Content-based Recommendations Using Hype Machine Listening History

Mike Winters

August 24, 2015

What is Hype Machine?

- **Summary:** A service that scrapes songs from a curated list of music blogs and makes the songs streamable from a web app and mobile app.

- **From hypem.com:** "*Hype Machine keeps track of what music bloggers write about. We've carefully handpicked a set of 809 music blogs and then present what they discuss for easy analysis, consumption and discovery. This way, your odds of stumbling into awesome music or awesome blogs are high.*"

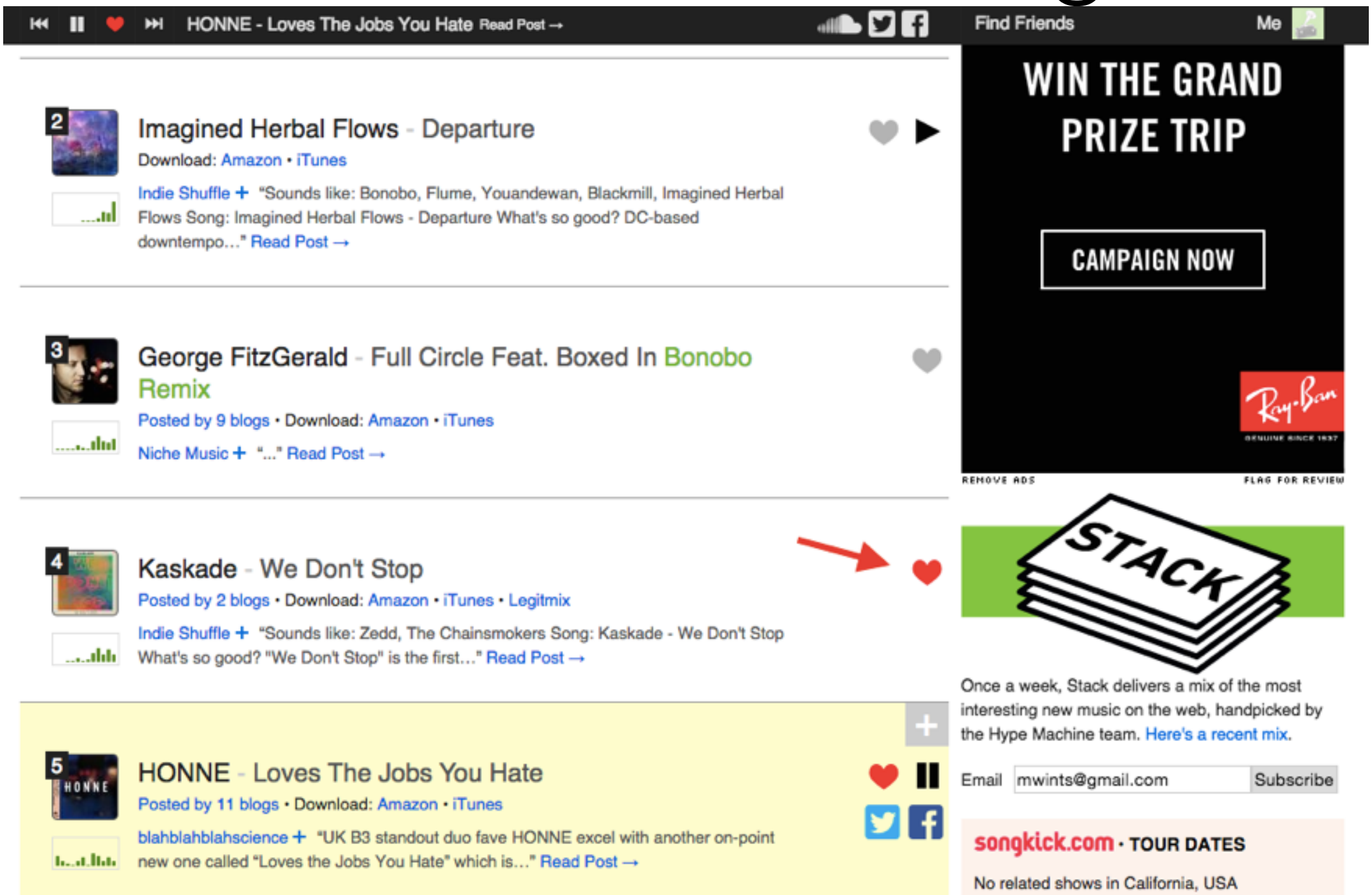
Why is Hype Machine interesting to me?

- **Unique collection:** Many songs that aren't included in larger streaming services

- **Personal relevance:** I've been listening to Hype Machine off and on for 6+ years

- **No recommendations:** Personalization tools on the site are limited, so I could build something I'd find useful

Hype Machine: ‘Popular’ feed & ‘Loved’ flag



The burning question

- ***Using 6 years of my personal Hype Machine listening and 'loved' history, can I build a recommender to identify new songs on the site that I should listen to?***
- Supervised ML; Categorical response; classification problem; content-based filtering
- **The grand vision:**
    1. Build recommender using personal history
    2. Scrape a long list not-yet-heard song IDs
    3. Create a smart list of songs that I'm likely to enjoy
    4. Build a web app that allows other listeners to do the same using their own history (way out of my depth)
- But tonight, we're just going to focus on phase 1

Step 1: ‘History’ and ‘Loved’ Data

- Use requests to grab JSON listening and loved history from URLs
- Create a dictionary with relevant song data; convert to dataframe
- Ready-to-use features: ‘loved count’ and ‘posted count’

```
import requests
from collections import defaultdict
new_dict = defaultdict(list)
for page in range(1,16):
    url = "http://hypem.com/playlist/history/wints/json/"+str(page)+"/data.js"
    song_json = requests.get(url).json()
    del song_json['version']
    columns = ['mediaid', 'artist', 'title', 'sitename', 'loved_count', 'posted_count',\
    'time']
    for column in columns:
        new_dict[column] += [song[column] for song in song_json.values()]
```

Step 2: Tags Pull

- Tags, unfortunately, are buried in the site's HTML and aren't available in JSON feed
- So I used requests, BeautifulSoup, and HTMLParser to build a dict containing media ID and song tags; converted tag list to dataframe then appended tag list to song_history dataframe
- Note: often more than one tag per song

```
import requests
from bs4 import BeautifulSoup
from HTMLParser import HTMLParser
h = HTMLParser()

songids = song_history['mediaid'].tolist()

def songtags_list_2(songids):
    tags_data = []
    for id_ in songids:
        r = h.unescape(requests.get('https://hypem.com/track/'+id_).text)
        soup = BeautifulSoup(r)
        data = {}
        data['mediaid'] = id_
        data['tags'] = [child.text
                        for tag in [soup.find('ul',{'class': "tags"})]
                        if tag
                        for child in tag.findChildren('a')]
        tags_data.append(data)
    return tags_data
```

Step 3: The Echo Nest Pull

- The Echo Nest is a Spotify-owned music service that provides, among other things, quantified song attributes (e.g. ‘speechiness’ or ‘acousticness’)
- Used song search (title, artist) and audio_summary bucket to pull song attributes
    - Created list of URLs that were formatted for Echo Nest API
        - http://developer.echonest.com/api/v4/song/search?api_key=XSJBT3OPV9ZXRHT8J&format=json&results=1&title=Ambition+%28feat.+Meek+Mill+%26+Rick+Ross%29&artist=Wale&bucket=audio_summary
    - Wrote function to get URLs (one per four seconds due to rate limiting) and store song attribute data where available


```
{"response": {"status": {"version": "4.2", "code": 0, "message": "Success"}, "songs": [{"artist_id": "ARXYXJ311A348EFC1A", "artist_name": "Wale", "id":
"SOCXCOR136C58B8C5B", "audio_summary": {"key": 8, "analysis_url": "http://echonest-analysis.s3.amazonaws.com/TR/LPzRIgRA_9V3waWJND83gP-
VXplkx9NvWXYhk2cE_6MnRSy270FejynwZEYMwq-5FTdyIpX7Nr3EklMV0%3D/3/full.
json?AWSAccessKeyId=AKIAJRDFEY23UEVW42BQ&Expires=1439861553&Signature=2ROrFJOZqZ1eAOwg87KbcDEs6zg%3D", "energy": 0.752496, "liveness": 0.141643, "tempo": 162.
367, "speechiness": 0.393506, "acousticness": 0.406961, "instrumentalness": 1.7e-05, "mode": 1, "time_signature": 4, "duration": 302.72, "loudness": -5.105,
"audio_md5": "", "valence": 0.31563, "danceability": 0.530457}, "title": "Ambition (feat. Meek Mill & Rick Ross)"}]}}
```

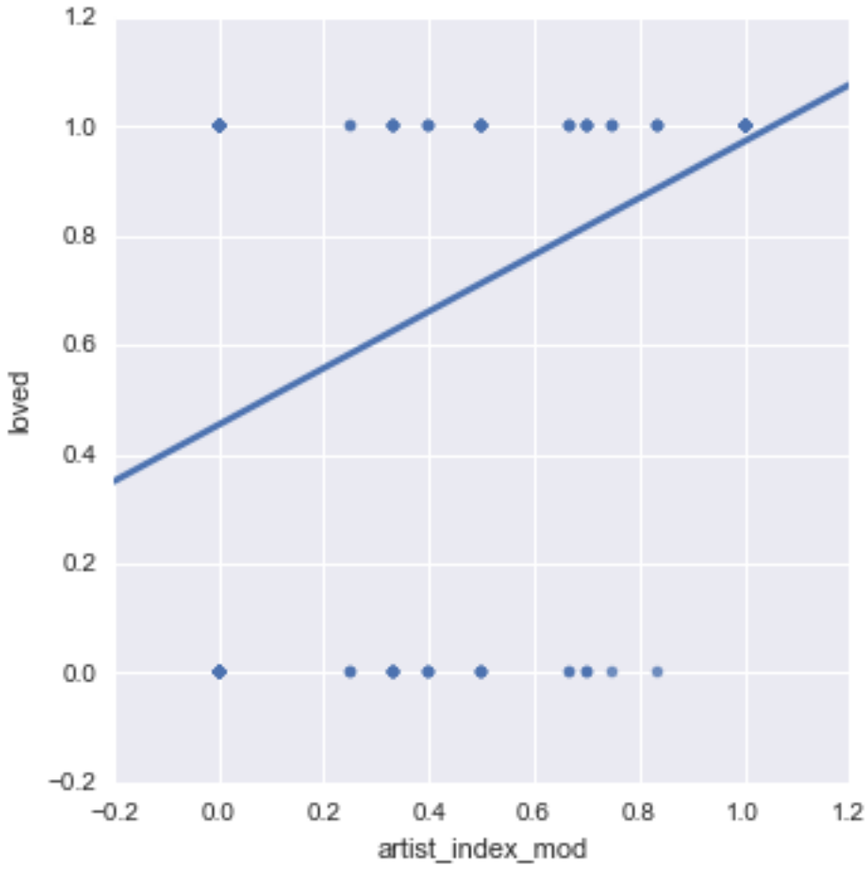
Data Gathering Summary

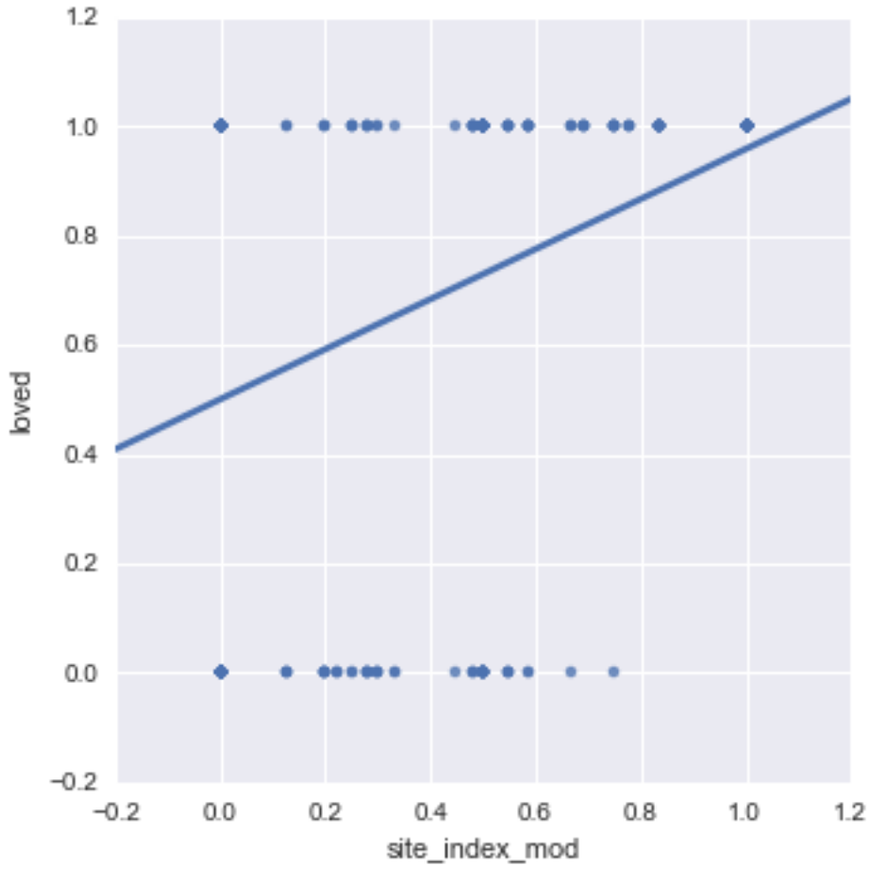
- By the numbers:
  - **850** songs total
  - ‘Loved’ **485/850** songs (57%)
  - Tag data on **759/850** songs (89%)
  - Echo Nest data on **381/850** songs (44%)

Features: Site and Artist Index

- Formula: (number of times a site's / aritst's song is loved) / (number of times a site / artist appears)

  - A site or artist's 'batting average'





Features: Tags Data

- Tags are a bit more gnarly; for 850 songs, there are over 1000 distinct tags, and a single song has anywhere from 0-10 tags

- A lot of tags only show up once and aren't super helpful. For example:
  - "vintage kanye production"
  - "zum durch ikea poebeln"
  - "weed smokers anthem"
  - "yyeeeesssssss"
  - "surprisingly good"

Feature Analysis: Tag Feature Importances (Top 10)

Tag	Importance
soul	0.02910088
mashup	0.024870872
alternative	0.02249014
femalevocalist	0.020801898
cover	0.020474033
rock	0.018013178
indiepop	0.017970319
dreampop	0.01784795
indietronica	0.016985728
altcountry	0.016736501

Features: Tags Data, cont.

- Tagsum: an attempt to combine many tags into one feature

    1. Use decision tree feature importances to determine which tags have predictive value

    2. Create 1 dummy column for each valuable tag

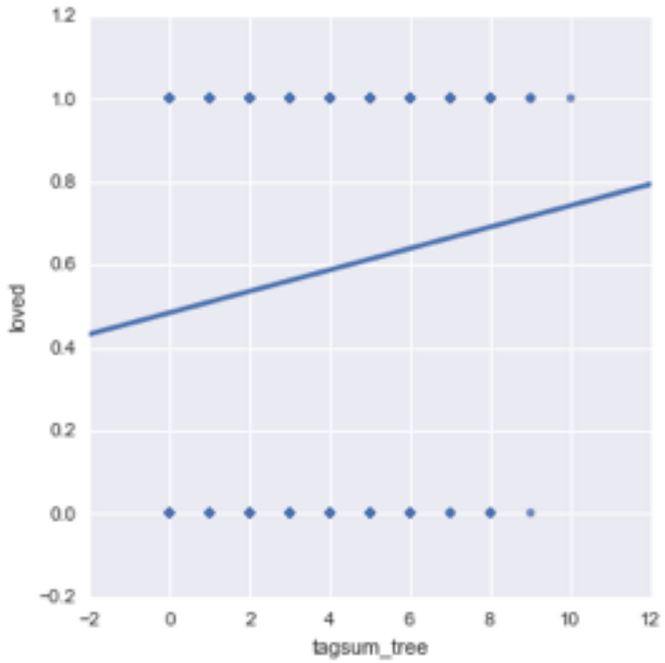
    3. Add a column to sum number of occurrences of valuable tags; this tagsum is what I used in my dataset

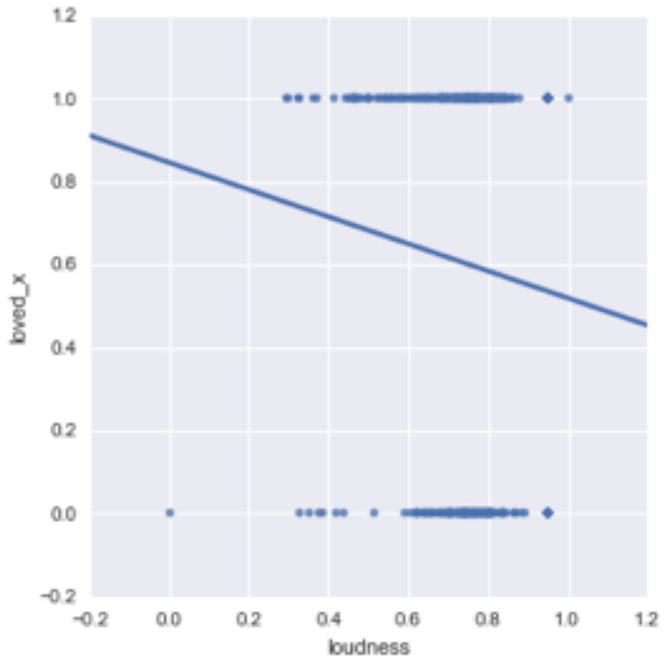
	soul	french	remix	cover	tagsum
Song A	1	0	1	1	3
Song B	0	1	0	0	1
Song C	0	0	0	0	0
Song D	1	1	0	0	2

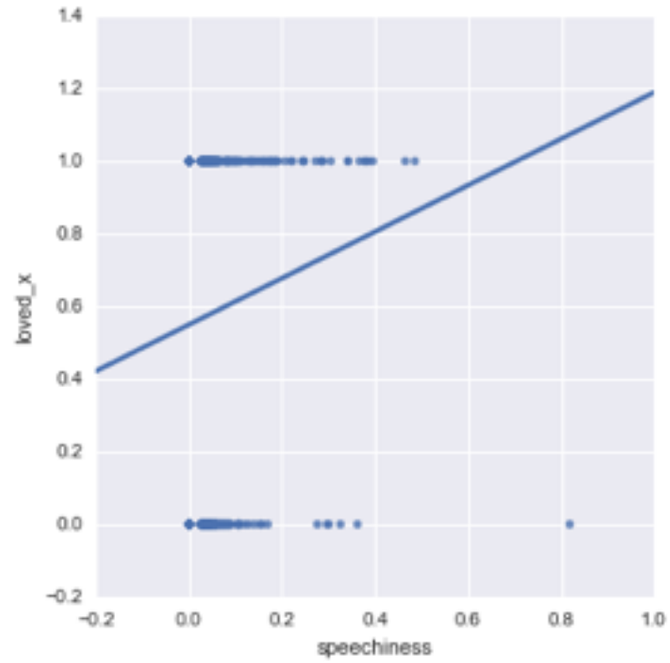
Feature Analysis: Echonest Feature Importances

EN Attribute	Importance
loudness	**0.185245202**
speechiness	**0.141293422**
energy	0.121033888
valence	0.120521514
danceability	0.089479982
acousticness	0.087563681
liveness	0.08152229
duration	0.061721814
instrumentalness	0.054555727
tempo	0.033398619
time_signature	0.023663861

Feature Analysis: Tagsum and Echonest Visualizations







Data Dictionary / Feature Definitions

Feature	Definition
loved_count	# of times a song was ‘loved’ on Hype Machine
posted_count	# of distinct blogs that posted a song
artist_index_mod	an artist’s ‘loved rate’ in my listening history
site_index_mod	a site’s ‘loved rate’ in my listening history
tagsum_tree	# of ‘important’ tags attached to a song, per decision tree
loudness	from Echo Nest: ‘The overall loudness of a track in decibels (dB).’
speechiness	from Echo Nest: ‘Detects presence of spoken words…Values above 0.66 describe tracks that are probably made entirely of spoken words.’

Source: http://developer.echonest.com/acoustic-attributes.html

Model Notes

- Top-performing dataset: 381 songs with Echo Nest data
    - Null accuracy on EN dataset = 61% (a bit higher than 57% on aggregate set)
- Specificity vs sensitivity tradeoff
    - Cost of hearing a song I don’t like is relatively low
    - Cost of missing out on a song I might enjoy is relatively high
    - My data likely contains a significant number of misclassified songs

Cross-Validated Model Scoring

model	accuracy	roc_auc	specificity	sensitivity	notes
logistic regression	76.67%	0.8213	72.22%	81.67%	
knn	75.07%	0.8181	77.77%	78.33%	n_neighbors: 24
naive bayes	74.05%	0.8205	63.80%	86.60%	
decision tree	76.37%	0.8008	69.44%	70.00%	entropy, max_depth: 4, max_feat: 4
SVM	75.85%	0.8255	75.00%	81.67%	kernel: lin, C: 2, gamma: 0.0

What else to explore?

- **Changing tastes over time:** if I break data into ‘eras’, is my model more accurate?

- **Collaborative filtering:** Hype Machine doesn’t require authentication to access users’ listening / loved histories

- Collect usernames; build collaborative model