Autoencoder-derived Features as Inputs to Classification Algorithms for Predicting Well Failures
Jeremy Liu, ISI USC, Ayush Jaiswal, USC, Ke-Thia Yao, ISI USC, Cauligi S.Raghavendra, USC

Abstract
This paper presents the results of using autoencoder-derived features, rather than hand-crafted features, for predicting rod pump well failures using Support Vector Machines (SVMs). Features derived from dynamometer card shapes are used as inputs to the SVM algorithm. Hand-crafted features can lose important information whereas autoencoder-derived abstract features are designed to minimize information loss. Autoencoders are a type of deep neural network with layers organized in an hourglass shape of contraction and subsequent expansion; such a network eventually learns how to compactly represent a data set as a set of new abstract features with minimal information loss. When applied to card shape data, we demonstrate that these automatically derived abstract features capture high-level card shape characteristics that are orthogonal to the hand-crafted features. In addition, we provide experimental results showing improved well failure prediction accuracy by replacing the hand-crafted features with more informative abstract features.

Introduction
In machine learning, effective classification of events into separate categories relies upon picking a good feature set to describe the data. For various reasons, dealing with the raw data’s dimensionality may not be desirable so the data is often reduced to a smaller space known as a feature set. Feature sets are typically selected by subject-matter experts through experience. In our experiments, we use dynamometer card shape data reduced to {card area, peak surface load, and minimum surface load} hand-crafted features to predict well failures using previously developed SVM technology. An alternate method of generating a good feature set is to pass the raw data through a type of deep neural network known as an autoencoder. Compared to selecting a feature set by hand, there are two advantages to using autoencoders. First, the process is unsupervised; so, even without expertise in the data being classified, one can still generate a good feature set. Second, the autoencoder-generated feature set loses less information about the raw data than a hand-selected feature set would. Autoencoders minimize information loss by design, and the additional information preserved in autoencoder features is carried through to the classification algorithms, manifesting as improved classification results.

In our experiments, we generate two feature sets from the raw dynamometer card shapes. One set is hand-selected and the other set is derived from an autoencoder. We then use the feature sets to train and test a support vector machine that classifies each feature vector as a normally operating well or a well that will experience failure within the next 30 days. In an extended experiment, we present the results of combining the two feature sets to produce a concatenated version containing both autoencoder-derived features and hand-selected features.

Autoencoders
General Concepts Autoencoders are a type of deep neural network that can be used to reduce data dimensionality. Deep neural networks are composed of many layers of neural units, and in autoencoders, every pair of adjacent layers forms a full bipartite graph of connectivity. The layers of an autoencoder collectively create an hourglass figure where the input layer is large and subsequent layer sizes reduce in size until the center-most layer is reached. From there until the output layer, layer sizes expand back to the original input size.

Data passed into an autoencoder experiences a reduction in dimensionality. With each reduction, the network summarizes the data as a set of features. With each dimensionality reduction, the features become increasingly abstract. (A familiar analogy is image data: originally an image is a collection of pixels, which can first be summarized as a collection of edges, then as a collection of surfaces formed by those edges, then a collection of objects formed by those surfaces, etc.). At the center-most layer, the dimensionality is at a minimum. From there, the network reconstructs the original data from the abstract features and compares the reconstruction result against the original data. Based on the error between the two, the network uses backpropagation to adjust its weights to minimize the reconstruction error. When reconstruction error is low, we can be confident that the feature set found in the center-most layer of the autoencoder still carries important information that accurately represents the original data despite the reduced dimensionality. In figure 2 we can see that much of the original cardshape’s information is retained within the abstract features generated by the autoencoder,
Figure 1: Autoencoder structure composed of 9 layers. Every layer in the network is fully connected with its adjacent layers. The layer sizes are 30 units (input), 60 units, 40 units, 20 units, 3 units, 20 units, 40 units, 60 units, and 30 units (output/reconstruction). Autoencoder-derived features are pulled from the center-most layer composed of 3 units.

enough to reconstruct the original data relatively accurately.

Performing a similar reconstruction may not be feasible with hand-selected features. In our case, the hand-selected features we use are card area, peak surface load, and minimum surface load. Using just these three features loses some important information. For example, it would be hard to determine that gas-locking is occurring in the well pictured in Fig. 2. There are many possible card shapes we can draw that have the same card area, peak surface load, and minimum surface load, most of which will not necessarily show indications of gas-locking. But if we use autoencoder-derived abstract features and look at the reconstruction, such as in Fig. 2, we can clearly see the stroke pattern that indicates that the gas-locking behavior is preserved.

Dimensionality Reduction Reducing the dimensionality of data is helpful for many reasons. An immediately obvious application is storage: by representing data using fewer dimensions, we can reduce the amount of memory required while suffering only minor losses in fidelity. While storage capacity is of less concern nowadays, limited bandwidth may still be an issue, especially in oilfields. We can look at the savings we can achieve with an autoencoder. The rod pumps used in this paper, for each card shape, transmit 30 points of position versus load. Once trained, an autoencoder can represent the 30 original values using only 3 values. Compression using autoencoders is not a lossless process, but as Fig. 2 shows, the error is small.

We also want to avoid the curse of dimensionality in which machine learning algorithms run into sampling problems, reducing the predictive power of each training example. As the number of dimensions grows, the number of possible states (or volume of the space) grows exponentially. Thus, to ensure that there are several examples of each possible state shown to the learning algorithm, we must provide exponentially greater amounts of training data. If we cannot provide this drastically increased amount of data, the space may become too sparse for the algorithm to produce any meaningful results.

Constructing and Training Autoencoders The final form of an autoencoder is built in two steps. First, the overall structure is created by stacking together several instances of a type of artificial neural network known as a Restricted Boltzmann Machine (RBM). These RBMs are greedily trained one-by-one and form the layered structure of the autoencoder. After this greedy initial training, the network begins fine-tuning itself using backpropagation across many epochs.

An RBM is an artificial neural network that learns a probability distribution over its set of inputs. RBMs are composed of two layers of neural units that are either “on” or “off.” Neurons in one layer are fully connected to neurons in the other layer but connections within a single layer are restricted (see Fig. 3). This restricted property allows RBMs to utilize efficient training algorithms that regular Boltzmann Machines cannot use.

The two layers within an RBM are known as the visible and hidden layers. The goal of training an RBM is to produce a set of weights between the neural units such that the hidden units can generate (reconstruct) the training vectors with high probability in the visible layer. An RBM can be described in terms of energy, and the total energy is the sum of the energies of every possible state in the RBM. We define the energy $E$ of a network state $v$ as
Figure 2: Example of autoencoder reconstruction. The original card shape (top) is composed of 30 points, and the reconstruction (also 30 points) is generated from only 3 abstract features.

Figure 3: A Restricted Boltzmann Machine. There are no intra-layer connections, and the network can be described as a bipartite graph. The first (black) layer is called the visible layer and the second (red) layer is called the hidden layer.

\[ E(v) = -\sum_i s_i^v b_i - \sum_{i<j} s_i^v s_j^v w_{ij} \]

where \(s_i^v\) is the binary (0 or 1) state of unit \(i\) as described by the network state \(v\), \(b_i\) is the bias of unit \(i\), and \(w_{ij}\) is the mutual weight between units \(i\) and \(j\). The total energy of all possible states, then, is

\[ \sum_u -E(u) \]

and we can find the probability that the network will produce a specific network state \(x\) by taking the log expression

\[ P(x) = e^{-E(x)} / \sum_u e^{-E(u)} \]
The method of training RBMs is known as contrastive divergence, or CD. Each iteration of CD is divided into positive and negative phases. In the positive phase, the visible layer’s state is set to the same state as that of a training vector (a card shape in our case). Then, according to the weight matrix describing the connection strengths between neural units, the hidden layer’s state is stochastically determined. The algorithm records the resulting states of the hidden units in this positive phase. Next, in the negative phase, the hidden layer’s states and the weight matrix stochastically determine the states of the visible layer. From there the network uses the visible layer to determine the final state of the hidden units. After this, we can update the weights according to the equation

$$\Delta w_{ij} = \varepsilon (\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{reconstruction}})$$

where $\varepsilon$ is the learning rate, $\langle v_i h_j \rangle_{\text{data}}$ is the product of visible and hidden units in the positive phase, and $\langle v_i h_j \rangle_{\text{reconstruction}}$ is the product of visible and hidden units in the negative phase.

Once the first RBM is trained using the CD method, we show all the training vectors to the RBM once more and record the resulting hidden unit states corresponding to each vector. Then we move to the next RBM in the “stack” within the autoencoder and use the hidden states as input vectors into the new RBM, beginning the process anew. From there, we can train the new RBM, gather new hidden states, and move onto training the next RBM in line. This is a greedy training method because the CD process only requires local communication between adjacent layers.

Once all RBMs in the autoencoder have been trained, the process of standard gradient descent using backpropagation begins. Normally, gradient descent requires labels to successfully backpropagate error, which implies supervised training. However, due to the special function and structure of the autoencoder, the data labels happen to be the data itself: the autoencoder’s goal is to accurately reproduce the data using lower dimension encodings.

Data

Dynamometer card shape data is two-dimensional and measures rod pump position versus load. Each oil well generates card shapes every day, and these card shapes are used to classify wells into normal and failure categories. From these card shapes, we hand-select the following three features: card area, peak surface load, and minimum surface load. These three features are used as inputs for an SVM model. The results represent the typical case where we use hand-selected features as inputs to the classification algorithm.

To generate a feature set derived from autoencoders, we first need to process the raw data. Because we are concerned more with the general shape of a card rather than absolute values of position or load, and because we want to compare card shapes across many different wells, we normalize card shapes to a unit box. Furthermore, we interpolate points in the card shapes so that each shape contains 30 points: 15 points for the upstroke and 15 points for the downstroke.

The autoencoder we use to generate the abstract features is composed of 9 layers. The layer sizes are 30 units (input), 60 units, 40 units, 20 units, 3 units, 20 units, 40 units, 60 units, and 30 units (output/reconstruction). After autoencoder training and testing, we collect the abstract features from the center-most layer that consists of 3 units. Thus, from the original raw card shapes, we choose a 3-feature abstract representation to pass to the SVM model (because we only want to replace the hand-selected features). The results represent the case where we use autoencoder-derived features as inputs to the classification algorithm.

A final experimental setup uses a mix of autoencoder-derived features and hand-selected features. One dataset uses 3 autoencoder features concatenated with card area, peak surface load, and minimum surface load features to generate 6-dimensional data vectors. Another reduced dataset uses 3 autoencoder features concatenated with just card area to generate 4-dimensional data vectors.

Results

Whenever a well reports downtime for any reason, it is considered a failure scenario. When the SVM model, upon reviewing a day’s card shape, makes a failure prediction, we look ahead in a 30-day window in the data to see whether there is any well downtime reported. If there exists at least one downtime day within that window, we consider the prediction to be correct. This is how we calculate the failure prediction precision. Furthermore, we compress failure predictions on consecutive days into a single continuous failure prediction (e.g. failure predictions made for day $x$, day $x + 1$, and day $x + 2$ would be considered a single failure classification).

For calculating the failure prediction recall, we examine each reported failure date and the 30 days preceding the failure. If there is at least one failure prediction during this period of time, we consider the failure correctly predicted. Otherwise, the failure is missed.

Using the 3 hand-selected features (card area, peak surface load, minimum surface load), we get a failure prediction precision of 81.4% and a failure prediction recall of 86.4%.

After passing the raw data through an autoencoder to obtain 3 abstract features describing the shapes, we use the new features as inputs to the SVM. Under this arrangement, we get a failure prediction precision of 90.0% and a failure prediction recall of 86.1%.

The results show that the use of autoencoder-derived features as input to an SVM produces better results than using hand-selected features. A precision improvement from 81.4% to 90.0% will almost halve the number of false alerts in a failure prediction system. At the same time, the improved precision does not come at any significant cost to recall.

We conducted additional experiments by altering the size of our failure prediction window. The results are in Table 1 and Table 2.

The learning task is more difficult with a smaller failure window due to the size of the date range in our data. Our data spans only half a year, so a window of 60 days already spans one-third of the data. Simply predicting failure randomly would still produce a
Table 1: Precision and recall results for differing failure window sizes using 3 hand-selected features.

<table>
<thead>
<tr>
<th></th>
<th>30 days</th>
<th>40 days</th>
<th>50 days</th>
<th>60 days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>81.4</td>
<td>85.0</td>
<td>99.6</td>
<td>100.0</td>
</tr>
<tr>
<td>Recall</td>
<td>86.4</td>
<td>88.1</td>
<td>92.9</td>
<td>97.0</td>
</tr>
</tbody>
</table>

Table 2: Precision and recall results for differing failure window sizes using 3 autoencoder-derived features.

<table>
<thead>
<tr>
<th></th>
<th>30 days</th>
<th>40 days</th>
<th>50 days</th>
<th>60 days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>90.0</td>
<td>94.4</td>
<td>99.6</td>
<td>96.0</td>
</tr>
<tr>
<td>Recall</td>
<td>86.1</td>
<td>89.8</td>
<td>93.2</td>
<td>97.6</td>
</tr>
</tbody>
</table>

Table 3: Precision and recall results for differing failure window sizes using a hybrid feature set consisting of 3 autoencoder-derived features and 3 hand-selected features.

<table>
<thead>
<tr>
<th></th>
<th>30 days</th>
<th>40 days</th>
<th>50 days</th>
<th>60 days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>65.8</td>
<td>71.1</td>
<td>78.6</td>
<td>83.0</td>
</tr>
<tr>
<td>Recall</td>
<td>63.3</td>
<td>63.2</td>
<td>71.0</td>
<td>75.3</td>
</tr>
</tbody>
</table>

Table 4: Precision and recall results for differing failure window sizes using three autoencoder-derived features and card area for a total of 4 dimensions.

<table>
<thead>
<tr>
<th></th>
<th>30 days</th>
<th>40 days</th>
<th>50 days</th>
<th>60 days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>86.7</td>
<td>87.8</td>
<td>96.5</td>
<td>99.3</td>
</tr>
<tr>
<td>Recall</td>
<td>86.4</td>
<td>88.9</td>
<td>93.3</td>
<td>97.2</td>
</tr>
</tbody>
</table>

The results from using a hybrid feature set are poor compared to using solely autoencoder features or hand-selected features. We believe this can be partially explained by the curse of dimensionality: given the same amount of data, we are trying to compare the results of a 6-dimensional system against a 3-dimensional system. The feature space becomes increasingly sparse with additional dimensions and our training data may be insufficient to properly cover this space. To test this, we have included in Table 4 the results from using 4 dimensions: 3 autoencoder features and card area.

The results from using a 4-dimension mixed set are better than those from using a 6-dimension mixed set. They are still not as good as using purely autoencoder-derived features, though they do fare better than using only hand-selected features. There could be many reasons for this beyond simply dimensionality issues - attempting to combine disparate feature sets may increase the difficulty of learning, for example. Exploring why there is a performance difference between mixed feature sets and purely autoencoder-derived feature sets can possibly be the topic of future papers or discussions.

**Discussion**

Despite the power of machine learning, simply throwing raw data at various algorithms will produce poor results. Picking a good feature set to represent the raw data in machine learning algorithms can be difficult: to avoid the curse of dimensionality, the feature set should remain small, yet if we use too few dimensions to describe the data, we risk losing important information that is helpful for making correct classifications in machine learning. Hand-selecting features works but requires extensive experience or experimentation with the data, which can be time-consuming or technically difficult. But if we use autoencoders to generate feature sets, we can achieve comparable results even though the process is unsupervised.

Using autoencoder-derived features as inputs to machine learning algorithms is a generalizable technique that can be applied to any sort of data. In our case, we use it for dynamometer data, but in principle the technique can be applied to myriad types of data. Originally autoencoders were applied towards pixel and image data; here we have modified it to use position and load dynamometer data; and in future work, we will apply it to time-series data gathered from electrical submersible pumps. If a problem involves complex, high-dimensional data and there exists potential for machine learning to provide a solution, using autoencoder-derived features as input to the learning algorithm might prove beneficial.
A potential downside is that the abstract nature of autoencoder features can make the classification process of the machine learning algorithms more opaque than usual. Autoencoder features are generated through an unsupervised process and are unlikely to have a direct physical interpretation despite carrying important information. With hand-selected features, we might be able to casually guess how an algorithm is generating its predictions; with abstract autoencoder features, this is more difficult. But if we are primarily concerned with the classification results rather than the classification process itself, as usually is the case, using autoencoder generated features is a viable alternative to using hand-selected features.

**Acknowledgment**

We would like to thank Chevron U.S.A. Inc. for providing data and funding for this research project. We would also like to thank other collaborators whose work has indirectly helped in this project: Dong Guo, Yintao Liu, Anqi Wu, and Jingwen Zheng.

**References**