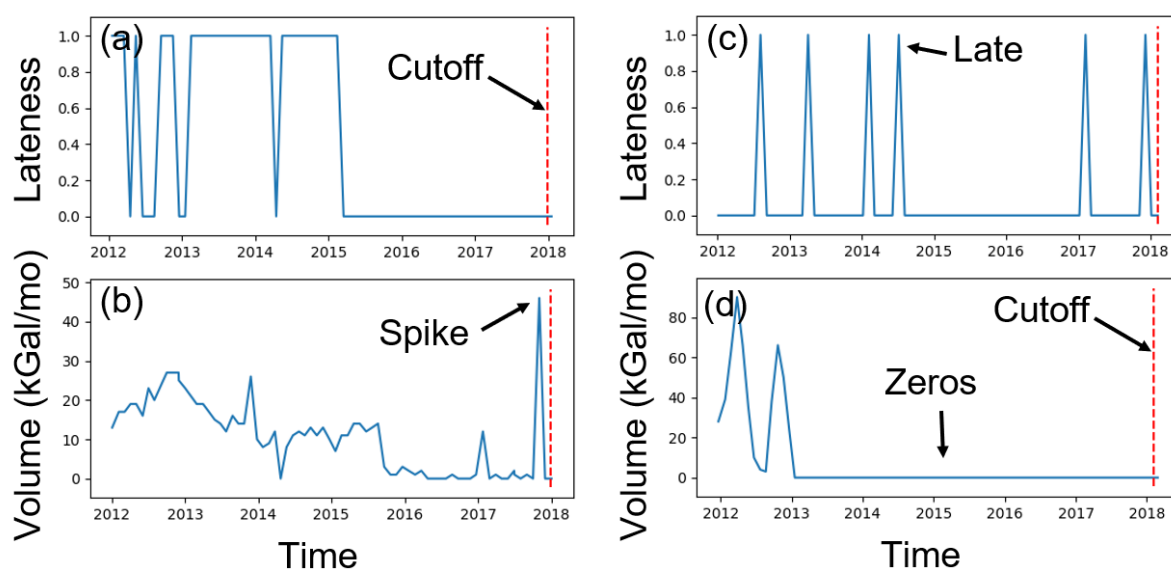
**Fig. 2.** Schematic illustrating the placement of sample windows during feature engineering in v1.0. Sample windows are N months long. Each sample window is assigned a binary label as follows: 0 = no cutoff occurs at any time in the record; 1 = a cutoff occurs immediately following the window.



**Fig. 3.** Schematic illustrating the placement of sample windows during feature engineering in v2.0. Sample windows are N months long. Each sample window is assigned a trinary label as follows: 0 = no cutoff occurs at any time in the record; 1 = a cutoff occurs more than N months after the window; 2 = a cutoff occurs immediately following the window.

### 3.2.2. Features

I revisited exploratory data analysis (EDA) for the various features computed from sample windows and from customer metadata. Before looking at features, it is useful to look at example plots of the timeseries data (Fig. 4). Cutoffs (shown in red dashed lines) are often preceded by late payments (Fig. 4c), spikes in usage (Fig. 4b), and months of zero usage (Fig. 4d). Therefore, features derived from timeseries variables should capture an anomaly of some statistic of the variable relative to its long-term behavior. The features I explored are listed in Table 1.

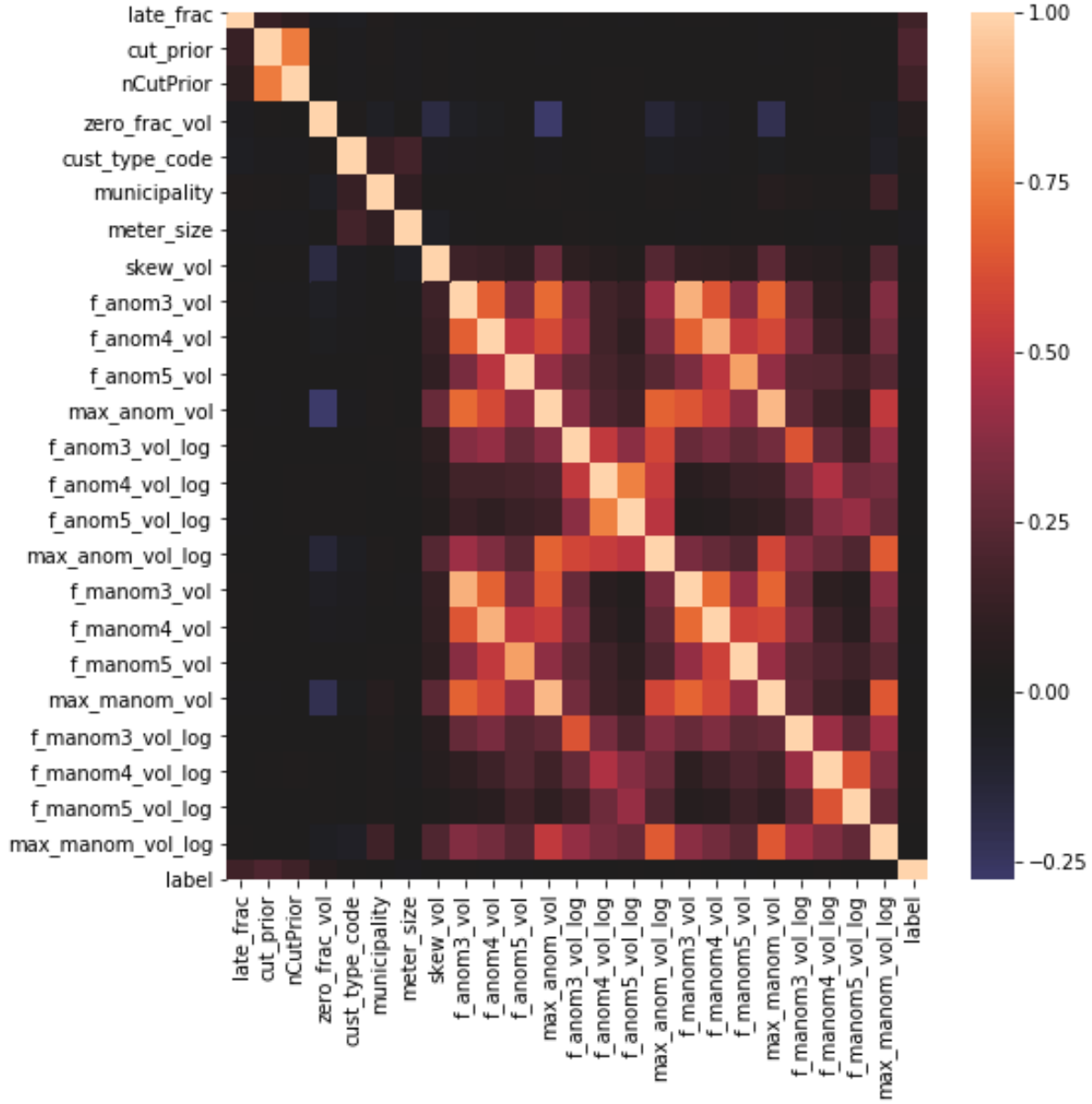


**Fig. 4.** Example monthly time series of lateness of payment (a, c) and volume used (b, d). Lateness is a binary variable, set to 1 if there was a late payment charge and 0 if not.

**Table 1.** Features explored in this study. ‘anom\_vol’ = standardized anomaly of volume, =  $(x - \text{long\_term\_mean}) / \text{long\_term\_std\_dev}$ . ‘manom\_vol’ = standardized anomaly of volume computed separately for each of the 12 months of the year (only computed for time series at least 24 months long).

Name	DependsOn	Type	Definition
label	window	int; trinary	Trinary window label; 0 = no cutoff ever; 1 = eventual cutoff; 2 = immediate cutoff
cutoff	occupant	int; binary	Occupant cutoff label; 0 = no cutoff ever; 1 = cutoff at some point
cutoff_strict	window	int; binary	Binary window label; trinary label = 0 or 1; 2 = immediate cutoff
nCutPrior	window	int	Number of prior cutoffs (before the current window)
cut_prior	window	int; binary	Binary flag; 0 = no prior cutoffs; 1 = at least one prior cutoff
cust_type_code	occupant	int; non-ordinal categorical	Customer type code
meter_size	occupant	float; non-ordinal categorical	Meter size
municipality	occupant	int; non-ordinal categorical	Municipality code
f_late	window	float	Fraction of payments that were late
skew_vol	window	float	abs(skewness) of the values of volume_kgals
f_zero_vol	window	float	Fraction of months with zero volume_kgals usage
f_anom{3,4,5}_vol	window	float	Fraction of months where $\text{abs}(\text{anom\_vol}) > \{3,4,5\}$
max_anom_vol	window	float	Maximum value of $\text{abs}(\text{anom\_vol})$
f_manom{3,4,5}_vol	window	float	Fraction of months where $\text{abs}(\text{manom\_vol}) > \{3,4,5\}$
max_manom_vol	window	float	Maximum value of $\text{abs}(\text{manom\_vol})$
f_anom{3,4,5}_vol_log	window	float	Fraction of months where $\text{abs}(\log(\text{anom\_vol})) > \{3,4,5\}$
max_anom_vol_log	window	float	Maximum value of $\text{abs}(\log(\text{anom\_vol}))$
f_manom{3,4,5}_vol_log	window	float	Fraction of months where $\text{abs}(\log(\text{manom\_vol})) > \{3,4,5\}$
max_manom_vol_log	window	float	Maximum value of $\text{abs}(\log(\text{manom\_vol}))$

In an effort to capture both positive spikes and zeros in volume usage, I created 18 different volume-related features (Table 1). We should expect them to be highly correlated with each other, but I had hoped that a small subset of them would contribute noticeably more predictive power than the rest. To check for predictive power and collinearity, I plotted the correlation matrix of all features, for N = 6 months (Fig. 5).



**Fig. 5.** Correlation matrix for all features and labels explored in this study, for sample window length  $N = 6$  months. 'label' is the trinary label described in Fig. 3. 'late\_frac' and 'zero\_frac\_vol' correspond to  $f_{\text{late}}$  and  $f_{\text{zero\_vol}}$  from Table 1.

From Fig. 5, we can see that:

1. The following features have non-trivial correlations with cutoffs (represented by trinary label label):  $f_{\text{late}}$ ,  $\text{cut\_prior}$ ,  $\text{nCutPrior}$ ,  $f_{\text{zero\_vol}}$ .

2. All of the anomaly metrics aside from `f_zero_vol` are highly correlated with each other. Out of all of them, the two with the highest correlation with the trinary `label` are `f_anom3_vol` and `f_manom3_vol`. The rest of them should just be ignored.
3. `cut_prior` and `nCutPrior` are highly correlated (0.95), so probably only one of these should be used.
4. `f_anom3_vol` and `f_manom3_vol` are highly correlated (0.84), so probably only one of these should be used; since `f_manom3_vol` is not meaningful for records less than 2 years long, we probably should go with `f_anom3_vol`.
5. `f_late`, `f_zero_vol`, and the pair (`f_anom3_vol`, `f_manom3_vol`) are essentially independent of each other.
6. The customer metadata have low correlations with the trinary `label` due to their being non-ordinal categorical variables; thus the correlation matrix doesn't accurately characterize their predictive power.

We ultimately can't use the prior cutoff features `cut_prior` or `nCutPrior` with most utility databases. However, I chose to evaluate models with and without prior cutoff information just to see how much predictive power they add.

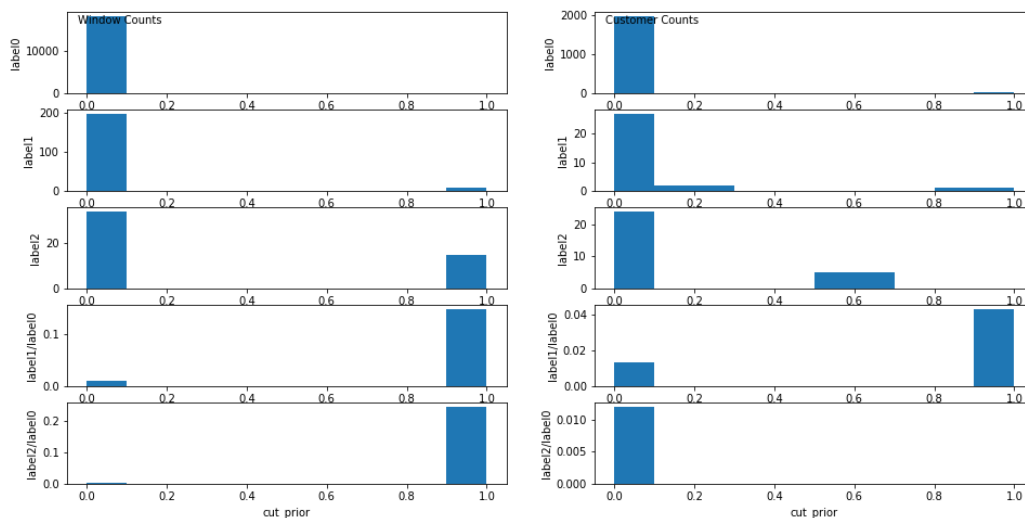
I was interested in seeing whether features from windows with trinary `label = 1` were more similar to windows with `label = 0` or `2`. If more similar to windows with `label = 0`, then those features would be good predictors of imminent cutoff. Histograms of values of the major features from windows with different `label` values are shown in Figs. 6-13. In each figure, the left column shows histograms of counts of windows for each bin of values of the feature, and the right column shows the same histograms, but of counts of unique occupants (since most occupants have multiple windows in each bin). The first 3 rows show histograms of windows or occupants with `label` values of 0, 1, and 2, respectively. The 4<sup>th</sup> and 5<sup>th</sup> rows show the ratios of the counts of `label 1` or `2` to `label 0`, respectively, which give some idea of the probabilities of windows having those label values.

From Figs 6-10 (the time-varying features), we can see that the ratios of `label=1` and `2` to `label=0` generally increase as feature values increase, indicating that these features have predictive value. However, in Figs 7-10, the rates of `label=2` do not differ substantially from those of `label=1`; indicating that these features are not good at distinguishing between immediate and eventual cutoffs. But in Fig. 6 (`cut_prior`), we can see that windows with `label=2` have `cut_prior` values of 1 at twice the rate of windows with `label=1`. This suggests that `cut_prior` could be used to distinguish between immediate cutoffs and eventual cutoffs. Again, `cut_prior` is not available in most utility databases, but this could be a motivation for a utility to add them (and for a commercial tool to have the option to use them).

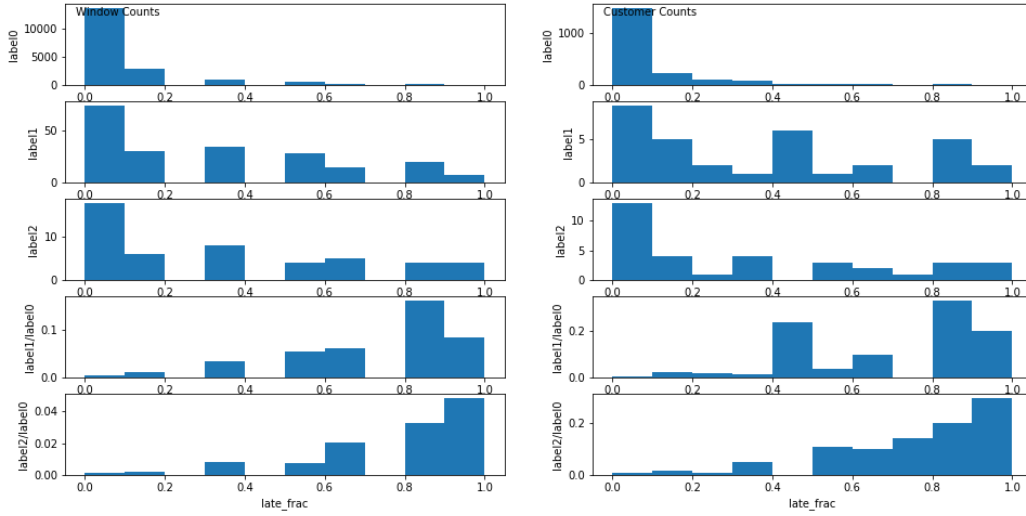
Because our deliverable is a prediction of cutoff risk, I needed to convert the trinary label back to a binary label. Because the only feature that can distinguish between immediate and eventual cutoffs is `cut_prior`, which we cannot depend on being available in most utility databases, I decided to assign trinary `label` values 0 and 1 to a binary `cutoff` value of 0, and trinary `label = 2` to binary `cutoff = 1`. ***This means that our models will only be able to estimate the probability of a customer having a cutoff at some point in the future, rather than an imminent cutoff.***

Figs 11-13 show the histograms of the customer metadata features. From these, we can see that `cust_type_code` (Fig. 11) is very imbalanced, with single-family residential (`code=1`) severely

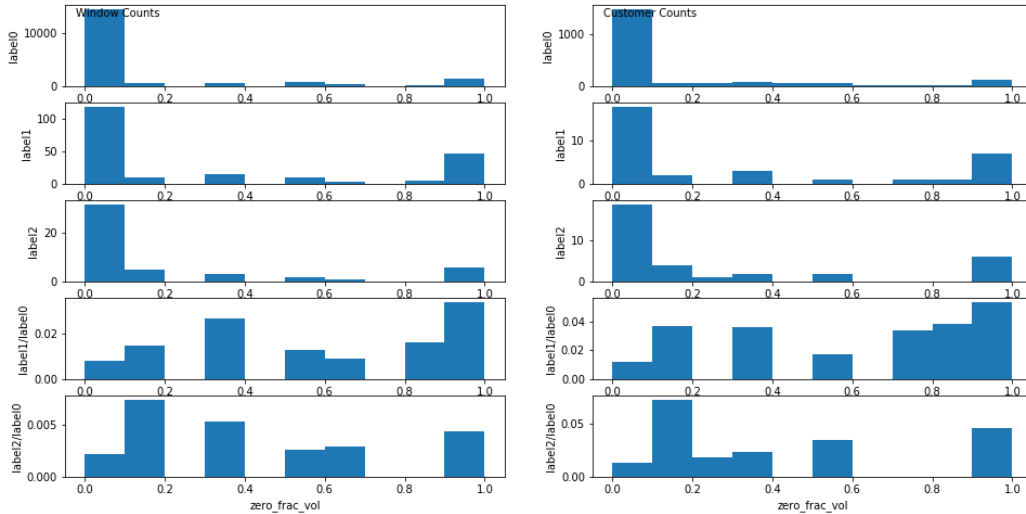
underrepresented relative to commercial (code=3), and rates of cutoffs (`label=1,2`) 10 times those of commercial. It seems likely that the small number of single-family residential customers might lead to sample error in the rates of cutoffs in the training set. Therefore, while I will continue exploring the value of customer metadata in the models, I strongly recommend using a larger, more representative training set to train the models for production.



**Fig. 6.** Histograms of counts of `label` values and their ratios across values of `cut_prior` feature. Left column: counts of values across all sample windows. Right column: counts of unique occupants having those `label` values. 'label0', 'label1', and 'label2' are values 0, 1, and 2 of the trinary `label` described in Fig. 3.

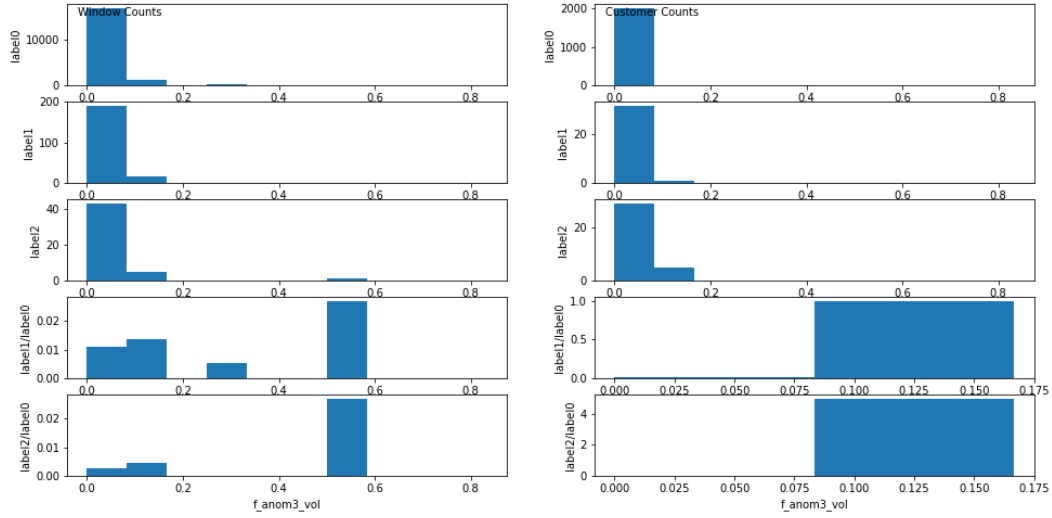


**Fig. 7.** Histograms of counts of `label` values and their ratios across values of `f_late` feature (labeled as '`late_frac`' here). Left column: counts of values across all sample windows. Right column: counts of unique occupants having those `label` values. '`label0`', '`label1`', and '`label2`' are values 0, 1, and 2 of the trinary `label` described in Fig. 3.

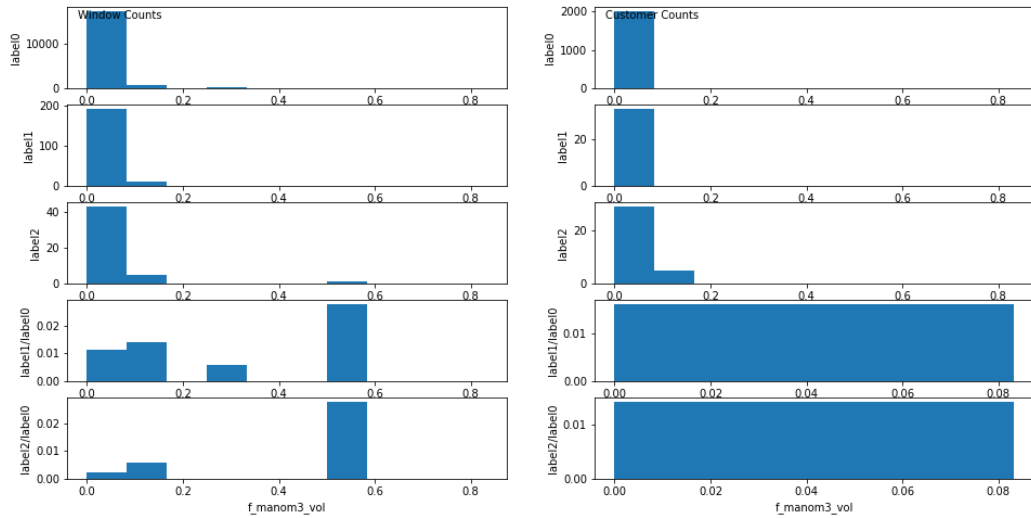


**Fig. 8.** Histograms of counts of `label` values and their ratios across values of `f_zero_vol` feature (labeled as '`zero_frac_vol`' here). Left column: counts of values across all sample windows. Right column: counts of unique occupants having those `label` values. '`label0`', '`label1`', and '`label2`' are values 0, 1, and 2 of the trinary `label` described in Fig. 3.



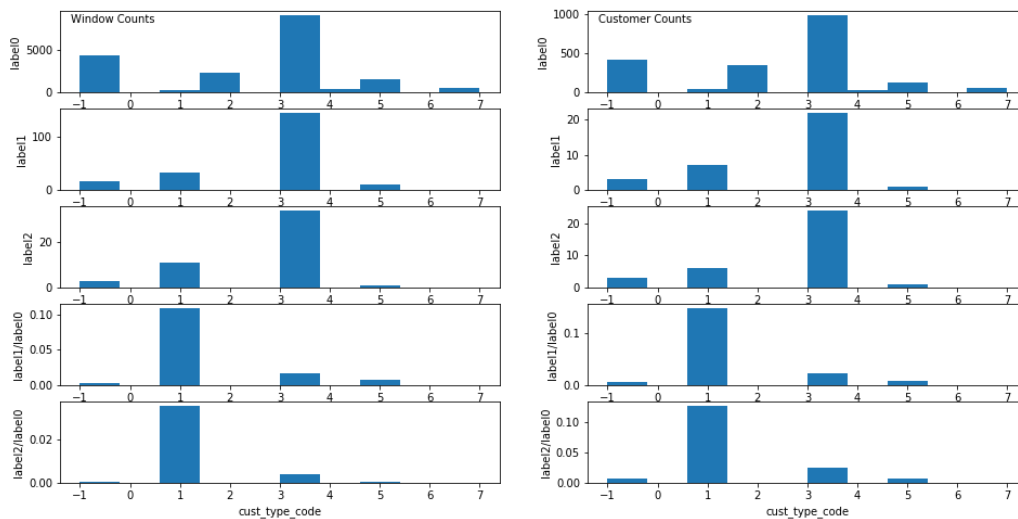


**Fig. 9.** Histograms of counts of `label` values and their ratios across values of `f_anom3_vol` feature. Left column: counts of values across all sample windows. Right column: counts of unique occupants having those `label` values. 'label0', 'label1', and 'label2' are values 0, 1, and 2 of the trinary `label` described in Fig. 3.

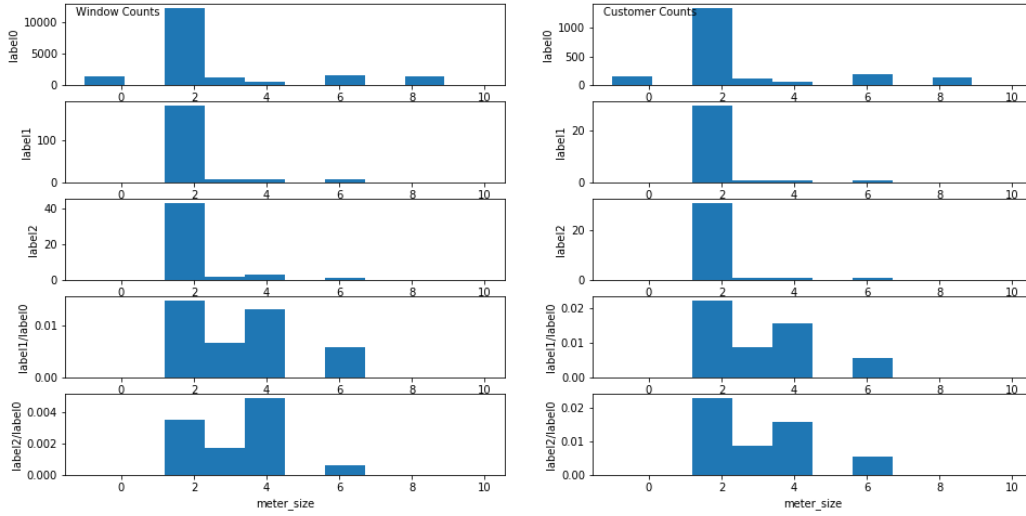


**Fig. 10.** Histograms of counts of `label` values and their ratios across values of `f_manom3_vol` feature. Left column: counts of values across all sample windows. Right column: counts of unique

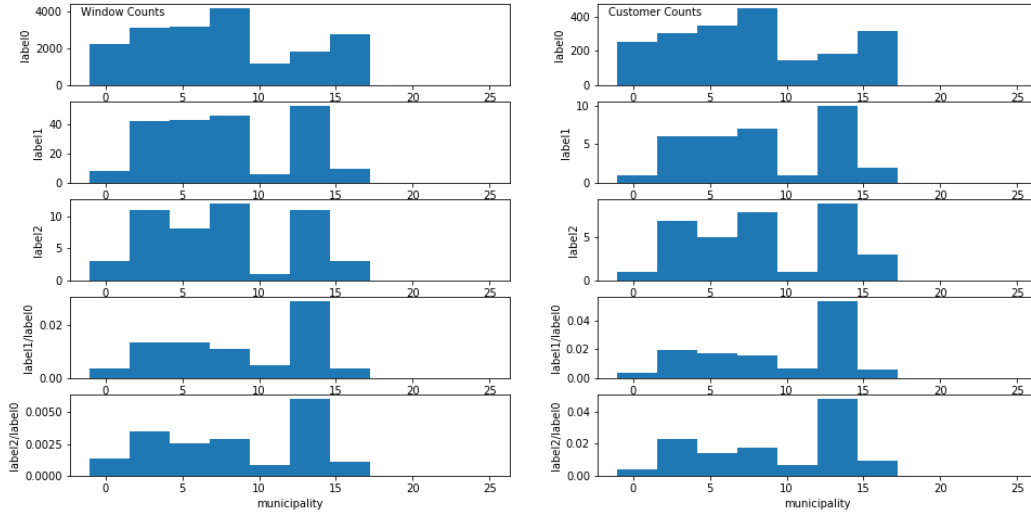
occupants having those `label` values. 'label0', 'label1', and 'label2' are values 0, 1, and 2 of the trinary `label` described in Fig. 3.



**Fig. 11.** Histograms of counts of `label` values and their ratios across values of `cust_type_code` feature. Left column: counts of values across all sample windows. Right column: counts of unique occupants having those `label` values. 'label0', 'label1', and 'label2' are values 0, 1, and 2 of the trinary `label` described in Fig. 3.



**Fig. 12.** Histograms of counts of `label` values and their ratios across values of `meter_size` feature. Left column: counts of values across all sample windows. Right column: counts of unique occupants having those `label` values. ‘`label0`’, ‘`label1`’, and ‘`label2`’ are values 0, 1, and 2 of the trinary `label` described in Fig. 3.



**Fig. 13.** Histograms of counts of `label` values and their ratios across values of `municipality` feature. Left column: counts of values across all sample windows. Right column: counts of unique occupants having those `label` values. ‘`label0`’, ‘`label1`’, and ‘`label2`’ are values 0, 1, and 2 of the trinary `label` described in Fig. 3.

### 3.3.3. Distribution of Cutoffs as a Function of Window Length

As window length  $N$  increases, the number of cutoffs will decrease, due to (a) some customers having histories shorter than 1 window and (b) customers with multiple cutoffs having some cutoffs separated by less than 1 window length. Given the imbalance in the dataset, and in particular the potential for sample error when the number of labels is small, we might expect model performance to decrease as  $N$  increases.

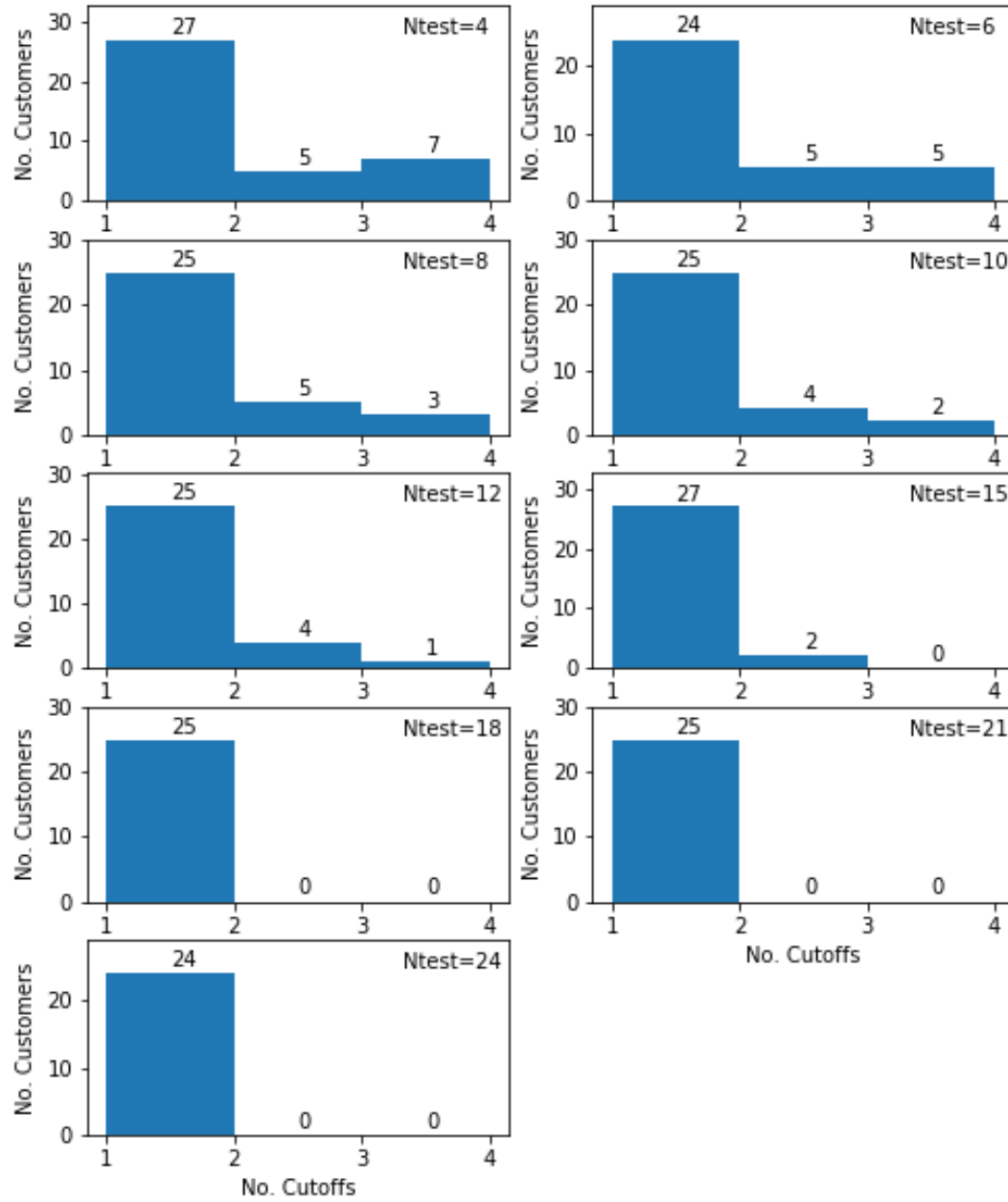
Another issue of potential concern is that I have chosen to include all cutoffs in the training/test dataset, as if they are all independent, because we have so few of them. The problem with this is that, if a small subset of customers has very many cutoffs, this could bias the results towards those customers' behaviors.

A further problem is that, again due to the small number of cutoffs, splitting into train/test subsets runs a risk of placing several cutoffs from the same customer into the same subset, leading to bias in training and in testing. The worst case for this is if the number of repeated cutoffs is neither extremely large nor extremely small. If there are very few repeated cutoffs, this whole issue is not important. If there are a large number of repeated cutoffs (more than half the total number), then test/train split is more likely to place some repeated cutoffs into each subset. The worst case is for the number of repeated cutoffs to fall somewhere in between (e.g., 25% of the total).

Therefore, I have plotted histograms of the numbers of cutoffs per customer for values of  $N$  ranging from 4 to 24 months (Fig. 14). From these histograms, we can see that at short window lengths, there are a substantial number of customers with multiple cutoffs (e.g. for  $N=4$  months, 9 out of 38 customers, or  $> 25\%$ ; in terms of cutoffs, the numbers are: 23 out of 54 cutoffs come from customers with multiple cutoffs, or nearly  $50\%$ ; if we only took one cutoff per customer, we'd have 38 cutoffs to work with). For window lengths longer than 12 months, there are very few with multiple cutoffs. This would imply that the longer window lengths would be better for avoiding bias from repeated cutoffs from a single customer.

But at longer window lengths, there are also fewer cutoffs in general (23-28 cutoffs at lengths  $\geq 15$  months, as opposed to 54 for a 4-month window). This might adversely impact model performance.

Another interesting trend is that window lengths from 6 to 15 months all have the same set of 27 customers with a single cutoff. The only changes in the distribution as window length increases is the decline in the number of customers with multiple cutoffs.



**Fig. 14.** Histograms of the numbers of cutoffs per customer (for customers with at least 1 cutoff), for different values of window length  $N$  (labeled as 'Ntest' here).

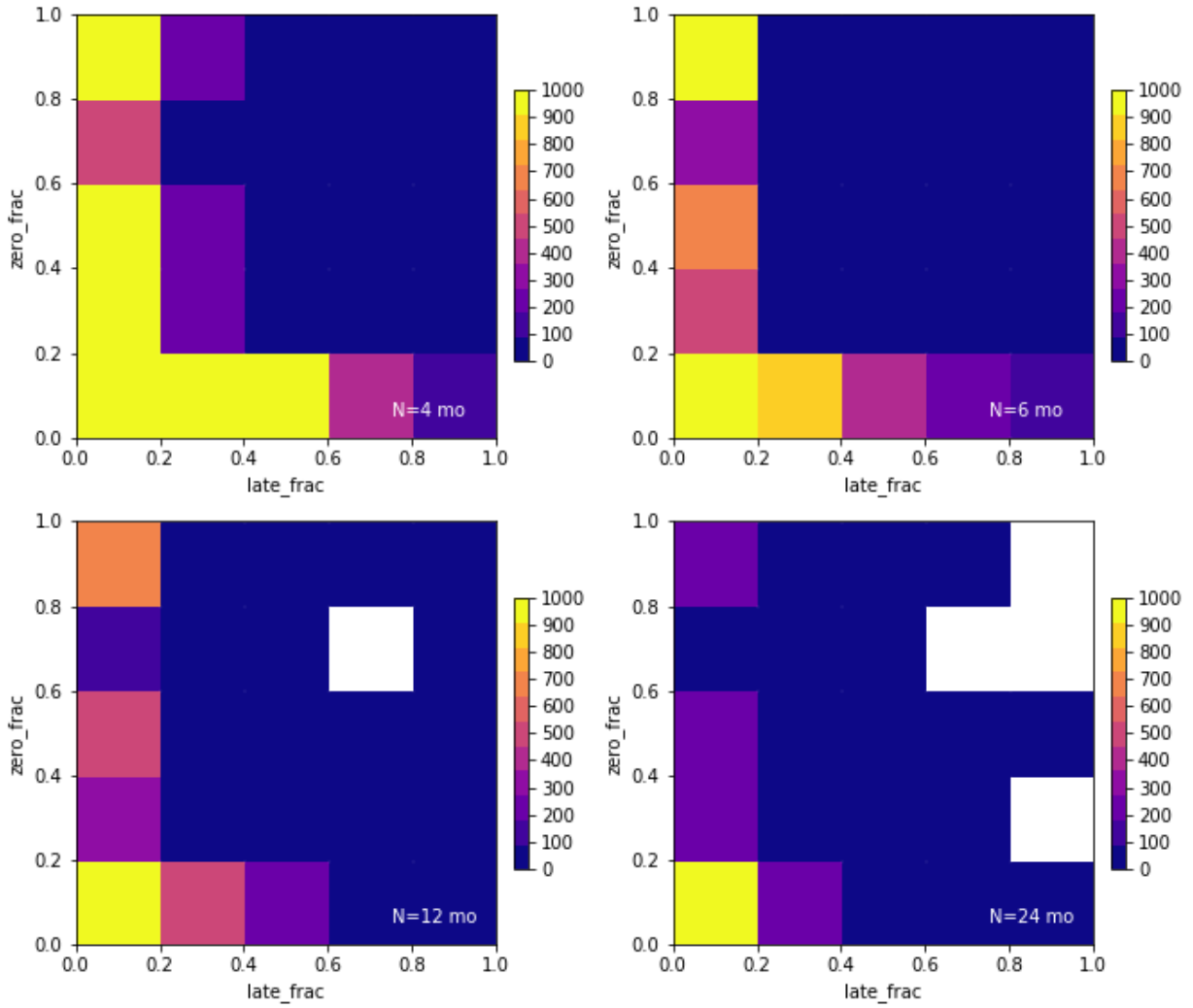
#### 3.3.4. Cutoff Rates as Empirical Functions of 2 Features

Before fitting models to the features and labels, I wanted to get an idea of how the labels related to the features, empirically, to gain insight into which models might perform better or worse and why.

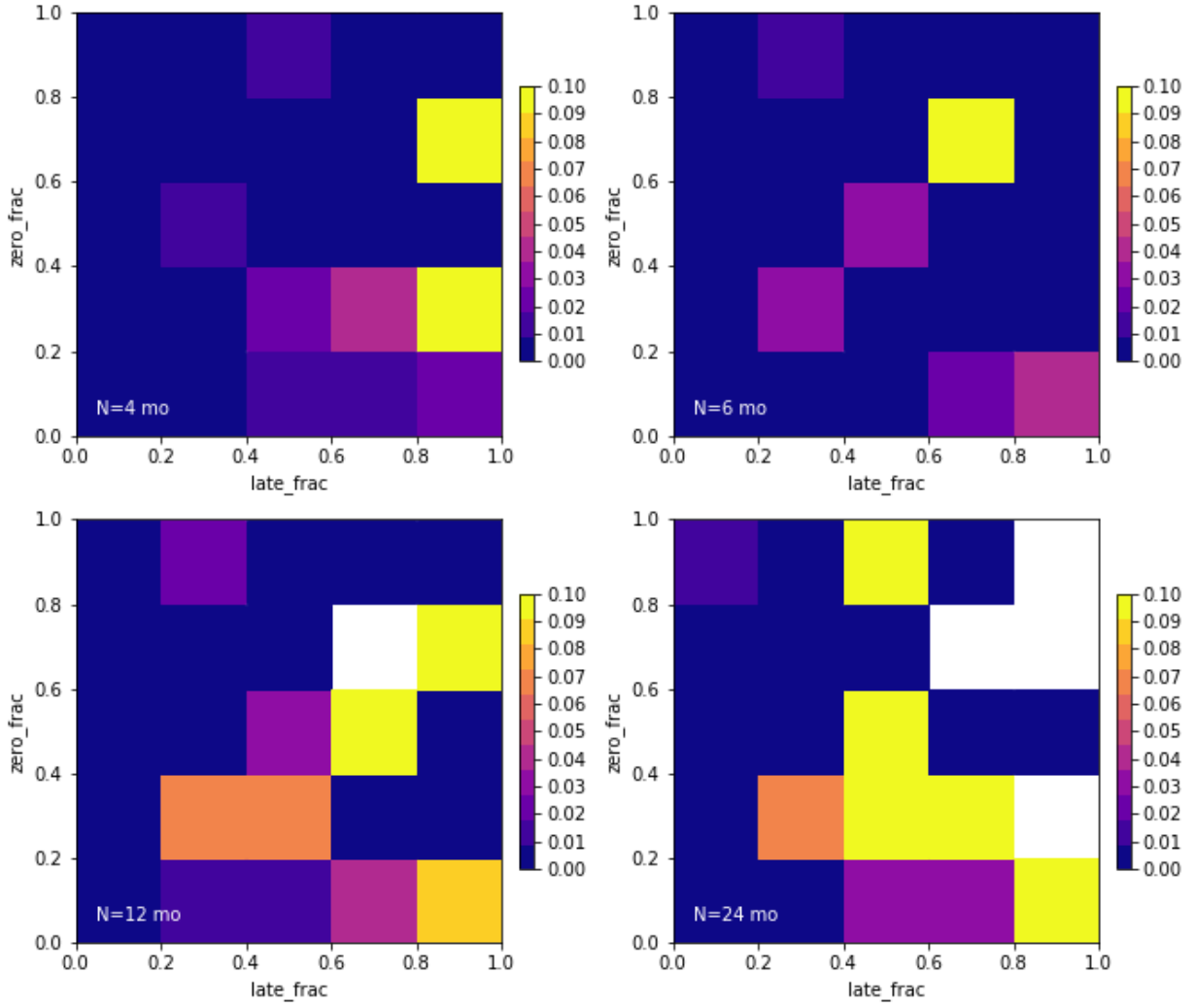
Fig. 15 shows 2-dimensional histograms of values of `f_late` and `f_zero_vol` from all windows, regardless of label values. As we would expect, windows with large values of both are rare, due to the low correlation between `f_late` and `f_zero_vol`.

Cutoff rates tend to increase with both `f_late` and `f_zero_vol` (Fig. 16-17), but at a bin width of 0.2, the relationship is quite noisy (Fig. 16). Increasing the bin width to 0.5 smooths out the relationships (Fig. 17) and shows that `N = 12` months has the cleanest joint relationship between cutoff rates and the two features. This implies that a model with a simple shape (e.g., logistic regression or a shallow-depth random forest, i.e. few divisions per feature) will be less susceptible to overfitting.

The relationship between cutoff rates, `f_late`, and `f_anom3_vol` (Fig. 18-19) exhibit similar behavior, with more noise at smaller bin widths, but here the correlations are weaker.

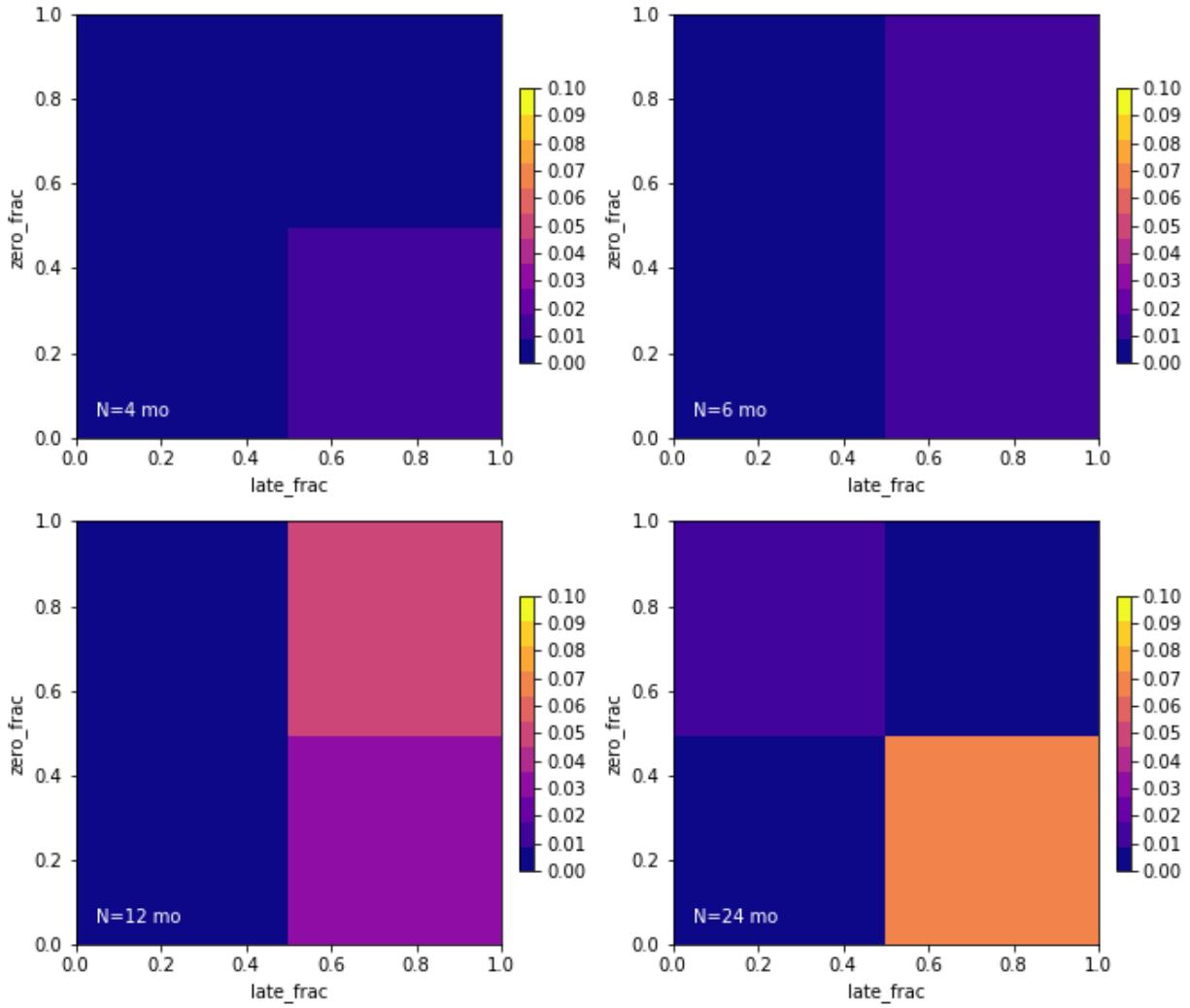


**Fig. 15.** 2-dimensional histograms of values of  $f_{\text{late}}$  and  $f_{\text{zero\_vol}}$  ('late\_frac' and 'zero\_frac' here) from all windows, regardless of `label` values.

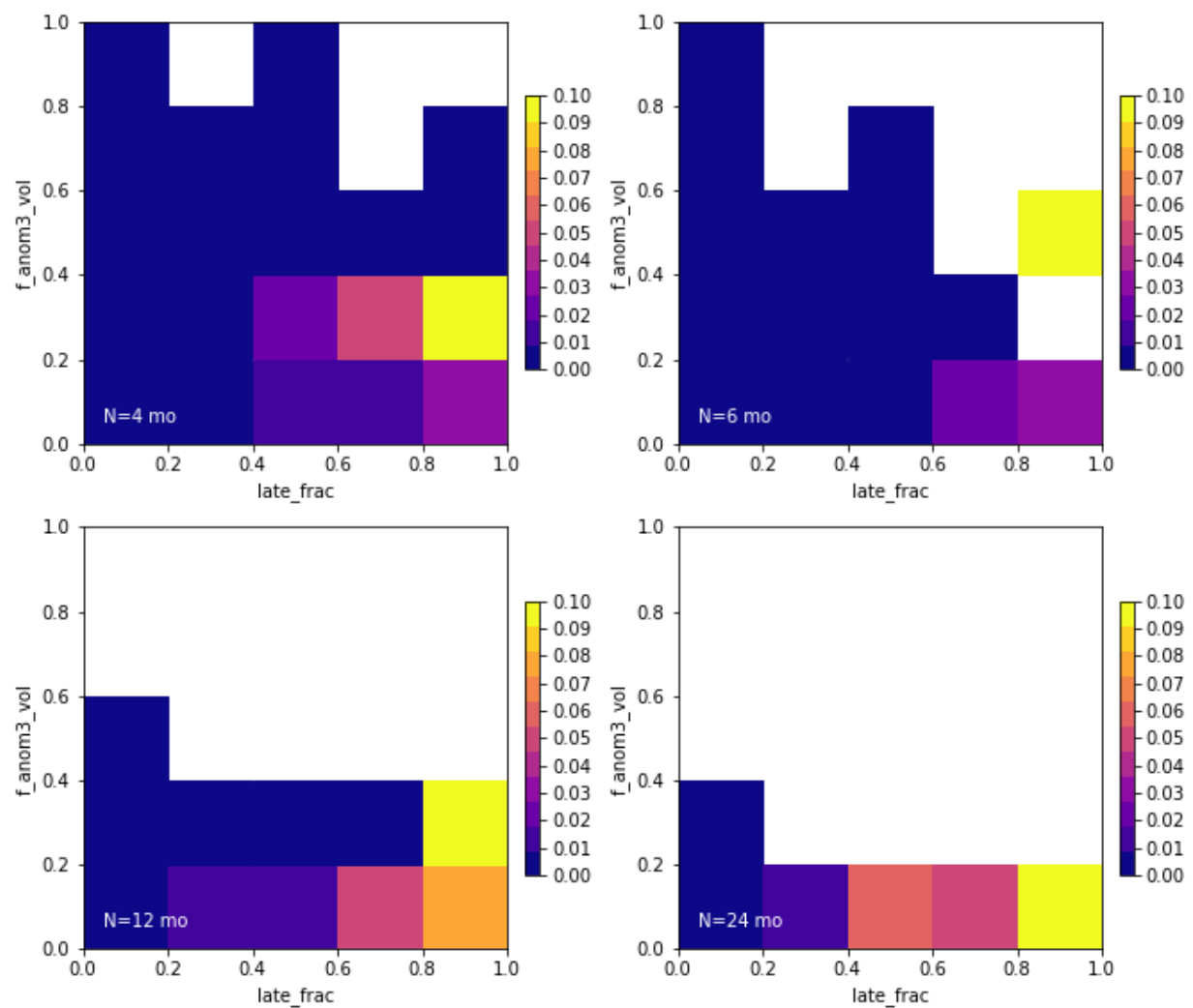


**Fig. 16.** Cutoff rates as a function of  $f_{\text{late}}$  and  $f_{\text{zero\_vol}}$  ('late\_frac' and 'zero\_frac' here), with bin size=0.2.

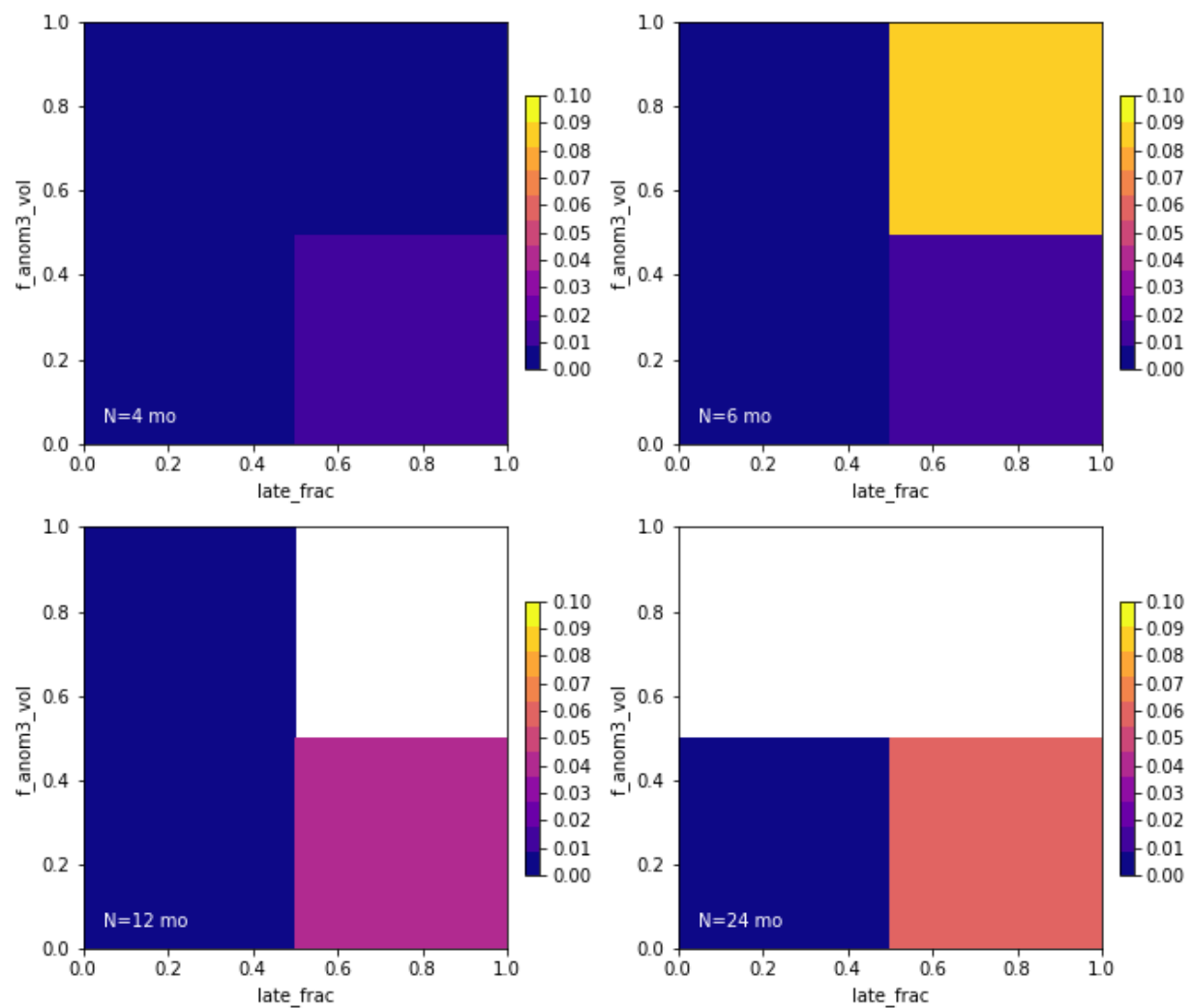




**Fig. 17.** Cutoff rates as a function of  $f_{\text{late}}$  and  $f_{\text{zero\_vol}}$  ('late\_frac' and 'zero\_frac' here), with bin size=0.5.



**Fig. 18.** Cutoff rates as a function of  $f\_late$  ('late\_frac' here) and  $f\_anom3\_vol$ , with bin size=0.2.



**Fig. 19.** Cutoff rates as a function of  $f_{\text{late}}$  ('late\_frac' here) and  $f_{\text{anom3\_vol}}$ , with bin size=0.5.

### 3.3.5. Major Feature Combinations Explored in this Study

To explore the predictive value given by various features, I have implemented 8 feature combinations to evaluate in the models, listed in Table 2.

All cases share a “base” feature list of: [f\_late, f\_zero\_vol]. Furthermore, all cases will include one of [f\_anom3\_vol, f\_manom3\_vol], determined by the ‘FEATURES\_ANOM’ config option. If ‘FEATURES\_CUT\_PRIOR’ is set to True, cut\_prior will be added to the feature list. If ‘FEATURES\_METADATA’ is set to True, [cust\_type\_code, municipality, meter\_size] will be added to the feature list.

**Table 2.** Major Feature Combinations.

Case	FEATURES_CUT_PRIOR	FEATURES_METADATA	FEATURES_ANOM
1	True	True	anom
2	True	True	manom
3	True	False	anom
4	True	False	manom
5	False	True	anom
6	False	True	manom
7	False	False	anom
8	False	False	manom

## 4. Models and Parameters

CutoffPredictor searches across several options for an optimal model in terms of cross-validation performance metrics (specifically the AUC). The options include: the sample window length `N` used in feature computation; the model structure (random forest or logistic regression); and key values of model parameters (for random forest, the maximum node depth parameter `max_depth`; for logistic regression, the regularization coefficient `lambda`). These models and parameters are summarized in Table 3.

In version 1.0, I used models from the SciKit-Learn package. However, I later discovered that the SciKit-Learn implementations have several limitations; in particular, SciKit-Learn's random forest algorithm does not handle non-ordinal categorical features correctly (it treats them as if they are ordinal, requiring deeper trees to adequately separate groups of non-adjacent classes, potentially leading to over-fitting). Therefore, for version 2.0, I switched to the H2O package. H2O's random forest algorithm handles non-ordinal categorical features correctly, and also internally handles one-hot encoding of these features for the logistic regression, as well as handling standardization of features internally for both types of models.

For the grid search over `max_depth`, I was unable to implement H2O's internal grid search correctly as it involved a call to a separate class (a grid search class), for which it was not clear how to invoke the cross-validation options that I wanted. Therefore, I implemented a loop over `max_depth` values that fit a random forest model separately for each value.

For model training and validation, I divided the training period randomly into 80%-20% training and test subsets, while ensuring that cutoff windows and non-cutoff windows were each divided in the same proportions. I further ensured that first cutoffs (a customer's first cutoff, if the customer had more than one, or otherwise the customer's sole cutoff) and subsequent cutoffs (for customers that had more than one) were divided in those same proportions. This minimized the risk that multiple cutoffs from a single user ended up in one or the other subset, potentially biasing the performance metrics.

In version 1.0, I balanced the labels in the training subset by calling the SMOTE oversampling package, which created synthetic cutoff records to achieve equal numbers of records. In version 2.0, I decided to instead achieve balance by assigning weights to the records (H2O models can take a weights column as an additional feature). Weights were computed as the ratio of the total number of records / the number of records with that value of the label.

**Table 3.** Models and parameter values in this study.

Model	H2O Class	Parameter	Value
Random Forest	H2ORandomForestEstimator	ntrees	100
		max_depth	grid search over values 3-20; best value (from cross-validation) almost always = 3
Logistic Regression	H2OGeneralizedLinearEstimator	family	'Binomial'
		link	'logit'
		lambda_search	True (elastic_net optimal lambda value is found via grid_search)
		alpha	0.5 (L1 coefficient, i.e. strength of L1/(L1+L2))
		solver	'AUTO' (for lambda_search=True, 'AUTO' chooses solver='COORDINATE_DESCENT')
Both		nfolds (cross-validation folds)	5
		fold_assignment (cross-validation fold assignment)	'Stratified'

Model performance statistics are plotted as a function of window length N, for the 8 feature set cases described in Table 3, in Figs 20 (cross-validation) and 21 (test subset). From cross-validation (Fig. 20), we can see the following:

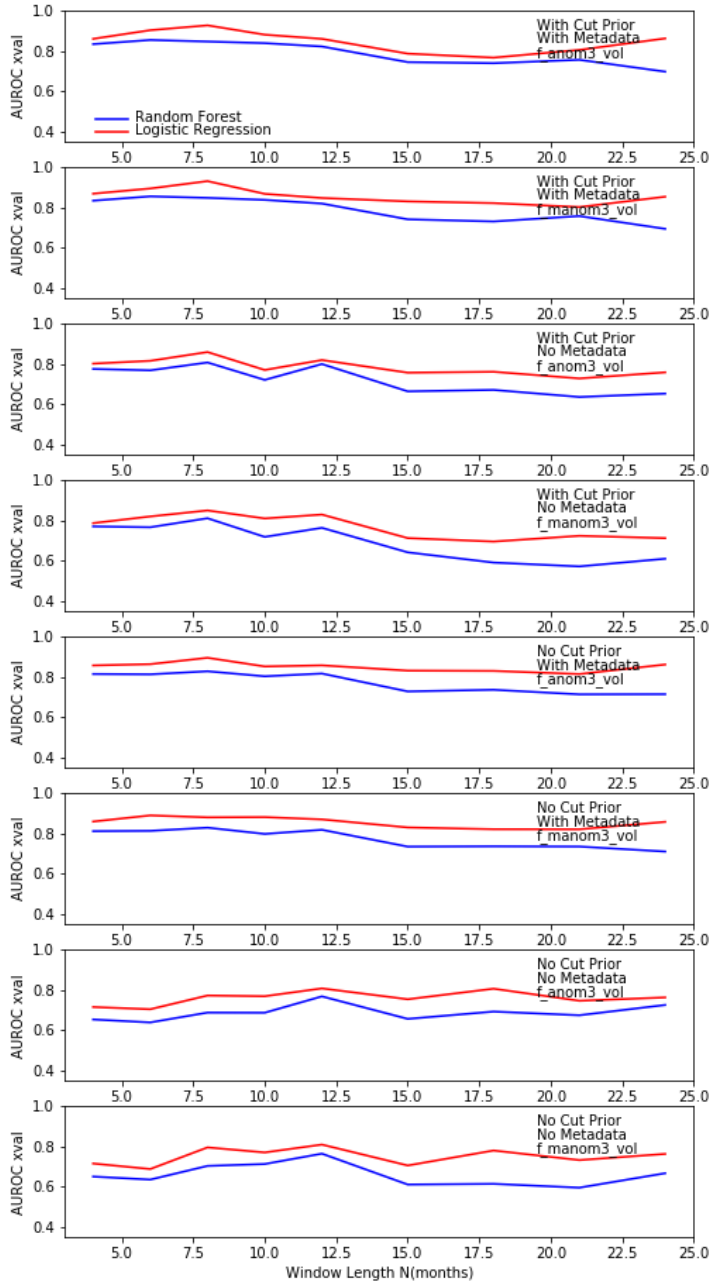
1. Logistic regression outperforms random forest for all cases.
2. When including either prior cutoffs or customer metadata in the feature set, performance peaks at shorter window length N (< 12 months) and declines gradually with N thereafter.
3. If NOT including either prior cutoffs or metadata, performance is worst at shorter N and increases gradually with N; it's hard to tell because of noise from the random partitioning of cross-validation, but performance appears to peak for both logistic regression and random forest at N=12 months; for logistic regression it remains roughly constant at longer N, while for random forest it declines somewhat.
4. For both types of models, the best performance occurs when including both prior cutoffs and metadata as features, at N = 6-8 months (AUC=0.92-0.93 for logistic regression; 0.84-0.85 for random forest).
5. For both types of models, excluding either prior cutoffs or metadata reduces performance modestly.
6. For both types of models, excluding both gives the worst performance, particularly at short window lengths; the best performance in this case is for N = 12 months (and also 18 and 24 months for logistic regression) (AUC=0.81 for logistic regression and 0.77 for random forest).
7. Using `f_anom3_vol` yields slightly better performance than `f_manom3_vol`, with the added advantage that `f_anom3_vol` does not require at least 2 years' worth of data to compute and thus is more useful for new customers.

I implemented a grid search over the `max_depth` parameter over the range 3-20 for each window length `N` and feature combination, using the cross-validation AUC score as the metric to optimize. For most cases, the optimal `max_depth` parameter was 3, even when there were 7 features (4 of which were categorical). It seems that limiting `max_depth` prevented over-fitting; however, it is surprising that the optimal value is less than the number of features. It might be that restricting the depth to 3 results in some trees that use the 3 continuous features and which perform as well as the no-metadata case, and other trees that use few or no continuous features and which perform poorly and possibly cancel out, yielding similar performance to the no-metadata case.

Given that window lengths  $> 12$  months rarely yield better performance than  $N = 12$  months, and that longer windows reduce the model's ability to recognize changes in customer behavior that might predict cutoffs, I recommend limiting the range of `N` to search over to 4-12 months. Similarly, given that the optimal `max_depth` tended to be 3, I recommend limiting the range of `max_depth` to search over to 3-5 nodes. I still think it's OK to explore both logistic regression and random forest in practice, particularly if/when a larger training set with better representation of all customer types is available; random forest might perform better if non-ordered categorical features such as customer type or municipality play a large role.

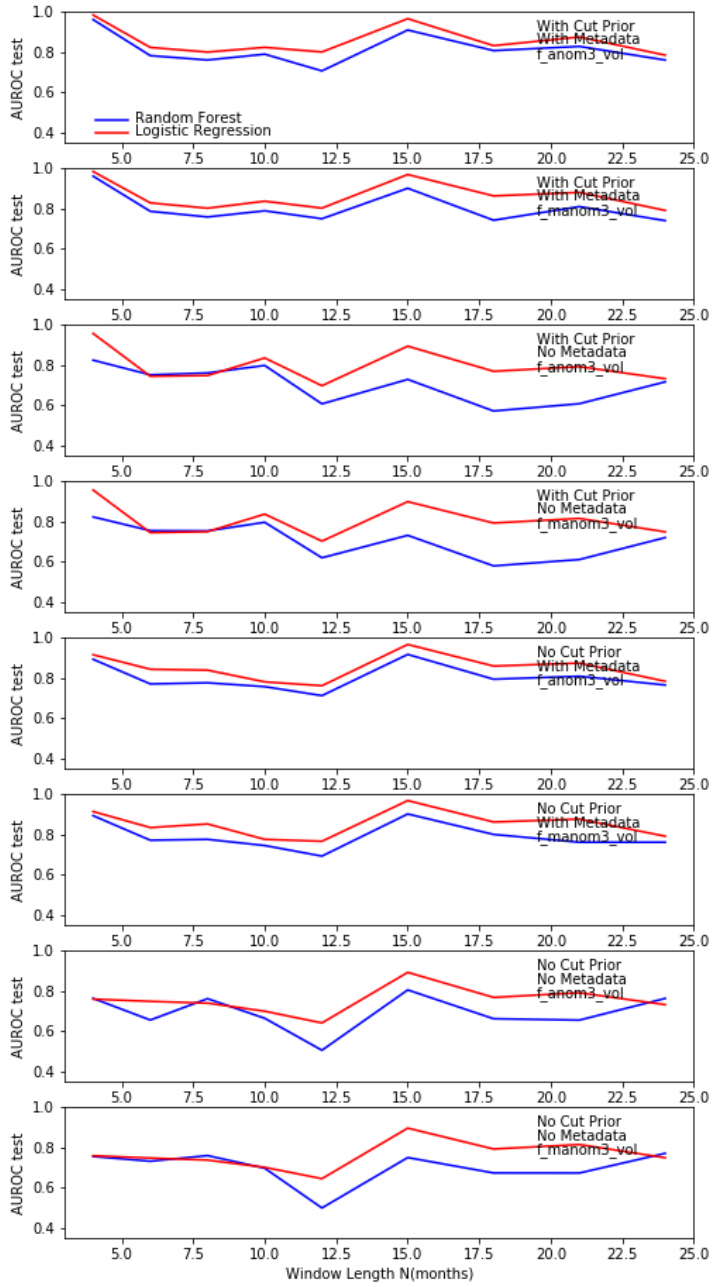
I have implemented options for the user to choose whether to use customer metadata or information about prior cutoffs in the model. This will allow the model to take advantage of these when they are present, and ignore them otherwise.

Model performance on the test subset (Fig. 21) was much noisier than for cross-validation. For the best-fitting `N` values from cross-validation, AUC scores on the test subset were lower (AUC ranging from 0.7 to 0.8). However, for other values of `N`, the AUC scores were quite high in the test subset ( $> 0.9$ ), even higher than for the cross-validation. This suggests that the scores for the test subset are very uncertain, likely due to the small number of cutoffs available to work with. A larger train-test dataset could give a fairer assessment of model performance.



**Fig. 20.** Cross-validation AUC scores for random set forest and logistic regression models as a function of sample window length N, for the 8 feature set cases described in Table 3.





**Fig. 21.** Test subset AUC scores for random forest and logistic regression models as a function of sample window length  $N$ , for the 8 feature set cases described in Table 3.

## 5. Tool/Dashboard

### 5.1. Requirements

CutoffPredictor is written in Python, version 3.4 or later. It imports several non-default packages, which in turn have some external dependencies. These requirements are listed in Table 4.

**Table 4.** CutoffPredictor System Requirements

Category	Needed For...	Component
<b>Accounts</b>		
Amazon Redshift	Database access	
Google Maps	Geocoding	
MapBox	Map plotting in dashboard	
<b>Software</b>		
PostGre SQL	Database access	
Java 64-bit JDK, version 11	Required by Python H2O package	
Flask	Dashboard access in browser	
Plotly/Dash	Dashboard	
Python 3.4 or later	CutoffPredictor code	Default distribution, including: math, scipy, numpy, datetime, pandas, requests, shutil, argparse, os
		psycopg2 (PostgreSQL interface)
		sklearn (for ML utilities; aka SciKit-Learn)
		H2O (for ML models)
		imblearn (for SMOTE oversampling)
		flask (browser support)
		plotly (dashboard code)
		dash (dashboard code)

### 5.2. CutoffPredictor Installation and Setup

CutoffPredictor does not currently have a formal installation or setup process. Only 3 steps need to be taken:

1. Download the code from github (<https://github.com/tbohn/CutoffPredictor>) and make sure all of the dependencies have been installed.
2. Create a data directory tree with the following structure (where DATA\_DIR should be replaced by the actual path on your machine):

/DATA\_DIR/

data\_tables/ (tables queried from the utility database are stored here as .csv files)

data\_tables\_clean/ (cleaned versions of the database tables)

feature\_tables.train/ (tables of features computed for the training/testing period)

predictions.train/ (tables of predictions and probabilities for the training/testing period)

saved\_models/ (best-performing models saved here as json files)

model\_perf/ (model performance statistics)

feature\_tables.pred/ (tables of features computed for the prediction period)

predictions.pred/ (tables of predictions and probabilities for the prediction period)

3. Create a config file by copying from the example config template and replacing the placeholders for file paths and database and map account names with the appropriate information. Then, CutoffPredictor is ready to run.

## 5.3. CutoffPredictor Process Flow and Usage

### 5.3.1. Components and Process Flow

CutoffPredictor has two components: backend and dashboard. The backend performs the functions of data access, data processing, model training, and model prediction. The backend code is executed from the command line in a linux shell. The dashboard displays model predictions. It must be executed by a linux shell command and then viewed inside the user's web browser.

The CutoffPredictor process flow is outlined in Table 5. The CutoffPredictor models must be trained on a user-defined subset of the utility database, and ideally this training dataset should be updated periodically. Once the models are trained and the best model identified, the final CutoffPredictor model can make a prediction at any time, for example as new data arrive.

The reason I designed this tool to have a model training step is so that Valor Water Analytics can continue working with the models to refine this process. Thus CutoffPredictor in its present form is more of a research tool than a product for general consumption.

**Table 5.** CutoffPredictor process flow.

Stage	Component	Description	Execution	Frequency
Data Download	Backend	Download data from database.	CutoffPredictor.py in linux shell	Whenever models need to be retrained or new predictions need to be made (incorporating new data).
Data Preparation	Backend	Clean/prepare the records for later stages.	CutoffPredictor.py in linux shell	Whenever models need to be retrained or new predictions need to be made (incorporating new data).
Feature Preparation	Backend	Prepare features from the data over the training period. Several feature tables will be prepared over a range of sample window lengths.	CutoffPredictor.py in linux shell	Prior to each model re-training.
Model Training	Backend	Train and evaluate models over the features prepared from the training data. Select the best model and parameters.	CutoffPredictor.py in linux shell	Whenever models need to be retrained.
Prediction	Backend	Apply the best model to the data within the chosen prediction window.	CutoffPredictor.py in linux shell	Whenever a new prediction needs to be made (incorporating new data).
Display	Dashboard	Display predictions in dashboard.	CPdashboard.py in linux shell to start dashboard application; web browser to view	Whenever the most recent prediction is to be viewed.

### 5.3.2. Usage

To execute the various stages of CutoffPredictor process flow, one must execute the following command within a linux shell:

For all non-dashboard stages:

```
python CutoffPredictor.py config_file >& log_file
```

where

`config_file` = text file that provides option settings and instructions for the application. An example config file template is provided with the code distribution in [github](#)

`log_file` = text file containing information about process steps, warnings, and errors

For the dashboard:

```
python CPdashboard.py config_file
```

where `config_file` is the same `config_file` used for the backend prediction step

In addition, once the CPdashboard.py application is running, the dashboard can be viewed by starting a browser and navigating to URL = 127.0.0.1:8050.

The config file consists of sections of related key-value pairs in the format:

```
[SECTION1]
KEY1 : value1
KEY2 : value2
...

[SECTION2]
...
```

To prepare the config file for a particular stage, one can make a copy of the example template provided with the code distribution and replace the placeholders for file paths and database and map account names with the appropriate information (in the [PATHS], [DATABASE], and [MAPPING] sections). Then, appropriate option settings should be chosen according to Table 6.

**Table 6.** Config file option settings for each processing stage

Stage	Section	Key	Value
Download	[STAGES]	DOWNLOAD	True
		All other keys	False
	[DATABASE]	All keys	Valid account and database information
Data preparation	[STAGES]	PREP_DATA	True
		All other keys	False

	[MAPPING]	All keys	Valid account information
Feature preparation	[STAGES]	PREP_FEATURES	True
		All other keys	False
	[TRAINING]	REF_DATE	Desired reference date, delineating the end of the training period (the beginning is assumed to be the first date in the database).
		N_SAMPLE_LIST	Comma-separated list of sample window lengths to evaluate during model training.
Model Training	[STAGES]	TRAIN_MODELS	True
		All other keys	False
	[TRAINING]	REF_DATE	Desired reference date, delineating the end of the training period (the beginning is assumed to be the first date in the database).
		N_SAMPLE_LIST	Comma-separated list of sample window lengths (in months) to evaluate during model training (recommend values spanning from 4 to 12).
		MODEL_TYPES	Comma-separated list of model types (recommend setting to 'random_forest, logistic_regression').
		MAX_DEPTH_LIST	Comma-separated list of maximum tree node depths to consider (recommend 3,4,5).
		FEATURES_CUT_PRIOR	True to include cut_prior in the feature set; else False.
		FEATURES_METADATA	True to include customer metadata in the feature set; else False.
		FEATURES_ANOM	'anom' for ordinary volume anomalies

			(recommended); 'manom' for monthly anomalies.
		REGULARIZATION	True
Model Prediction	[STAGES]	PREDICT	True
		All other keys	False
	[TRAINING]	All keys	Same values as for 'Model Training' stage.
	[PREDICTION]	REF_DATE	Set to reference date for prediction, e.g., current date.
		For FEATURES_*, model must have been trained with the same combination of values, else prediction will fail.	
		FEATURES_CUT_PRIOR	True to include cut_prior in the feature set; else False.
		FEATURES_METADATA	True to include customer metadata in the feature set; else False.
		FEATURES_ANOM	'anom' for ordinary volume anomalies (recommended); 'manom' for monthly anomalies.

### 5.3.3. Dashboard

The CutoffPredictor dashboard is illustrated in Fig 22. The dashboard gives the user the ability to select a desired risk threshold (upper left corner). For a given risk threshold, the dashboard will display analytics for those customers whose cutoff probability is above the threshold:

- the total number of predicted cutoffs
- the monthly total lost revenue and water (computed as expected value, i.e., sum over customers of cutoff probability \* mean monthly charges or volume used)
- the mean monthly lost revenue and water per customer
- a map of at-risk customers, colored by their cutoff probability
- a map of at-risk customers, colored by their mean monthly charges or volume used
- a breakdown (pie chart) of the customers by metadata such as customer type, municipality, or meter size
- a table of all relevant information about the at-risk customers

The ability to select a risk threshold enables the user to decide which customers to prioritize and how to allocate resources.

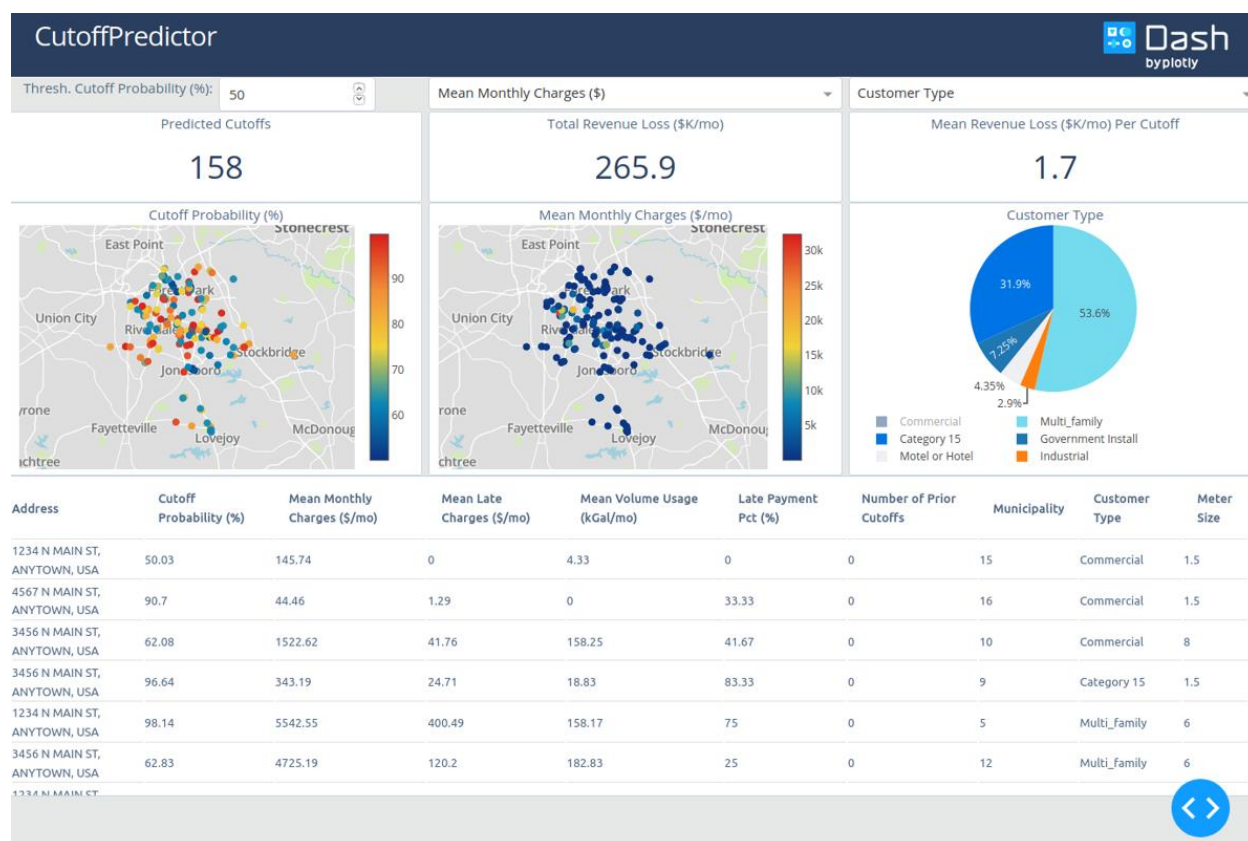


Fig. 22. CutoffPredictor dashboard.



## 6. Recommendations for Future Releases

For future releases, I have some recommendations:

- Implement a formal setup.py
- Use a larger, more representative training dataset (in particular, ensure better representation in the single- and multi-family residential classes, i.e., codes 1 and 2).
- Replace the training period REF\_DATE with START\_DATE and END\_DATE in the config file. Currently, all records prior to the REF\_DATE are included in the training set, but more control over the training period would be desirable.
- Add N\_CUST and N\_CUTOFFS options to the config file. These would control the total number of customers and total number of cutoff customers to include in the training set (currently all available customers and cutoffs are included). Ideally, with a larger database to work with, we would like to randomly select a subset of customers for training.
- Eliminate the date alignment processing step (but this might require more sophisticated logic around window clipping).
- Explore a wider variety of features derived from variables in the **charge** table (e.g., late fee amount, total charge).
- Explore replacing the municipality code with other features that are correlated with municipality but whose values are not specific to the pilot test utility (e.g., median income of census tract). This would have the advantage that the model could be trained in one region and applied to other utilities without requiring re-training of the models.
- Explore adding a lead-time dimension to the predictions, along the lines of survival analysis. I.e., incorporate a probability distribution of the time until a cutoff into the model prediction.
- Given that prior cutoffs are strongly predictive of future cutoffs, and that CutoffPredictor has the option to use prior cutoff information, try to convince utilities to add this information to their databases.