CS230

Pump It Up: Mining the Water Table

Alex Pham
Department of Computer Science
Stanford University
apham7@stanford.edu

Ben Backus
Department of Computer Science
Stanford University
mbackus@stanford.edu

Lauren Zhu
Department of Computer Science
Stanford University
laurenz@stanford.edu

Abstract

Used a dataset that contains data on approximately 60,000 water pumps in Tanzania, we applied deep learning methods to predict the functionality of pumps given a set of features. We were able to achieve 78% accuracy on our test set after testing many fully connected neural networks of different shapes and sizes.

1 Introduction

Despite advances in clean water technology, millions of people today continue to rely on unreliable water sources, including wells and pumps that may or may not be functional. Our goal is to predict the functionality of pumps, given a set of features. We can then improve maintenance operations and ensure that potable water is available to communities across Tanzania.

We decided that we wanted to pursue a project that reflected how AI, more specifically deep learning, can be wielded to enact social good. While deep learning can be applied for incredibly lucrative purposes, we wanted to use this technology to help people solve complex problems that affect their local communities.

This dataset was published a few years ago for a datamining competition hosted by DrivenData. There are a number of articles and papers that discuss how others have attempted to tackle this problem, and based on the most successful current approaches, the best accuracy for this dataset is around 82 percent. The most successful model out there seems to have been the random forest model. Though many groups implement different machine learning models, it seems that no group has attempted to apply deep learning to this dataset. We thought it would be worth our time to try and see if a deep neural network could achieve a higher accuracy than the other approaches.

The input to our model is a matrix that represents 40 different categories of features. The majority of these features are strings while the rest were integers and floats. We then used a fully connected neural net to predict a pump's functionality. The output of our model is a ternary classification; water pumps and wells are classified as functional, non-functional, or functional but needs repair.

2 Related work

When reviewing the literature[1][2][3][4][5], we found that in other examples of approaches to this problem, most groups used other machine learning algorithms. By using the random forest model,

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

groups were able to achieve upwards of 82 percent accuracy. Additionally, many of these groups used grid search to aid their hyperparameter tuning process. (See any of our references)

The strength of this model is definitely that the implementation seems to be relatively straightforward: using scikitlearn, one can set up a model with relative ease, leaving the brunt of the work to how one decides to clean up the data. Additionally, implementing grid search seems to be much more straightforward for a basic machine algorithm than for what we were trying to accomplish, because much of our time constructing the model was spent trying to find the right sizes of the hidden layers, and how many hidden layers was best.

The main potential weakness of this approach is that it lacks the complexity that a neural network could offer. We figured that a neural network with many layers and nodes could reach a higher accuracy. Furthermore, since it seems that all published solutions to this problem seem to have settled on the implementation of the random forest model, we thought we could expect better results from a relatively untouched strategy.

In these other attempts, the researchers all acknowledged that cleaning and parsing the data given was perhaps the most time consuming and difficult aspect of the project. A number of strategies and ideas were written about, some of which we implemented in our data preprocessing, and some of which we decided weren't reasonable. We also came up with some original means of processing the data that we thought might have helped our results.

3 Dataset and Features

Our dataset consists of 59,400 total examples, which we split into a 93:7 train to test ratio, with 55,000 training data points and 4,400 test data points.

We preprocessed the data so each example contains 432 feature values (most of which are one-hot encodings), and a corresponding one-hot label of size 3. The first steps of our pre-processing were to properly one-hot encode all non-numeric features in the dataset. Since most of the features were string-types, this also required some extra data analysis of the potential one-hot encodings available in each feature's column. Some features had thousands of entries, while others only consisted of a few dozen entries.

For the features that consisted of thousands of entries, we had to determine a cutoff point for which entries would be included in our one-hot encodings, and which entries would not. For example, we found that there were thousands of wards listed in the data, but no ward was entered more than two-hundred times, so we one-hot encoded for all wards that appeared more than one-hundred times, which came out to about two dozen new encodings.

For the date recorded features, we used the year and month, and omitted the day. We figured that the months could be indicative of cyclical patterns, so we one-hot encoded each month of the year, while the years were processed as integers. For construction year, a lot of examples entered 0 instead of the actual year, so we replaced 0 with the median construction year of all examples. We did a similar median replacement for population, since many 0 was also entered as the population for many examples when that's clearly not valid. But, for population, instead of taking the median population of the entire data set, we used the median population of all the water pumps that corresponded to that district code.

While many of the resources we read chose to eliminate some of the categories so as to eliminate false positives and noise, we found our best results in our processed dataset which omitted no categories. While this dataset contained some features that were arguably redundant, such as the three categories acknowledging scheme names, it makes sense to us that our deep learning model would thrive off of as much data we could give it.

We obtained our dataset from the Taarifa and the Tanzanian Ministry of Water. More information can be found here.

4 Methods

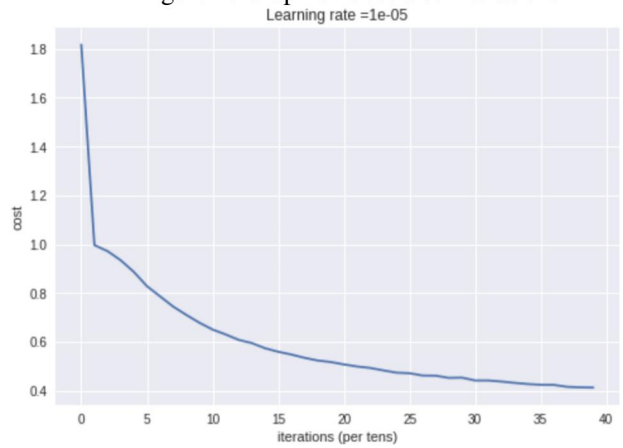
Because our dataset was relatively straightforward, we decided to use a fully connected neural network. We tested many different neural network architectures using various layer shapes and sizes.

Our final and most accurate model was a 9 layer fully connected network (layer shapes are 432, 4000, 192, 96, 48, 24, 12, 6, 3). We use the Adam Optimizer, which implements RMSprop and momentum for faster learning, and an exponentially decaying learning rate. For the last layer of our neural net we used a softmax function and for the remaining layers we used a ReLu function.

We used a cross-entropy loss function. For three-class classification our loss function was calculated as $L = \Sigma_{c=1}^{3} y_{o,c} log(p_{o,c})$ where $y_{o,c}$ represents the binary indicator (0 or 1) if class c is the correct classification for observation o and $p_{o,c}$ represents the predicted probability of observation o being in class c.

5 Experiments/Results/Discussion

Figure 1: Graph of Cost Over Iterations




```
Parameters have been trained!
('Train Accuracy:', 0.78505456)
('Test Accuracy:', 0.7861364)
```

After testing different model architectures, our final model was a fully connected neural network with the input layer (432 neurons), 7 hidden layers (4000, 192, 96, 48, 24, 12, 6 neurons), and an softmax output layer (3 neurons).

We had trained it to 560 epochs with a decaying learning rate initialized to 10^{-5} and decays at rate .99 with step size 2000. This decaying learning rate was effective in converging in later iterations, as our mini-batch size of 250 created noise that led cost to diverge when using a constant learning rate.

While certian hyperparameter tuning methods suggest that a decaying learning rate creates results analagous to increasing the batch size, we found that a batch size of 250 was optimal as it increased training speed (our GPU timed out after extensive computation), and our learning was not impeded by noise.

Learning Rate	No. Epochs	Batch Size	Train Acc.	Test Acc.	Hidden Layer Shapes
10^{-5}	100	250	71.66	71.88	100/40/15/6
10^{-5}	40	250	73.23	74.27	1000/40/15/6
10^{-5}	60	250	75.46	76.41	4000/40/15/6
10^{-5}	49	250	76.19	76.88	4000/40/15/6
10^{-5}, .9	300	250	77.01	77.47	4000/40/15/6
10^{-5}, .8	150	250	72.91	73.07	1000/400/100/30/15/6
10^{-5}, .89	181	920	77.36	78.36	4000/40/15/6
10^{-5}, .99	400	250	78.51	78.61	4000/192/96/48/24/12/6

We don't think that our algorithm overfitted to the training set because in almost every instance, our training accuracy is very close to our test accuracy. This means that our model generalized well to unseen data.

6 Conclusion/Future Work

After testing different model architectures, our final model was a fully connected neural network with the input layer (432 neurons), 7 hidden layers (4000, 192, 96, 48, 24, 12, 6 neurons), and an softmax output layer (3 neurons). We use the Adam Optimizer, which implements RMSprop and momentum for faster learning as well as a decaying learning rate to improve accuracy. We found a large increase in accuracy after implementing a decaying learning rate. We believe this was the case because the decaying learning rate helps us reduce the loss quickly at the beginning but as parameters become increasingly accurate, we reduce the importance of the batch we train on.

For future work, we saw three main routes to take: (1) Identify Important Features. Although we did do some pre-processing, by using data visualization techniques to better understand the data, we think we could improve accuracy by looking at the importance of individual features. This would lead to better representation of the data, faster training, and higher accuracy. (2) Estimate Decay Rate and Lifetime of a Pump. Instead of doing a three-class classification, we could attempt to estimate the lifetime and decay rate of a pump. By working with Taarifa to get the approximately 20,000 missing construction years in the dataset, we could see trends in pump decay based on other features. (3) Improve Success Metrics. Although our work helps to see the functionality of a pump, we could improve our success metric to calculate the lives improved per pump. In doing so, we could tell which pumps are crucial to water access for Tanzanians.

7 Contributions

We each contributed equally. For the coding section, Ben focused mostly on the preprocessing and Alex and Lauren focused on the deep learning model. We all took part in the hyperparameter training. Lauren did most of the poster, Ben did most of the writeup, and Alex helped on both.

References

[1] Addison, Joseph. “Pump it up – Data Mining the Water Table.” Yet Another Compsci Guy, 12 Jul. 2017, https://jitpaul.blog/2017/07/12/pump-it-up/.

[2] Agrawal, Karishma, et al. Pump it Up Data Mining for Tanzanian Water Crisis. andrew.cmu.edu, 2015, pp. 1–12, Pump it Up Data Mining for Tanzanian Water Crisis.

[3] Kim, Joomi. “Predicting Non-Functional Water Pumps in Tanzania.” Joomi K Blog, 3 Feb. 2017, joomik.github.io/waterpumps/.

[4] Kremonic, Zlatan. “Predicting Status of Tanzanian Water Pumps.” Zlatan Kremonic – My Mathematical Mind, 23 Jan. 2017, zlatankr.github.io/posts/2017/01/23/pump-it-up.

[5] Malone, Katie. “Workflows in Python: Getting Data Ready to Build Models.” Civis Analytics, 27 Dec. 2015, https://www.civisanalytics.com/blog/workflows-in-python-getting-data-ready-to-build-models/.