Unsupervised Learning for Computational Phenotyping

Chris Hodapp (chodapp3@gatech.edu)

***Abstract*—With large volumes of health care data comes the research area of computational phenotyping, making use of techniques such as machine learning to describe illnesses and other clinical concepts from the data itself. The "traditional" approach of using supervised learning relies on a domain expert, and has two main limitations: requiring skilled humans to supply correct labels limits its scalability and accuracy, and relying on existing clinical descriptions limits the sorts of patterns that can be found. For instance, it may fail to acknowledge that a disease treated as a single condition may really have several subtypes with different phenotypes, as seems to be the case with asthma and heart disease. Some recent papers cite successes instead using unsupervised learning. This shows great potential for finding patterns in Electronic Health Records that would otherwise be hidden and that can lead to greater understanding of conditions and treatments. This work implements a method derived strongly from Lasko *et al.*, but implements it in Apache Spark and Python and generalizes it to laboratory time-series data in MIMIC-III. It will be released as an open-source tool for exploration, analysis, and visualization.**

***Index Terms*—Big data, Health analytics, Data mining, Machine learning, Unsupervised learning, Computational phenotyping**

I. Introduction & Background

The field of *computational phenotyping*[1] has emerged recently as a way of learning more from the increasing volumes of Electronic Health Records available, and the volume of this data ties it in naturally with fields like machine learning and data mining. The "traditional" approach of supervised learning over classifications has two noted

- It requires the time and attention of that domain expert in order to provide classification information over which to train a model, and this requirement on human attention limits the amount of data available (and, to an extent, its accuracy.)
- It tends to limit the patterns that can be found to what existing classifications acknowledge. If a disease treated as a single condition really has multiple subtypes with different phenotypes, the model will not reflect this - for instance, asthma and heart disease[9].

Some recent papers[13], [9], [5] cite successes with approaches instead using unsupervised learning on time-series data. In Lasko *et al.*[9], such an approach applied to serum uric acid measurements was able to distinguish gout and acute leukemia with no prior classifications given in training. Marlin *et al.*[13] examines 13 physiological measures from a pediatric ICU (such as pulse oximetric saturation, heart rate, and respiratory rate). Che *et al.*[1] likewise uses ICU data, but focuses on certain ICD-9 codes rather than mortality.

This approach still has technical barriers. Time-series in healthcare data frequently are noisy, spare, heterogeneous, or irregularly sampled, and commonly Gaussian processes are employed here in order to condition the data into a more regular form as a pre-processing step. In [9], Gaussian process regression produces a model which generates a continuous, interpolated time-series providing both predicted mean and variance. In [4] it is used to instead derive a latent representation from correlated time-series. Che *et al.*[1] does not use Gaussian process regression, but rather fills irregularities in the data by propagating forward or backward.

Che *et al.*[1] notes that "shallow" clustering models such as Gaussian mixture models (as in [13]) are used too, deep learning succeeds more here at finding latent factors and extracting concepts - which is particularly important given the complexity of bodies and processes. Lasko *et al.*[9] applies a two-layer stacked sparse autoencoder (compared with a five-layer stacked denoising autoencoder in [1]), but the approach that looks like it could extend to a deep learner with the addition of layers.

II. Problem Formulation

The goal undertaken here is to reimplement a combination of some earlier results (focusing mainly on that of Lasko *et al.*[9]) using Apache Spark and the MIMIC-III critical care database[6], and able to run on a "standard" Spark setup such as Amazon Elastic MapReduce. The software behind this work is also intended to be released as an open source tool for accomodating exploration, analysis, and visualization using the techniques described herein.

Most of the needed algorithms are available in existing machine learning libraries in languages such as MATLAB or R, however, these are less accessible to the aforementioned "standard" setup or may be more cumbersome to integrate with Apache Spark. While Apache Spark runs on top of Java/JVM infrastructure and has access to a considerable ecosystem of Java-based, relevant libraries do not always exist or do not exist in a form which works well with Spark.

The main problems that the implementation tries to address within these parameters are:

- Loading the MIMIC-III database into a form usable from Spark
- Identifying relevant laboratory tests, admissions, and ICD-9 codes on which to focus
- Preprocessing the time-series data with Gaussian process regression
- Using a two-layer stacked sparse autoencoder to perform feature learning

- Visualizing the new feature space and identifying potential clusters

III. Approach and Implementation

A. Loading and Selecting Data

The MIMIC-III database is supplied as a collection of `.csv.gz` files (that is, comma-separated values, compressed with gzip). By way of `spack-csv`, Apache Spark 2.x is able to load these files natively as tabular data, i.e. a `DataFrame`. All work described here used the following tables[6]:

- `LABEVENTS`: Timestamped laboratory measurements for patients
- `DIAGNOSES_ICD`: ICD-9 diagnoses for patients (per-admission)
- `D_ICD_DIAGNOSES`: Information on each ICD-9 diagnosis
- `D_LABEVENTS`: Information on each type of laboratory event

From these the laboratory time-series for `ITEMID` of 50820 (blood pH) were extracted, under the conditions that at least 3 samples were available for a given admission, and that the admission involved at least one ICD-9 code 518 (other lung diseases), or at least one ICD-9 code 584 (acute renal failure), but not both. The field `VALUEUOM` also was checked to ensure that units of measure were consistent. (This combination of tests and codes was not chosen for any particular reason except that the dataset was able to accomodate it.)

All processing at this stage was done via Spark's DataFrame operations, aside from the final conversion to an RDD containing individual time-series.

This produced a total of 9,035 unique admissions, 8,219 unique subjects, and 187,374 time-stamped samples. 70% of these admissions were randomly selected for the training set, and the remaining 30% for the testing set.

B. Preprocessing

1) Time Warping: The covariance function that is used in Gaussian process regression (and explained after this section) contains a time-scale parameter τ which embeds assumptions on how closely correlated nearby samples are in time. This value is assumed not to change within the time-series - that is, it is assumed to be *stationary*[9]. This assumption is often incorrect, but under the assumption that more rapidly-varying things (that is, shorter time-scale) are measured more frequently, an approximation can be applied to try to make the time-series more stationary - in the form of changing the distance in time between every pair of adjacent samples in order to shorten longer distances, but lengthen shorter ones[9]. For original distance d, the warped distance is $d' = d^{1/3} + b$, using $a = 3, b = 0$ (these values were taken directly from equation 5 of [9] and not tuned further).

Thomas Lasko also related in an email that this assumption (that measurement frequency was proportional to how volatile the thing being measured is) is not true for all medical tests. He referred to another paper of his[8] for a more robust approach, however, this is not used here.

2) Gaussian Process Regression: In order to condition the irregular and sparse time-series data from the prior step, Gaussian process regression was used. The method used here is what Lasko *et al.*[9] described, which in more depth is the method described in algorithm 2.1 of Rasmussen & Williams[15].

In brief, Gaussian process regression (GPR) is a Bayesian non-parametric, or less parametric, method of supervised learning over noisy observations[3], [9], [15]. It is not completely free-form, but it infers a function constrained only by the mean function (which here is assumed to be 0 and can be ignored) and the covariance function $k(t, t')$ of an underlying infinite-dimensional Gaussian distribution. It is not exclusive to time-series data, but t is used here as in this work GPR is done only on time-series data.

That covariance function k defines how dependent observations are on each other, and so a common choice is the squared exponential[3]:

$$k(t,t') = \sigma_n^2 \exp\left[\frac{-(t-t')^2}{2l^2}\right]$$

Note that k approaches a maximum of σ_n^2 as t and t' are further, and k approaches a minimum of 0 as t and t' are closer. Intuitively, this makes sense for many "natural" functions: we expect closer t values to have more strongly correlated function values, and l defines the time scale of that correlation.

The rational quadratic function is used here instead as it better models things that may occur on many time scales[9]:

$$k(t,t') = \sigma_n^2 \left[1 + \frac{(t-t')^2}{2\alpha\tau^2}\right]^{-\alpha}$$

Rasmussen & Williams[15] supplies a formula for computing the log marginal likelihood of the target values $\mathbf{y}$ given the inputs X (both $n \times 1$ row vectors of target values and input values, respectively), where K is a $n \times n$ matrix for which $K_{ij} = k(X_i, X_j)$:

$$\log p(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^\top (K+\sigma_n^2 I)^{-1} - \frac{1}{2}\log|K+\sigma_n^2 I| - \frac{n}{2}\log 2\pi$$

An equivalent version that uses Cholesky decomposition rather than $n \times n$ matrix inversion is given in algorithm 2.1 of the aforementioned text, and this is what is actually used in the code.

In specific, in order to optimize the hyperparameters σ_n, τ, and α of the covariance function, $\sum \log p(\mathbf{y}|X)$ was roughly maximized over every time-series in the training set (following the time warping of the prior section) using a grid search. The values found for this were $\sigma_n = 1.4$, $\alpha = 0.1$, and $\tau = 3.55$.

Hyperparameter optimization was likely the most time-consuming step of processing. However, this step also is a highly parallelizable one, and so it was amenable to the distributed nature of Apache Spark.

3) Interpolation: The remainder of algorithm 2.1 is not reproduced here, but the code directly implemented this method using the rational quadratic function (and hyperparameters given above) as the covariance function. This inferred for each individual time-series a continuous function producing predictive mean and variance for any input t (for which they use the notation $\mathbf{x}^*$ for "test input").

As in [9], all of these inferred functions were then evaluated at a regular sampling of time values (i.e. via the test input $\mathbf{x}^*$) with padding added before and after each time series. For this data set, the sampling was done at intervals of 12 hours, with 15 samples of padding (that is, 7.5 days) at the beginning and end of each time-series.

In effect, this mapped each individual time-series first to a continuous function, and then to a new "interpolated" time-series containing predicted mean and variance at the sampled times described above (a total of 341,819 interpolated time-series samples). These interpolated time-series were then the input to later steps.

C. Feature Learning with Autoencoder

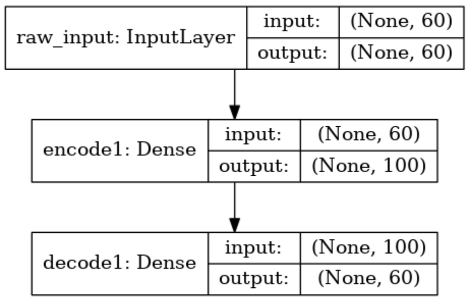
A stacked sparse 2-layer autoencoder was then used to perform feature learning. The implementation used here was a combination of what was described in [9] (which closely follows the UFLDL Tutorial from Stanford[14]), and François Chollet's guide[2] on using the Python library Keras to implement autoencoders. Specifically, Keras was used with Theano (GPU-enabled) as the backend, however, it should work identically using TensorFlow as the backend.

These autoencoders have a fixed-size input and output, and in order to accomodate this, fixed-size contiguous patches were sampled from the interpolated time-series. As in [9], the patch size was set to twice the padding (thus 30 samples here), and patches were sampled uniformly randomly from all contiguous patches. A total of 18,801 patches were produced here, all of which were then standardized to mean 0 and variance 1. 70% of these patches were then selected for autoencoder training, and the remaining 30% for validation.

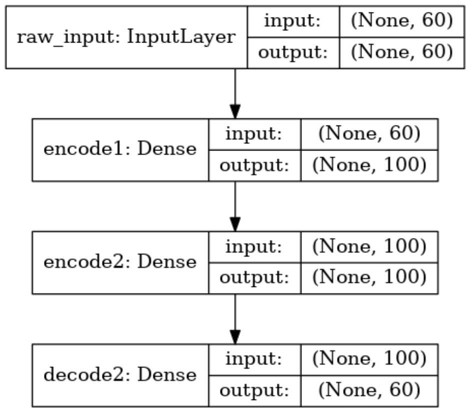
To accomodate this patch size, the autoencoder's input and output layers had 60 units (as each sample contained both variance and mean).

As in [9], the encoder and decoder layers used sigmoid encoders and linear decoders, and all hidden layers had 100 units. The sparsity and regularization parameters given there did not correspond precisely with Keras' model, so instead L1 activity regularization of 10^{-4} and L2 weight regularization of 10^{-4} were used.

In order to perform greedy layerwise training[14], the network was then built up in stages, starting first with this:



This model was trained with the raw input data (at both the input and the output), thus learning "primary" hidden features at the layer `encode1`. The first decoder layer `decode1` was then discarded and the model extended with another encoder and decoder:



This model then was similarly trained on raw input data, but while keeping the weights in layer `encode1` constant. That is, only layers `encode1` and `decode2` were trained, and in effect they were trained on "primary" hidden features (i.e. `encode1`'s activations on raw input). Following this was "fine-tuning"[14] which optimized all layers, i.e. the stacked autoencoder as a whole, again using the raw input data.

The final model then discarded layer `decode2`, and used the activations of layer `encode2` as the learned sparse features.

IV. Experimental Evaluation

Presently, not many results are available as further work is needed.

V. Conclusions & Further Work

This work is still in need of further experimentation and refining. The code is intended to be released as an open-source tool that allows exploration in MIMIC-III data but could easily be generalized to other time-series data, however, the code is not yet released.

Several optimizations also may help; for instance, hyperparameter optimization is currently done with a grid search, but would more sensibly done with a more intelligent optimization algorithm (such as SGD). The time warping function has parameters that could be tuned, or more extensive changes[8] could be made to try to make the time-series more stationary. Other covariance functions may be more appropriate as well.

The intention was that as much as possible could be executed within Apache Spark on "standard" infrastructure such as Amazon EMR or Databricks. The original intention was to use DL4J, a native Java library which already supports Apache Spark, but problems prevented this from working correctly. It may be possible to still use DL4J, there may be other Java-compatible libraries accessible from Apache Spark, or a closer integration with Keras may be possible via Spark's built-in `pyspark` support (and performance may still be acceptable if GPU support is unavailable).

Some other areas should perhaps be explored further too. One incremental change is in the use of multiple-task Gaussian processes (MTGPs); the work done here handles only individual time-series, while MIMIC-III is rich in parallel time-series that correlate with each other. Ghassemi *et al.*[4] explored the use of MTGPs to find a latent representation of multiple correlated time-series, but did not use this representation for subsequent feature learning. Another incremental change is in the use of Variational Autoencoders (VAEs) to learn a feature space that is sufficiently low-dimensional that techniques such as t-SNE are not required for effective visualization.

A more extensive change could involve using recurrent neural networks (RNNs). Deep networks such as RNNs such as Long Short-Term Memories (LSTMs) have shown promise in their ability to more directly handle sequences[16] and clinical time-series data, including handling missing data[10], [12], [11]. However, they are primarily used for supervised learning, but could potentially be treated similarly as autoencoders (as in [7]), that is, trained with the same input and output data in order to learn a reduced representation of the input. This approach would avoid some of the need to perform Gaussian Process Regression, however, it still may not cope well with time-series data that is very irregular.

Acknowledgment

Thank you to Dr. Jimeng Sun and the TAs for their time and effort in this course.

References

[1] Z. Che, D. Kale, W. Li, M. Taha Bahadori, and Y. Liu. Deep Computational Phenotyping. *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 507–516, 2015.

[2] F. Chollet. Building Autoencoders in Keras.

[3] M. Ebden. Gaussian Processes: A Quick Introduction. (August), 2015.

[4] M. Ghassemi, T. Naumann, T. Brennan, D. a. Clifton, and P. Szolovits. A Multivariate Timeseries Modeling Approach to Severity of Illness Assessment and Forecasting in ICU with Sparse , Heterogeneous Clinical Data. *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 446–453, 2015.

[5] A. E. W. Johnson, M. M. Ghassemi, S. Nemati, K. E. Niehaus, D. Clifton, and G. D. Clifford. Machine Learning and Decision Support in Critical Care. *Proceedings of the IEEE*, 104(2):444–466, 2016.

[6] A. E. W. Johnson, T. J. Pollard, L. Shen, L.-W. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark. MIMIC-III, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.

[7] M. Klapper-Rybicka, N. N. Schraudolph, and J. Schmidhuber. Unsupervised Learning in LSTM Recurrent Neural Networks. *Icann*, pages 674—-681, 2001.

[8] T. A. Lasko. Nonstationary Gaussian Process Regression for Evaluating Clinical Laboratory Test Sampling Strategies. *Proceedings of the ... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence*, 2015(8):1777–1783, jan 2015.

[9] T. A. Lasko, J. C. Denny, and M. A. Levy. Computational Phenotype Discovery Using Unsupervised Feature Learning over Noisy, Sparse, and Irregular Clinical Data. *PLoS ONE*, 8(6), 2013.

[10] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzell. Learning to Diagnose with LSTM Recurrent Neural Networks. *Iclr*, pages 1–18, 2015.

[11] Z. C. Lipton, D. C. Kale, and R. Wetzel. Directly Modeling Missing Data in Sequences with RNNs: Improved Classification of Clinical Time Series. 56(2016):1–17, 2016.

[12] Z. C. Lipton, D. C. Kale, and R. C. Wetzell. Phenotyping of Clinical Time Series with LSTM Recurrent Neural Networks. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 705–714, 2015.

[13] B. M. Marlin, D. C. Kale, R. G. Khemani, and R. C. Wetzel. Unsupervised Pattern Discovery in Electronic Health Care Data Using Probabilistic Clustering Models.

[14] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, and C. Suen. UFLDL Tutorial.

[15] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning.*, volume 14. 2004.

[16] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112, 2014.