Keeping Bias at Bay(es): A Naive Bayes Exploration on Retaining Diverse Talent in the Tech Industry

Project Link: https://katherinevivianw.github.io/biasatbayes/
Stanford CS109 Project Challenge

Katherine Wong
Department of Symbolic Systems
Stanford University
kvwong@stanford.edu

December 7, 2023

1 Introduction

The technology industry's workforce diversity gap is a long-time problem that many have attempted to combat. Less than a quarter of employees in computing and engineering are women; furthermore, only 5% of engineering jobs are comprised of Black workers, while 9% of engineering are Hispanic (Fry 2021).

Current solutions for increasing diversity in the tech industry workforce — such as the annual Grace Hopper Conference and diversity programs (e.g. Microsoft Explore and Meta University) — are often hyper-fixated on increasing the pipeline of initially helping marginalized individuals land roles. However, hiring is just one part of the equation. Largely neglected when evaluating the diversity gap problem is the retention of diverse talent — and what factors might influence an underrepresented employee's decision to leave a company.

Utilizing a Naive Bayes approach, this project aims to create an accessible machine learning-based tool to demonstrate what factors influence an employee's attrition status, along with how attrition status can impact other factors such as job satisfaction, income, and working overtime. Since company Human Resources (HR) datasets are often not publicized, we see this tool being internally used by tech companies' HR departments — who would train the model on their own private data. Once the model is trained, this tool can then help companies evaluate a possible a diversity talent retention issue at their own company and what factors they may need to consider in order to address attrition problems.

2 Addressing Simplifications

This project makes a couple simplifications in order to fit the scope of CS109 material. First off, in this project, we focus on gender disparities in tech between male versus female employees. It is important to note that the diversity gap

applies to several other underrepresented groups across race, sexuality, disability status, etc. This project holds many possible areas of expansion in other domains.

In addition, we only take into account the following variables when evaluating how they impact attrition: gender, working overtime, job satisfaction, and monthly income. In the data set, job satisfaction is represented on a discrete scale from 1-4, while monthly income is a continuous value. In our Naive Bayes program, each of these have been interpreted as Bernoulli random variables:

- Female: 1 if 'Gender' is 'Female', 0 otherwise
- OverTime: 1 if 'OverTime' is 'Yes', 0 otherwise
- JobSatisfaction: 1 if 'JobSatisfaction' $\geq$ the Job Satisfaction median (3.0), 0 otherwise
- MonthlyIncome: 1 if 'MonthlyIncome' $\geq$ the Monthly Income median (4933.0), 0 otherwise

Finally, by using Naive Bayes, we implicitly assume that each of these factors are independent, conditioned on the attrition label. If we were to expand this project to not make this assumption, we could utilize other machine learning algorithms such as logistic regression.

3 Implementation

3.1 Accessible UI/UX Design

In order to combat the daunting nature of machine learning, I aimed to make the frontend design as accessible and user-friendly as possible — especially considering that potential users of this tool are likely coming from a non-technical, HR background.

On the tool testing pages, the user is able to select the variable they want to predict (aka, the y variable) from a series of radio buttons. In addition, the user is also able to select variables that they want to condition on via a list of check-boxes (aka, the x variable vector). Once the user clicks the "Submit" button, the user inputs are sent to a Naive Bayes program in JavaScript that finds the given conditional probability. This probability is then printed out on the page for the user to easily view.

3.2 The Naive Bayes Logic

First, the input training dataset is loaded into the Naive Bayes script and translated into a dictionary data structure. Then, the model parameters $P(Y)$ and all possible $P(X_i|Y)$ for the training dataset are computed via standard counting. Finally, the probabilities $P(Y = 0|X)$ and $P(Y = 1|X)$ are calculated using

these model parameters. If $P(Y = 0|X)$ is greater than $P(Y = 1|X)$, the final prediction is Y=0 given the x inputs; if $P(Y = 1|X)$ is greater than $P(Y = 0|X)$, the final prediction is Y=1 given the x inputs. These probabilities along with the final prediction is sent back to the HTML/CSS frontend, which is displayed for the user to see. See section 4 for more details on calculations.

In order to keep my code clean/readable, I decomposed these steps into several helper functions. This allowed for me to both gain a deeper understanding of Naive Bayes' multi-step process and easily debug any issues that came up.

4 Calculations

Let Y be the user's selected choice for the "What do you want to predict?" question on the tool's testing page. In addition, let X represent all X_i values, where X_i is 1 if checked off by the user in the "Check off the factors that you want to be true" question and 0 otherwise. Our goal is to find $P(Y|X)$.

We can find $P(Y|X)$ using Bayes' Theorem, which can be expressed as

$$P(Y|X) = \frac{P(Y)P(X|Y)}{P(X)} \tag{1}$$

Here, we make the Naive Bayes assumption that

$$P(X|Y) = P(X_1, X_2, ..., X_n|Y) = \Pi_{i=1}^{n} P(X_i|Y) \tag{2}$$

We obtain the values $P(Y)$ and $\Pi_{i=1}^{n} P(X_i|Y)$ by iterating through the dataset and counting. Note that the $P(X)$ denominator (the normalization constant) is typically dropped during Naive Bayes classification algorithms, since removing it makes the computation significantly lighter and would not change how the probabilities compare to each other. Also note that $\Pi_{i=1}^{n} P(X_i|Y)$ is calculated using the joint probability. When calculating, we must apply Laplace smoothing in order to avoid zero probabilities. Thus,

$$P(Y) = \frac{(\text{Number of training examples where Y}) + 1}{(\text{Number of training examples}) + 2} \tag{3}$$

$$\Pi_{i=1}^{n} P(X_i|Y) = \Pi_{i=1}^{n} \frac{(\text{Number of training examples where } X_i, Y) + 1}{(\text{Number of training examples where } Y) + 2} \tag{4}$$

Computing these values is how we 'train' the Naive Bayes model. We can plug the values back into our Bayes' Theorem equation to get

$$P(Y|X) = P(Y|X_1, X_2, ..., X_n) = \frac{P(Y)\Pi_{i=1}^{n} P(X_i|Y)}{P(X_1, X_2, ..., X_n)} \tag{5}$$

$$\hat{y} = \text{argmax} P(Y)\Pi_{i=1}^{n} P(X_i|Y) \tag{6}$$

We calculate the probability for both P(Y=0 | X) and P(Y=1 | X) and compare them in order to make our final prediction ($\hat{y}$).

5 Conclusion

Overall, I had a really fun time doing this project. It was my first time using JavaScript and creating a whole web application from scratch, so there was a bit of a learning curve at first. However, doing this project allowed me to solidify my understanding of probability topics, learn some new web development skills, and engage with a topic I'm extremely passionate about (diversity in tech!). I hope you enjoy this project as much as I enjoyed making it!