Wrangler

Present: Jake Fisher, Scribe: Qianrui Zhang

Background - Data cleaning

Related work

- Programming-by-demonstration
- Toped++
- SWYN

Technical Parts

- Interface: natural language descriptions of transforms, visual previews
- Interactions
- Inference Engine: ranking transformation suggestions, using user corpus, ranking criteria, filter for diversity

Q: Why they focus on text?

Data cleaning is a broad topic, but Wrangler focus on Text - an important aspect of data. They want to distinguish their techniques by calling it 'Data Wrangling'.

Evaluation

Compare against Excel: popularity, same capability.

12 participants skilled in data.

10 minute tutorial on Wrangler.

3 tasks.

Q: Who is Wrangler's target audience? How is Wrangler comparing with writing scripts?

Wrangler helps to understand the data at the beginning comparing with writing codes. And programming-by-examples is more friendly to use (future direction: programming-by-example for regular expressions) and makes people feel more comfortable.

There is another direction to combine tools like Wrangler with programming scripts.(e.g. Jupyter Notebook) It would be cool to switch between Wrangler and programming interfaces.

Q: Is the ranking algorithm effective?

The categrization helps, the order is not helping.

It's easy to find what you want, but it's not always at the top.

IT's not easy to find what we want in suggestions.

Easy transformation is easy for the system to find, but it is difficult to find tricky/complicated transformations.

Q: Was the evaluation only against Excel sufficient?

The evaluation is very briefing in this paper. Should they do that with more other systems? (Given the evaluation, what does it imply the contribution of this paper is?)

The tasks they choose may be easy to do in Wrangler.

No individual thing in the paper is what matters. The main contribution is putting things all together and then it works better than industrial alternatives(Excel). In this sense, they did a good job in their paper.

It would be better to have some evaluations on the reproducable scripts. (And other features)

Texture Notes — 4/1/2020

- Texture — Data Extraction over Print Documents
- Defines a language similar to SQL
- Related works:
  - Refiner: uses anchors and distance to define area of information in print document
  - PDFFigure2.0: Finding figures in scientific papers
- Texture uses a 2 stage approach
- First stage is identifying structural elements
- Then, extract info over the structure
- Paper focuses on first stage
- Alternatives:
  - Training an ML model
  - Crowd-sourcing
  - Hiring freelance devs
- Texture combines components from end-users, crowd workers, and developers
- Developers contribute heuristics as code to repository
- Texture provides unit test and test doc set to them
- Crowd workers: collect structure labels from crowd
- Used CrowdCollect
- Provided basic, initial heuristics from CS ugrads
- **Discussion:**
- Q: How will it be used in the world?
- Q: Why would developers want to provide heuristics to the system?
- Refiner is still flexible, but here there is a notion of structure. But if structure is fixed, will this limit the capabilities?
- Use cases are not exactly the same
- Texture doesn't do a good job of providing a real problem, but this problem does exist.
- People who want to get something out of data aren't same people as those who write programs
- A lot of redundancies between documents, and someone needs to write reusable heuristics
- How is text/document styling take into account?
- Q: How is the evaluation good for accessing the system? Should it be looking more at the main contribution, not just the heuristics?
- Evaluation is hard to talk about because it's so little
- (Context: this is a workshop paper, so evaluation will be rather small)

- Could be helpful to run a user study
- You could also get a company or group to use it and report on their results
- This paper doesn't focus on a small piece, but the combination of all of them
- Not totally unique, but they do a nice job
- Diagram is overly complicated
- No "end-users" — people who use this are developers or data literatre
- Have documents that you can't directly query, and need people to write extractors
- Their job is to implement a bunch of operators
- They're evaluating if people can write operators, and if they're good
- Operators create database of "boxes" and, those are fed to a query executor
- Like in databases, a developer will write functions for you and give you functionality to extend
- **General discussion**
- More perspective on 3 systems — Wrangler, Texture, and Refiner
- Predictive Interaction: a way of thinking about them, especially Wrangler
- Begin with data, and easiest way to deal with it is to write code
- You could instead visualize data, and allow user interaction to accomplish goals, choose rendered results, and compile to regular results (same form as regular code data flow)
- 2 spaces: Vis/UI and Data + Code
- Connected around Domain Specific Language
- How do you best combine interfaces and AI/recommendations in an effective way?
- Shared representation designed by DSL
- AI's job is to generate and rank suggestions
- User can review, select, edit, or ignore suggestions
- Autocomplete common everyday example
- Many examples we've seen in class
- You need:
  - Cases where it helps
  - Low overhead (doesn't hurt in anyway, can be ignored)
  - Gets better over time

  s
- How do you evaluate positive value?
- Question is, does there exist a task where we provide positive value
- More difficult to show for decreasing value, requires end-to-end user study
- Why combine HDI and AI?

- Data tasks unclear until you start trying it out
- Helpful when task itself is a large space
- If you know what the task is, you don't need an interface (just algorithm to make task better)
- SQL is example: common set of operation, but class of all queries is very large
- Compare how 3 systems modeled the problem