Dealing with Imbalance in Computer Vision

1st Jeffrey Xu
Department of Computer Science and Engineering
Texas A&M University
College Station, TX, US
jeffreyxu@tamu.edu

***Abstract*—Data imbalance is a universal problem in the world of machine learning and computer vision. Classes that are represented less in the data may get overlooked during training resulting in poor performance on those classes. Many different methods and models have been created over the years to help against imbalanced data such as sampling methods or new models. In this paper, we explore the world of imbalanced data and its effects on training and different methods that can help improve the performance of models.**

***Index Terms*—Imbalanced Data, Computer Vision, CNN, Logistical Regression, Random Forests**

I. Introduction

Collecting large amounts of data is very important when training models. However, this data is often imbalanced and some classes are represented more than others. This creates some major problems when training machine learning models as the imbalanced dataset may not lead to amazing performance overall across all the different classes. As stated in [1], the issue of imbalanced data has been a issue within machine learning for a long time that goes across all sections of machine learning.

Finding efficient and simple solutions to imbalanced data could help data scientists across the world better train their models and achieve better performance. Often the data collected from the real world is imbalanced. For example, collecting data about people's favorite food by surveying people over the internet may only give us data from first-world countries as those are usually the countries where a mass-majority of the population has access to the internet. The amount of people from third-world countries will be greatly under-represented and this poses as an issue if data scientists and machine learning engineers want to use this data to train models to predict people's favorite food.

In this paper, we want to explore the effects of imbalanced datasets in Computer Vision problems. We will specifically be using the MNIST digits dataset to test our models and methods on since the MNIST dataset is a very popular dataset to perform testing on. Many other papers such as [2] have specifically used the MNIST datasets to determine the models that perform the best over a variety of MNIST datasets.

The issues of imbalanced data have been explored before. Japkowicz [3] tried to determine how imbalanced datasets affect multilayered perceptron models and concluded that the multi-layered perceptron was more sensitive to imbalanced datasets when the complexity of the domain increased (linear separable domains were found to not be that sensitive to the data imbalance issues). Japkowicz also found that sampling methods such as down-sampling or up-sampling was very effective with dealing with imbalanced data, though it was found that down-sampling majority classes were slightly more effective.

Computer Vision is very useful within real-world applications. It has been applied to problems such as autonomous driving, facial recognition, image compression and many other fields of industry. However, problems in computer vision are still prone to the general issues that imbalanced datasets give.

In the *Difference of Methods* section, we talk about the implementation of our models, datasets and any techniques that were used to help counteract the negative effects of the imbalanced datasets. In the *Test Results* section, we go over the overall performance of the models and the techniques used for imbalanced datasets. Within the *Comparison of Techniques* section, we analyze the performance of each model along with the imbalanced dataset techniques used and determine the effectiveness of them. We also analyze the difference of the performance of models with balanced and imbalanced datasets and see how much imbalanced datasets can affect models. Finally, in our *Conclusion* section, we wrap up our discussion on imbalanced datasets and see how the specified techniques in this paper could help machine learning applications in the future.

II. Difference of Methods

The MNIST digit dataset was used for this paper. The data was first loaded into a torch dataset then converted to a torch data loader (the data loader is used during CNN to load the data in batches for training). We also converted the torch dataset tensors into numpy arrays to make the compatibility with the *sklearn* models an easier transition. To do this, we essentially looped through all the given torch datasets and converted each feature array into a flattened numpy array and the labels tensor into a 1D numpy array containing all the labels.

For this paper, we tested three distinct models on our MNIST dataset. The models used were Logistical Regression, Random Forests and Convolution Neural Networks. All three models can be used for classification problems.

A. Logistical Regression

Logistical Regression is a classic model used in classification problems. It uses a non-linear logistical function to convert a linear combination of the input data to likelihoods of each respective output class. It also has some regularization to prevent the coefficients of the linear combination to explode and overfit to the training data. This model is very simple but very powerful within the realm of machine learning.

The *sklearn* library was used to implement the Logistic models. Cross Validation was used to determine the best regularization parameter and the *lbfgs* solver was used for multi-class classification. Two different models were fitted for the two different provided datasets (one model for the linear dataset and one model for the step dataset).

B. Random Forest

Random Forests are an extension of decision trees. Decision trees split the dataset into partitions and assigns each equivalence class an output class. Decision trees are very interpretable to the human mind and are often perform well with classification problems. However, decision trees can tend to overfit and perform poorly on output classes that are less represented in the training data. This can be solved using post or pre-pruning but pre-pruning often doesn't give the best performance and post-pruning is very computational heavy as it has to perfectly fit to the training data and then begin reducing the model bottom-up. Luckily, the Random Forest classifier solves this issue by creating multiple decision trees and using this list of decision trees to predict. Within each decision tree, only certain features can be used to predict to prevent every single decision tree from looking the exact same as the others. Doing so allows the Random Forest classifier to handle smaller output classes and overall performs much better than simple decision trees.

The *sklearn* library was also used to implement our Random Forest Classifier and also performed cross validation to determine the optimal number of classifiers to use in our Random Forest Classifier model. Once again, we fitted models for both datasets and for each dataset. During cross validation, we cross validated models between the Gini index and the entropy loss function and use the model that had the best performance during cross validation to create our actual model.

C. Convolutional Neural Network

Convolutional Neural Networks are a specific form of neural networks that are used for computer vision problems. At each convolutional layer, a filter is used to compute output values for that layer. At the end, fully-connected layers are used to compress the output data from the convolutional layers down to likelihoods of each output class which can then be used to predict. This model is the most complex model out of the three but performs very well with image classification and computer vision problems.

Finally, we used *pytorch* to implement our Convolutional Neural network. A template [4] was used to generate the code for the CNN class used in this paper. We used a 1D convolutional layer, followed by two 2D convolutional layers, finally finishing with two fully connected layers that output the likelihoods for each of the ten output classes. We also used a momentum optimizer along with stochastic batch learning for our CNN when performing gradient descent.

D. Imbalanced Data Techniques

Along with implementing our models, we also implemented some additional sampling methods to counteract the imbalanced dataset. Specifically, undersampling was implemented to help balance the dataset and give better results. For our dataset, *imblearn*[5] was used to implement the undersamping. We used the *RandomUnderSampler* to perform all undersampling. Oversampling wasn't performed in this paper since it is very computationally heavy. Note that access to large servers or GPUs were not possible during the time of implementing these models and methods and computationally efficient algorithms had to be used.

III. Test Results

For our models, the *AUCROC* score was decided to determine the performance of the models since it is a generally unbiased form of measurement. We had to implement a multiclass version of the *sklearn* ROC score function. To do this, every single class had an ROC score (all vs. one) and the scores were averaged all classes to produce one final performance value.

A. Logistical Model

For our imbalanced logistical model trained using the linear dataset, the best C value determine through a 5-fold cross validation was $C = 0.01$ and it had a multiclass AUC ROC score of around 0.936. For the step dataset, the best C value also seemed to be $C = 0.01$ but had a higher AUC ROC score of around 0.943. We notice that having a small regularization parameter seemed to perform best with the logistical models. The plots showing the performance of various C values are shown below.

As noted in the previous section, two models were fit for each dataset: a model using the normal imbalanced dataset

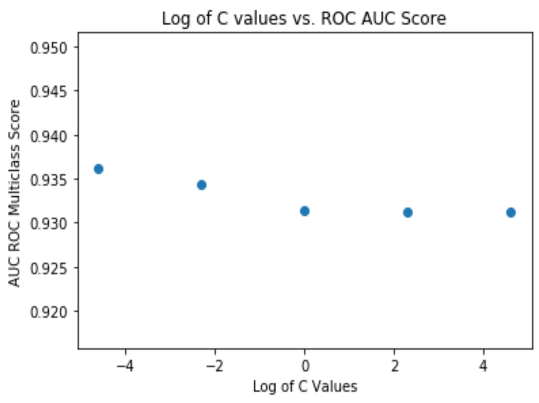

Fig. 1. AUC ROC Scores for varying C values for the linear dataset. The smallest C values seems to have to best overall performance

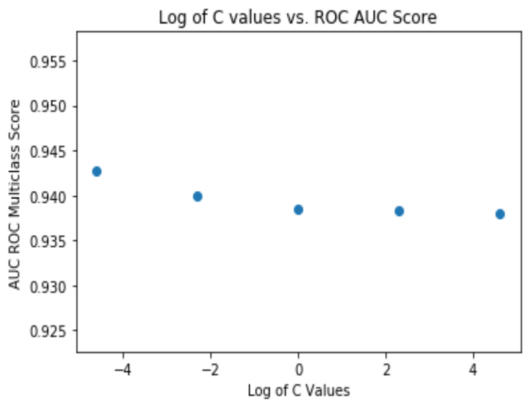

Fig. 2. AUC ROC Scores for varying C values for the step dataset. The smallest C values seems to have to best overall performance

and another dataset that was generated through sampling using *imblearn*[1] through undersampling. This gives a total of four distinct logistical models.

For the linear dataset, the imbalanced model gave an AUC ROC score of 0.9472 and the undersampled linear dataset gave a model that had an AUC ROC score of 0.9484. The step dataset seemed to give slightly better performances as the imbalanced model gave a score of 0.9521 and the undersampled step dataset model gave an AUC ROC score of 0.9523. Overall, the logistical model did a good job of predicting the digits on the test set but the step dataset seemed to have better performances. This can mainly be attributed to the fact that the step dataset is slightly larger than the linear dataset.

B. Random Forest

Once again, a 5-fold cross validation was performed to determine the best number of estimators within our random forest classifier as well as the best criteria to partition the space with (Gini Index vs. Entropy loss). Figure 3 and 4 show the results of performing cross validation on the linear and step datasets.

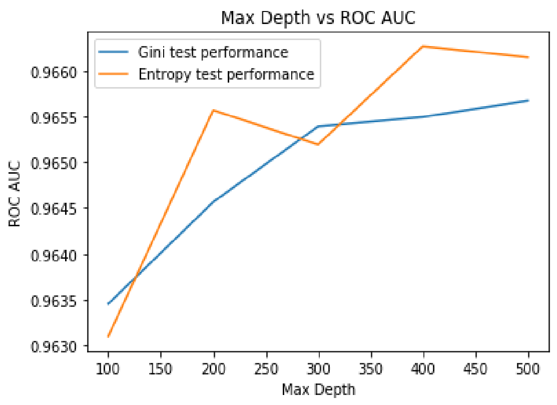

Fig. 3. Cross Validation results for Random Forest on the Linear dataset

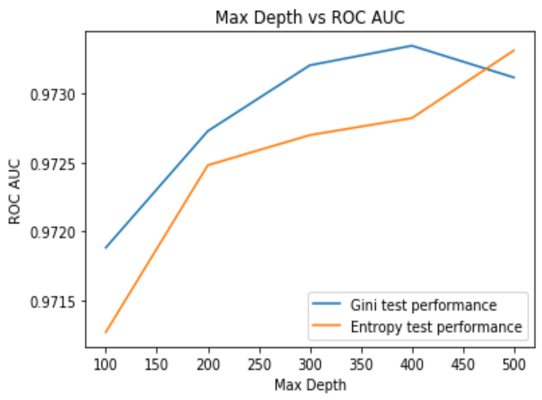

Fig. 4. Cross Validation results for Random Forest on the Step dataset

The Random Forest classifiers all did a very good job of handling the MNIST dataset and predicting on the test set. For the linear dataset, the entropy criterion was used and the imbalanced model had an AUC ROC score of 0.9730 while the balanced model had a performance of 0.9714. For the step dataset, the imbalanced model gave an AUC ROC score of 0.9792 and the balanced model gave a performance score of 0.9795. Overall, all of these models did a very good job of predicting but did take a lot of computational power.

C. Convolution Neural Network

No cross validation was performed on the Convolutional Neural Network. However, stochastic batch gradient descent was implemented and the train losses for the imbalanced datasets are shown in figure 5 and 6.

The linear imbalanced CNN model gave an AUC ROC score of 0.9903 and the linear balanced model gave a score of 0.9930. The step imbalanced CNN model performed with a score of 0.9917 while the balanced step CNN model had

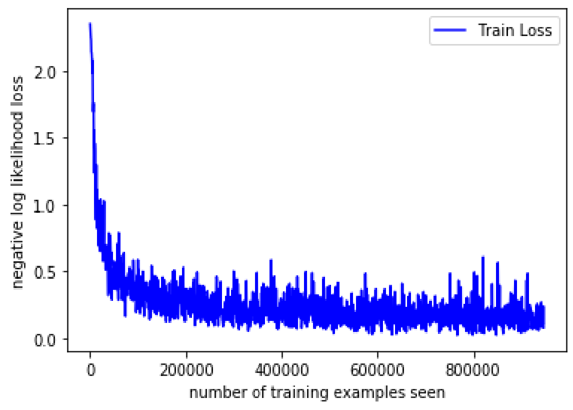

Fig. 5. Training losses during gradient descent for the linear imbalanced CNN model

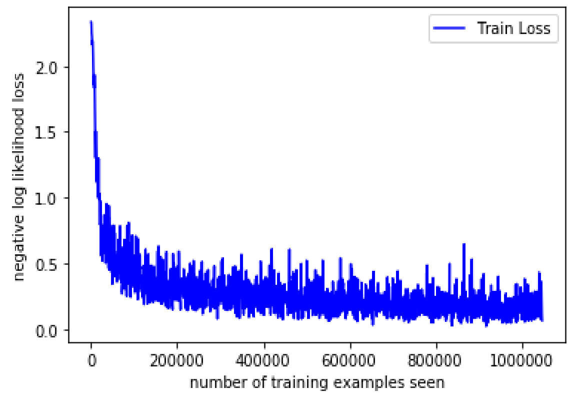

Fig. 6. Training losses during gradient descent for the step imbalanced CNN model

an AUC ROC score of 0.9932. The CNN models performed drastically better than the other two provided models.

D. Sampling Methods

Two different sampling methods were used within this paper to battle against the imbalanced datasets that were provided. First, the *imblearn* module was used for the logistical and random forest classifiers. Specifically, the *RandomUnderSampler* was used to balance out the dataset. Overall, these samplers seemed to have improvements to the models, though the improvements weren't huge. However, this algorithm was a simple randomized algorithm for resampling which gives hope for other more sophisticated algorithms.

For the CNN model, a different module was used to perform sampling methods. The *ImbalancedDatasetSampler* from *torchsampler*[6] was used. This module did basically the same thing that the *RandomUnderSampler* did in *imblearn*[5] but performs these operations for the data loader needed for the CNN model using *pytorch*. Sampling also improved the performance of the CNN models.

Overall, sampling definitely helped improve the performance of the models but it seemed to help the CNN models the best. The Random Forest Classifiers already handled imbalanced data very well, therefore adding sampling methods didn't help very much on the testing set. The Logistical models also had some small improvements but the improvements weren't very big.

IV. Comparison of Techniques

Now we want to analyze the models and techniques used in this paper and compare the performance of them. We noticed that the CNN model outperformed the other two models. Since the CNN is the most complex model mathematically speaking, it definitely had advantages to performance. Note that we also only used 3 convolutional layers along with 2 fully connected layers and more complex models could be used to generate better performances.

The Random Forest Classifier is already known to handle data imbalance very well since it can generalize well and handle edge cases. Therefore, the sampling techniques didn't have that much of a difference on the test performance. However, the Random Forest Classifiers were very computationally heavy and took up a significantly larger amount of memory compared to the other two models. Training and cross validating the Random Forest classifiers took much longer compared to the CNN or logistical model despite the interpretabillity of the Random Forest classifier.

The logistical model is the simplist of the three models and is also the fastest to train out of the three. However, with the increase in efficiency and time complexity, it compromises in performance. It easily has the lowest AUC ROC score out of all the models. This is mainly due to the simplicity of the model as it is essentially a linear model and not much non-linearity is introduced into the model.

The sampling methods definitely improved performance for most of the models but didn't have a drastic improvement. The models were originally doing decently and the sampling methods simply fine-tuned some of the features for some of the models. Sampling techniques also seem to be bery computationally expensive for extremely large datasets. Undersampling can help the performance but oversampling for large datasets take very long and the performance increase compared to undersampling may not be worth it.

V. Conclusions

Imbalanced data within computer vision can definitely decrease the overall performance of models and sometimes produce ineffective predictors. However, methods such as resampling the data can help increase the performance of these models. Also some machine learning models can still perform computer vision tasks well despite the issues regarding imbalanced data. The sampling methods introduced in this paper seem to have potential with improving the performance of models with imbalanced data. With access to better hardware and machines, some more sophisticated sampling and data generation methods could be implemented and produce even better results than the randomized samplers used in this paper.

Methods and techniques that can deal with imbalanced data could go a long way for computer vision models. The collection of perfectly balanced data for computer vision problems usually are not possible, and having methods to train very powerful and complex models using the imbalanced data could be applied to many computer vision applications. Issues such as tumor detection could benefit greatly from methods like these. Usually, data is very imbalanced as most people don't have malignant tumors, but being able to deal with imbalanced data could allow models to accurately determine if a tumor is present given an MRI image or Xray. A model that can predict the presense of a tumor can help doctors quickly determine the diagnosis of a potential cancer patient without using too many resources or performing too many tests (some patients may be sensitive to some testing methods). The applications of methods that deal with imbalanced datasets are endless. These techniques could be applied to applications such as autonomous driving, medical imaging, facial recognition, and many more. The issues dealt here with computer vision problems could even be extended to other fields of AI such as natural language processing, robotics, economics and many others.

References

[1] L. Abdi, S. Hashemi, et al. "Learning from Imbalanced Data: Open Challenges and Future Directions." Progress in Artificial Intelligence, Springer Berlin Heidelberg, 1 Jan. 1970, link.springer.com/article/10.1007/s13748-016-0094-0?ref=hackernoon.com.

[2] Chen, Feiyang, et al. Assessing Four Neural Networks on Handwritten Digit Recognition Dataset (MNIST). June 2018, arxiv.org/pdf/1811.08278.pdf.

[3] Japkowicz, Nathalie. "Learning from Imbalanced Data Sets: A Comparison of Various Strategies." Arvix, 2000, www.aaai.org/Papers/Workshops/2000/WS-00-05/WS00-05-003.pdf.

[4] Koehler, Gregor. "MNIST Handwritten Digit Recognition in PyTorch." Nextjournal, 17 Feb. 2020, nextjournal.com/gkoehler/pytorch-mnist.

[5] Nogueira, Fernado. "Learn API." Imblearn, 2016, imbalanced-learn.readthedocs.io/en/stable/api.html.

[6] Ufoym, Ming. "Ufoym/Imbalanced-Dataset-Sampler." GitHub, 2018, github.com/ufoym/imbalanced-dataset-sampler.