Stas Sajin Follow
Data Scientist at a Fin Tech Company
Oct 31 · 7 min read

Preventing Machine Learning Bias

Machine learning algorithms are increasingly used to make decisions around assessing employee performance and turnover, identifying and preventing recidivism, and assessing job suitability. These algorithms are cheap to scale and sometimes can be cheap to develop in an environment filled with good quality research and tooling. Unfortunately, one thing these algorithms don't prevent and automate is how to structure your data and training pipeline in such a way that it does not lead to bias and negative self-reinforcing loops. This article will cover solutions for faithfully removing bias in your algorithms using **bias-aware** rather than **blindless** approaches.

Photo by moren hsu on Unsplash

Why Preventing Bias Is Important

Predictions from an algorithm are generally not used in a vacuum. Instead, they are used to drive decisions and optimize for certain outcomes. **If the data you work with has some inherent biases, the**

model will not only learn those biases but will end up amplifying them. With biased data, the outcomes of the model will end up in self-fulfilling prophecies, which in many instances can lead to disastrous consequences. I don't think I can give adequate coverage to the topic of ethics in AI, since I want to provide more practical solutions to how one can design their training pipelines to be more robust to bias. Instead, I highly recommend going through the AI ethics resources by Rachel Thomas from fast.ai for a more detailed discussion on the topic.

Blindless Algorithms

The most common approach for removing the bias from an algorithm is to explicitly remove variables that are associated with bias. For example, if you want to predict who should be hired for a position, you might include relevant inputs such as the skills and experience an applicant has and exclude irrelevant information such as gender, race, and age. The issue with this approach is that it does not work. Some of the variables that you want to include are influenced by latent inputs. In other words, you can't say "Let's just exclude race from the model" while including twenty other variables that help encode someone's race. I will provide two examples where a blindless approach like this fails.

Amazon Hiring

Recently Reuters reported that Amazon has been trying to develop a tool to automate resume screening and hiring. The project had to be scrapped because it showed a heavy prejudice against women. I'm fairly sure that the engineers who worked on the model did not use any explicit variables that would identify someone's gender. Nonetheless, it is challenging to remove the model's ability to identify other latent variables that tie to gender and not discriminate job applications that contain "Women's Chess Club captain" in the resume. In other words, if the data is biased, which is very likely in this case, the model will find a way to encode that information when given a large and comprehensive set of features.

Predicting Recidivism

Two years ago Pro Publica has provided a very good analysis of the software used to predict recidivism rates. They found that the model embedded in that software had a much higher rate of false positives among African American than White population. The company that developed the software pointed out that none of the 137 questions they use in order to score a person's recidivism risk includes race as an input. Nonetheless, if you take a look at the questionnaire, it does not take a lot of insight to notice that some of the topics might be correlated with race and encourage bias when judging an applicant's risk. For example, when you ask someone if their parents were separated while growing up, you are indirectly encoding their race (see this article by Raley et. al, 2015, to understand just how mind-blowingly different are divorce rates among different races). Again, this is another example where blindless approaches do not work because of a failure to account for latent variables.

Bias Aware Algorithms

Instead of removing variables that are directly tied to biased outcomes, I suggest we do the opposite. In other words, **in order to know how much bias exists in your data, you need to allow your model to measure it correctly and then subtract the effect of that bias on the outcome**. I will go through a contrived simulation through which we can examine how effective is this technique.

Our simulated data will contain the following features: gender (we'll focus on the majority genders: male and female), years of experience, and career (we'll focus on two careers, software engineering and consulting). Our goal is to predict what salary we should pay someone given these features.

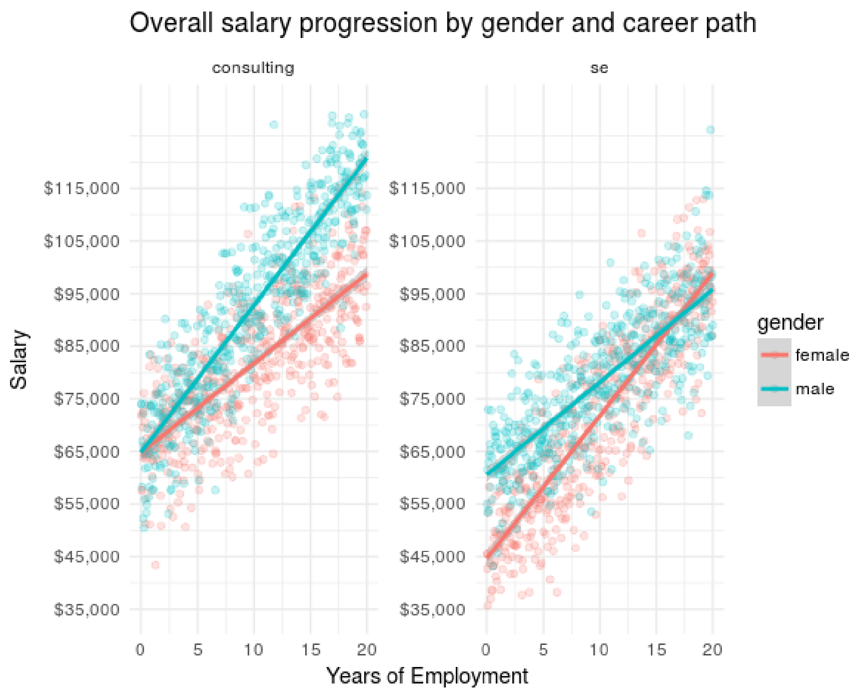
Our data will have the following relationships:

- Overall, there is a clear positive relationship between years of experience and salary
- Being a consultant will earn you more money relative to being a software engineer (SE)
- Being a male will earn you more money than being a female
- There are several key interactions we want to account. In consultancy, both males and females start with the same salary, but the salary growth among males is higher than among females.

In other words, for each year of experience (or for each promotion), the male is expected to get a higher bump in salary

- In software engineering, females start at the lower end and then have to play a period of catch up

You can find all the relationships described above depicted below.



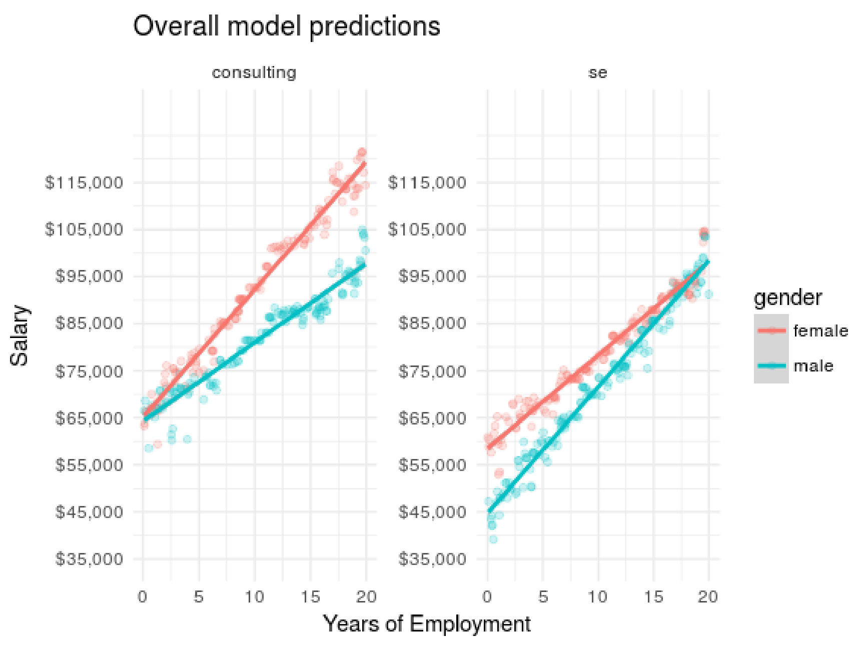
Note that we are purposefully creating a scenario that "seems" real and depicts different levels of interactions because I want to show the strength of a bias-aware approach in generalizing to problems that have hundreds of variables with many layers of complexity.

In order to remove the effect of prejudice in our data, we need to build a model that faithfully represents the gender bias. In the code below, we have a very simple training pipeline that does just that.

```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split

data = pd.read_feather('data.feather')
data = pd.get_dummies(data, drop_first=True)
X_train, X_test, y_train, y_test = train_test_split(d
                                                     n
                                                     t
                                                     r
dtrain = xgb.DMatrix(data = X_train, label=y_train)
dtest = xgb.DMatrix(data = X_test, label=y_test)
```

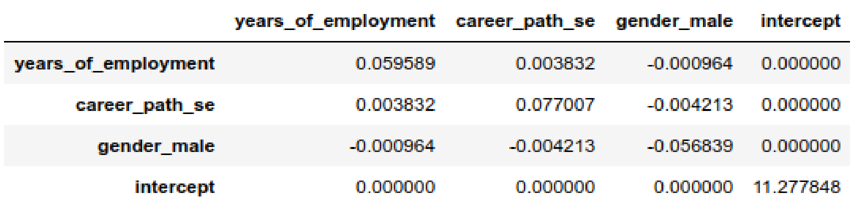
Below you can find the model predictions on the test set. As you can see, the model generalizes very well and manages to capture all the main effects and interactions in the data.



Next, I will extract the prediction-specific interactions in order to cancel their effect. In this instance, I'm using XGBoost, which has already implemented Shapley Additive Explanation method—a consistent, fast, and deterministic method for extracting feature contributions at the individual prediction level. This method is already available in other popular boosting libraries such as CatBoost and LightGBM. You can also use LIME if determinism in your predictions is not an issue, or extract the weights from hand-coded interactions if you're using other types of algorithms such as L1 or L2 linear regression derivatives.

```
feature_names = dtest.feature_names
interactions = xgb_model.predict(dtest, pred_interacti
pd.DataFrame(interactions[0],
             index = feature_names + ['intercept'],
```

The `interactions` object will contain a numpy array of shape (n_samples, n_features + 1, n_features + 1), which includes all the main effects, interactions, and the intercept. Here is how that array looks like for the first observation in the test set:



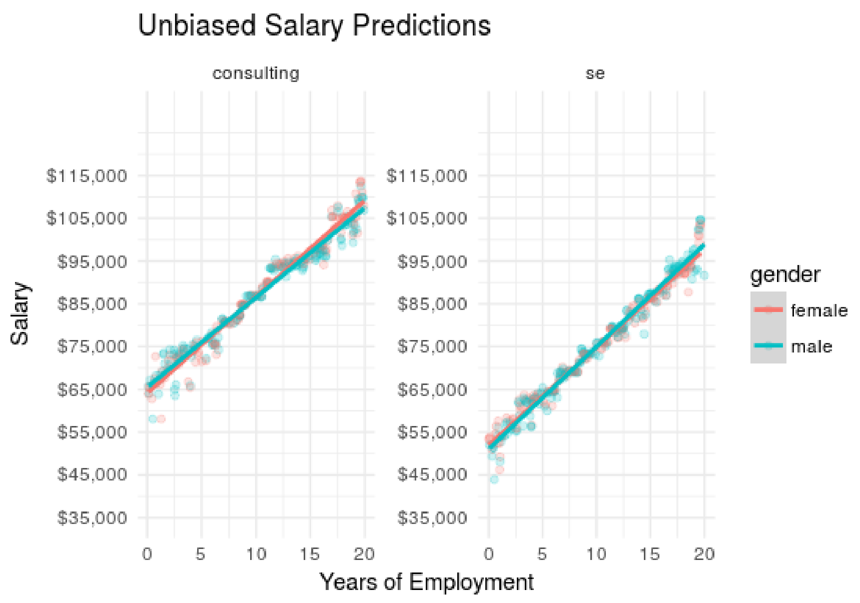
	years_of_employment	career_path_se	gender_male	intercept
years_of_employment	0.059589	0.003832	-0.000964	0.000000
career_path_se	0.003832	0.077007	-0.004213	0.000000
gender_male	-0.000964	-0.004213	-0.056839	0.000000
intercept	0.000000	0.000000	0.000000	11.277848

Note that the numbers look small because the outcome was log-transformed before training.

Next, we simply cancel all the influence driven by the gender variable.

```
bias_var = np.array('gender_male')
bias_idx = np.argwhere(np.isin(np.array(feature_names)
interactions[:, bias_idx, :] = 0
interactions[:, :, bias_idx] = 0
```

We can then calculate our predictions and plot them. We can immediately see that any sign of prejudice is completely removed and that the model faithfully captures the variation driven by years of employment and career path.



A bias-aware approach to modeling can be applied to other types of input data: text, images, sounds. If you're using NLP models that rely on entity embeddings, you can explicitly encode concepts that are more sensitive and keep track of how embeddings related to those concepts interact with other embeddings. Note that a similar approach to this seems to be used by Google.

Conclusions

I would like to point out from the outset that there is no question that this approach will lead to a decrease in model performance on your validation data. In our contrived example, the RMSPE is 12% for predictions that encode bias and 14% for predictions where we removed the gender contribution. Nonetheless, this decrease in performance is acceptable and encouraged in many settings. After all, the purpose of your models is not only to make good predictions but to also allow you to identify ways to pull levers, such as modify user behavior on your website or prevent harmful things from happening when diagnosing a disease. Hence, if you want to build a model that is not prejudiced by your data, you can't go wrong with letting the model first measure the amount of prejudice and then resetting all the bias contributing factors to zero.