Keith Chewning
Systematicity project
07.07.22

Definitions

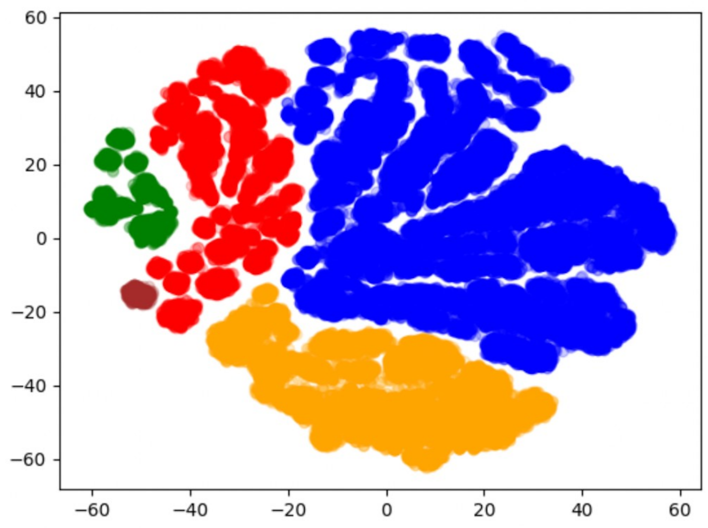
- Y is the target variable, application_status
- X is the set of input features and $X = (Z, D)$
- D is the set of demographic features
- Z is the set of of features in X that are not in D

Assumptions and expectations

- model adaptions for bias treatment are not considered
    - fairlearn could be used in this regard
    - if we were optimizing a function, we could add a penalty term $(f(X) - f(Z))^2$. In other words we want to learn $p(Y \mid X) = p(Y \mid Z, D) = p(Y \mid Z)$, where $f(x) = p(Y \mid X = x)$.
    - model accuracy was not an explicit concern in this project

Data inspection

- Cluster demographic data, D
    - t-SNE for dimensionality reduction, DBSCAN for clustering



- Share per class by cluster: classes are relatively evenly distributed

```
cluster          | interview     | hired           | pre-interview
0 [0.28444181 0.28236342 0.43319477]
1 [0.28942538 0.30561362 0.404961  ]
2 [0.30748299 0.26598639 0.42653061]
3 [0.31916903 0.30122757 0.3796034 ]
4 [0.31733333 0.37066667 0.312 ]
```

- top share of pre-interview by demographic categories

```
candidate_demographic_variable_9=0                    0.423808
candidate_demographic_variable_4=0.0                  0.430847
ethnicity_White Gypsy / Irish Traveller=1             0.432738
```

candidate_demographic_variable_3=0	0.438879
candidate_demographic_variable_5_citizenship=1	0.456482

- for illustration purposes, we consider candidate_demographic_variable_3=0 as a variable of interest

Data preprocessing
- change the target variable into a numeric categorical feature
- remove ID fields, i.e. where the number of unique values equals the number of records
- stratify data by data['candidate_demographic_variable_3'] == 0 and its complement.
- hold out 10% of the data in each stratification group as a test set
- numeric missing values
  - set as the median value
  - find the KNN take the median value, from the original data, excluding NaN values, which is meant to address *treat like cases alike*
- categorical features turn into a one-hot encoding

Model
- Random forest classifier
- build N models where
  - number of trees is a random number between 10 and 50
  - max depth is a random number between 3 and 20
  - the criterion is either gini or entropy

Addressing Systematicity
- random model selection
  - randomly select one of the N models
  - predict with this model
- random jitter to output probabilities
  - select the best performing model
  - multiply the output probabilities by a uniform random number between 1e-5 and 0.1; the scaling factor could be tuned
  - the argmax of the scaled probabilities is the predicted value
- in each case it is advantageous that for a given x, the output is deterministic. To this end I hash the input and set this as a random seed. This allows us retain our stochasticity, but it is deterministic for a given input.

Evaluation

Systematicity treatment:

No treatment (best performing model):

	interview	hired	pre-interview
Demo var3=0	0.3312	0.4534	0.6441
Demo var3!=0	0.4492	0.5348	0.5262
Global	0.3794	0.4876	0.6142

Random classifier:

	interview	hired	pre-interview
Demo var3=0	0.2115	0.3112	0.6198
Demo var3!=0	0.3087	0.4522	0.5119
Global	0.2510	0.3726	0.5908

Jitter probability:

	interview	hired	pre-interview
Demo var3=0	0.3299	0.3845	0.5515
Demo var3!=0	0.3772	0.4541	0.4212
Global	0.3479	0.4114	0.5160

- We look at F1 scores stratified by our demographic variable and also globally
- When the demographic variable = 0 the F1 is 0.6441 for pre-interview
- For the random classifier and jitter probability, this 0.6198 and 0.5515 respectively
- Incorporating systematicity allows for more variation in our predicted values at the expense of F1

Open question

Compared to the best performing model, incorporating systematicity in this manner will reduce our evaluation metrics compared to the gold labels. So are we saying that the real issue that we are trying to guard against is the “incorrect” labeling of the gold labels? The randomness addresses this in an indirect manner. Another way to look at this is that if the gold labels are mislabelled (noisy), then our classifier can be overly confident in what it learned from the labels during training. A way to make a classifier less confident is to use soft labels. For example map 1 → 0.9 and 0 → 0.1 in the binary case. We could also add Gaussian noise to the input features, which can help a model generalize better in the presence of noisy data.