Software in Society

Personal responsibility in the engineering workplace

Lere Williams

Overview

Policy vacuums, conceptual vacuums and invisibility in software

Algorithmic complexity (ethical not computational)

Arguments for inclusion and personal responsibility in the software industry

A framework for computer ethics

"The mark of a basic problem in computer ethics is one in which computer technology is essentially involved and there is an uncertainty about what to do and even about how to understand the situation." [Moor, 1985]

Computation is a very general purpose tool, and is increasingly and idiosyncratically involved in our lives

Its involvement leads us to conceptual vacuums and policy vacuums

Policy vacuum: Facebook and fake news

Big controversy about whether or not the dissemination of fake news on Facebook affected the outcome of the election [https://goo.gl/JduJbT, https://goo.gl/9yfakX]

False claims (in actuality the Pope did not endorse Trump or anyone else)

Evidence to suggest that a lot of people get news from Facebook (44% of U.S. adults by one estimation)

Also evidence to suggest that content on Facebook leads to action

Zuckerberg says (and I agree): we need to be extremely careful about becoming arbiters of truth

Invisibility in software

Invisible abuse

Use of invisible operations to conduct malpractice, misuse of private information, surveillance, etc.

Invisible decisions and assumptions

Things left to the interpretation of the developer: substantive implementation decisions, input data, etc.

Invisible complexity
Wisconsin Supreme Court sentencing case

Judges were using algorithmically-generated risk assessments that predict rates of recidivism as input for sentencing.

Proprietary algorithm was found by one study to have a 40% error rate and to be biased against African-Americans.

Ultimate ruling was for what is effectively a warning label.

How might this sort of unfairness happen?

Supervised learning algorithm (roughly): takes historical instances of a problem (training data) and produces a function (classifier) that can be used to decide future instances of the problem.

In order to do this, the algorithm picks out certain attributes of the input data (features). The set of all features is called the feature space.

Biased training set? Likely biased classifier

Learning algorithms are designed to detect statistical patterns in the training data.

If the training data reflects existing biases, then the learned classifier will likely reinforce those biases.

Critically, this can be true regardless of whether a sensitive attribute is explicitly included in the feature set.

Problems of sample size

It gets better. Classifier error often decreases as inverse square root of sample size:

But of course, by definition, minority populations have smaller sample sizes.

So if the classifier learned on the majority group does not in fact apply well to the minority group, then the classifier will be more accurate for the majority than the minority.

Overall then, it might appear that a classifier is highly accurate while still biasing against a minority population.

A case for data due process

Angwin advocates for due process protections with respect to data used in algorithmic decision making.

Apparently, the credit industry is the only one currently subject to such legislation.

In light of the knowledge of how learning algorithms work, that’s maybe not a bad suggestion.

Of course, not all modelling uses learning algorithms (that was just an example). Specifics of the model used are critical.

We should probably advocate for seeing the code too.
A case for diversity in software organizations

Choice of training data is largely a decision left up to the developer.

Perhaps then, more diverse development teams can help to combat bias in training sets

Team diversity drives innovation in new areas [https://goo.gl/YCSS9M]

Educate people about software internals

Remember Moor’s thesis. Paraphrasing: complexity creates obscurity, and unnecessary ethical disputes often result from lack of hard facts.

Think carefully about the software you build, and the contexts in which it might be applied

Very, very hard to consider all the possible ramifications, but awareness is the first step.

More chances than ever to apply software to big challenges

Government applications
- Code for America’s food stamp application
- Brigade’s voter networks

Poverty applications
- Kiva’s microfinancing
- Enveritas’s smallholder verification process

Many, many other problem domains

Recap

Brave new world, full of policy vacuums

Fairness is not guaranteed by “neutral” software

We have a responsibility to take an active stance in how software shapes the world

Educate people about software

Build carefully