ONLINE DETECTION OF UNUSUAL EVENTS IN AUDIO VIA DYNAMIC SPARSE CODING

COURTNEY GRAZZINI
MEREDITH MILLER
SEAN WILSON

Abstract. Sparse representation of a natural signal in terms of a dictionary learned from the signal itself has been shown to be highly effective. Online methods for dictionary learning mitigate scalability issues with traditional batch learning methods, and offer the ability to learn dictionaries for streaming signals. We review the key concepts of sparse dictionary learning and important results related to the online extension of such methods. We show that online dictionary learning can be used to detect unusual events in audio signals, such as a change in meter or the introduction of a new instrument, and that such methods are robust to concept drift in the target signal.

1. Introduction

1.1. **Dictionary learning.** We can represent a signal as a linear combination of basis elements. We refer to the basis elements as *atoms*, and to the collection of these elements as a *dictionary*. We do not require that the atoms be linearly independent, and we allow the dictionary to be *overcomplete*, i.e., it may contain more basis elements than the dimension of the signal it is used to represent.

Predefined dictionaries are often used to represent signals. Let $\mathcal{D} = \{\mathbf{u}, \mathbf{v}\}$ for some linearly independent $\mathbf{u}, \mathbf{v} \in \mathbb{R}^2$. Then, we can represent any signal $\mathbf{x} \in \mathbb{R}^2$ in terms of the atoms $\mathbf{u}$ and $\mathbf{v}$ of the dictionary $\mathcal{D}$. We might similarly consider the two-dimensional Haar wavelet basis on the space of real, 2×2 matrices, whose elements can be used to encode images. In place of a predefined dictionary, we may also choose to learn a dictionary directly from the signal of interest, a procedure that has been shown, e.g. by [Elad and Aharon, 2006], to perform as well as or better than the predefined approach.

A signal $\mathbf{x} \in \mathbb{R}^m$ is said to have a *k-sparse representation* in a dictionary $\mathcal{D} = \{\mathbf{v}_i\}_{i \in \mathcal{I}} \subset \mathbb{R}^m$ if there is a system of coefficients $\{c_i\}_{i \in \mathcal{I}}$ such that $\mathbf{x} = \sum_{i \in \mathcal{I}} c_i \mathbf{v}_i$ and $c_i \neq 0$ for at most k many indices $i \in \mathcal{I}$. A sparse representation of $\mathbf{x}$ may be preferable to a "dense" representation (assuming that both representations have similar reconstruction performance) for two primary reasons. The first is the principle of parsimony (or model interpretability): if we can represent the signal with relatively few atoms (the coefficients of the remaining atoms being zero), our understanding of the signal is easier to characterize than if we had to interpret the effects of a larger number of atoms. That is, we have identified redundancy in the atoms relative to the signal. The second is computational efficiency: fewer nonzero coefficients means that we can represent and manipulate the signal using fewer resources. [Elad and Aharon, 2006] and others have shown sparse representions of natural signals to be very effective. Thus, our task in *dictionary learning* given some signal $\mathbf{x}$ is to learn a dictionary $\mathcal{D}$ such that $\mathbf{x}$ has a sparse representation in terms of the atoms of $\mathcal{D}$.

1.2. **Online learning.** A dictionary is typically learned from a training set (possibly the signal to be reconstructed) using a constrained optimization procedure. We can implement this optimization as a *batch* procedure, wherein the entire training set is used at each iteration. While such methods have been shown to be effective, they do not scale well to very large training sets, or to training sets that vary over time, e.g., streaming audio or video signals. [Mairal et al., 2009] propose an *online* approach to overcome these limitations, wherein the elements of the training set are accessed one at a time or in small batches.

When learning a dictionary for a streaming signal, our approach must also account for the phenomenon of *concept drift*, wherein the statistical properties of the signal change unpredictably over time. If our learning method does not allow the dictionary to adapt to these changes, then reconstruction performance will degrade as we attempt to represent portions of the signal that the dictionary has not previously "seen." For example, if we wish to respresent the video feed of a traffic camera, we are likely to learn a dictionary well suited to representing various motor vehicles. If the same street is closed for a parade, our dictionary is not likely to perform acceptably when reconstructing a marching band. [Zhao et al., 2011] have shown that online algorithms are robust to concept drift precisely because they permit the dictionary to adapt to new phenomena.

2. Methodology

Suppose that we wish to learn a dictionary for a signal $\mathbf{x} \in \mathbb{R}^m$. Let $\mathbf{D} \in \mathbf{M}_{m,k}(\mathbb{R})$ be the matrix whose columns are the atoms (basis vectors) of the dictionary, and consider the loss function

$$l(\mathbf{x},\mathbf{D}) \coloneqq \min_{\boldsymbol{\alpha}\in\mathbb{R}^k} \frac{1}{2}\|\mathbf{x}-\mathbf{D}\boldsymbol{\alpha}\|_2^2 + \lambda\|\boldsymbol{\alpha}\|_1,$$

which is known as the Least Absolute Shrinkage and Selection Operator (LASSO), and where λ is a regularization parameter. We see that $l(\mathbf{x},\mathbf{D})$ optimizes the sum of the squared error in reconstructing $\mathbf{x}$ using the dictionary $\mathbf{D}$ and the 1-norm of the coefficient vector $\boldsymbol{\alpha}$. To avoid arbitrarily small values of $\boldsymbol{\alpha}$, we adopt the constraint that the basis vectors $\{\mathbf{d}_j\}_{j=1}^k$ have 2-norm of at most one. Let $\mathcal{C}$ be the set of matrices satisfying this constraint, i.e.,

$$\mathcal{C} \coloneqq \left\{\mathbf{D} \in \mathbf{M}_{m,k}(\mathbb{R}) : \|\mathbf{d}_j\|_2 \le 1 \ \forall j \in \{1,2,\ldots,k\}\right\},$$

so that the optimization becomes

$$\min_{\mathbf{D}\in\mathcal{C},\boldsymbol{\alpha}\in\mathbb{R}^k} \frac{1}{2}\|\mathbf{x}-\mathbf{D}\boldsymbol{\alpha}\|_2^2 + \lambda\|\boldsymbol{\alpha}\|_1. \tag{2.1}$$

While this problem is not jointly convex, it is convex with respect to $\mathbf{D}$ and $\boldsymbol{\alpha}$ separately, so that a solution may be obtained by alternately optimizing one variable while the other is held constant, as proposed by [Lee et al., 2007].

Now suppose that we wish to learn a dictionary to represent a set of discrete *samples* $\{\mathbf{x}_i\}_{i=1}^n$ of some analog signal, where n is possibly unknown, as in the case of a streaming signal. Then, the optimization (2.1) becomes

$$\min_{\mathbf{D}\in\mathcal{C},\boldsymbol{\alpha}\in\mathbf{M}_{k,n}(\mathbb{R})} \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{2}\|\mathbf{x}_i-\mathbf{D}\boldsymbol{\alpha}_i\|_2^2 + \lambda\|\boldsymbol{\alpha}_i\|_1\right),$$

which is equation (4) from [Mairal et al., 2009]. The same authors propose a *mini-batch* extension of their algorithm 1 that considers $\eta > 1$ samples at each iteration $t \in \{1,2,\ldots T\}$. We adopt this mini-batch method to process η samples of a streaming signal at a time, alternating between optimizing $\boldsymbol{\alpha}$ and $\mathbf{D}$.

As in [Zhao et al., 2011], we will learn an initial dictionary $\mathbf{D}_0$ using the first N samples of the signal, then use a *sliding window* of width w to obtain the samples $\left\{\mathbf{x}_i^{(t)}\right\}_{i=1}^{\eta}$ for each iteration t. We then compute the reconstruction coefficients $\boldsymbol{\alpha}_t$ using Orthogonal Matching Pursuit (OMP) and use these coefficients to compute the updated dictionary $\mathbf{D}_t$ using Least Angle Regression (LARS), with $\mathbf{D}_{t-1}$ as a warm restart. Algorithm 1 presents our approach.

Algorithm 1 Online Dictionary Learning

Require: $\{\mathbf{x}_i\}_{i=1}^{n} \in \mathbb{R}^m$ (set of signals), $\lambda \in \mathbb{R}$ (regularization parameter), $\mathbf{D}_0 \in \mathbf{M}_{m,k}(\mathbb{R})$ (initial dictionary), w (sliding window width).
Ensure: learned dictionary $\mathbf{D}_T$
Calculate the number of iterations T and the number of samples per iteration η based on the signal dimension m, the number of signals n, and the window width w.
for $t = 1$ to T **do**
    Sparse coding: compute $\boldsymbol{\alpha}_t$ as in algorithm 2 for the samples $\left\{\mathbf{x}_i^{(t)}\right\}_{i=1}^{\eta}$.
    Dictionary update: compute $\mathbf{D}_t$ using LARS, with $\mathbf{D}_{t-1}$ as a warm restart.
end for
return $\mathbf{D}_T$

Algorithm 2 provides the coefficient update.

Algorithm 2 Orthogonal Matching Pursuit

Require: $\{\mathbf{x}_i\}_{i=1}^{\eta} \in \mathbb{R}^m$ (set of samples), $\lambda \in \mathbb{R}$ (tolerance parameter), $\mathbf{D} \in \mathbf{M}_{m,k}(\mathbb{R})$ (unit-norm dictionary).
Ensure: coefficients $\boldsymbol{\alpha} \in \mathbf{M}_{k,\eta}(\mathbb{R})$
for $i = 1$ to η **do**
    $\mathcal{I} \leftarrow \emptyset$
    $\boldsymbol{\alpha}_i \leftarrow \mathbf{0}$
    $\mathbf{r} = \mathbf{x}_i$
    while $\|\mathbf{r}\|_2 > \lambda$ **do**
        $i' \leftarrow \arg\max_{j \in [k] \setminus \mathcal{I}} |\langle \mathbf{r}, \mathbf{d}_j \rangle|$, where $\mathbf{d}_j$ is the jth column of $\mathbf{D}$
        $\mathcal{I} \leftarrow \mathcal{I} \cup \{i'\}$
        $\mathcal{B} \leftarrow \operatorname{span}\left\{\{\mathbf{d}_j\}_{j \in \mathcal{I}}\right\}$
        $\boldsymbol{\alpha}_i \leftarrow \Pi_{\mathcal{B}}(\mathbf{x}_i)$, where $\Pi_{\mathcal{B}}(\mathbf{x}_i)$ is the projection of $\mathbf{x}_i$ onto $\mathcal{B}$
        $\mathbf{r} \leftarrow \mathbf{x}_i - \boldsymbol{\alpha}_i$
    end while
end for
return $\boldsymbol{\alpha}$

As the window slides along the set of signals, the optimization at each iteration will produce a dictionary that is *adaptive*, i.e., it can learn to represent new signal behavior, and given some specified number of atoms, can also "forget" previously observed signal behavior that becomes rare or ceases to occur altogether after some specified time. In this way, our approach accommodates concept drift.

2.1. **Unusual event detection.** As the window slides along the set of signals, we will update our dictionary as described in the previous section. When we slide to the $(t+1)$ th window composed of samples $\left\{\mathbf{x}_i^{(t+1)}\right\}_{i=1}^{\eta}$, we will use our dictionary from the previous window $\mathbf{D}_t$ to reconstruct the samples. If the $(t+1)$ th window is *unusual*, i.e., it represents signal behavior that is either rare or novel, then our dictionary will not be able to represent it accurately (the dictionary will lack elements well-suited to approximating the new behavior). In this case, and following [Zhao et al., 2011], we expect that either a greater number of atoms will be required to reconstruct the samples for a specified reconstruction error, or that the error in reconstruction will increase for some desired sparsity k. In particular, we observe that this method is *unsupervised* insofar as we have given no prior definition for what constitutes an unusual event; we have not even provided prior information regarding the signal class.

[Zhao et al., 2011] determine the "normality" of an event $\mathbf{X}_i = \left\{\mathbf{X}_i^1, \ldots, \mathbf{X}_i^{n_i}\right\}$ by computing the objective function

$$J\left(\mathbf{X}_i, \boldsymbol{\alpha}_i, \mathbf{D}\right) \coloneqq \frac{1}{2}\sum_{j=1}^{n_i}\left\|\mathbf{X}_i^j - \mathbf{D}\boldsymbol{\alpha}_i^j\right\|_2^2 + \lambda_1 \sum_{j=1}^{n_i}\left\|\boldsymbol{\alpha}^j\right\|_1 + \lambda_2 \sum_{j=1}^{n_i}\sum_{k=1}^{n_i}\mathbf{W}_{jk}\left\|\boldsymbol{\alpha}_i^j - \boldsymbol{\alpha}_i^k\right\|_2^2,$$

where $\boldsymbol{\alpha}_i = \left\{\boldsymbol{\alpha}_i^1, \ldots, \boldsymbol{\alpha}_i^{n_i}\right\}$ are the reconstruction weight vectors (coefficients), $\mathbf{D}$ is the current dictionary, $\mathbf{W}$ is the adjacency matrix of $\mathbf{X}_i$, and $\lambda_1, \lambda_2 \in \mathbb{R}$ are regularization parameters. The first term gives the (squared) reconstruction error, the second term implements sparsity regularization, and the third term implements "smoothness" regularization, which arises from the authors' application to video signals. Then, a newly observed event $\mathbf{X}'$ with reconstruction weight vectors $\boldsymbol{\alpha}'$ learned using the current dictionary $\mathbf{D}$ is said to be unusual if $J\left(\mathbf{X}', \boldsymbol{\alpha}', \mathbf{D}\right)$ is greater than some specified threshold ε.

It may not be necessary to implement smoothness regularization for other signal classes, e.g., audio, and in practice we find that the reconstruction sum of squared errors (SSE) tends to dominate the sparsity regularization. Thus, for each iteration t, we will compute the reconstruction SSE and define an event to be unusual if the SSE for the tth iteration is greater than a specified threshold.

3. Application

3.1. **Overview.** To test the strengths of online dictionary learning for detecting change points, we apply the concept of initial dictionary learning with sequential dictionary update to an audio signal. As stated above, we hypothesize that the reconstruction of a signal window with the dictionary learned prior will perform poorly if new elements were encountered during that window (e.g. new instruments, key changes, etc.). This will result in a spike in the reconstruction error for that window, allowing for the identification of change points. For a test case, we consider the song "Bohemian Rhapsody" by the British rock band Queen, due to its frequent, noticeable (to the human ear) changes in style, range, and tempo.

Online dictionary learning seems well suited to this application since the sequential updates to the dictionary inherently provide a temporal aspect to the learning and reconstruction process. If a dictionary were instead learned on the entire song, the major change points might be masked since the dictionary "knows" what comes after. With online learning, however, the dictionary has no *a priori* information on what features will be in the next signal window, so is better suited to identify changes.

3.2. **Audio signal structure.** Analog audio is continuous as a function of time, so is digitized by taking discrete *samples*. The basic properties of this digitization include:

- Each sample quantifies the *amplitude* of the signal at a point in time.
- The *sampling rate* is the average number of samples obtained in one second.
- The resolution of each sample corresponds to the number of possible values of the amplitude, which is given by the *bit depth*.
- The number of *channels* used to represent a signal correspond to separate data streams that must be sampled.

For our application, we obtained an uncompressed, 16,000 Hz (sampling rate), 16-bit (bit depth), monaural (1 channel) waveform audio file (WAV) for "Bohemian Rhapsody." At 16-bit depth, amplitudes range from -2^{15} to $2^{15} - 1$, so pre-processing the signal involves normalizing these amplitudes to range from -1 to 1.

3.3. **Unusual event identification.** For comparison with the peaks in the reconstruction errors and assessing the performance of our implementation of online learning, we examined the sheet music of "Bohemian Rhapsody," i.e., "ground truth," for major musical changes and mapped them to timepoints in our audio, see table 1. The song begins in 4/4 time in the key of B-flat major. Two key changes occur, as well as frequent meter and instrumentation changes, providing numerous opportunities to assess the algorithm's performance.

Time	Description
0:07	meter change (5/4)
0:11	meter change (4/4)
0:15	new instrument (piano)
0:38	new instrument (cymbal swell)
0:48	new instrument (bass)
1:23	new instrument (set)
1:43	meter change (2/4)
1:45	meter change (4/4)
2:19	new instrument (guitar)
2:34	guitar solo
3:01	end guitar solo, key change (A major)
3:04	lead vocals rejoin
3:07	backing vocals rejoin
3:22	key change (E-flat major)
3:24	bass, set rejoin
4:05	meter change (12/8)
4:13	vocals rejoin

TABLE 1. Unusual events in "Bohemian Rhapsody"

3.4. **Implementation.** The algorithms discussed in [Mairal et al., 2009] have been implemented in the `MiniBatchDictionaryLearning` class of Python's `scikit-learn` library, so could be readily utilized for our application. However, our use case requires a hybrid implementation of the two methodologies of [Mairal et al., 2009] and [Zhao et al., 2011].

[Mairal et al., 2009] contains the description and recommendations for applying the basic online learning framework to vector data (by vectorizing image data), similar to the format of the monaural channel audio signal involved in this effort, but utilizes random sampling of the images. [Zhao et al., 2011] uses online learning to detect unusual events in video, paralleling our application well, and details the use of a sliding window during processing, rather than random samples. Further guidance is provided by the application of online learning to speech samples in [Santos, 2014], which directly addresses processing audio. The final Python code is contained in Appendix A, and the basic structure is detailed in the pseudocode of Algorithm 3.

Algorithm 3 Change Point Detection with Online Dictionary Learning

Require: $\mathbf{y} \in \mathbb{R}^M$ (audio signal, WAV format, 16,000 Hz, monoaural channel), $k \in \mathbb{N}$ (number of dictionary components)

Reshape $\mathbf{y}$ to $\mathbf{Y} \in \mathbf{M}_{k,n}(\mathbb{R})$, where $n = \lfloor M/k \rfloor$
INITIAL TRAINING
Extract initial training set: $\mathbf{X}_0 = \{\mathbf{Y}_{\mathcal{I}}\}$, where $\mathcal{I} \in \{1, 2, ..., N\}$
Learn the initial dictionary $\mathbf{D}_0 \in \mathbf{M}_{k,k}(\mathbb{R})$ from $\mathbf{X}_0$ using least angle regression (LARS) to compute the LASSO solution
Compute the reconstruction matrix $\boldsymbol{\alpha}_0 \in \mathbf{M}_{k,N}(\mathbb{R})$ using Orthogonal Matching Pursuit (OMP) with $\mathbf{D}_0$
Reconstruct the training set data and vectorize: $\widehat{\mathbf{X}}_0 = \mathbf{D}_0\boldsymbol{\alpha}_0 \Rightarrow \hat{\mathbf{x}}_0 \in \mathbb{R}^{Nk}$
ONLINE LEARNING
$T = k(n - N)/(16,000w)$, where w is the length of each sample window in seconds
$\eta = 16000w/k$, the number of vectors to extract
for $t = 1$ to T **do**
    Extract $\mathbf{X}_t = \{\mathbf{Y}_{\mathcal{I}}\}$, where $\mathcal{I} \in \{N + 1 + (t - 1)\eta, ..., N + 1 + t\eta\}$
    Compute the reconstruction matrix $\boldsymbol{\alpha}_t \in \mathbf{M}_{k,\eta}(\mathbb{R})$ with $\mathbf{D}_{t-1}$
    Reconstruct the window of data and vectorize: $\widehat{\mathbf{X}}_t = \mathbf{D}_{t-1}\boldsymbol{\alpha}_t \Rightarrow \hat{\mathbf{x}}_t \in \mathbb{R}^{\eta k}$, append to previous reconstructions
    Compute the sum of squared errors (SSE), $SSE_t = \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|_2^2$
    Update dictionary, $\mathbf{D}_{t-1} \Rightarrow \mathbf{D}_t$, using $\mathbf{D}_{t-1}$ as a warm restart
end for
return SSE (Reconstruction Errors), **Reconstructed audio**

Optimization of the algorithm runtime was initiated with recommendations from the literature, which were modified empirically until performance was improved to acceptable runtimes. The following parameters required empirical optimization:

- **Sparsity controlling parameter of Equation 2.1:** Due to the length of the signals used during online learning, the suggested value in [Mairal et al., 2009] of $1.2/\sqrt{m}$ did not perform well and was changed back to the default value of 1.
- **Number of iterations to perform during each training window:** Reduced to 200 from the default of 1000 to improve runtime.
- **Algorithm used to fit the dictionary to the input signal:** least angle regression (LARS, default); the coordinate descent alternative performed much more slowly.

- **Algorithm used to transform the input signal and generate the reconstruction vector:** orthogonal matching pursuit (OMP); LASSO-LARS was also tested but performed much more slowly.
- **The tolerance parameter of the OMP algorithm:** Set to 0.001 to ensure strong reconstruction.
- **Window size:** The window size is of particular concern because it directly affects the time resolution of our ability to detect unusual events. That is, by having a window size that was too large, it would be difficult to identify the true location of changes, since a single window could potentially contain multiple novel musical features. These features would be learned in one iteration of the algorithm, but recorded using only a single error value, inhibiting change point detection. However, runtime prevented the use of a window that was too narrow. We chose a window size of 2 seconds to obtain an acceptably short runtime while maintaining adequate resolution.
- **Number of dictionary components extracted:** 500, a value that provided a good trade-off between runtime performance and strong reconstruction capabilities.

We implemented this algorithm on one core of an Intel Xeon 2.00GHz CPU and 96 GB of RAM. Initial dictionary training on the first 10 seconds of the song, i.e. $\mathbf{X}_0 \in \mathbf{M}_{320,500}(\mathbb{R})$, took 44 seconds. The online dictionary learning with a window size of 2 seconds, i.e. $\mathbf{X}_t \in \mathbf{M}_{64,500}(\mathbb{R})$, cycled through 172 windows to complete the song, taking 163 minutes total.

4. Discussion

4.1. **Results.** Figure 4.1 plots the sum of squared errors in reconstructing the signal in each 2-second window using the dictionary learned during the previous window. The "epochs" in the song are displayed based on [Wikipedia, 2016], and the key and instrumental changes are as in table 1. We also plot the 75th and 90th percentiles of the SSE. R code to reproduce this figure is contained in Appendix B.

Unusual event detection requires that we choose some threshold ε, such that any window with reconstruction error greater than ε is said to contain unusual signal behavior. We see that at the 90th percentile of SSE (corresponding to $\varepsilon_{90} \approx 253$), we would classify as unusual

- the initial appearance of the guitar,
- the ensuing guitar solo,
- the beginning of the "opera" epoch,
- and the simultaneous key change to E-flat major and reintroduction of the bass guitar and drum set.

We also observe several windows with reconstruction error greater than ε_{90}, which may represent *false positives*, or possibly actual unusual events not considered in the limited classification of table 1. Further, we do *not* detect as unusual the initial appearances of the piano, bass, and drum set.

At the 75th percentile of SSE (corresponding to $\varepsilon_{75} \approx 192$), we would classify as unusual all the events classified as unusual at the 90th percentile, as well as

- the initial appearance of the piano,
- the initial appearance of the drum set,
- the reappearance of vocals during the "hard rock" epoch,
- and the beginning of the "outro" epoch.

The false positive rate will go to 100% as ε goes to zero because we are reconstructing the signal from a limited dictionary and have explicitly favored sparse reconstructions, i.e., those

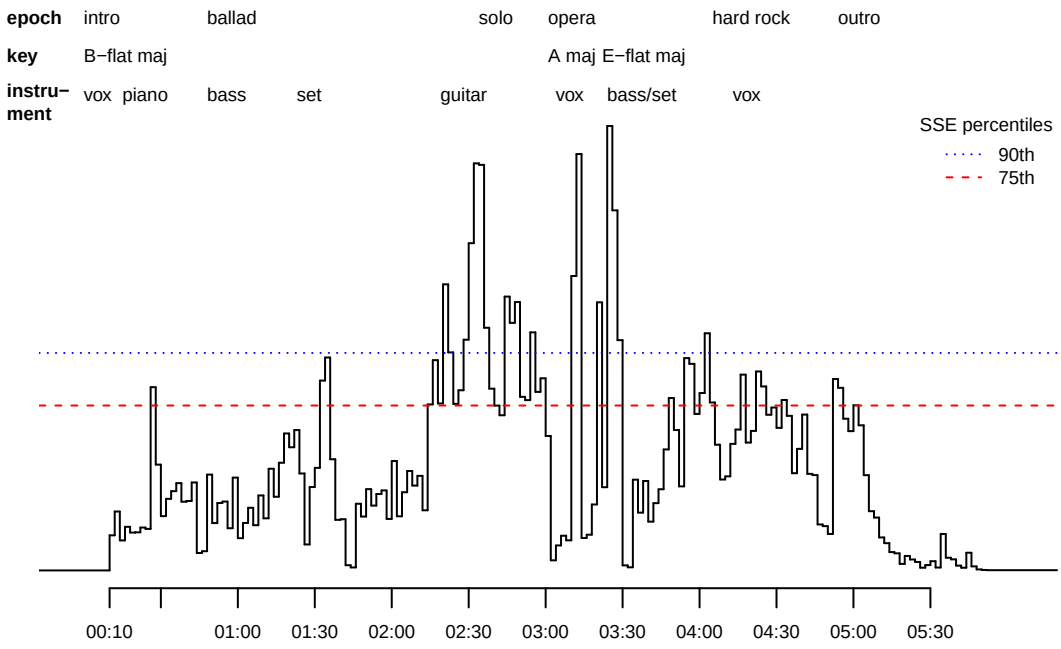

FIGURE 4.1. Reconstruction error for test data

with a higher number of zero coefficients. It is thus unsurprising that we observe more false positives at the 75th percentile than at the 90th percentile. The unusual events that we detect at the 75th percentile, but not at the 90th percentile, may also be better represented by the dictionary learned during the previous window. For example, that we detect the initial appearance of the piano only at ε_{75} is possibly explained by the similar range during the "intro" epoch of the piano arrangement and the human voice, where the latter constitutes the bulk of the training data available to the dictionary upon the introduction of the piano.

Thus, we observe reasonable performance from the algorithm on "Bohemian Rhapsody." In particular, we successfully detect both the guitar solo and the opera epoch, which are two of the most identifiable passages from the song. The algorithm also successfully and quickly learns these new features, and the error drops when similar data are encountered in the subsequent windows. This is clear from the plot of the SSE, Figure 4.1, where we get spikes, rather than steady or increasing error as the song progresses. That the detection was performed in an unsupervised fashion makes the result that much more significant. Further application of this algorithm to a particular signal would likely require empirical tuning of the parameters and selection of an appropriate reconstruction error threshold.

4.2. **Next Steps.** The concept of online dictionary learning is both flexible and powerful, so can be applied to a wide range of use cases and signal types. However, optimization to any one specific application does require extensive testing and tuning, as evidenced by the results detailed above. For example, further empirical testing of the seven parameters detailed in Section 3.4 would be required before finding optimal performance in both run-time and reconstruction behavior. Eventually, as [Zhao et al., 2011] did, our application might require a modification to the underlying objective function utilized by the algorithms

in [Mairal et al., 2009]. This would require forgoing the use of the functions available in Python's `scikit-learn` library, and instead necessitate building code from first principles. Similarly, other error measures could be developed to improve change point detection performance, rather than relying strictly on a threshold of the sum of the squared error.

In addition to optimization work, this use case readily suggests extensions in multiple directions. For example:

- Classification of change point types (e.g. instrument entrance/exit, key change, etc)
- Classification of song elements (e.g. highly specific genre/characteristics, a la Pandora's music genome project)
- Using a learned dictionary on one song to compress other songs by the same artist or in the same genre
- Using a learned dictionary to predict a song's popularity

Each of these applications could leverage the past work done on sparse representations, dictionary learning, and online processing techniques to develop new algorithms and further the cutting-edge of signal processing technology.

References

[Elad and Aharon, 2006] Elad, M. and Aharon, M. (2006). Image Denoising Via Sparse and Redundant Representations Over Learned Dictionaries. *IEEE Transactions on Image Processing*, 15(12):3736–3745.

[Lee et al., 2007] Lee, H., Battle, A., Raina, R., and Ng, A. Y. (2007). Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, pages 801–808.

[Mairal et al., 2009] Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2009). Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 689–696, New York, New York, USA. ACM Press.

[Santos, 2014] Santos, J. F. (2014). Dictionary learning and sparse coding for speech signals. [Online; retrieved on 7-August-2016 from `http://www.seaandsailor.com/dictlearning.html`].

[Wikipedia, 2016] Wikipedia (2016). Bohemian rhapsody — wikipedia, the free encyclopedia. [Online; retrieved on 7-August-2016 from `https://en.wikipedia.org/w/index.php?title=Bohemian_Rhapsody&oldid=732647147`].

[Zhao et al., 2011] Zhao, B., Fei-Fei, L., and Xing, E. P. (2011). Online detection of unusual events in videos via dynamic sparse coding. In *CVPR 2011*, pages 3313–3320. IEEE.

Appendix A. Python implementation

```
import numpy as np
from sklearn.decomposition import MiniBatchDictionaryLearning
from scipy.io import wavfile

fs, data = wavfile.read("Bohemian_Rhapsody_16Khz_16bit_mono_cut.wav")

# normalize sound pressure values to (-1, 1)
data = data / 2.0**15
data_len = data.shape[0]

# initialize reconstruction vector
recon = np.zeros(data_len)

# set training parameters & extract sample
train_factor = 10
end_train = fs * train_factor

n_components = 500          # number of atoms
n_features = n_components   # choose a "square" dictionary
n_samples_training = int(end_train / n_features)

training_data_vec = data[0:end_train]   # 160,000 samples

# reshape the training data to an array having dimensions
# (n_samples, n_features) = (320,500)
training_data = training_data_vec.reshape(n_samples_training, n_features)

# build the initial dictionary
dico = MiniBatchDictionaryLearning(n_components = n_components,
                                   alpha = 1, n_iter = 200,
                                   transform_algorithm = 'omp',
                                   transform_alpha = 0.001)
dico = dico.fit(training_data)

# reconstruct the training set with the initial dictionary
D = dico.components_
alpha = dico.transform(training_data)
out = np.matmul(alpha, D).flatten()
recon[0:end_train] = out

# extract testing signal & set online learning parameters
testing_data_vec = data[end_train:data_len]

# window width in seconds
window_width = 2
window_samples = window_width * fs
```

```
# length of the test signal in seconds
sig_length = len(testing_data_vec) / fs

# calculate the number of windows
num_windows = int(sig_length / window_width)

# calculate n_samples
n_samples_learn = int(window_samples / n_features)

# initialize SSE vector
sse = np.zeros(num_windows)

offset = end_train
# begin online learning with SSE calculation
for i in range(0, num_windows):
    # windows do not overlap
    start_idx = window_samples * i
    end_idx = window_samples * (i + 1)

    # extract window of data; window_width = 2, so 32,000 samples
    orig = testing_data_vec[start_idx:end_idx]

    # reshape the new data to an array of samples (64, 500)
    new_data = orig.reshape(n_samples_learn, n_features)

    # learn optimal reconstruction vector (64, 500)
    alpha = dico.transform(new_data)

    # compute SSE
    out = np.matmul(alpha, D).flatten()
    sse[i] = sum((out - orig)**2)

    # append to the reconstruction
    recon[offset + start_idx:offset + end_idx] = out

    # update the dictionary
    dico = dico.partial_fit(new_data)
    D = dico.components_

# write out the SSE
np.savetxt('sse.csv', sse.T, delimiter = ",")
wavfile.write('recon.wav', fs, recon)
```

Appendix B. R code to plot SSE

```
dat <- read.csv("sse.csv", header = F)
colnames(dat) <- c("SSE")

sf <- stepfun(x = (seq_along(dat$SSE) - 1) * 2, y = c(0, dat$SSE))
par(mgp = c(1.5,0.5,0), mar = c(2,1,3,0))
plot(sf, do.points = F, main = NA, cex.lab = 0.75, cex.axis = 0.75,
     xlab = NA, ylab = NA, xaxt = "n", yaxt = "n", frame.plot = F,
     xaxs = "i")

z <- (seq_along(dat$SSE) - 1) * 2

sec <- z + 10
labs <- c("00:10",
          paste(sprintf("%02d", sec[sec %% 30 == 0] %/% 60),
                c("30", "00"), sep = ":"))

axis(side = 1, at = c(0, z[z %% 30 == 0][-1] - 10),
     labels = labs, cex.axis = 0.6)

labs.epoch <- c("intro", "ballad", "solo", "opera",
                "hard rock", "outro")
sec.epoch <- c(-10, 38, 144, 171, 235, 284)

mtext("epoch", side = 3, line = 2, cex = 0.6, at = -40, adj = 0, font = 2)
for(i in seq_along(labs.epoch)) {
  mtext(labs.epoch[i], side = 3, line = 2, cex = 0.6,
        at = sec.epoch[i], adj = 0)
}

labs.key <- c("B-flat maj", "A maj", "E-flat maj")
sec.key <- c(-10, 171, 192)

mtext("key", side = 3, line = 1, cex = 0.6, at = -40, adj = 0, font = 2)
for(i in seq_along(labs.key)) {
  mtext(labs.key[i], side = 3, line = 1, cex = 0.6,
        at = sec.key[i], adj = 0)
}

labs.inst <- c("vox", "piano", "bass", "set", "guitar", "vox",
               "bass/set", "vox")
sec.inst <- c(-10, 5, 38, 73, 129, 174, 194, 243)

mtext("instru-\nment", line = -0.5, side = 3, cex = 0.6, at = -40,
      adj = 0, font = 2)
for(i in seq_along(labs.inst)) {
  mtext(labs.inst[i], side = 3, cex = 0.6, at = sec.inst[i], adj = 0)
```

```
}

col.qt <- c("red", "blue")
lty.qt <- 2:3

legend("topright", legend = c("90th", "75th"), col = rev(col.qt),
       lty = rev(lty.qt), bty = "n", cex = 0.6,
       title = "SSE percentiles")

eps <- quantile(x = dat$SSE, probs = c(0.75, 0.9))
abline(h = eps, col = col.qt, lty = lty.qt)
```