**How to make a recommender system: For a beginner!**

Part 1: Intro and Data Exploration

When I first started learning about machine learning, one of the parts that I found the most challenging was the lack of information on how to structure a project, particularly when do to what steps and why. This blog is meant to provide both some basics on the code, but more importantly a clear, non presumptive explanation of why each step is occuring, to help the reader learn along with me.

**Intro to recommender systems**:

Reccomender systems are used everywhere - from netflix to amazon to spotify, they are working to curate content for you based on who they think you are. Or rather, based on what your peers like and what you’ve shown you like in the past.

Thus, there are two types of systems:

**Collaborative filtering**, which is basically a way of collecting information about preferences from users and then looking to see how similar those preferences are to other users. Aka, if me and Lebron James both buy the same brand of banana costume, I might find myself being recommended a lot more basketball related products in the future.

Obviously this approach relies on a lot of data from users about their specific likes and dislikes, so is often employed by companies that have access to this kind of data - like Spotify (you’ve listened to thousands of songs) and Amazon (it’s a monopoly so, yes, you’ve probably bought thousands of products).

**Content-based filtering** is another common approach which uses the users past behavior to recommend other products to them. Aka - Did you previously buy a brand new air purifier? You might enjoy *another* air purifier! This approach also relies on user ratings - think Amazon’s search results that usually display the highest rated products at the top.

For this project, we’ll have to determine which type makes the most sense for our particular dataset. One of the biggest shortcomings for new data scientists is trying to fit models on data that just doesn't have the right information - you have to design the project around the data, otherwise you end up with a square-peg-round-hole situation!

**The Project:**

The data for this sample project was obtained from a kaggle competition for creating recommender systems, meaning that it is perfect for learning with. It is comprised of information about articles and user preferences around those articles. There will be much more in depth exploration of the data below, but first I want us to start thinking about project planning.

In planning for this project, there are a number of things to consider immediately.

What is the question I actually want to answer?

* Even when you think you have a good sense of what you want to find out, it’s best to write it out to ensure there is no ambiguity in your project’s goal. Nothing is worse than going into a project realizing you don’t actually know what you’re looking for!
* Here, we know want to ***“create a recommender system that will recommend articles to users based on their preferences for other articles”***. Even this seemingly straightforward question begs the next question - how exactly will it make this recommendation, which leads us to ….

Which model can be used to best answer that question?

* Its best to brainstorm a number of ideas and narrow it down. Chris Albon has an excellent graphic detailing what common model options are available to answer different types of problems. Here, we know that we’re focusing on a recommender system, which seriously narrows our options:

**Install Dependencies**: As with all python projects, we need to think about what dependencies we will need for this project. Doing this also provides a way to start mapping out the project because it involves consideration of the tools and models that will be used.

In this project, we’ll be using:

* Standard python dependencies like numpy and pandas
  + Pandas: to let us interact with the csv data as a dataframe
  + Numpy: to
  + Matplotlib: for graphs and visualizations
* Natural language processing tools for the text features.
  + Spacy: stopwords
  + Sklearn.feature\_extraction: TfidfVectorizer
* Several different sklearn packages:
  + Preprocessing: MinMaxScaler
  + Model\_selection: train\_test\_split
  + Metrics.pairwise: cosine\_similarity

import numpy as np

import scipy

import pandas as pd

import math

import random

import sklearn

from nltk.corpus import stopwords

from scipy.sparse import csr\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from scipy.sparse.linalg import svds

from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt

**Data Exploration**:

Before anything further is done, it is extremely important to LOOK AT THE DATA. Even initial exploration can help to start you thinking about the data story, and perhaps more importantly, sets up a framework in your mind of what is missing from that story and how you might go about filling in those gaps.

Thinking about the data and how it relates to the real world is vitally important - it will inform decisions you make about how to deal with everything from missing values to how to evaluate your model.

Here, we have two datasets: shared\_articles.csv and user\_interactions.csv.

It is important to note immediately that these datasets both contain a user id, which means that they can be connected on that id using a join - aka, there is an easy way to make those tables talk to one another.

Shared\_articles.csv provides information about articles that are shared on the platform.

A quick df.head() provides us with a snapshot of the first five rows and all the column headers. Right off, we can see that there is information about:

* **Timestamp**: The date the article was shared, stored as a timestamp
* **Url:** The link to the article itself
* **Title**: Text
* **Content**: Text
* **Language:** Stored as
* Author: information specifically about the author such as

User\_interactions.csv

* **View**: Any time the article has been opened.
* **Like**: Any time the article has been liked.
* **Comment Created**: The user created a comment in the article.
* **Follow**: The user chose to be notified on any new comment in the article.
* **Bookmark**: The user has bookmarked the article for easy return in the future.

**Connecting and storing the data**:

Now that we have some context about what we’re working with, it’s time to consider the best way to physically handle this data.

Are the csv’s large, giving the computer a hard time to process?

In this case, the tables are relatively small. Instead of putting them into a SQL database like MySQL, we can merge them into a single dataframe and work with that information within pandas.

SHOW CODE

**Munging/Wrangling/Cleaning:** Which basically just means getting data into a workable form for the project.

For recommender systems, an important part of the data wrangling process is figuring out what data is most important for making the recommendations. Is a comment of equal value to a like? What about a starred review? Should we always rate a purchase highest?

We also need to figure out how to mitigate a common issue called **user-cold start**, which occurs when there is not enough information about a new user’s preferences to make appropriate recommendations. Imagine a person purchases only

**Cross Validation:**

As in all machine learning, it is vitally important to have a method to evaluate the outcome of the model. This is done through cross validation, the most common method being train-test-split. Here, I split the data into a typical 80-20, which means I ‘held out’ 20% of the data to test the model on later in the process.

**Evaluation Metrics:** Recommendation systems are evaluated in a number of ways:

Top-N accuracy metrics evaluate the the ***accuracy*** of the top recommendations to the user. Basically, for each user