Defending Credit for the World's Poor with Data Science

Everett Wetchler · Follow

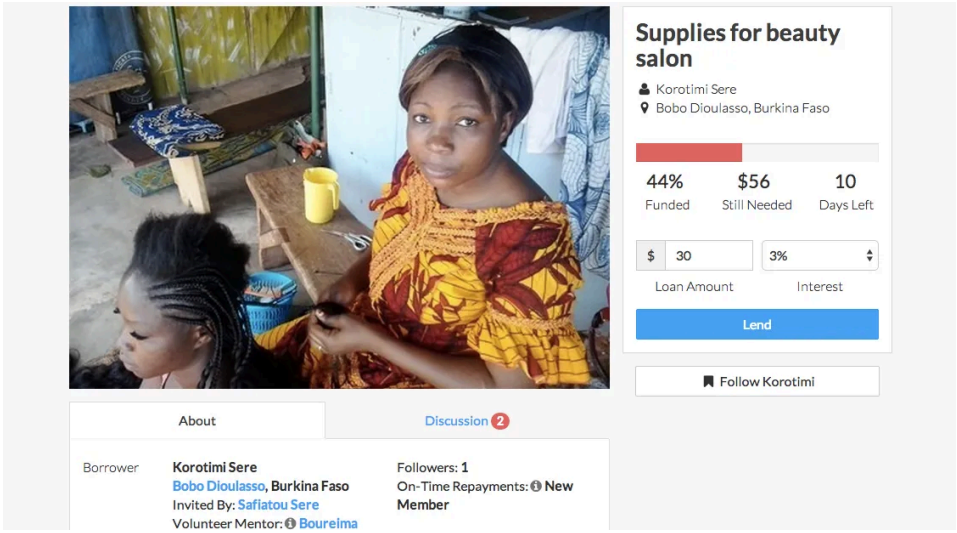
Published in Bayes Impact · 9 min read · Oct 14, 2014

As a Westerner, getting a credit card is only slightly more complicated than tying my shoes. My world is raining with opportunities to borrow money to go to school, open a store, consolidate loans, or buy an iPhone 6. For poor global citizens, though, the game is different. In fact, there isn't a game. A small business loan in developing countries usually requires a bank account (which 80% of sub-saharan Africans do not have), carries exorbitant interest rates (50%+), and are otherwise unreachable or unaffordable to the majority. The reality is that it's simply not profitable for lending agencies to accommodate these high-risk, small-scale borrowers.

The irony is that these are precisely the people whose lives can be most dramatically improved by a small amount of liquidity. Their hardship is enormous, and emerging from it is often complex or impossible due to systematic constraints (poverty traps). Enter the world of microfinance, where nonprofit agencies like Kiva and Grameen seek out and create loans for those in the greatest need. With a small amount of cash, these industrious borrowers seek to start small businesses and pull themselves out

of poverty. Microfinance loans may not be the best ROI, but in terms of human impact, the dollars are well invested (catalytic philanthropy).

For the past few months, Bayes Impact worked with one such organization and novel player in the microfinance game,Zidisha. Zidisha provides peer-to-peer loans from individual (usually US) lenders directly to borrowers in Africa and Asia. Borrowers sign up on the Zidisha website (typically from an internet cafe) and in a matter of days can have fresh cash in their mobile money accounts (M-Pesa). By connecting borrowers and lenders directly, Zidisha is able to avoid involving intermediary agencies that add cost, overhead, and time delay.



The Problem

The hard part of lending money is (surprise!) getting paid back. Because Zidisha does not employ a partner agency on the ground to vet candidates, it's difficult to tell a reliable borrower from a negligent one solely from their online application. Zidisha combats this by requiring a valid Facebook account, local referrals, and a range of other criteria, but still a substantial proportion of first-time loan applicants end up paying nothing back. This is

an especially bad experience for lenders: it's one thing if you loan money to a person who falls into hardship and only partially repays you, but it's entirely different if someone takes your generous loan and runs away. These "fraudsters" are a threat to the faith of lenders in the institution as a whole.

Bayes Impact undertook the task of creating a model to predict which borrowers were likely to completely skip town with their loan money. Being a complete-solution-driven organization, we also worked with their development team to create a serving solution that completely integrates borrower scoring and into their existing workflow.

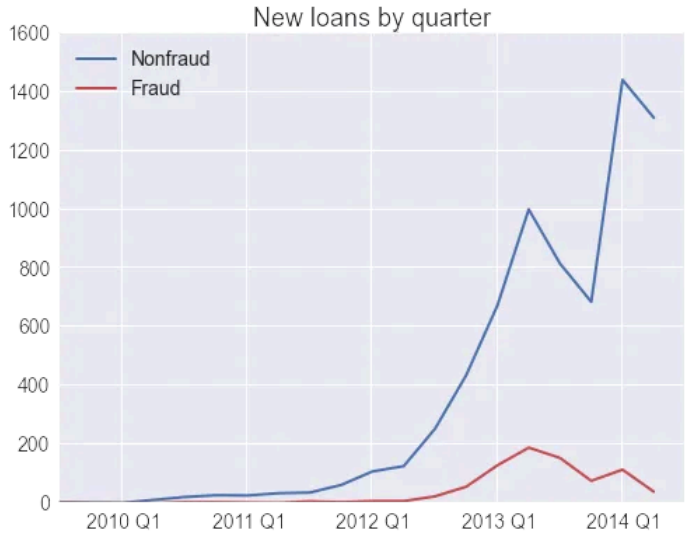
The Dataset

Zidisha has historical data of borrower applications and repayment behavior over their 5 year existence. This includes information about all borrowers who applied, loans that were disbursed and at what interest rates, borrower repayment performance, and borrower default. From this we need to model repayment based on borrower information and loan application details.

We defined "fraud" as any loan where the borrower has repaid nothing and is at least two months past due on their first payment. While this is a simplification (as some borrowers may have been honest but simply encountered hardship early), it is meant to capture the group of loans that are most damaging to Zidisha as a whole.

Step 1: Exploratory Data Analysis

The best place to start is always a descriptive big-picture view. For one, this is a sanity check that our data is sound and we haven't egregiously failed in extracting it correctly. When we have to join data from multiple sources, tables and interpret columns/keys correctly, ETL is easy to botch. Second, it's important to confirm that we can see, in the data, the problem our client claims to have. For example, we might plot the loans over time:



The non-smoothness can be attributed to policy changes in the Zidisha application process. For example, in Q2 2013, the maximum first loan size was reduced, and Facebook linking became required. This explains the drop in new loans in that period, which quickly recovered with Zidisha's growth. EDA helps us flag quirks in the data like this, ensuring that we ask all the necessary questions to interpret the data correctly. This is only one of a number of descriptive summary analyses we did, but it gives an idea. It was also useful to compare our numbers with Zidisha's published statistics and account for any discrepancies.

Step 2: Feature Extraction

Model building and evaluation isn't a completely linear process — we may loop back from model evaluation to feature extraction to EDA and back again. That said, we do start with an initial feature set, and add more later as we think of ideas. While for security reasons we can't share the exact features we used in the model, some general areas we considered were:

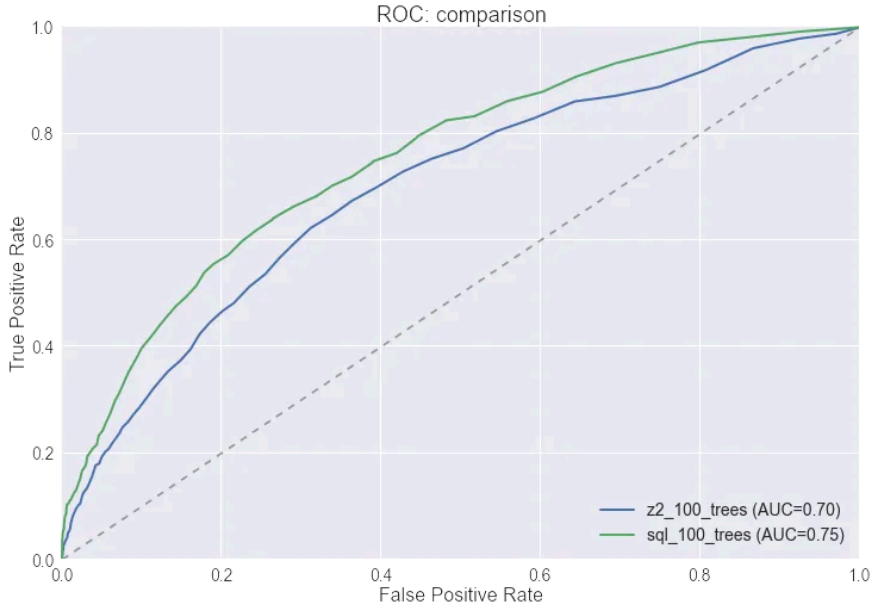
- Loan amount, interest rate, and repayment timeline

- Geographical features
- Information from the self-description and business description
- Relationships with other Zidisha borrowers

Step 3: Model Evaluation

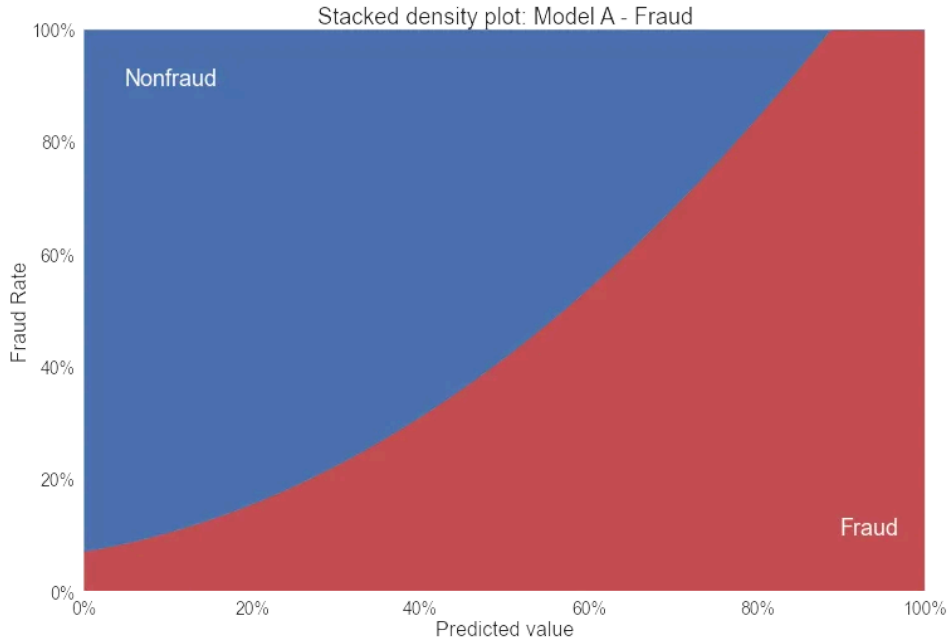
There are many ML algorithms to apply to our data, and many possible parameters and tweakings of each algorithm. In comparing, we need a clear sense of which classifier is "best" — a title that is highly dependent on the application.

The most obvious thing to do is to plot ROC curves comparing the classifiers' performance on a held-out test set. (Well, actually the average performance of each classifier over a 5-fold cross validation of the training set; we save the test set purely for evaluation before launching, so it has no influence on model selection.) Here's a comparison of two of our final model candidates — both random forests using different features and parameters.



In this case, one model is clearly better at all thresholds. If, instead, the curves cross (and especially if they have similar AUCs), the “best” one is very dependent on how the model will be used.

Model stability over its range is also important. If we predict that an applicant has (say) a 20% risk of fraud, then we would like to see 20 fraudsters for every 100 applicants given that score (no more, no less). Same goes for all other predicted values. We can visualize this like so:



The ideal curve would be a diagonal line. Ours does reasonably well, but we can see that it is a bit biased, particularly in the extremes. This may or may not be an issue depending on the expected operating threshold (see below).

Step 4: Determining the Objective

We have our final model, it looks reasonable, but we're not done. Our goal was never to produce a good AUC score — it was to help Zidisha's organizational objectives. Their goal is to allow or deny a loan based on risk score. But at what risk score do we deny people a loan, and why?

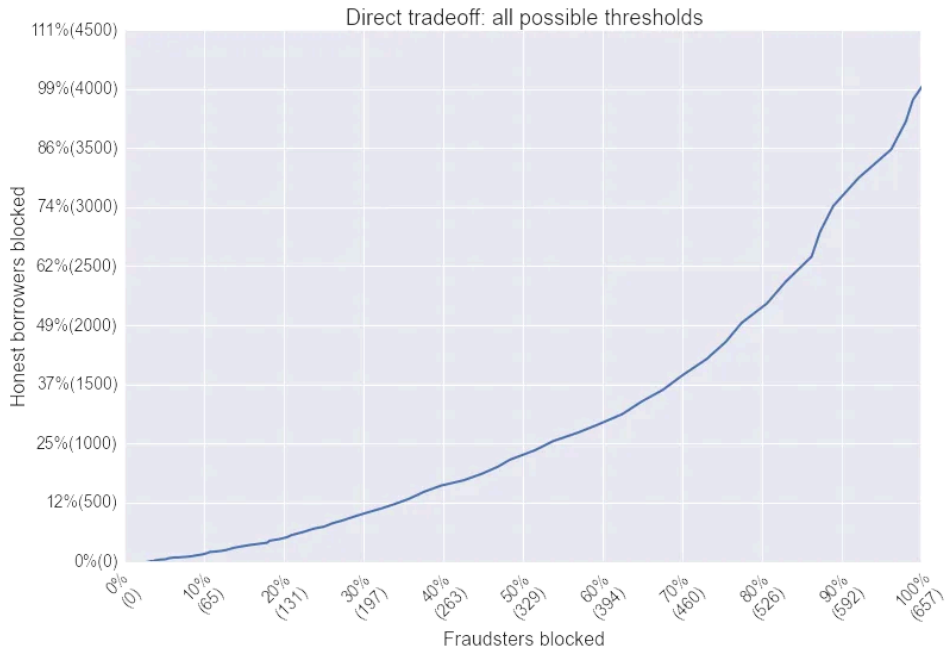
If this were a normal lending agency, we would (roughly) just allow borrowers that we expect to be profitable (e.g. do some math based on interest rate and default probabilities). However, Zidisha is looking to help people in need, not produce financial return. These loans are not gifts, though, and it's not acceptable to never be repaid.

Zidisha has two interests that we have to trade off:

1. Give loans to as many good people as possible.
2. Reduce fraud as much as possible.

In the extreme, we can accomplish either. With a very lenient threshold, everyone gets a loan, but fraud persists unabated. With a very stringent threshold, fraud is squashed, but nobody gets a loan. The in-between is what's interesting.

As an outside agency, we can't (and shouldn't) make this decision for Zidisha. However, it IS our job to inform them well enough to make it. The ROC curve is abstract and hard to interpret for this purpose, so we translated its information into a plot that directly measures the trade-off at every possible threshold value.



We can choose any point on this curve. The bottom-left represents a threshold where everyone gets a loan. All honest people still get their loans,

but so do all the fraudsters. Neither is reduced (hence 0%, 0%). The top-right is the opposite — nobody gets a loan. We successfully prevent 100% of fraud, woohoo! Unfortunately, we also block every honest person who wants a loan. Sad face.

The intermediate points show what happens if we only give loans to applicants with risk of fraud below X% (where we vary X from 0 to 100, drawing this line). For example, with one threshold, we can reduce fraud by 20% but also block 5% of honest borrowers. At another threshold, we can reduce fraud by 50% and block about 20% of honest borrowers. Which is better?

It's hard to say, as we're trading apples for oranges. In parentheses, the axes also show the raw number of people associated with each percentage. So moving from a 20% fraud reduction to 50% will block about 200 additional fraudulent loans and also 200 honest loans. Is that okay? What's the ideal point? When it's not dollars-vs-dollars, this is a judgment call. Before deploying, someone (on the client side) has to make the hard decision of how many apples are worth an orange.

Step 5: Productization

The final question is to how to go from this model (running in an ipython notebook on a laptop) to a live scoring system in Zidisha's production system. We follow projects through until they're live; handing off some model parameters or charts is not where our work stops.

At first we built a basic cloud API for them. (We spun up a single-core EC2 instance with a light Flask/gunicorn webserver handling JSON requests/responses, redis for message queueing, and a custom serving process keeping the model hot and reading/writing scoring requests to the queue.) After discussing who would maintain it, we realized that with Zidisha's current modest throughput (tens of applicants per day, so << 1 QPS), the whole thing was overkill. Waiting a minute to score an application is not

a big deal, so we simply wrote a python script for Zidisha to run locally for each new applicant. The script that loads the model from disk, computes the borrower features from a series of db queries, scores the borrower, and writes the scoring results to another table whose schema we designed. It turns out that sometimes (usually, even), fancy architectures are just not required.

Further Thought

We talked about trade-off between fraud prevention and giving loans to people in need, but there is always more complexity. For example, in addition to loan availability there is also the question of capital availability (how many loans can be funded?). If capital is especially tight on a given month and only half the loans would be able to get funded, then it's OK to tighten the application approvals, ensuring that the limited funds are directed to the least-risky borrowers. Having such a dynamic threshold adds extra challenge, but we need to consider all real-world facets of the problem we're solving.

Conclusions

There is no "typical" Bayes project, but this one spans a broad range of things we might need to do. We researched the relevant domain of the social problem in question (microfinance, poverty, credit risk, fraud). We rolled up our sleeves to collect, extract, and convert the client's dataset into a clean and intelligible format. We brainstormed and tested different features, algorithms, and parameters to find the most appropriate model for our use case, carefully evaluating our performance. We translated the model's value into real business concerns for the client, making critical decisions about its use. Somewhere in there we took a break and ate ice cream. Finally, we helped with the surgery of getting this model into a live, beating-heart system until it breathed air.

Social problems are messy, complicated, and subjective to measure, but that's exactly why they're so crucial to work on. Many social issues may never

be fully “solved,” but by amassing teams of talented and deeply motivated individuals, we can get surprisingly far.

Originally published at www.bayesimpact.org.

Data Science Credit

Written by Everett Wetchler

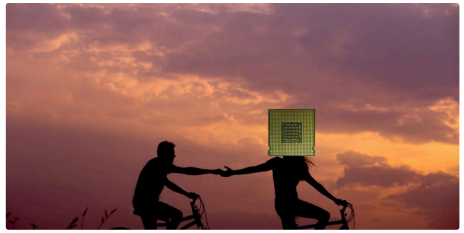
116 Followers · Writer for Bayes Impact

Follow

More from Everett Wetchler and Bayes Impact

Everett Wetchler in Bayes Impact

Announcing our first product to bridge the divide between police...



Pascal Corpet in Bayes Impact

Bob AI: How Computer Chips Have a Social Impact

Today, with the California Department of Justice, we launched URSUS — an all-digital...

Sep 22, 2016 50

To serve many people efficiently, Bob is using AI, artificial intelligence. If this rings like a...

Mar 11, 2020 8

Sinclaire Prowse in Bayes Impact

Working Faster to Harness the Power of Big Data in Government

There is growing consensus that big data will radically change the way we address...

Sep 19, 2017 1

Pascal Corpet in Bayes Impact

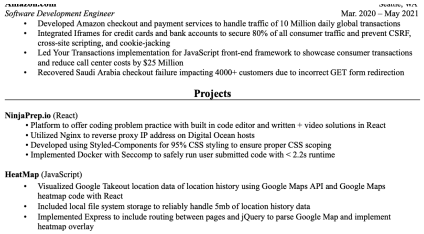
Impact vs Complexity: my answer to engineers eager to have a soci...

A lot of people come to me asking “Where can I apply my knowledge in engineering to...

Dec 4, 2019 176 1

See all from Everett Wetchler | See all from Bayes Impact

Recommended from Medium



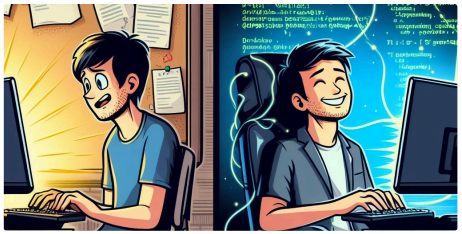
Software Development Engineer Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by $25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)
- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

HeatMap (JavaScript)
- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay



Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

1-page. Well-formatted.

May 31 · 23K · 454

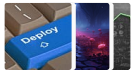
Abhay Parashar in The Pythoneers

17 Mindblowing Python Automation Scripts I Use Everyday

Scripts That Increased My Productivity and Performance

Aug 25 · 7.9K · 75

Lists



Predictive Modeling w/ Python

20 stories · 1559 saves

Practical Guides to Machine Learning

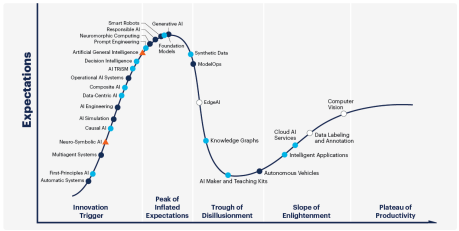
10 stories · 1893 saves

Coding & Development

11 stories · 824 saves

ChatGPT prompts

48 stories · 2038 saves



Vishal Rajput in AIGuys

Why GEN AI Boom Is Fading And What's Next?

Ethan Duong

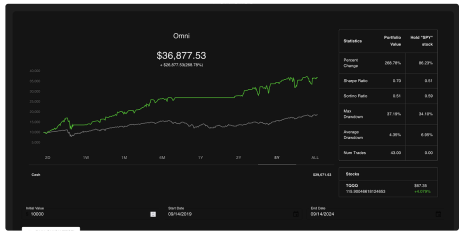
Stop aiming at "Data Analyst" position

Every technology has its hype and cool down period.

Sep 3 1.7K 55

You won't be a Data Analyst and that is ABSOLUTELY FINE!

Jul 14 553 23

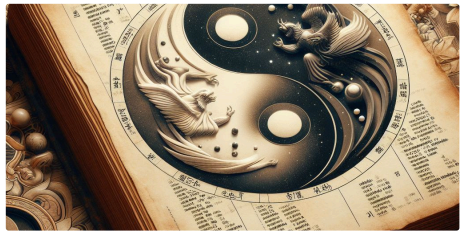


Austin Starks in DataDrivenInvestor

I used OpenAI's o1 model to develop a trading strategy. It is...

It literally took one try. I was shocked.

Sep 15 2.6K 83



Aaron Freeman in Soul Journey Publication

THE SEVEN CONCEPTS OF THE SPIRITUAL CONSTITUTION

CONCEPT OF OPPOSITES

Apr 30 42

See more recommendations