Talent Model

Monika Singh, Ph.D.

11/07/2022

Objective

- Using a dataset build a hiring system (predict “hire”, “rejected at interview”, or “rejected at pre-interview”)
- Deploying models that impact people requires careful ethical decision making.
- One of the components of ethical AI is Systematicity

Systematicity

- **Arbitrariness**: The inherent error in the model due to overfitting/underfitting or issues in the data (unpredictability, unconstrained, and unreasonable)
- **Systematicity**: A model (algorithm) with inherent arbitrariness used on a large scale within an entire sector (e.g. hiring, education, and loans) can produce systematic discrimination.
  - When deploying talent models at scale, we have to design systems that have some degree of randomness in order to ensure that individuals are not arbitrarily blocked from economic opportunities
  - Since we are unable to completely remove arbitrariness from a model, we may at least reduce its impact.

Addressing systematicity

- The authors of the paper* outline two primary ways of addressing systematicity:
  1. Ensemble model method: Training a set of models (with similar accuracy) and randomly drawing from the set of models at prediction time **(I chose to implement this one)**
  2. Directly introducing (bounded) randomness to scores at prediction time

*Creel, Kathleen and Hellman, Deborah, The Algorithmic Leviathan: Arbitrariness, Fairness, and Opportunity in Algorithmic Decision Making Systems (February 15, 2021). Virginia Public Law and Legal Theory Research Paper No. 2021-13, Available at SSRN: https://ssrn.com/abstract=3786377

Dataset – features not used

- Observations: 50,000
- Total Features: 60
- Features not used: 18

Unique Features

- candidate_id
- occupation_id
- company_id
- application_atribute_1

Contains only zeros and nulls

- candidate_interest_2

Demographic Features

- ethnicity
- gender
- age
- candidate_demographic_variable_1
- . .
- candidate_demographic_variable_10

Dataset – features used

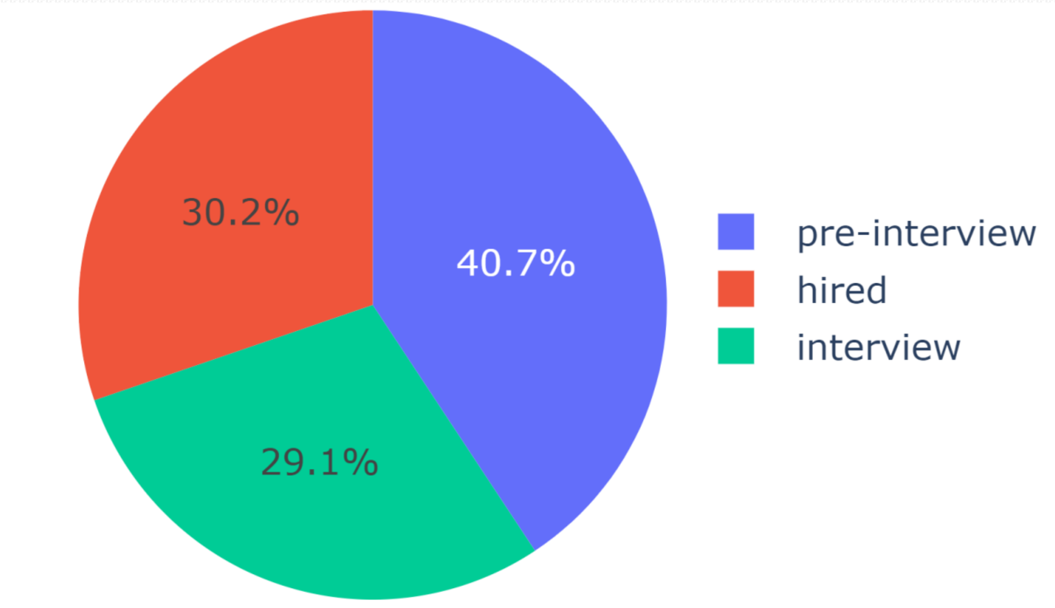
Feature	**Description**	**Type**	**Missing Value Imputation**
number_of_employees_log	Log of the number of employees at given company	Numerical	No nulls
occupation_skill_1_count … occupation_skill_10_count	Count of skills associated with attribute for given role	Categorical (binary)	Fill with most common
candidate_attribute_1 … candidate_attribute_8	Attribute of applicant	Categorical (binary) and two numeric	Fill with most common and mean
candidate_interest_1 , candidate_interest_3 … candidate_interest_8	Attribute of applicant	Categorical (numeric)	Fill with most common

Dataset – features used

Feature	Description	Type	How to deal will null values
number_years_feature_1 … number_years_feature_4	Attribute of applicant	Numerical	Mean
candidate_skill_1_count … candidate_skill_9_count	Count of skills within applicant's CV related to attribute	Numerical	Mean
candidate_relative_test_1, candidate_relative_test_2	Attribute of applicant. Performance on test relative to school of origin	Numerical	Mean

Dataset

- Application status:



Challenge

- A **predict()** function that implements one or more mechanisms for addressing systematicity

- A **set of metrics** and associated charts that allow someone to measure and evaluate the effectiveness of my systematicity mitigation strategy.

Predict() function

The Predict() function requires two phases:

1. **Training:** Training and building an ensemble model that contains a set of models (with similar accuracy). Storing the set of models to use at prediction time.

2. **Predicting:** Randomly select one model from the ensemble model set at prediction time.

Phase 1: Training models for Predict() function

The predict function works in the following steps:

Step 1:

1. Split the dataset randomly into 2 parts: one with 90% and one with 10% of the data
   - The 90% is used to create models and the 10% is used to address systematicity
   - Let's call the 10% "systematicity data" and the 90% "train-test data"
2. 500 times randomly split the train-test data into 80% for training and 20% for testing
3. Train model using random forest classifier on each split
4. Select the models that have a recall (of hired candidates) and f1-score(of hired candidates) above a certain threshold

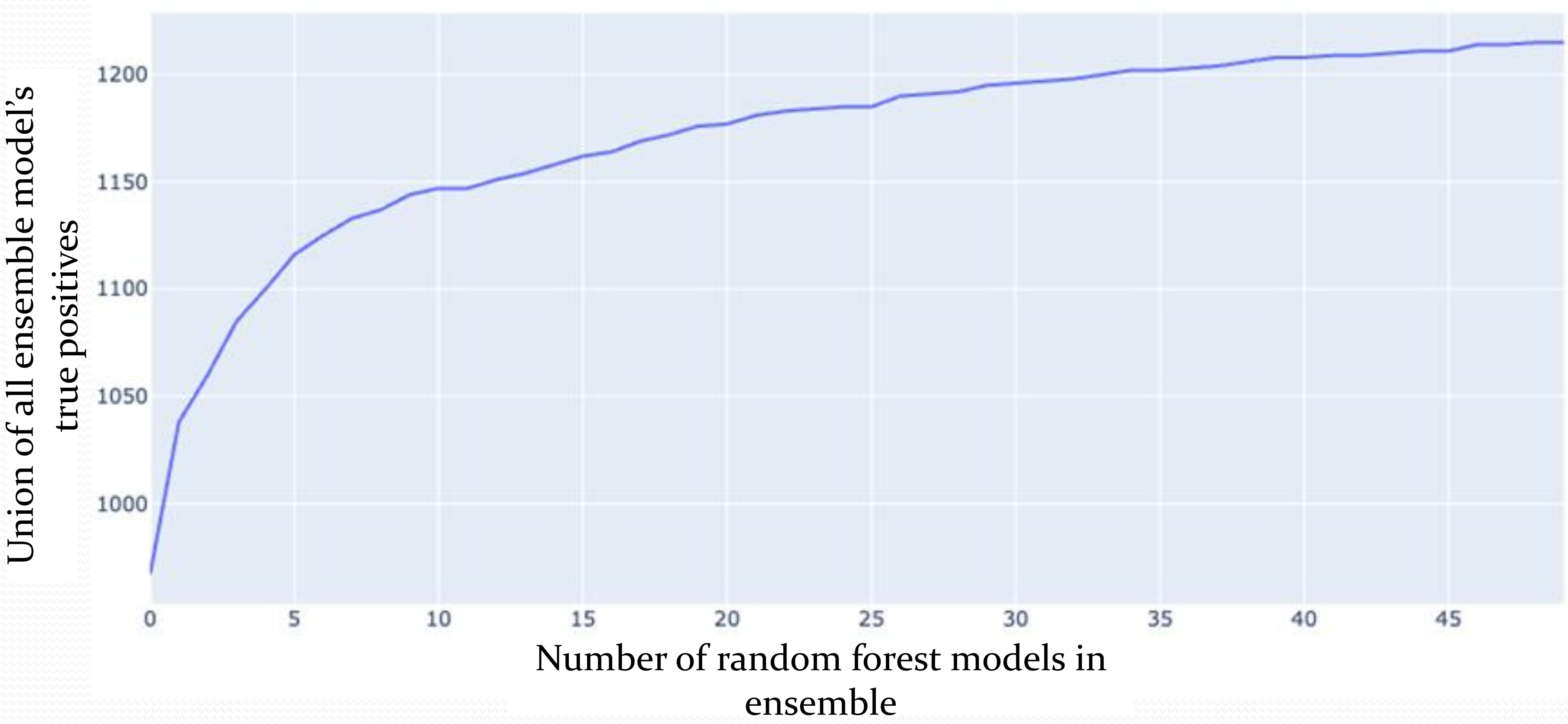
Phase 1: Training models for Predict() function

Step 2:

1. Once we have selected models, compute the set of True Positives (TP) (hired) of each selected model using the systematicity data (the remaining 10% data)
    - Add the model to the ensemble if its set of true positives is not fully a subset of any other ensemble model's true positives
2. We stop adding new models, when one of the following is true
    1. The union of all ensemble model's true positives covers all of the possible true positives in the systematicity dataset
    2. The change in the union of all ensemble model's true positives becomes very small (zero or close to zero)
        - In our implementation we compute the running average of the change in TP set size of the last 10 accepted models
        - We stop when the running average is 0
    3. We run out of trained models to add to the ensemble (computational budget reached)
        - All 500 models added

Phase 1: Training models for Predict() function

- It may not always possible to cover all positives
- The total number of possible true positives equals = 1,573 (people actually hired)



Phase 2: Predict()

1. Select a random model from the ensemble model set
2. Use the model to predict the given candidate

Evaluation Metrics

The following are the two desirable behaviors of an ensemble technique addressing systematicity:

1. **Fairness**: ensures that every deserving candidate is at least ‘hired’ once by an ensemble model.
2. **Arbitrariness**: ensures that all deserving candidates have an equal chance of being “hired” by our ensemble model.

Fairness Evaluation Metric

- I propose a metric to compute fairness of our ensemble model called ‘*coverage*’
- Coverage= $\dfrac{number\ of\ All\ True\ Posive\ (hired)}{number\ of\ Actual\ Positive\ (hired)}$
- Coverage within [0, 1], where 1 indicates the maximum coverage and 0 indicates no coverage
- We want coverage to be close or equal to 1.
- This represents the fraction of all people hired in reality that would have been hired by this algorithm had it been used to make the hiring decision

Fairness Evaluation Metric: example

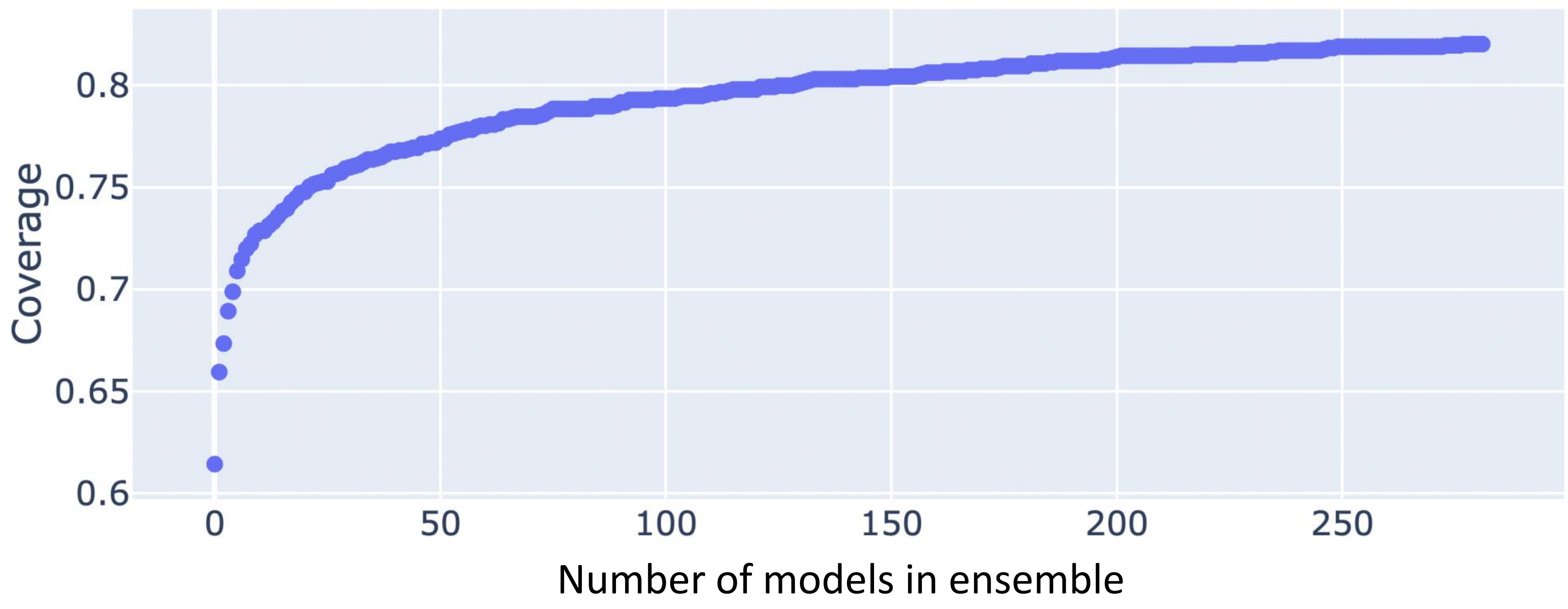
- For example our dataset contains the following candidates: {**a**,**b**,c,**d**,**e**,f,g,h,**r**}
- The following are hired (Actual positive) ={a,b,d,e,r}
- We have 3 models (M1,M2,M3) in our ensemble model and the following are their ‘hired’ predictions
- M1 = {a,b,c}
- M2 = {c,d}
- M3 = {a,b,e,f}

Fairness Evaluation Metric: example

- $All\ True\ Posive\ (hired) = \{a, b\} \cup \{d\} \cup \{a, b, e\}$
  $= \{a, b, d, e\}$
- $Actual\ Positive\ (hired)$= $\{a, b, d, e, r\}$
- Coverage= $\frac{|\{a,b,d,e\}|}{|\{a,b,d,e,r\}|} = \frac{4}{5}$ = 0.8
- Notice candidate 'r' was not selected by any model
- This means that 20% of people that are hirable are not identified as such by the classifier

Fairness Evaluation Metric

- The coverage metric improves asymptotically as the number of models in the ensemble increases



Arbitrariness Evaluation Metric

- Let us compute an array:

  TruePositiveFrequency= [f_1,f_2,f_3,f_4, ..., f_n],

  where n is the number of all candidates who are hired in reality
- f_i indicates the number of times candidate$_i$ has been selected by our ensemble model set
- $Arbitrariness = \frac{\sum_{i=1}^{n}(|mean(TruePositiveFrequency) - f_i|)}{n}$
- Where $mean(TruePositiveFrequency) = \frac{\sum_{i=1}^{n}(f_i)}{n}$
- |a| indicates the absolute value of a

Arbitrariness Evaluation Metric

- Using the previous example:
- M1 = {a,b,c} (TP={a,b})
- M2 = {c,d} (TP={d})
- M3 = {a,b,e,f} (TP={a,b,e})
- All TP in the dataset={a,b,d,e,r}
- TruePositiveFrequency =[f_a, f_b, f_d,f_e,f_r] =[2,2,1,1,0]

- $Arbitrariness = 0.64$

Arbitrariness Evaluation Metric

There two way to interpret the arbitrariness metric :

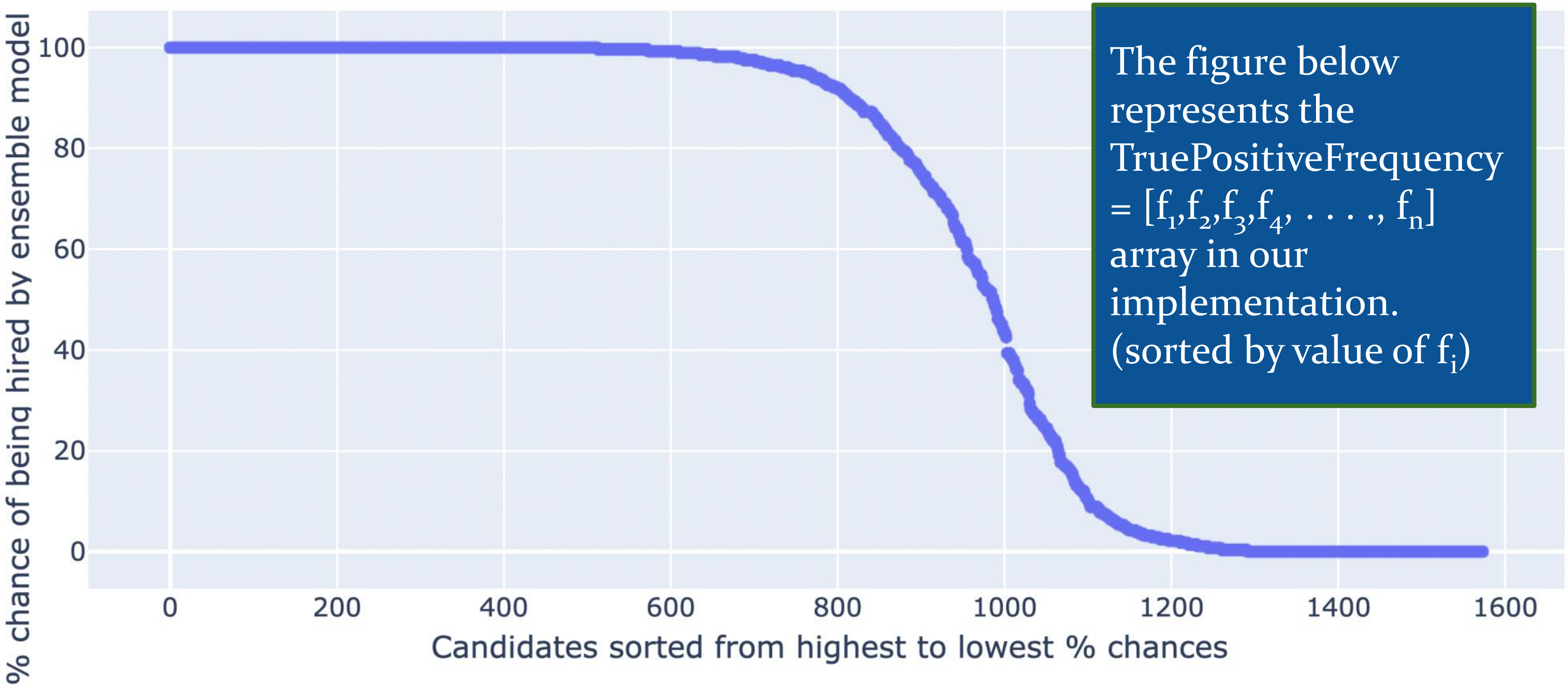
- Case 1: if all candidates with application status ‘hired’ in the dataset are equally skilled.
    - In this case a low value of ‘*Arbitrariness*’ is desirable
- Case 2 (most likely): if candidates with application status ‘hired’ in the dataset are not equally skilled, then there are some borderline candidates.
    - In this case the high value of ‘*Arbitrariness*’ is desirable along with high ‘*Coverage*’
- **Future research:** How to quantify the variation in candidate skill and relate this to the desirable level of arbitrariness?

Evaluation Metrics

- In the implementation there is an ensemble of random forest classifiers
  - The method generalizes to any type of classifier
- The highest accuracy observed was 0.62 (f1-score) and 0.67 (recall)
- The f1-score and recall threshold for model acceptance into the ensemble was set at 0.58 and 0.63
- Given more time, more data, and information about the features, the accuracy could be improved and a variety of classification techniques could be experimented with

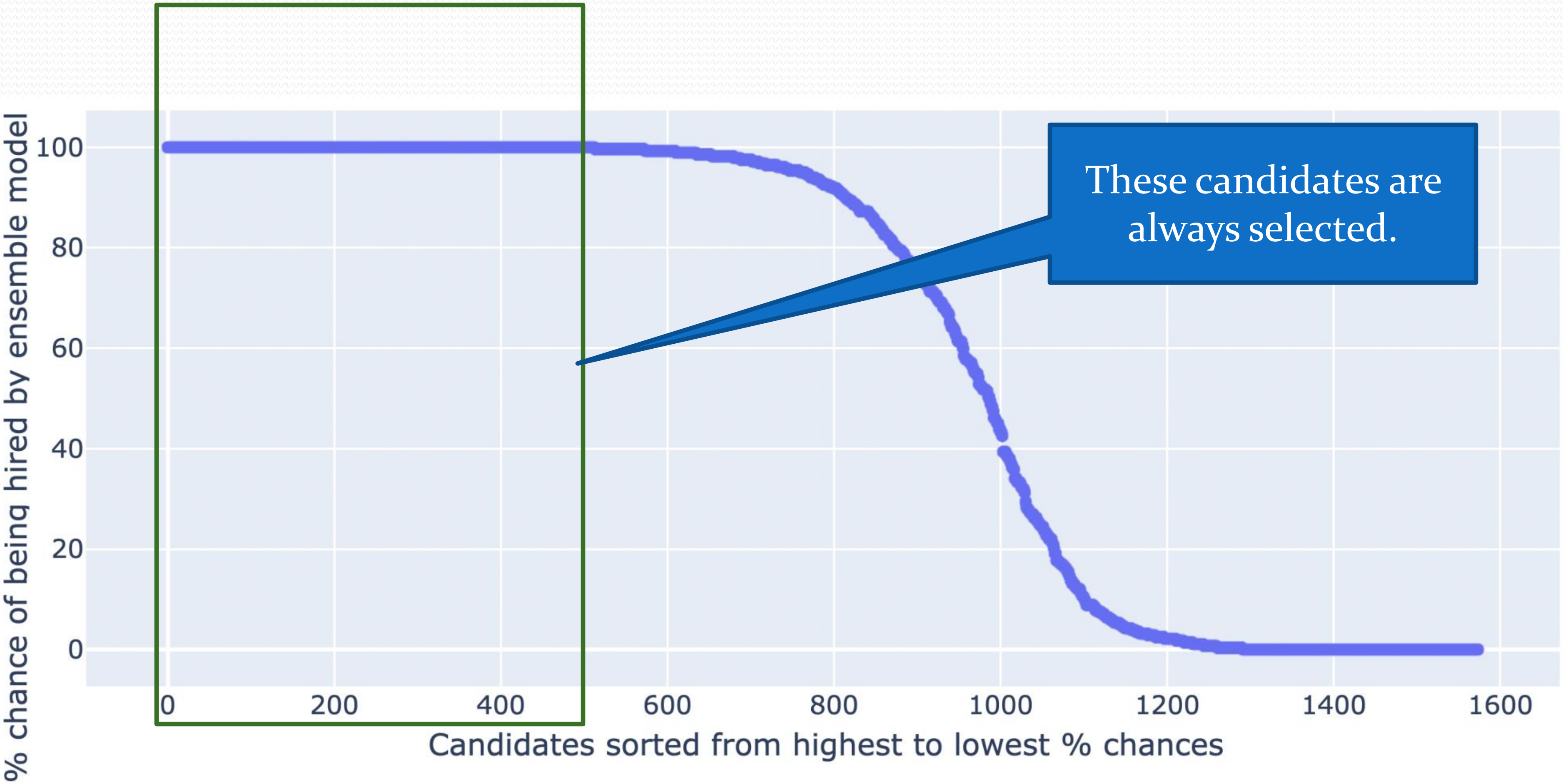
Arbitrariness Evaluation Metric

All of the candidates (1,573) shown here were hired in the systematicity data, so this is their percent chance of being hired by the ensemble model as well.



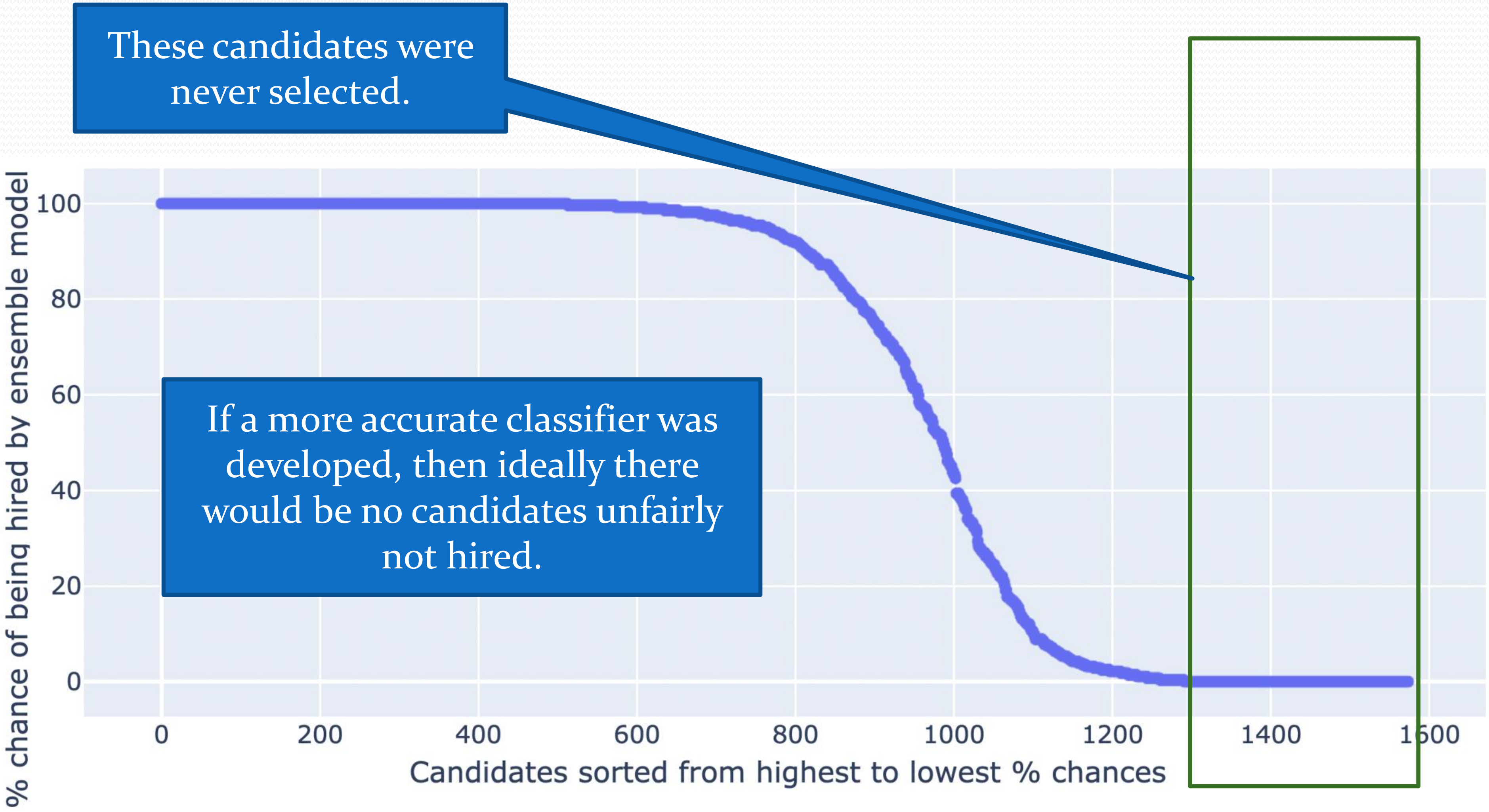
The figure below represents the TruePositiveFrequency = [f_1,f_2,f_3,f_4, ..., f_n] array in our implementation. (sorted by value of f_i)

Arbitrariness Evaluation Metric



These candidates are always selected.

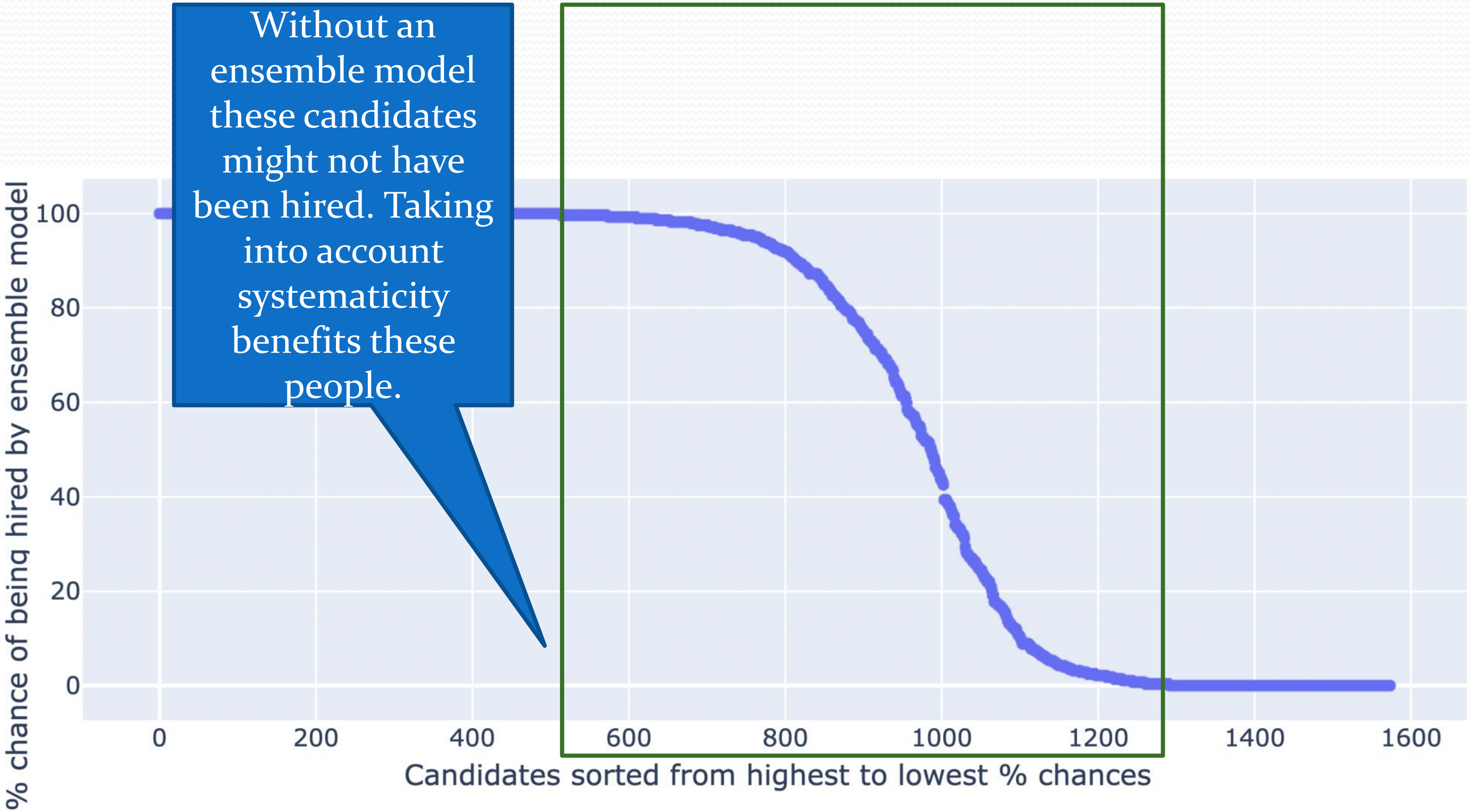
Arbitrariness Evaluation Metric



These candidates were never selected.

If a more accurate classifier was developed, then ideally there would be no candidates unfairly not hired.

Arbitrariness Evaluation Metric



Without an ensemble model these candidates might not have been hired. Taking into account systematicity benefits these people.

Future Work

1. How to quantify the variation in candidate skill and relate this to the desirable level of arbitrariness?
2. Build a more accurate model (with a larger dataset if available) by experimenting with other classification techniques (XGBoost, TensorFlow, logistic regression classification, etc.)
3. Convert into binary classification problem (hired/not hired); currently using multi-class classification (hired/interview/pre-interview)
4. Given more data, consider job performance after hiring to predict <u>who would do the best job</u> **not** <u>who is most likely to be hired</u>