Riley Matthews
CS 229, Fall 2013

Determining Appliance Use From Power Draw Decomposition

(Modification of Belkin Kaggle Contest)

Background & Motivation:

The high level aim here is to monitor (out of benevolence!) a household's appliance usage without cumbersome and intrusive sensors.

The concrete technical problem at hand is to infer from sensor data the answer to the following question: which home appliances are ON and which are OFF at a given time *t*? The sensor data we are given to answer this question is limited to 8 timeseries: real power, reactive power, apparent power, and the power factor ratio, from each of "phase 1" and "phase 2". If you have an EE background feel free to understand all that, but for the purposes of this problem, it suffices to understand that the value of this 8-tuple at a given time characterizes the power draw in a richer way than say, the current alone. The hope then is that each appliance has an ON/OFF signature hidden in the overall power draw data. As a starting point we are given a training set of such power draw data in which the ON/OFF status of each appliance is (loosely) labeled, and are given a much larger "testing" set without labels.

Basic Approach:

The two key choices of our approach are (1) to build a separate classification model of ON/OFF status for each appliance and train it on the labeled data (instead of trying to have one simultaneous multi-class model) and (2) to determine ON/OFF status by identifying ON/OFF *toggle events*. (1) simplifies the thinking in what turns out to be a fairly difficult problem. (2) is almost necessary simply because the power draw at a given time gives us very little information. Indeed, when multiple appliances are ON at once, the power power draw's of those appliances that happen to be on. So unless you know the status of all other appliances, it will be hard to classify on this basis alone. It is much better to use the fact that we have time series data, and can therefore identify transitions from ON to OFF and vice versa. Essentially we want to baseline the the power draw of an appliance to the moment before it turned ON (OFF).

As a brief summary, the approach is to implement supervised learning procedure in which we train a model to classify *toggle events*. The ON/OFF status at any given time would then follow without much difficulty, though we do not get that far here.

Building Usable Data for Supervised Learning

In our problem we do not receive a neat training dataset ready made for supervised learning. We need to decide on and build the training set X from the time-series data.

Again we aim to build a model for each appliance/toggle-direction combination (write model $M_{a,d}$ as model of event appliance a toggles status in direction d), which will classify toggle events $x^{(i)} \in \Re$ as either a toggle event of (a,d) or not. So our training set needs both examples of toggle event

(a,d), and all other toggle events we will encounter. Further, we would like a sample that is representative or rather consistent with that which we would test on. However, this latter desideratum is difficult in our particular case because the labeled data is drawn from a separate distribution. Namely, the labeled timeseries power draw data is drawn from an "experimental" situation wherein at most one appliance is one at time, but the data allocated to testing is without such a constraint. With this in mind we review the steps taken to define and build a usable example set for our model.

(1) *Defining Toggle Events*

Letting $z(t) \in \Re^8$ be the raw time series powerdraw data, a toggle event will roughly be a time t_e where $\| z(t_e - \varepsilon) - z(t_e + \varepsilon) \|_\infty = \Delta_\varepsilon(t)$ is large. Intuitively, this will correspond to times when there is at least one component of the power-draw which jumps significantly in the moments before and after t_e. To isolate the event time precisely, we take "large" here to be any local (in t) extrema of $\Delta_\varepsilon(t)$. We have glossed over one step, it should be noted: in this event detection procedure we first smooth $z(t)$ along each component with a gaussian convolution/filter of parameter σ. When then have two parameters which determine the "events" over any timespan: σ, for smoothing, and ε the delay between measurements that determine $\Delta_\varepsilon(t)$ at any given time t.

(2) *Pairing Events with Labeled Times*

In the provided training set (the various control/experimental time series, $z(t)$), there are human labeled toggle times. I.e., for each appliance/toggle-direction pair, (a,d), there is a set of times $L_{a,d}$ where $t \in L_{a,d}$ if an (a,d) toggle event occurred within +/-30 seconds of t. So a central task of our problem is to make accurate and consistent estimates of the event times t_e that correspond to human labeled times $t \in L_{a,d}$.

The estimation method that yielded the most successful results is as follows. For each $t_i \in L_{a,d}$, we take estimate the corresponding toggle event to be $\widehat{t_i} = \sum_{t_e \in I_{t_t}} t_e \, \Delta_\varepsilon(t_e)^2$ where I_{t_i} is the set of detected event times (which are, remember, local extrema of $\Delta_\varepsilon(t)$) in the search interval around t_i. That is we estimate the toggle event time corresponding to the noisey labeled time as the sum of detected events times within a window surrounding t_i each weighted by the square of $\Delta_\varepsilon(t_e)$. **(SEE FIGURE 1 FOR INSTANCE OF EVENT DETECTION)**

(3) *Building a Training Set*

Fixing our smoothing and differential window parameters σ, and ε, respectively, we construct the supervised-learning-amenable example matrix X

and classification vector Y for $M_{a,d}$.

For each $t_j \in L_{a,d}$ and it's estimate $\widehat{t_j}$, there is a row vector $x^{(i)} = \delta_\varepsilon(\widehat{t_j}) = z(\widehat{t_j} - \varepsilon) - z(\widehat{t_j} + \varepsilon) \in \Re^8$ with classification $y^{(i)} = 1$. Think of this vector as capturing of an appliance's power draw "looks" as it is toggling ON/OFF status. The rest of the examples in X correspond to detected events (local extrema in $\Delta_\varepsilon(t)$ which are not near the estimated labeled times. That is, for each detected toggle event time t_e in the training series $z(t)$ which is not too close to a labeled time, there is a corresponding row vector and classification entry, $x^{(i)} = \delta_\varepsilon(t_e) \in \Re^8$, and $y^{(i)} = 0$.

A Note on Testing

Briefly, one of our difficulties is that we don't have a convenient way to cross validate on the larger test set. We can submit predictions to through the kaggle site that will be evaluated, but that is limited to twice a day and we would need to provide prediction for all 4 houses to get useful feedback, whereas we were focused on one house to get my bearings.

Model Experimentation

Ahhh, now that we've built a usable example set (take care to note, though, that X, Y are in fact parameterize by σ, and ε), we can experiment with different supervised learning (and some non-parametric) approaches.

(1) *Logistic Regression & Linear SVM*

Results with logistic regression and SVM were both mixed. When training on a (a,d) with a particularly distinct $x^{(i)} = \delta_\varepsilon(\widehat{t})$ signature, then often regression/SVM could perfectly separate the classes in the training set, but recorded unreasonably many false positives (checked by inspection) on the noisier non-controlled testing environment. On (a,d)'s with lower power draw signature (and thus smaller signal to noise ratio) the regression could not satisfactorily separate the testing data. I did not have time, though, to experiment with enriching the feature space in the logistic case by adding, e.g., all monomials of the features or other method.

(2) *r Thresholding Metric (locality)*

After frustration with the above two approaches, we tried to leverage the observation that signatures $x^{(i)} = \delta_\varepsilon(\widehat{t}_i)$ of a given (a,d) were in fact very similar to each other. We thought we could simply classify (a,d) examples based on their distance to their mean xample, $\bar{x}_{(a,d)}$ (this distance was computed after normalizing the individual components of $x^{(i)}$ by either the std or mean of the columns of X, so that scaling ofto depend on one or two components). The threshold at which we decide would be either proportional or geometrically proportional to

$$r = \frac{min_{i \notin (a,d)} ||x^{(i)} - \bar{x}_{(a,d)}||_2}{max_{i \in (a,d)} ||x^{(i)} - \bar{x}_{(a,d)}||_2}$$

i.e., the ratio of the distance from $\bar{x}_{(a,d)}$ to the closest non (a,d) event, to

the maximum distance of to $\bar{x}_{(a,d)}$ the an (a,d) event. This yielded more promising results but this approach was superseded by the thought that the SVM with rbf kernel was in an abstract sense also using a nearness metric, and might be easier to implement as it was a standard method.

(3) *SVM with R.B.F. Kernel*

SVM with the rbf kernel function was the most successful, however, not nearly successful enough and it was sensitive to overfitting, particularly problematic if my event detection algorithm mis-estimated and event -- the SVM with rbf could and would usually separate the data this data if the box constraint was high and the rbf sigma value was low enough, and consequently bake in that misclassification.

Tuning (Hyper)Parameters for SVM w/Gaussian Kernel

Given the time constraints we had after completing the infrastructure to build X,Y (given σ, ε), we wanted to implement a parameter optimization routine to get the best results with the SVM rbf as it was the most promising. The aim is to find the parameter set $(\sigma, \varepsilon, c, \sigma_{rbf})$--where c is the box constraint of the SVM--which minimizes misclassification rate of stratified k-fold cross validation on the training set (where k is typically 3, the number of examples of per appliance of a verified toggle events, i.e., the number of examples with $y^{(i)} = 1$).

We can't truly consider all possible parameters so we did a small grid search over a few possible σ, ε parameters and within that search we optimized the other two parameters over a larger but still small possibility set.

The results were tepid. Like the other examples it yielded too many false positives on the testing set. The issue was two-fold: (1) since the m.c.r. under multiple parameters reached zero (only 3 y=1 examples), there was not a good apriori tiebreaker, so to that numerous c, σ parameters achieved 0 m.c.r, then we can't say we were optimizing in choosing between them. (2) the testing test seems to have much noise and types of events, so the SVM is in a sense classifying events too much unlike those in the traing set -- it can't know where to set the boundary.

Visualization with PCA

We used PCA to visually explore underlying patterns and to quickly verify results. When we project the examples in X onto it's first to principal components we see some interesting structure. Clearly there is strong correlation between certain features. Out interpretation is that there is only a few types of electronic systems and these types produce a typical power-draw signature which is unique up to a scaling factor.

It was also an easy sanity check to see how classifiers are performing on the test set is to inspect the projection onto the principal components (See note about testing issues). Presumably projection of true (a,d) toggle events in the test set will follow a similar pattern as those of the training set upon projection onto the first two principal components. The

other slow but more precise approach was to inspect the $z(t)$ plots around the predicted (a,d) toggle event times -- indeed each (a,d) signature behavior is usually distinct enough to the eye.

Figures: 1-3