CS230

Identifying DNA Sequencing Anomalies

Ryan Humble*
Institute for Computational and Mathematical Engineering
Stanford University
ryhumble@stanford.edu

Chuck Seberino*
Roche Diagnostic Systems
seberino@stanford.edu

Abstract

We used a deep neural network to classify nanopore insertion during a pre-sequencing setup step for nanopore-based DNA sequencing. Several networks and loss formulations are considered to achieve higher classification accuracy. Lastly, we explore an explainability analysis technique to analyze which features are most important in creating a more ideal sequencing setup.

1 Introduction

DNA sequencing is a complex, multi-step process that is prone to errors and anomalies. Modern nanopore-based DNA sequencing heavily relies on a successful pre-sequencing setup to create the ideal conditions from which to measure the polymerase incorporation activity (i.e. identifying nucleotide bases). Incorrect setup conditions can lead to poor or corrupted DNA reconstruction and identification. Each correctly calibrated sequencing sensor should have exactly one fully formed nanopore inserted into the measurement region. Of significant concern is when more than one pore is inserted, causing mixed signals to be received, contaminating results. A lesser concern is when no pores are inserted, leading to no measurement. The goal is to optimize this pre-sequencing setup to provide ideal initial conditions that minimize sequencing errors. This requires identifying which features of the setup are most salient to nanopore insertion and, ultimately, sequencing error.

Our primary contribution is a neural network to classify the nanopore insertion as no, single, or multi-pore, given measurements from the various pre-sequencing setup steps. We also perform an explainability analysis to determine which feature of the sequencing setup are most impactful.

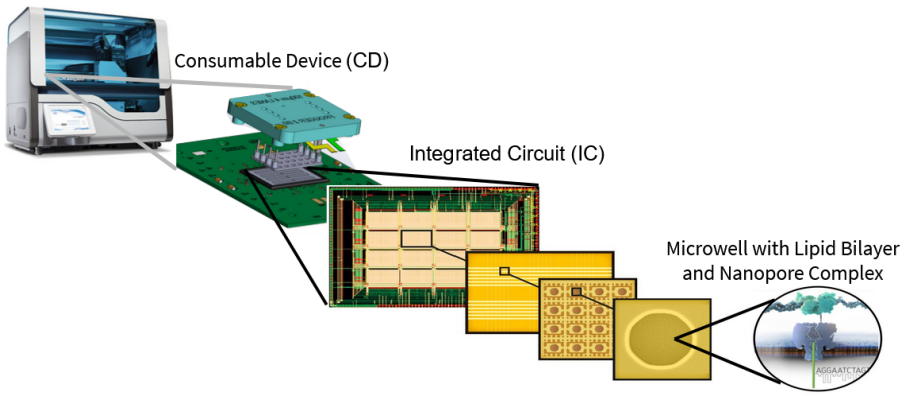


Figure 1: Illustration of nanopore sequencing platform.

CS230: Deep Learning, Fall 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

2 Related work

Using deep neural networks for classification is increasingly ubiquitous in nearly every field. For large, well-balanced datasets, deep neural networks are incredibly accurate. However, many real-world applications struggle from highly imbalanced datasets. As summarized in [1], many approaches use sampling techniques to balance the classes. Most common are random oversampling of minority classes or undersampling of majority classes. Typically, oversampling suffers from overfitting while undersampling drops important information from the majority class. SMOTE [2] uses a more sophisticated oversampling technique whereby synthetic data is created between minority samples and its nearest minority neighbors in the feature space, thereby ideally avoiding overfitting. More recently, [3] uses an SVM's support vectors to undersample the majority class without losing important information, but this general technique is limited to methods that emit "hard" examples from the majority class. Although each of these techniques has its drawbacks, they have been shown to significantly improve classification accuracy across a wide array of benchmarks.

For explainability analysis of fully-connected networks, a common technique is gradient fingerprinting. As demonstrated in [4] and [5], by considering the gradient of the loss function with respect to the input features, the most salient features can be inferred. Another explainability approach is LIME [6], which learns a simple (typically linear) local classifier near a test point to more explain the classification result. A related method, SHAP [7], seeks to quantify each feature's contribution of the prediction using Shapley values.

3 Dataset

The dataset will comprise of measurements collected from a nanopore sequencing instrument over the course of 2 runs, where each run started with 2 million active cells. A final active cell count of 954368 and 937984 samples, respectively, was recorded. Each cell sample contains 59 input features, 32 of them as categorical state classifications, and 27 as numeric electrical measurements. These features are selected from 10 different measurement captures, taken as part of the pre-sequencing setup and calibration of the sequencer. Optimal setup of the instrument requires precise application of both a lipid bilayer substrate over a sensor well (mimicking a cell wall), and insertion of a single nanopore within the bilayer. Small electrical fields are generated to coax these formations along. This time consuming and delicate process is prone to errors and is not fully understood. Nearly half of the active cells (1.88 million values) are labeled with an initial classification label, identifying a no pore (0), singlepore (1), or multi-pore (2) state. The nanopore labels are highly imbalanced, as shown in Table 1.

Label	Count	Percentage
No Pore	29,806	1.586%
Single Pore	1,848,867	98.410%
Multi Pore	72	0.004%
Total	1,878,745	

Table 1: Statistics on prevalence of nanopore labels

We split the available data into train, validation, and test sets at the ratio $80 : 10 : 10$ and standardized each feature to have zero mean and unit variance.

4 Methods

Our problem is a highly imbalanced classification problem. We construct our classification algorithm from a combination of a sampling method, a neural network architecture, and a loss formulation.

For our sampling method, we considered several popular techniques: random undersampling, random oversampling, and SMOTE. Consider a classification problem with C classes with counts c_i. Random undersampling discards data from the original dataset; at the extreme, it removes data from each class until all classes have exactly $\min_i c_i$ data points. Random oversampling adds duplicate data from the minority classes; at the extreme, it adds until all classes have exactly $\max_i c_i$ data points.

SMOTE is a variant of oversampling that creates new data for minority classes by interpolating between two existing data points of a minority class, in the hope of avoiding overfitting; however, we found SMOTE to be too computationally expensive for a dataset with over a million points. We instead elected for a balance of undersampling and oversampling and used a weighted random sampling scheme where the sampling probability was the inverse of the class counts. This effectively rebalances the training set so that each class is approximately equally represented.

For our neural network, we used the deep feed-forward network architecture presented in Figure 2. We used only 43 input features from the dataset and have 3 classes. Except for the final layer before the output, the hidden layers are each a DenseBlock unit, which is composed of a linear layer, ReLU activation, batch normalization, and a dropout layer. We choose ReLU activation to avoid vanishing gradient issues with tanh/sigmoid. We added batch normalization to prevent covariate shifts. Lastly, we add a dropout layer with a configurable drop percentage.

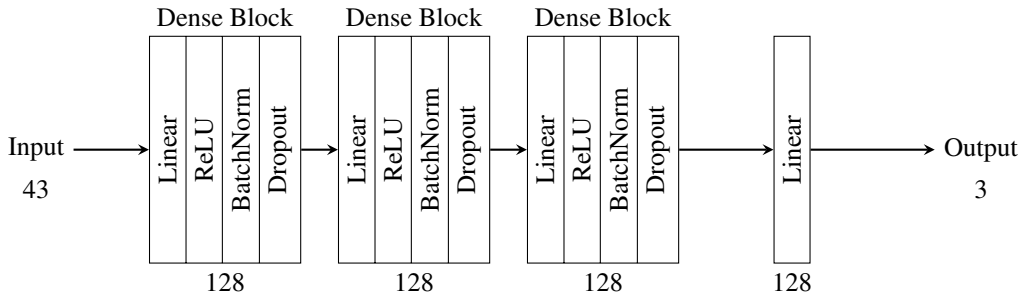


Figure 2: Nanopore classification feed-forward neural network architecture.

For our loss formulation, we consider three different approaches. Consider a batch of size n and number of classes cl. Let z be the output from the final layer and $\hat{y} = \text{softmax}(z)$. The first loss formulation is the standard cross entropy loss

$$\mathcal{L}_{CE}(\hat{y}, y = e_j) = -\log(\hat{y}_j)$$

The second loss formulation is the standard multi-margin loss

$$\mathcal{L}_{MM}(z, y = e_j) = \sum_k \max(0, 1 + z_j - z_k)^p$$

where $p \in \{1, 2\}$ distinguishes between a hinge and squared hinge loss. For both of these formulations, the batch loss is the mean of the individual losses. The third loss formulation follows [8] and effectively turns the last layer into several one-vs-rest linear SVMs. Let $W \in \mathbb{R}^{3\times 128}$ be the weight matrix from the final linear layer. Let $t_{ij} = \mathbb{1}(i = j) - \mathbb{1}(i \neq j) \in \{+1, -1\}$ be the target for the i^{th} training example with respect to the j^{th} SVM. For the entire batch, the loss for the j^{th} SVM, corresponding to class j, is

$$\mathcal{L}_{SVM,j}(Z, Y) = \frac{1}{2} \|w_j\|_2^2 + \frac{C}{n} \sum_i \max(0, 1 - z_{ij} t_{ij})^p$$

where p is the same as with the multi-margin loss loss and C is a hyperparameter that penalizes violations. The total loss is then

$$\begin{aligned} \mathcal{L}_{SVM}(Z, Y) &= \sum_j \mathcal{L}_{SVM,j}(Z, Y) \\ &= \frac{1}{2} \|W\|_F^2 + \frac{C}{n} \sum_{i,j} \max(0, 1 - z_{ij} t_{ij})^p \end{aligned}$$

5 Results

We use AUPRC (area under precision-recall curve) as our primary metric, where we use a one-vs-rest setup since there are more than two classes. As motivated in [9], we specifically avoid AUROC (area

under receiver operating curve) since our data is heavily imbalanced. A F1-score could also be used here, but this only captures the model performance at a single threshold along the PR curve. For visualization, we include the confusion matrix for the best models.

Our nanopore classification method has a number of hyperparameters. For training, we choose a class-weighted resampled dataset of the same size as the original training set, a batch size of 256, and the number of epochs as 10. These were chosen to achieve a moderate training time (roughly 20 minutes on a K80 GPU). We varied the learning rate in $\{10^{-4}, 5 * 10^{-4}, 10^{-3}, 5 * 10^{-3}, 10^{-2}\}$ and varied the dropout rate in $\{0, 0.2, 0.4\}$. We set $p = 2$ for the multi-margin and SVM-based loss and set $C = 5$ for the SVM-based loss as to heavily weight violations; we ran out of time to more carefully select these hyperparameters. We used PyTorch [10] and PyTorch Lightning [11] to define and train the models. Also, we use both logistic regression and linear SVM without any sampling as baseline methods; we define and train these models with scikit-learn [12].

Table 2 shows the performance of these methods; the confusion matrices are shown in Figure 3. For the deep methods, we chose the hyperparameter setting that maximized the average AUPRC. For each loss function, the best cross-validated dropout rate was in fact 0 . For the cross entropy and SVM losses, the learning rate was 0.001; for the multi-margin loss, the learning rate was 0.01.

	No Pore	Single Pore	Multi Pore
Logistic Regression	0.452	0.999	0.407
Linear SVM	0.439	0.999	0.292
NN (CE Loss)	0.736	0.999	0.461
NN (MM Loss)	0.716	0.999	0.585
NN (SVM Loss)	0.733	0.999	0.418

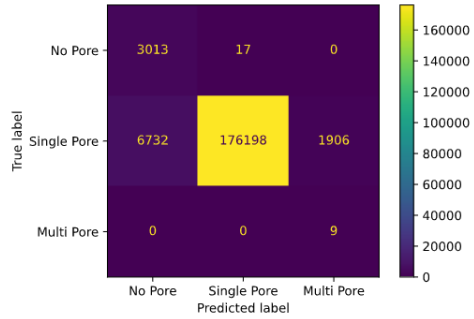
Table 2: Experiment results on classifying nanopore label. AUPRC is calculated as one-vs-rest.

We also seek to derive insights into how to create a more ideal sequencing setup. For the best model above, the deep network with the multi-margin loss, we now employ SHAP [7] to explain the classification results. We specifically consider the explanation for no pore and multi-pore data points in the testset. Figure 4 shows an explanation summary. This very clearly identifies a single feature, *occal_oc_calibration_pos_oc*, as being highly explanatory. Small *occal_oc_calibration_pos_oc* is strongly associated with no pores while large *occal_oc_calibration_pos_oc* is strongly associated with multi-pores.

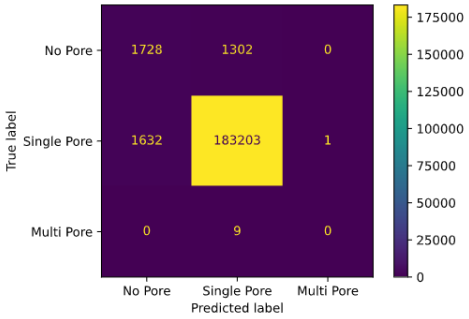
6 Conclusion/Future Work

The goal of this project was to identify bad pores and gain insights into better optimizing the pre-sequencing setup. We found that a deep feed-forward network with a multi-margin loss performed quite well despite a massive data imbalance. Using this model, we were able to infer an important explanatory feature that is strongly related to the pore quality.

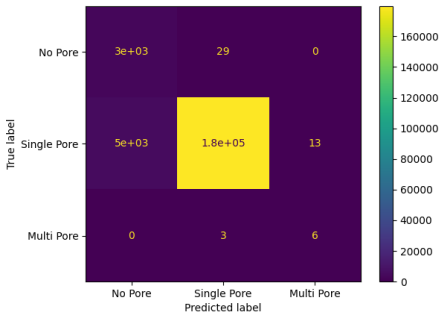
With more computational resources, we would have considered more expensive sampling techniques, like SMOTE, and a deeper neural network, as there were no indications of overfitting to the training set. Additionally, if we had more time, we would have considered the harder, but potentially more informative, regression problem. For roughly 0.5% of the data, we have a measure of the sequencing correctness: the composite edit distance between the measured DNA read and the "true" sequence. The goal of the pre-sequencing setup is to ultimately enable correct sequencing. Using a similar progression from simple to deep models above, and employing similar explainability analyses, we would hope to learn which setup features are most important to minimizing the ultimate sequencing error.



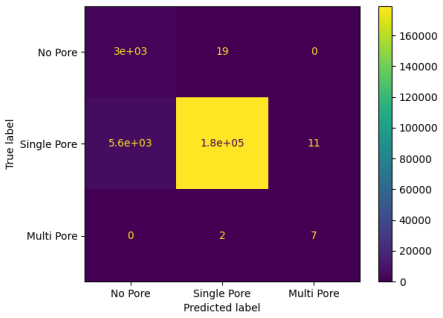
(a) Logistic regression classifier



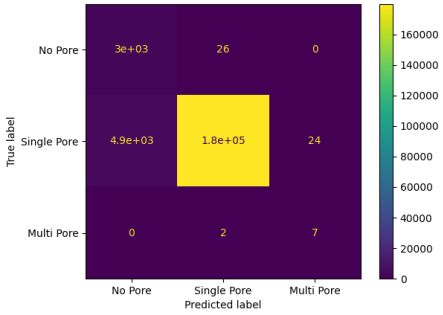
(b) Linear SVM classifier



(c) DNN (CE Loss)

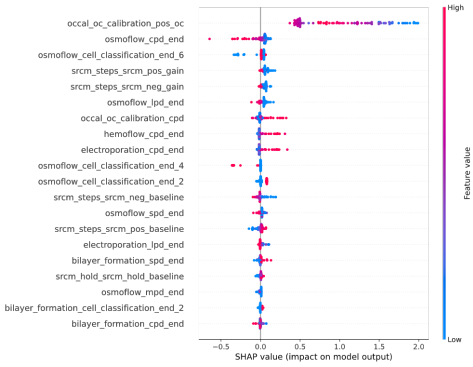


(d) DNN (MM Loss)

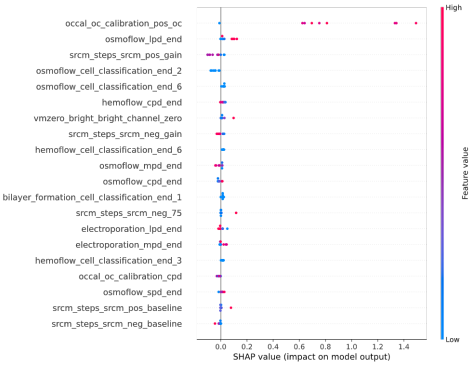


(e) DNN (SVM Loss)

Figure 3: Confusion matrices for selected classifiers.



(a) No pore explanation



(b) Multi pore explanation

Figure 4: SHAP explanation summaries for bad pores. Large SHAP values correspond to a large effect on the predicted class label. Feature coloring is only relative to the other data points in the plot.

7 Contributions

Ryan wrote the "Related work", "Methods", and "Results" sections. He also wrote the code to obtain the classification results and SHAP explanations.

Chuck wrote the "Introduction" and "Dataset" sections. He also did all of the data acquisition and wrote the associated code. He also put together the slides for the video and the nanopore platform graphics.

References

[1] H. He and E. A. Garcia. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, September 2009. Conference Name: IEEE Transactions on Knowledge and Data Engineering.

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002.

[3] M. Y. Arafat, S. Hoque, S. Xu, and D. M. Farid. An Under-Sampling Method with Support Vectors in Multi-class Imbalanced Data Classification. In *2019 13th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, pages 1–6, August 2019. ISSN: 2573-3214.

[4] Yuanchao Shu, Yinghua Huang, Jiaqi Zhang, Philippe Coue, Peng Cheng, Jiming Chen, and Kang G. Shin. Gradient-Based Fingerprinting for Indoor Localization and Tracking. *IEEE Transactions on Industrial Electronics*, 63(4):2424–2433, April 2016.

[5] Quoc Phong Nguyen, Kar Wai Lim, Dinil Mon Divakaran, Kian Hsiang Low, and Mun Choon Chan. GEE: A Gradient-based Explainable Variational Autoencoder for Network Anomaly Detection. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 91–99, June 2019.

[6] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *arXiv:1602.04938 [cs, stat]*, August 2016. arXiv: 1602.04938.

[7] Scott M Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[8] Yichuan Tang. Deep Learning using Linear Support Vector Machines. *arXiv:1306.0239v4 [cs, stat]*, page 6, February 2015. arXiv: 1306.0239v4.

[9] Takaya Saito and Marc Rehmsmeier. The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLoS ONE*, 10(3), March 2015.

[10] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[11] WA Falcon. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.