Well that's a Re-leaf: Reproducibility in the classification of leaves

Nicholas Brown[1]

[1] *Research School of Computer Science, Australian National University*

{u5563539@anu.edu.au}

Abstract. Replication is a crucial part of Science. We formulate an independent model to achieve similar results to Silva (2013). Our model is a feedforward neural network, trained using stochastic gradient decent. The data set under consideration has 14 shape features and 340 samples. Different network topologies were experimented with and a classification model was built. Various methods were used for understanding the network. A clustering approach using the k-means algorithm achieved a benchmark of 53%. Test set accuracy of our neural network achieved 81%, which is comparable to the original results of 87% [1].

Keywords: Artificial Neural Network, Leaf Dataset, Classification, Evaluation Metrics, pytorch

1 Introduction

One must have been living under a rock to not have noticed the recent replication crisis in psychology [2]. Indeed, medical research may be noticing similar concerns [3]. This paper is undertaken, to gain a deeper understanding of the tools and techniques involved. Clarity and interpretation is given precedent over raw predictive power.

It would be hubris to claim that Machine Learning suffers from none of the issues of the medical and psychology research community. The state of the art is still (currently) human researchers. Humans are known to suffer from some systematic bugs2. We all want our research to be novel, exciting and well received. While many of us abstain. The temptations are there; trading interoperability for prediction, using more and more of the test set, finding metrics that show good results or discarding inconvenient data.

It is in that spirit that the leaf dataset [1] was chosen. This author was not able to implement Lest Trimmed Squares (see section 2.3 below) So our aim is to replicate the original results in [1]. In doing so, we hope to gain an understand of the leaf classification problem. We also hope to explore different hyper-parameters and see what effect this has on the network's classification accuracy. This dataset has 340 examples of 36 different types of leaves.

---

[2] https://en.wikipedia.org/wiki/List_of_cognitive_biase

Fig. 1. Each of the different types of leaves, these form the classes used by our classifier. [1]

2 Method

2.1 Data Prep.

First some data processing was required. The data set from Silva et al [1] had already been converted from raw images into numeric values (csv). This was through the selection and measurement of particular features; such as Elongation and Lobedness.

The second column was discarded as it kept count of the number of specimens within each type of leaf (class). As we don't want larger features to bias the model we normalize and re-scale within each feature so that they fit between [0,1].

We normalize for the whole dataset, then partitioned the data into the training and test sets (75% and 25% respectively).

If we use more data in our training stage this will yield a more powerful predictor. However, this trades off against an accurate idea of how well our model performs on unknown data.

In our case we have 340 examples and 36 classes. This is approximately 10 examples of each type of leaf. At a minimum we desire 2 novel examples of each leaf. This is 72 examples, but for a margin of error as the split is chosen randomly ~75% was the chosen size of the test set. Hence, a quarter of the data, was reserved for testing.

The data was shuffled to remove the class ordering and split into the testing and training sets. The code scaffold was adapted from previous work [4]. Notable one-liners include:

```
data      = data.sample(frac=1).reset_index(drop=True)
selector  = np.random.rand(len(data)) < 0.75
train     = data[selector]
test      = data[~selector]
```

2.2 Network Architecture

We have 15 features to discriminate 36 different classes. Our 340 examples correspond to approximately 10 examples within each class. Due to the relatively small number of training examples there is a chance of high variance is great. Due to the ability of a large network to memorizing the training examples, we will choose a small architecture.

The number of neurons in the input layer was set equal to the number of features (16). The final layer was equal to the number of classes (36) standard practice for classification. It was decided that only one hidden layer would be needed as it added model complexity and did not yield much increased test set prediction accuracy. An additional hidden layer was tested, however no noticeable improvement was observed.

The cross-entropy loss function is an intuitive choice for our logistic regression problem:

$$J(\mathbf{w}) \;=\; -\frac{1}{N}\sum_{n=1}^{N}\Big[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\Big]$$

2.3 Technique

Many hours were spent in the fruitless implementation of a custom cost function. Attempts were made towards Least Trimmed Squares and the Absolute Criterion Method. These techniques were Taken from Joines [5] which was referenced by Gedeon [6] Unfortunately, this author's skill in pytorch was not enough for the network trained using the new cost function to yield any sensible results. He was adrift lost in a sea of tensors and variables, unable to write cost functions that were differentiable for the backward pass of the network.

LTS commonly formulated is:

$$S_k(\beta) = \sum_{j=1}^{k} r_{(j)}(\beta)^2$$

For each data point you calculate it's contribution to the error (the residual) $r_i(\beta) = y_i - f(x_i, \beta)$. Then you sort these residuals in order of size. After that take the sum of the first k or them. This will exclude (n-k) number of data points. The hope here is to discard the very noisy points. These will have the largest error.

2.4 Evaluation Metrics

At first a confusion matrix was considered. However due to the large number of different leaves involved in the classifier. Inspection of this matrix proved unpleasant. Instead for the most part we used the accuracy metric. Accuracy here is the sum of the correctly classified data pointes divided by the total number of examples.

Purity is a common metric for evaluating how "goodness of fit" for a cluster. Take the sum of the most frequent class in that cluster. Repeat this for each k clusters. Sum this then divide by the total number of examples. It's as if the most frequent class within the cluster is classified correctly. Formally purity is given by:

$$\frac{1}{N}\sum_{m\in M}\max_{d\in D}|m\cap d|$$

given a set of clusters M (=40) and some set of classes D (=40).

Further research might involve using cross-validation to measure the strength of the k-means against the constructed neural net. Currently we only have the testing set accuracy to compare them against each other. Perhaps also the k-means clusters could be scored probabilistically, yielding greater nuance in the cluster overlaps.

3 Results & Discussion

Using the architecture decided upon above, many different trials were run. To understand what the hyper-parameters should be set to the following graphs were generated.

We can see in Figure 2, that the larger the learning rate the faster it descends the cost function. If the cost function is too small, it can get stuck in a local minimum as is the case with $\eta = 0.0001$. Conversely if a learning rate is too large $\eta > 0.5$ (not shown), it begins to oscillate, with dramatic spikes in the cost as it leaps back and forward over a minimum.

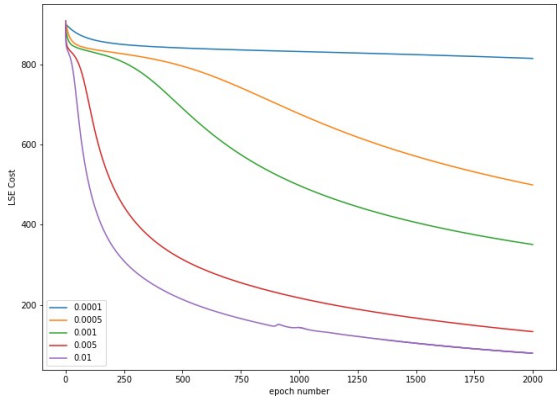

Fig. 2 Classifiers with different learning rates.

In Figure 3, we can see the effects of how many steps the classifier takes. The learning rate uniquely defines the shape of the curve, the number of epochs determines how far along the curve the classifier will go. As the epoch number is increase from 10-10,000, the classifier descends father. It is imaginable that with more complex datasets classifiers may not have time to converge due to the sheer number of epochs needed.

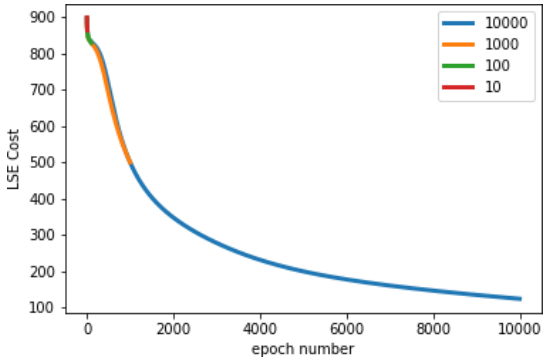

Fig. 3 The same Classifier running for different epoch lengths.

In Figure 4, we can see a contour plot. Each point on the plot corresponds to a different way of exploring the cost function, dictated by the epoch number and learning rate.

We can see that within the blank area a learning rate, $\eta \epsilon [0.02, 0.08]$ and epochN ϵ [1000, 2500], achieves a similar cost. Findings like these might allow for you to save time by running the classifier for less epochs.

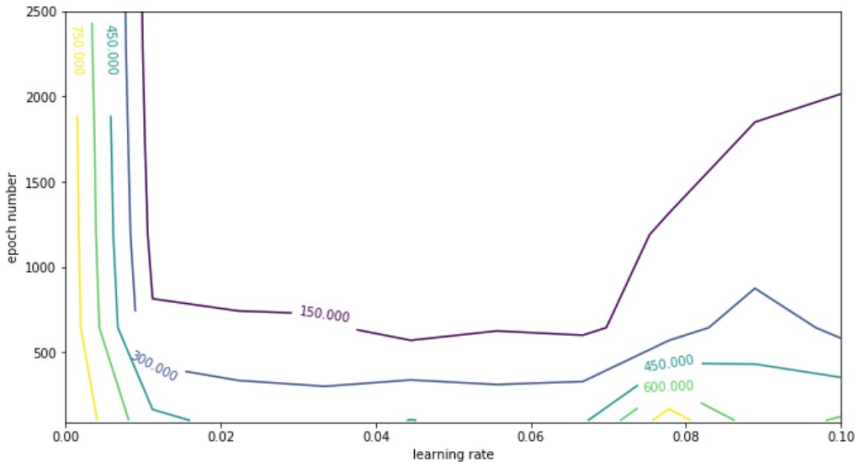

Fig. 4 The shape of the cost landscape under different hyper-parameters.

In Fig. 5 we have generated a similar contour plot to Fig 4. The contour height is test set accuracy instead of cost. Here we can see that staying within the yellow region will yield the greatest accuracy.

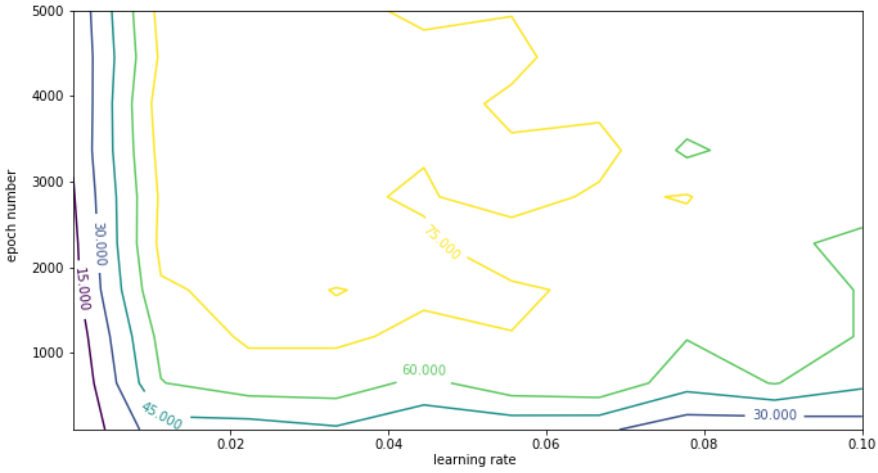

Fig. 5 Test set accuracy of different hyper-parameters.

Now we ask how well our network performed compared to the existing literature [1]. Test set accuracy of 81% was achieved, which was near, but not quite as good as the 87% accuracy in the original work [1]. While it is tempting to use more of the test set, the risk of overlearning on such a small data set is very great.

Additionally, a simple k-means classifier was trained as a baseline. Here k-means serves as a "naïve" benchmark to compare our model against. K was set to 40 as we have 40 different types of leaves. This classifier had a purity of .62 and a test set accuracy of 53%. This was 28% less than our neural architecture. Perhaps we're onto something.

4 Conclusion and Future work

We have begun to get a picture of how some kinds of classification work. First we chose the leaf dataset, and inspected the problem. A feedforward neural network was trained under gradient decent. Different network topologies were experimented with (extra hidden layers). The model was trained on 75% of the data.

We then took some time to investigate hyper parameters such as the learning rate, and the number of epochs. A few graphs were generated in an attempt to show how different settings of these parameters effect the outcome in terms of cost and test-set accuracy.

Our classifier achieved a test set accuracy of 81%. A K-means classifier was built which managed test set accuracy of 53%. It's clustering purity was 0.62.

Further research would involve using cross-validation to evaluate more general neural architectures. Perhaps we could compare our k-means classifier to the trained NN. What about other architectures, how might a decision tree hold up? Perhaps a Generative Adversarial Network to create new leaf images? [8]

An exciting direction this research could be taken is feature visualization [7]. This approach yields greater insight into how different parts of the network contribute to the classification. It does this by forcing a "bottleneck" of a few neurons in the middle of the network.

References

1. Pedro F.B. Silva, Andre R.S. Marcal, Rubim M. Almeida da Silva 'Evaluation of Features for Leaf Discrimination', In: Springer Lecture Notes in Computer Science, Vol. 7950, 197-204. (2013)
2. Open Science Collaboration, Estimating the reproducibility of psychological science. In: *Science*, 349 (6251), pp.aac4716-aac4716. (2015)
3. Ioannidis JPA, Why Most Published Research Findings Are False. In: PLoS Med 2(8): e124. (2005)
4. Gedeon, T. lab2/task1_glass_binary.py (2018):
5. Joines, M and White, M, "Improving generalisation by using robust cost functions," In: *IJCNN*, vol. 3, pp.911-918, Baltimore, (1992)
6. Slade, P. and Gedeon, T. Bimodal distribution removal. In: New Trends in Neural Computation, pp.249-254.(1993)
7. Olah, et al., "Feature Visualization", In: Distill, (2017)
8. Goodfellow, Ian J et al., *Generative Adversarial Nets*, In: NIPS (2014)