**Script**

Hello, my name is Zach Shim, for my Capstone I made a web application called Fakeology that detects fake reviews in Amazon products

**Purpose**

* So, just some background about the topic
* I’m sure some of you, when you go on Amazon, you look at the ratings and reviews of a product before buying it.
* That’s natural, because you want to believe in what you’re buying.
* That’s why its advantageous for sellers to have good reviews on their products
* But this also opens the door for malicious sellers to make dishonest reviews that may not accurately reflect the views of real people.
* Therefore, when you’re on amazon looking at the reviews of some product, it can be difficult to know whether a review you’re looking at was written by a dishonest person or even a bot.
* And it’s also impractical to closely analyze thousands of reviews yourself for the possibility of finding one that may seem out of place.
* Having some automated system, that can recognize patterns in reviews to help determine if they’re fake or not, can help shoppers have more confidence in what they’re buying.

**Project Background**

* The faculty research I worked on involved scaling a legacy project done by a previous graduate student that detects fake reviews from a static Amazon dataset.
* The basic flow of the classification system involved a collection of scripts taking reviews as input, running some data analysis algorithms on the text, and outputting relevant information about the product’s credibility.
* The system used three methods for analysis:
  1. detection of duplicate reviews,
  2. detection of incentivized reviews,
  3. and detection of anomalous reviews.

**Requirements**

* These were the requirements I set at the beginning of the project.
* The main objective I sought out to do was to create a full-stack application using Django as my main web framework.
* On the front-end I would use HTML, CSS, and Javascript to create three screens that centered around the core functionality of the application.
* Screen 1 would be a redesign of the original project’s interface, which allowed the user to analyze data from the static Amazon dataset.
* Screen 2 would be a new feature that would allow users to paste a product link and see real-time data scraped from the product site’s page
* Screen 3 would show the statistical results from analysis from either Screen 1 or 2
* In terms of the backend, I would be giving the system a structured architecture rather than it just being a loose set of scripts by leveraging Django and developing a database API that encapsulates SQLite operations that is used in the backend analysis work.

**Approach**

* And you can see here, Django nicely encapsulates the relationships and transitions between the front and backend.
* A user would send a request from their browser, then Django would process that request and send it to a view, which is just a script that maps a template (which is pretty much an HTML file) with data from the backend. It would then do all the heavy lifting in generating a response once analysis is done, which is then returned to the browser.
* Depending on which screen the request came from, the processing in the backend would differ.
* So if a request came from Screen 1, which was the static data analysis, it would simply pull the data as is from the database. This is because, in the previous project, all scripts had to be run on the data by an admin before presenting it to the user. This was a part of my motivation for scaling the system, so that data can be analyze and served dynamically.
* If a request came from Screen 2, which was the paste a link page, the request would be processed in a view, which would then send a signal to a scraping script that would pull relevant data from the link’s product page (like product title, category, rating, and review times) and send this freshly scraped data to a pipeline of pretrained models for analysis.
* Some calculations are made, sent to the database, and then pulled into a view and placed into a relevant template.

**Result**

Screen 1

* Here are a couple screenshots showing off the first screen of the final website’s design. One thing I wanted to highlight was this chained search bar
* Basically, I would use ajax requests which would pull data from the backend and asynchronously present it to the user based on the category of choice, and would also autofill possible product ASINs based on what the user was typing.

Screen 2

* Here are a couple screenshots showing off the second screen of the final website’s design.
* One thing I wanted to highlight here was the error/failure capabilities of the system, as in if it wasn’t able to process your request in any way, it returns an error message.

Screen 3

* This is a typical example of the type of data you’d see in the results screen.
* You would get a table of data pointing out different evaluation metrics as well as their associated graphs highlighting important data like rating anomalies or where there was a large influx of reviews and when it happened.

**Solution**

* I focused on implementing two machine learning methods for my research:
  + One was improving the efficiency of similarity analysis using an unsupervised machine learning model
  + The other was adding supervised sentiment analysis
* The general processing of data was similar for both methods, and followed this general pipeline of text normalization, vectorization, transformation, and then estimation.
* In my case, the data loader would be the review data being scraped from an Amazon product page.
* And then, after collecting this corpus, there are typically a number of preprocessing steps we want to do, which makes up the normalization stage.

**Normalization**

* There are many possible methods you can use for normalization, like stemming, entity recognition, removing stopwords, chunking, and lots of others.
* In my case, I used tokenization, lemmatization, and also removed punctuation.
* Pretty much, at this stage, each review is broken down into a list of words, and each word is broken down into its base dictionary form. This process is called lemmatization and each word is called a lemma.
* We then create a dictionary of lemmas, by assigning a unique integer ID to all words appearing in the corpus.
* Therefore, it can be said that a dictionary is made up of ‘n’ distinct words, which also means, each document is represented as an N-Dimensional vector, or you can also say that each review vector has N feature instances.

**Vectorization**

* Next is text vectorization. In order to perform machine learning on text, we need to transform our documents (or in my case, reviews) into a numeric feature space, because machine learning algorithms only know how to work with numbers, not strings.
* This process is called feature extraction or more simply, vectorization.
* Representing documents numerically gives us the ability to perform meaningful analytics and also creates the instances on which machine learning algorithms operate.

**Bag-of-Words**

* Here is example of vectorization I used called bag-of-words, which is just a numerical representation of text.
* In this representation, each review is represented by a single vector, and the frequency of each word in the review, makes up the elements in the vector according to the vocabulary in the dictionary.

**TFIDF**

* While convenient and simple, we can actually still improve from bag of words model.
* For example, two reviews could have the exact same words, but have completely different contexts.
* A better approach would be to consider the relative frequency or rareness of tokens in the document *against their frequency in other documents*.
* The central insight here, is that meaning is more likely encoded in the *rarer terms from a document*.
* This is the essence of the TFIDF text representation.

**Transformation**

* Different models will require different vector spaces, so I found that I often needed to transform from one vector space to another depending on the model I used.
* So, for example, for similarity, I changed from the tfidf vector space to another space called latent semantic indexing, because again, different vector spaces have different advantages.

**Similarity**

* For estimation, some heuristic is used for generating a score.
* For example, for similarity, the library I used utilized a cosine similarity heuristic.
* So, suppose the angle between the two vectors was 90 degrees. In that case, the cosine similarity would have a value of 0.
* As the cosine similarity measurement gets closer to 1, then the angle between the two vectors A and B gets smaller.
* The higher the score, the more similar two documents are.

**Sentiment**

* Sentiment involved binary classification, which means that the model outputted a score of between 0 to 1 for each review. If a review was marked was closer to 0, it was more negative, if it was closer to 1, then it was more positive.
* This whole pipeline of processes nicely encapsulates general modeling in machine learning.

**Challenges**

* It was often difficult finding the right machine learning models to use, because there are many great libraries and diverse sets of algorithms out there.
  + In the case of similarity, I found that the Gensim library was an easy to use and scalable solution for this problem and could quicky process many reviews in a relatively short amount of time.
  + For sentiment, I fortunately found a large dataset of tagged review data I could use for supervised analysis. I used the Spacy library, where they have a text classification model available.
* Another challenge I ran into was scraping data.
  + Scraping a website with no bot detection is a fairly simple process, but Amazon has very robust bot detection in place, so I had a lot of trouble getting around their captcha.
  + I eventually found a solution by rotating proxies and user agents, but it’s not 100% reliable, and I still run into captchas every so often, which is something I mention in future work for the project.
* The last thing was learning several new languages and libraries in such a short amount of time.

**What I have learned**

* + As you can see in this slide, I’ve learned so much in just a few-month timespan. Some examples I’ll point out are how to use several web scraping libraries like BeauitfulSoup, Scrapy, Lxml, and Selenium.
  + I also learned a lot about web development, and also machine learning theory, mostly in text classification. So, things like data frames, vectorization, natural language processing, and so on.

**Next steps**

* For future work, I think the system could always be improved on. Trying out different machine learning algorithms and libraries to optimize classification scores like accuracy, precision, recall, or f1 score will only prove to make the fraud detection more reliable.
* And like I mentioned earlier, the web scraper could be improved to avoid getting caught by captchas. I think using a paid proxy service that takes care of managing and rotating proxies for you would most likely be the easiest solution.
  + Another solution could be to just make a captcha solver, and I’m sure there’s other solutions as well.
* Another thing I feel like could be implemented is a loading bar. Scraping and analyzing data with multiple algorithms over a large dataset takes a long time to do. So something I wasn’t able to do but wish I could, is give the user some indication of feedback that the system is making progress.

**Helpful courses**

* Some helpful courses that were instrumental to my research were data structures and algorithms – I used a lot of concepts like inheritance, polymorphism, as well as class patterns.
* I also did a lot of UML diagramming like flow charts, class diagrams, and use case diagrams.
* 475 was also really important because it’s where I learned SQLite as well as how to do ER diagrams and table mappings.
* Another class I utilized when fleshing out screen designs was CSS 480. I did a lot of sketching, using the 10 plus 10 method, I also did wireframing, and kept a lot of design principles in mind like affordances or the gestalt laws of organization.

**Question & Answer**

And thank you that concludes my presentation. Please let me know if anyone has any questions, and thanks again for listening.