Capstone Project - COMPAS Fair Classifier

Problem Definition

In May 2016, ProPublica published a report regarding a judicial decision support algorithm that outputs a risk score for defendant redicivism (Angwin et al., 2016). This algorithm is called COMPAS, short for *Correctional Offender Management Profiling for Alternative Sanctions*. COMPAS has already assessed the risk of more than 1 million defendants since its inception in 1998. It has been shown that COMPAS is extremely biased toward african-american defendants when compared to caucasian offenders for the same prior/post offenses. For example, the false positive rate for medium/high risk of recidivisim is 40.4% for african-american defendants and 25.4% for caucasians defendants (Dressel & Farid, 2018).

With this in mind, and also the recently happenings around the death of George Floyd by Minnesota white police officers, the main objective of this capstone project is to train, tune and test a binary fair classifier using COMPAS data. By fair, my intentions are that race should not be a marker for medium/high recidivism risk.

Analysis

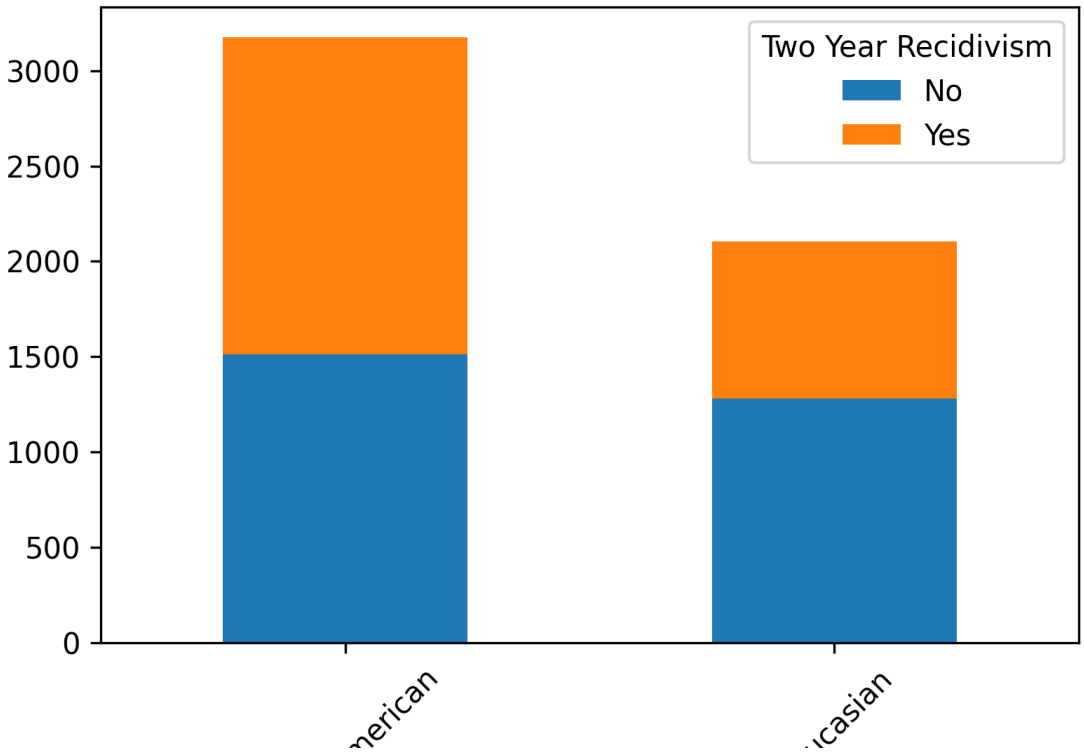
What follows are the calculations of the COMPAS Recidivism Risk Scores. These calculations are based on ProPublica's analysis methodology (Larson et al., 2016).

First, following ProPublica's analysis, I've selected only the features of everity of charge, number of priors, demographics, age, sex, compas scores, and whether each person was accused of a crime within two years. Also, I've filtered out instances where the days between screening and arrest are over 30.

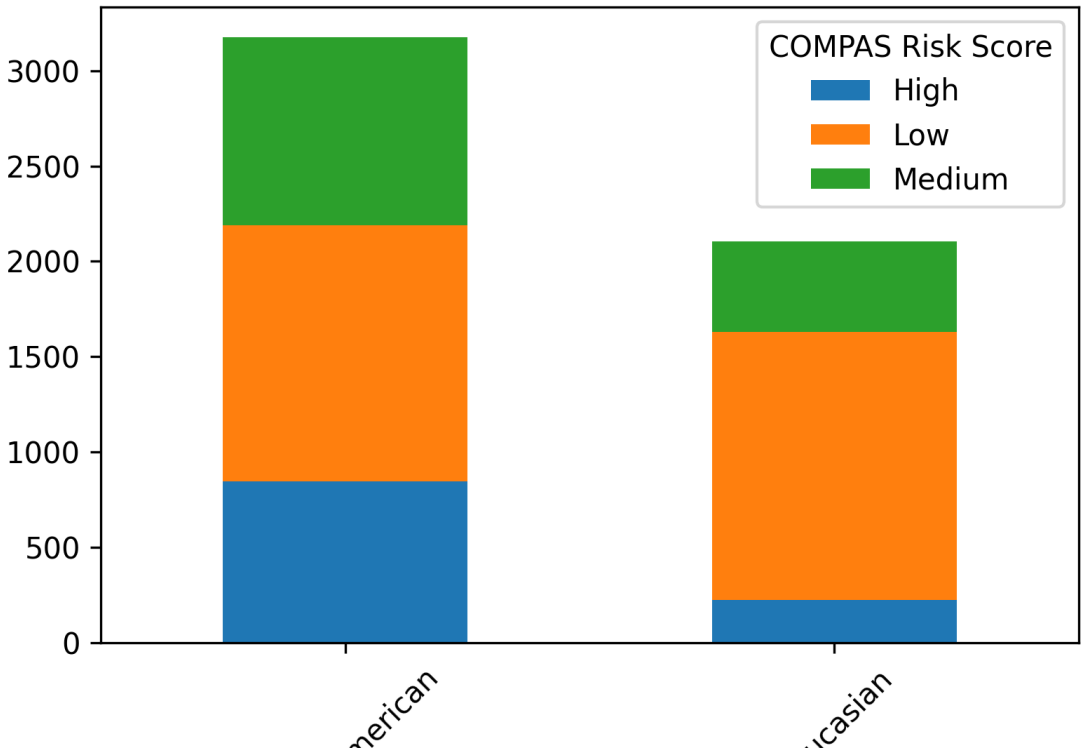
Second, following Chouldechova (2017), I've also filtered only the instances that the attribute race is either `African-American` or `Causasian`. Ultimately, the dataset had 5,278 instances.

Race Analysis

I've made some cross-tabulations and visualizations regarding some features by race. Below, we can see that, for our data, africa-american have more two year recidivism than caucasians.



Also, african-americans are predicted more as medium/high risk than caucasians.



Implementation

Data Splitting

Since, I've used a hyperparameter tuning procedure, I've split the data into 3 constituents: training, validation and test data. Training data was used to train the model. Validation data was used to validate by tuning the best model hyperparameters. Test data was used to assess the model performance on unseen data (out-of-bag - OOB predictions). The ratio used was 50/20/30 respectively for train/validation/test. The data was saved as a `.csv` file without header and index. Also, for training and validation data, the first entry in each row was the target variable. This is the standard approach that Amazon algorithms assumes when reading these kind of data.

Data Upload to S3

Inside a `SageMaker` notebook and using the `boto3` and `sagemaker` python libraries, I've uploaded all files that were inside a data directory folder containing all `.csv` files to the default S3 bucket for the sagemaker session. This is important because all of the following training, hyperparameter tuning and batch transform jobs will need to read the data from this S3 bucket.

Algorithm

The algorithm that I've chose was the `XGBoost`. There are two main reasons. The first, `XGBoost`, being a tree-based algorithm, can tolerate non-parametric and non-linearity quite robustly. Second, `XGBoost` allows for different objective metrics, instead of the default accuracy in a binary classifier context. The metric that I was interested was mean average precision in order to control and optimize for lower false positive rates. I do not want that the model to label someone (either african-american or caucasian) as medium/high risk for recidivism if his/hers ground truth is low risk.

Creating the Estimator

In order to train the `XGBoost` algorithm, first I've created an `estimator` object using Amazon's `XGBoost` uri. This `estimator` object had the container name, IAM role, 5 instances of `ml.m4.xlarge` type, and an S3 output path for the results.

After the `estimator` was created, I've set it's hyperparatemers by calling the method `.set_hyperparameters()`. The most important hyperparameter that I've used differently from the `XGBoost`'s default is the `objective` argument. Instead of using the default (`'reg:squarederror'`), I've used one for binary classification (`'reg:logistic'`).

Hyperparameter Tuning

With the `estimator` object completely set up, the next step was to create `hyperparameter tuner` object. To do this I've constructed a new object which contains each of the parameters I want SageMaker to tune. In this case, I wish to find the best values for the `max_depth`, `eta`, `min_child_weight`, `subsample`, and `gamma` parameters. Note that for each parameter that I want SageMaker to tune I've to specified both the *type* of the parameter and the *range* of values that parameter may take on. The most important arguments that were passed to this `hyperparameter tuner` object was `objetive_metric_name` being `'validation:map'` and `objective_type` being `'Maximize'`. This tells SageMaker that I want to find hyperparameters that maximize the mean average precision.

In addition, I specify the number of models to construct (`max_jobs`) and the number of those that can be trained in parallel (`max_parallel_jobs`). I've' chosen to train 20 models, of which I ask that SageMaker train 5 at a time in parallel.

Running the Hyperparameter Tuning Jobs

Now that I have my `hyperparameter tuner` object completely set up, it is time to train it. To do this I make sure that SageMaker knows my input data is in `.csv` format and then execute the `.fit()` method. This creates the hyperparameter tuning jobs. I can monitor the jobs if I go to the `AWS Console` and down to `Amazon SageMaker`, you shall see under `Training > Training jobs` tab. Or I can use the `.wait()` method.

Once the hyperamater tuner has finished, I've retrieved information about the best performing model by using the `.best_training_job()` method on the `hyperparameter tuner` object. Finally, to have the best performing model attached to a trained estimator, I've created a new estimator object called `xgb_attached` by assigning it to the `sagemaker.estimator.Estimator.attach(xgb_hyperparameter_tuner.best_training_job())`.

Evaluating the Model

Now that I have my best performing model, I can test it. To do this I've use the `batch transform` functionality. To start with, I need to build a `transformer` object from my fitted model. I do this by calling the `.transformed()` method on a trained estimator (in my case `xgb_attached`). I also specify the instance type and count that I want this batch transform to run on. I've chosen a single `ml.m4.xlarge` instance. The batch transform job will read the test data from the sagemaker session's default S3 bucket. To monitor the batch transform progression, I can monitor the `Batch transform jobs` tab under `Inference` in the `Amazon SageMaker` or I can use the `.wait()` method on a `transformer` object.

To assess the model, I've imported `Scikit-Learn`'s `accuracy_score`, `precision_score` and `confusion_matrix`. I've passed to each of those metrics the ground truth from the test data and also the predicted class from the `transformer` object's output.

Results

To my surprise, the hyperparameter tuned `XGBoost` performed marvellously. It achieved a whooping 100% accuracy (and consequently 100% precision) on the test set. Below, I show the original COMPAS confusion matrix and my model's confusion matrix.

Original COMPAS (Dressel & Farid, 2018)

Accuracy: 60.6%

	African-American	**Caucasian**
False Positive Rate	40.4%	25.4%

Proposed Model

Accuracy: 100%

	African-American	Caucasian
False Positive Rate	0%	0%

Conclusions

My conclusions are:

- SageMaker is a very simple and intuitive platform that has a wonderful and simple high-level API that makes easy to get the job done for model training, hyperparameter tuning, model assessment and model deployment.
- Despite not covered here, model deployment is very simple in Amazon AWS. Once the model is trained (either by straightforward model training or by hyperparameter tuning), I can deploy and endpoint using the `.deploy()` method on a trained `estimator` object. And then I can use Lambda and API Gateway in order to construct an API for model predictions.
- `XGBoost` is a high-performing algorithm that can handle well non-linearities and sparse data.

File Reference

The notebooks and data used in this capstone project can be found on this GitHub repository:

Description	File
Data Processing	`compas_data.py` and `Data-Exploration.ipynb`
SageMaker Estimator and Hyperparameter Tuning	`SageMaker.ipynb`
SageMaker Batch Transform	`SageMaker.ipynb`
SageMaker Model Assessment	`SageMaker.ipynb`
Capstone Approved Proposal	`capstone_project/proposal.pdf`
Project Report (this document)	`capstone_project/project_report.pdf` or `capstone_project/project_report.md`

References

Angwin, J., Larson, J., Mattu, S., & Kirchner, L. (2016). Machine Bias: there's software used across the country to predict future criminals. And it's biased against blacks. ProPublica. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

Chouldechova, A. (2017). Fair Prediction with Disparate Impact: A Study of Bias in Recidivism Prediction Instruments. Big Data, 5(2), 153–163. https://doi.org/10.1089/big.2016.0047

Dressel, J., & Farid, H. (2018). The accuracy, fairness, and limits of predicting recidivism. *Science advances*, *4*(1), eaao5580. https://doi.org/10.1126/sciadv.aao5580

Larson, J., Mattu, S., Kirchner, L., & Angwin, J. (2016). How we analyzed the COMPAS recidivism algorithm. ProPublica. https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm