Mphasis
The Next Applied

Using Automated Machine Learning to Detect and Mitigate Demographic-oriented Biases

Whitepaper by Prashanth Arun, Data Science Intern at Mphasis | Kaushlesh Kumar, AVP - Applied AI, Mphasis NEXT Labs

Contents

1.
Introduction

Machine Learning (ML) is steadily gaining popularity in the world across industries, governments and academia. Organizations leverage ML to make financial recommendations and hiring decisions. Legal entities, such as courts of law, use ML algorithms to decide which criminals have a high chance of committing a crime within a few years of their release. In the medical world, it identifies whether a growth is cancerous or not. Realtors use it to predict house prices, and search engines to identify user preferences and behaviors.

As ubiquitous as ML is becoming across the world, we must ask ourselves an important question. How much are ML-based services really improving our lives? Do ML models go beyond the human veil of cognitive biases and prejudices to treat every person – regardless of their gender, race, religion or any other attribute – fairly and equally? Over the last few years, it has been shown that multiple ML services have demonstrated biases against at least one demographic. For example, Amazon's ML software that aided in hiring new applicants was found to be biased against female applicants (Dastin, 2018). In the same year, many facial recognition algorithms – notably those from IBM and Microsoft – were found to display racial and gender biases (Najibi, 2020).

What makes ML models biased? Trained on real-world data generated or tagged by humans, each of whom have their own biases that interfere with their ability to make rational and fair decisions, it is not surprising that such biases are passed on to ML models. Biased data, if provided during training, induces the model to see them as significant, resulting in 'unjust discrimination' (Kim et al., 2019). Hence, the idea of Responsible AI was introduced. One of the main goals of Responsible AI is to make ML and Artificial Intelligence (AI) services fair to all demographics. To quote a paper by Peters et al. (2020), *"it is no longer acceptable for technology to be released into the world blindly, leaving others to deal with the consequences".*

In this paper, we integrate fairness metrics with an AutoML pipeline and use different ML packages to detect biases against a particular demographic in a dataset. Finally, a mitigation algorithm would be introduced to mitigate the demonstrated biases.

2.
The Methodology

This section focuses on project-specific details, including information on the dataset used, the steps followed and the fairness metrics (fairness scores) deployed. The fairness metrics and mitigation algorithm, provided by the Fairlearn package by Microsoft. AutoML (Automated Machine Learning) is a design idea that aims to (ideally) automate most of the steps of the ML process. ML problems can be broadly divided into four categories - (1) data collection and preprocessing, (2) model training, (3) hyperparameter tuning and (4) testing and deployment.

Data preprocessing is the step through which data is sorted into its necessary features, while discarding what is not needed. The model training step is self-explanatory: a model is trained on the data, using one of many different statistical and logical approaches. Hyperparameter tuning refers to using techniques to improve the performance of the model for a particular score. Testing and deployment refer to the validation step needed before the model can be released for commercial use.

Currently, AutoML covers the model training and hyperparameter tuning aspects of ML problems. Two Python-based AutoML packages were used for the project: PyCaret and TPOT.

Dataset

COMPAS dataset used for this project, which contains information about a sample of prisoners in the US. This dataset illustrates a racial bias in the American Judicial System. 6172 samples were present in the dataset, out of which 4069 were used in this project (the remaining data instances were discarded due to one or more missing fields). **[Source: kaggle.com]**

Column Name	Attribute Type
Two_yr_Recidivism	Binary
Number_of_Priors	Numeric (int64)
Score_factor	Binary
Age_Above_FortyFive	Binary
Age_Below_TwentyFive	Binary
Female	Binary
Misdemeanor	Binary
Race	Categorical

Table 1: A summary of the columns in the dataset, as well as their attribute types, and the data types of each element in that column.

The target column was the Two_yr_Recidivism of an individual, an indication of whether a person on parole committed a punishable offence within two years of his/her release. The race of the individual under consideration was termed as the sensitive attribute. A target value of 1 means that the person is predicted by the model to commit a crime within two years, while a value of 0 denotes the opposite.

Fairness Metrics

It is important to decide on how a model's fairness will be measured. Three fairness metrics were used in the experiment - demographic parity difference, demographic parity ratio and equalized odds ratio. Note that the terms *parity score* and *fairness score* are used interchangeably as a substitute for these metric names in the next two sections.

In all three cases, the predicted values are provided along with the corresponding sensitive attribute data. An underlying metric known as the *selection rate* is used to calculate all of these scores. The selection rate is the proportion of data instances from *any one* demographic that had a predicted value of 1 (in the context of this experiment, whether a person will commit a crime in two years).

The demographic parity difference is calculated as the difference between the highest selection rate and the lowest selection rate (note that each demographic of the sensitive attribute has an associated selection rate). The idea range for this score is between 0 and 0.2.

The demographic parity ratio is the ratio of the lowest selection rate to the highest selection rate and all of its values are in the interval [0, 1]. Here, a value of 0.8 or higher is considered acceptable.

The equalized odds ratio measures the degree to which the algorithm issues the same proportion of predictions to each demographic (true positives, false positives, true negatives and false negatives). A value higher than 0.8 is considered acceptable.

Steps Followed

The first step is to investigate or, in this case, verify that there is indeed a bias against one or more demographics in the dataset. Going straight to the model training and mitigation steps, when there is no bias will only be a waste of computational resources. This can be accomplished by first integrating one or more fairness metrics with an (Auto)ML pipeline and setting a threshold value below (or above) which it can be concluded that the dataset shows a bias. A model (or set of models) is/are trained on the data and the results are noted.

Since AutoML pipelines make use of cross-validation and ensemble techniques such as bagging and boosting, the entire training data is not used at a single stretch to train the model. Most scoring functions only require the predicted and true target values to calculate a score, so no matter how data is sampled during training, this does not pose a problem. Fairness functions, on the other hand, take the sensitive attribute data as well, so that they can make calculations for each demographic. Hence, integrating a fairness function is slightly more complicated. The solution is a wrapper function whose main aim is to ensure that the values from all three main parameters line up before it can be passed into the scoring function.

If the dataset shows a bias, then we remove the sensitive attribute from the data and retrain the model(s). After all, it follows logically that if we exclude the source of the bias from the dataset, then the model will be less biased. If the fairness score(s) do(es) not follow in the desired range, then the model is tuned to improve one or more of these metric scores. Finally, if the fairness score is still not satisfactory, we pass the model through a mitigation algorithm, in an attempt to reduce the biases shown by the model to a larger extent.

3. Producing a Fairness-optimized Model in TPOT

TPOT (Tree-based Pipeline Optimization Tool) is an AutoML package that makes heavy use of genetic programming algorithms to optimize ML pipelines (Olson & Moore, 2019). In this section, we will look briefly at how TPOT works, and whether it is suitable for the specific problem at hand.

The fundamental principle behind TPOT, as mentioned above, is genetic programming. Research by Dafflon et al. (2020) shows that this process is a “multi-generation approach”, starting with a collection of random models. At every stage, the best-performing models are “bred” (i.e., subject to crossover), and mutations are introduced to some of these models. Olson & Moore (2019) describe the initial machine learning pipeline operators (GP primitives) to relate directly to the algorithms found in the Python package scikit-learn, with the exception of XGBoost. They go on to detail it at every generation, and models that are detrimental to the classification accuracy are removed, while pipeline operators that improve the metric score are kept. At the end of 100 generations, the pipeline with the best metric score is selected.

Initial Experiments and Observations Using TPOT

Training a classification model in TPOT using common scoring metrics is very straightforward. Given that the data was already encoded (apart from Race, which was removed from the dataset prior to training), the only required procedures were to read in the data and split it into the training and test data. For this experiment, a non-stratified 80/20 split was used to train the models.

Initial observations with TPOT were not satisfactory. The resulting predictions were only 0s (i.e., *no* prisoner is expected to relapse within 2 years) or only 1s (i.e., *every* prisoner will relapse within 2 years). To investigate this, 9 models were trained, each for a different time, to determine if there was a relationship between the training time (in minutes) and the parity score. A graph of the results is shown below:

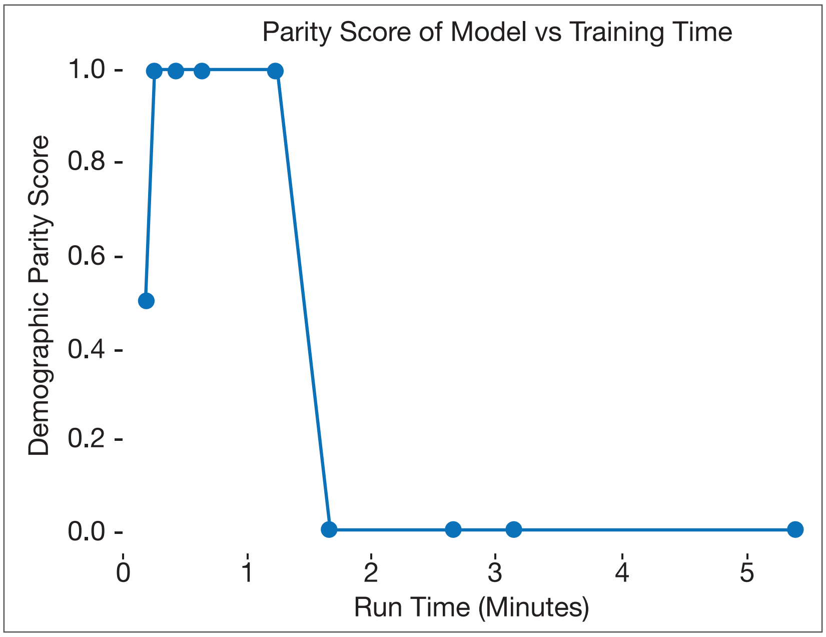


Figure 1: Graph showing the parity score of the model based on how long it was trained for. Models that were run for 30 minutes showed the same result (a demographic parity score of either 0 or 1).

It was necessary to determine, visually, how the algorithm optimized a particular metric. Scoring functions for parity, accuracy, precision and recall were written such that for each function call (i.e., for each model at the end of a generation) the corresponding score could be calculated. A scatterplot of the results (metric score vs iteration number) is shown below:

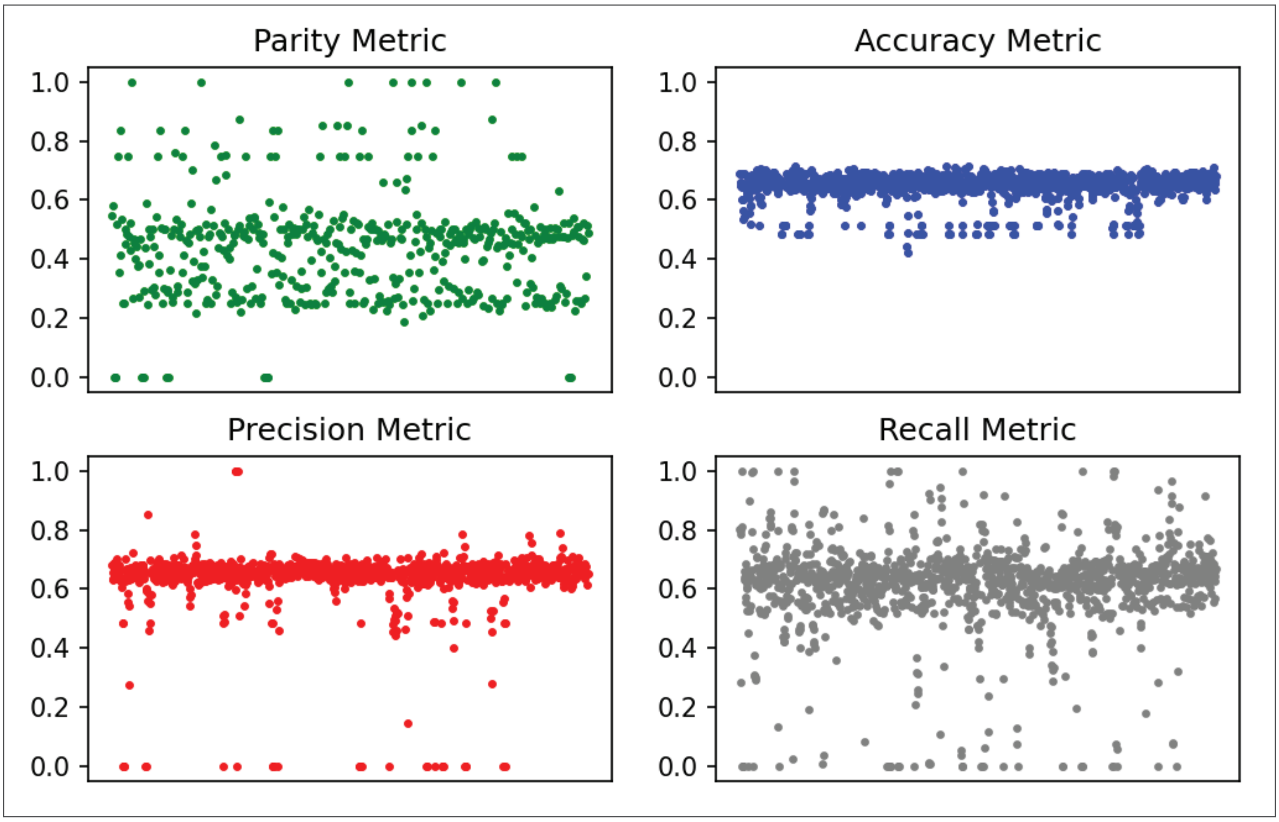


Figure 2: The score of each model plotted against the "i"th time the function was called. Each model was trained for 4 minutes.

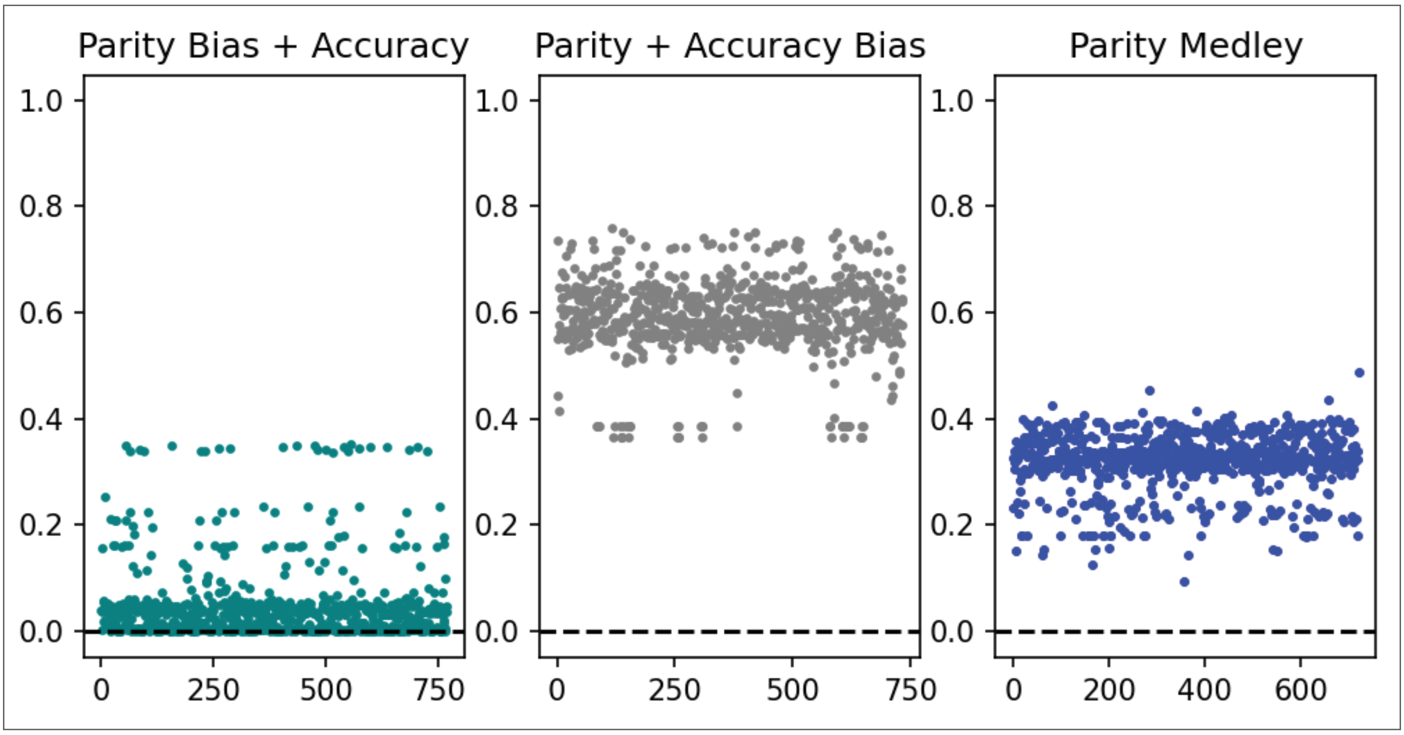
We notice that for the accuracy and precision metrics, the values converge to a small range very quickly. For the recall metric, a single band of values is visible. By contrast, there are two distinct bands visible for the parity metric (at 0.5 and ~0.2), and the scores are much more dispersed. Based on the workings of TPOT described in Section 3.1, this shows that, at the end of a generation, the algorithm may not be able to select the required number of effective pipelines to pass on to the next generation. As a result, it is not able to make reliable predictions for a given data instance.

TPOT – A Modified Approach

Given that the algorithm was unable to give reliable results for the parity metric alone, it was hypothesized that the algorithm would be able to select effective pipelines with a lower dispersion in metric scores, if the demographic parity difference metric were to be combined with at least one other metric that the algorithm is able to optimize.

Six custom metrics were chosen in pursuit of this hypothesis, and a detailed breakup of each of their components can be found in the appendix (table A(i)). A similar procedure to the one above was followed and a scatter graph was plotted to show how the six custom metrics behaved with regard to the dispersion in scores.

The results for this process showed significant improvement from the initial results. The model which used the combination of parity and precision showed a parity difference of 0.75. While this is nowhere close to the ideal parity difference for an optimized model (which would typically be in the range 0 – 0.1), it is not also a boundary value (0 or 1). The best accuracy was, as predicted, achieved by the model which weighed accuracy at 75% and parity at 25%.



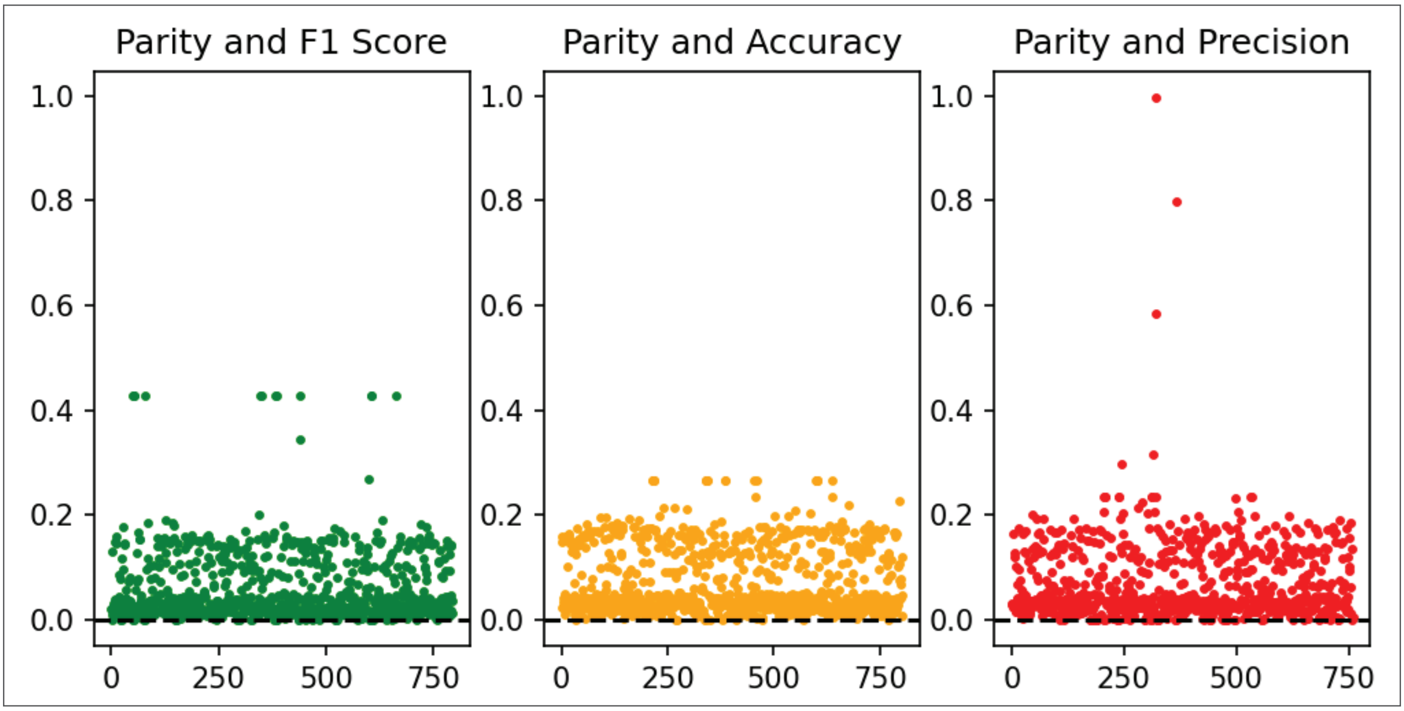


Figure 3 (Previous Page and Above): The scatterplot for the metric scores against the iteration number during the training process. Note that all values are much more localized when compared to the parity metric on its own.

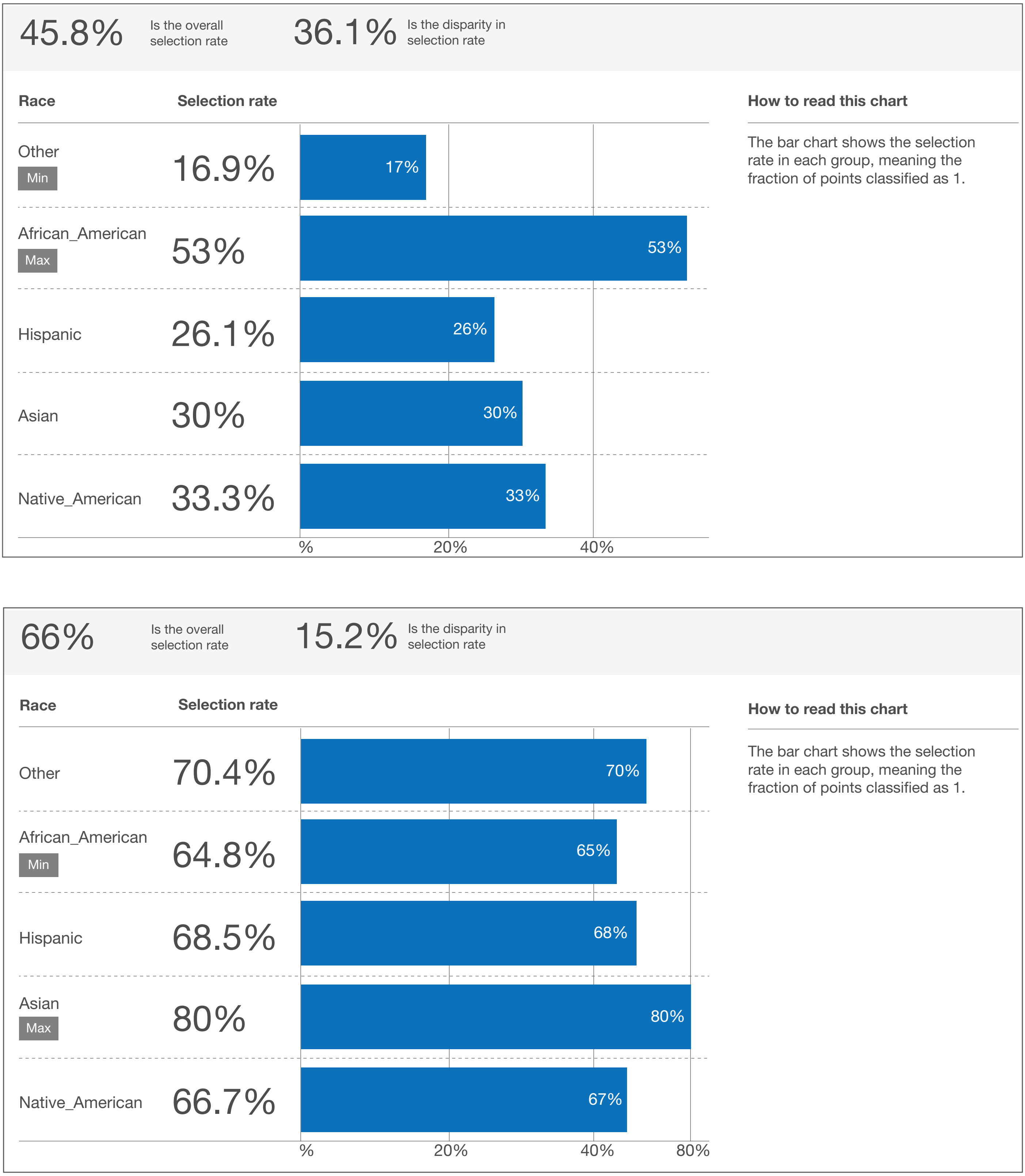
4.

Producing a Fairness-optimized Model Using PyCaret

PyCaret is a versatile, user-friendly ML package that also incorporates AutoML. In this section, we will look at how PyCaret was used in conjunction with a fairness metric to produce fairness-optimized models. PyCaret uses a total of 15 models for supervised learning that include LightGBM, CatBoost and XGBoost classifiers, as well as many classifiers found in scikit-learn. A setup process (which can be viewed in its entirety on *pycaret.org*) is required before training the model.

The scoring function used for this solution utilizes the same wrapper described in Section 2.3. PyCaret generates a table of metric scores once a model has been trained. Adding the fairness metric to this list was straightforward. During the implementation process, different combinations of metrics were experimented with, similar to what was described in the earlier section. The bulk of the total training time for this solution is concentrated in the initial training stage, when all 15 models are trained on the same data. The best AutoML pipeline for a particular metric can be obtained through a final function call. Given that more than one fairness metric was used, it was possible to tune all three separately to improve the model's fairness even more.

Bias detection trials using PyCaret were encouraging, and so the final model was passed through a mitigation algorithm to investigate the extent to which biases in the model could be reduced. The mitigation algorithm used was an exponentiated gradient, the principles of which are described by Agarwal et al. (2018), though similar results were obtained using a GridSearch mitigation algorithm. The results for the disparity in selection rates for the unmitigated and mitigated models are given in Figure 4.


45.8% Is the overall selection rate

36.1% Is the disparity in selection rate

How to read this chart

The bar chart shows the selection rate in each group, meaning the fraction of points classified as 1.

66% Is the overall selection rate

15.2% Is the disparity in selection rate

How to read this chart

The bar chart shows the selection rate in each group, meaning the fraction of points classified as 1.

Figure 4: The mitigated model (lower) – note the drop in disparity among the different demographics when compared to the unmitigated model (upper).

PyCaret vs TPOT

We will view the differences between PyCaret and TPOT – specifically, the training times for each, their performance and how well they trained models using a fairness score. Additionally, we will look at the possible avenues that can be explored to improve the performance of this solution. Here, we will compare the two packages from three different viewpoints: the training times required (on average) to produce optimized models, the performance of the models themselves and their performance after integrating the parity metric.

Training Time: There is a clear difference in the training times required to generate models in TPOT and PyCaret, with TPOT classifiers requiring much more time (to the tune of hours) to train well-performing models. That being said, TPOT classifiers produce acceptable models that closely rival the metric scores found in PyCaret after being trained for only 3-4 minutes. PyCaret, however, performs this job much quicker, and it is possible to obtain a pipeline in a matter of seconds.

Performance: On an average, pipelines returned by TPOT have higher metric scores than those returned by PyCaret. The graph below shows the comparison between TPOT and PyCaret for accuracy, precision and recall metrics. In this instance, the TPOT models were trained for 3 minutes each. Recall the research by Olson and Moore (2019), which stated that the algorithm behind the TPOT classifier keeps the pipeline operators that improve or maintain the metric score. TPOT only optimizes one metric score for a classifier (which is why three separate models were trained here), while PyCaret allows the user the option of tuning the model to optimize any metric score.

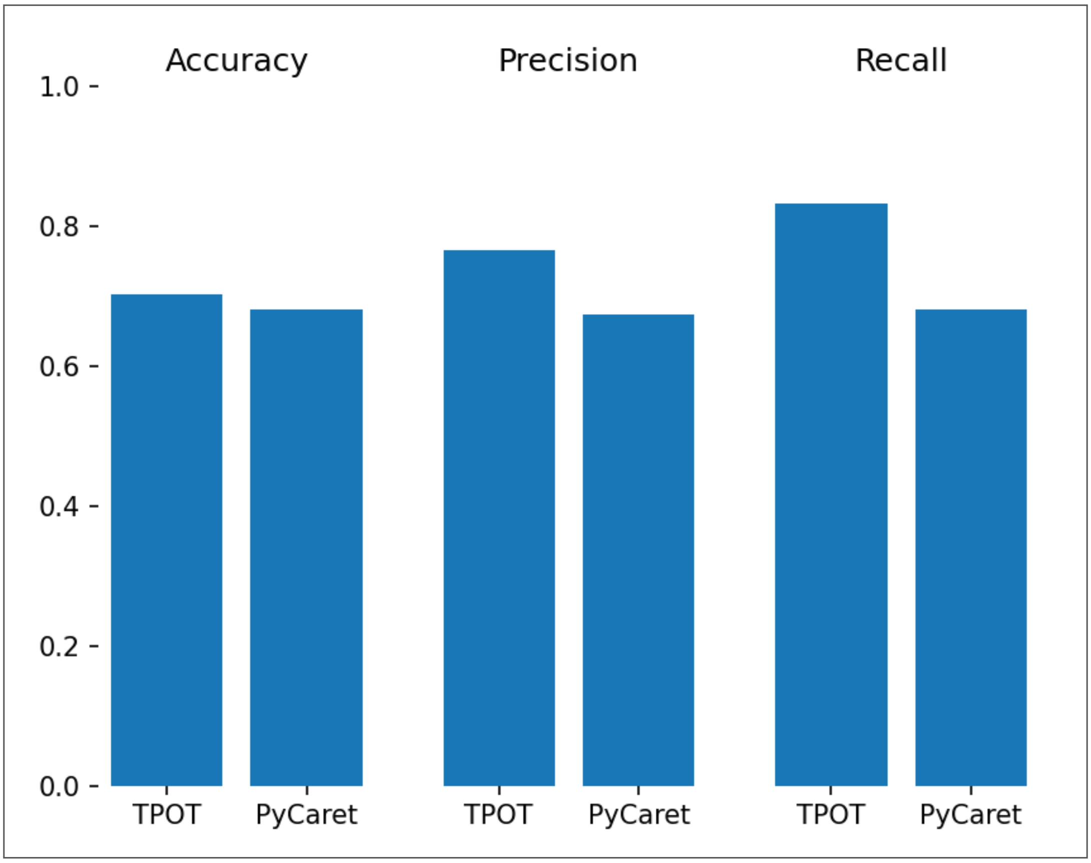


Figure 5: Note that TPOT shows better scores for each of the metrics compared to PyCaret.

Performance with a Parity Metric: Here, PyCaret comes out far ahead of TPOT. Using PyCaret, a fairness-optimized AutoML pipeline was returned, which could then be passed into a mitigation algorithm. By contrast, TPOT produced unreliable results, even after combining the parity metric with other scoring functions.

5.
Conclusion and Recommendations

We integrated fairness metrics with an AutoML pipeline and passed it through a mitigation algorithm to reduce the biases against a specific demographic. This solution can be improved by exploring new techniques for bias detection and mitigation. Approaching this problem from a Deep Learning perspective is a feasible and encouraging course of action. We established that removing the sensitive attribute alone from the dataset does not impact the fairness of the model to a large extent. This suggests that there are features in the dataset that act as proxy sources of bias (i.e., they do not explicitly contain the sensitive attribute, but are so closely related to it that they reflect the same bias). Investigations on how to detect and mitigate these proxy sources will be very effective in improving the fairness of a model.

6.
References

1. D. Peters, K. Vold, D. Robinson and R. A. Calvo, "Responsible AI—Two Frameworks for Ethical Design Practice," in IEEE Transactions on Technology and Society, vol. 1, no. 1, pp. 34-47, March 2020, doi: 10.1109/TTS.2020.2974991
2. Jessica Dafflon, Walter H. L. Pinaya, Federico Turkheimer, James H. Cole, Robert Leech, Mathew A. Harris, Simon R. Cox, Heather C. Whalley, Andrew M. McIntosh, Peter J. Hellyer, “An automated machine learning approach to predict brain age from cortical anatomical measures” (2020)
3. R. S. Olson; J. H. Moore, “TPOT: A Tree-Based Optimization Tool for Automating Machine Learning”, in F. Hutter et. al. (eds.) Automated Machine Learning, Springer, 2019
4. J. Dastin, “Amazon scraps secret AI recruiting tool that showed bias against women”, Reuters, 2018
5. A. Najibi, “Racial Discrimination in Face Recognition Technology”, Harvard SITN, 2020
6. A. Agarwal, A. Beygelzimer, M. Dudík, J. Langford, H. Wallach, “A Reductions Approach to Fair Classification”, 2018
7. Byungju Kim, Hyunwoo Kim, Kyungsu Kim, Sungjin Kim, Junmo Kim, "Learning Not to Learn: Training Deep Neural Networks With Biased Data", in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019
8. Article on Amazon’s Gender-Biased Model: https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G
9. Article on Facial Recognition Software Biases: https://sitn.hms.harvard.edu/flash/2020/racial-discrimination-in-face-recognition-technology/

Authors

Prashanth Arun

Data Science Intern at Mphasis

As a Data Science intern at Mphasis NEXT Labs, Prashanth completed projects revolving around Responsible AI. He is currently in the second year of his undergraduate studies at the University of Waterloo in Canada, majoring in Computer Science and minoring in German.

Kaushlesh Kumar

AVP - Applied AI, Mphasis NEXT Labs

In his career spanning more than 13 years, he has helped solve business problems for clients using innovative and structured approaches. In his current role, Kaushlesh designs, develops and executes solutions for transforming business services leveraging machine learning, data science and process. These initiatives deliver super normal value to the clients. He has a PG Diploma in Management and is a graduate engineer.

About Mphasis

Mphasis (BSE: 526299; NSE: MPHASIS) applies next-generation technology to help enterprises transform businesses globally. Customer centricity is foundational to Mphasis and is reflected in the Mphasis' Front2Back™ Transformation approach. Front2Back™ uses the exponential power of cloud and cognitive to provide hyper-personalized ($C = X2C^{2}_{TM} = 1$) digital experience to clients and their end customers. Mphasis' Service Transformation approach helps 'shrink the core' through the application of digital technologies across legacy environments within an enterprise, enabling businesses to stay ahead in a changing world. Mphasis' core reference architectures and tools, speed and innovation with domain expertise and specialization are key to building strong relationships with marquee clients. To know more, please visit www.mphasis.com

For more information, contact: marketinginfo.m@mphasis.com

USA
460 Park Avenue South
Suite #1101
New York, NY 10016, USA
Tel.: +1 212 686 6655

UK
1 Ropemaker Street, London
EC2Y 9HT, United Kingdom
T : +44 020 7153 1327

INDIA
Bagmane World Technology Center
Marathahalli Ring Road
Doddanakundhi Village
Mahadevapura
Bangalore 560 048, India
Tel.: +91 80 3352 5000

NR 05/08/21 US LETTER BASIL6979

www.mphasis.com