I. The Problem

Broadly, the problem is to infer, from written text, information about the demographic profile of the writer. For this specific project, I am limiting the text to be used to the text portion of Twitter statuses (tweets), and I am limiting the demographic information to be inferred to gender. In addition to extensions to other demographic attributes and to text from other media, the specific Twitter context suggests possible extensions to using non-text information (e.g. emojis included in a tweet, visual information from Twitter profiles, etc.), but I have made no deliberate attempts to extend the project beyond text.

My imaginary client is a consulting firm (Imaginary Marketing Associates, LLC, or IMA for short) that helps its clients with targeting their marketing efforts. IMA will use inferred demographic information to target marketing efforts for products expected to be of interest to particular demographic profiles.

For a couple of reasons, IMA will be interested in how the predicted demographic information associated with a tweet changes over time: first, because their clients will be reacting in real time; second, because they will be developing profiles of individuals that will depend on how their text is classified at different points in time. For example, if the typical characteristics of male and female tweets change over time, it would add confidence if the same Twitter profile produced tweets at different points in time that were classified as the same gender using models sensitive to different conditions that apply at those different points in time.

I have relied on streaming data to be obtained from Twitter via their API. I accessed these data via a developer account with Twitter, which enables me to access random streams of public tweets.

The approach has involved applying machine learning models to text embeddings. In a preliminary analysis, I used Google's Universal Sentence Encoder to produce embeddings and then applied a gradient boosted decision tree model to those embeddings to predict gender. (I can impute the "ground truth" for gender for some tweets from the display name of the user, using available gender imputation software, such as the Python gender-guesser package. For training and evaluation of the model, I have relied on tweets for which I can impute gender from the display name with reasonably high confidence.)

From a machine learning point of view, the initial problem (gender inference) is a supervised binary classification problem. Later extensions (applying it to different demographic information) could involve multi-class classification, or possibly regression.

II. Initial Data Preparation

Twitter's data stream consists of a continuous random stream of tweets in JSON format. The tweet itself is contained in the "`text`" field, with lots of metadata in the other fields. For example:


```
{"created_at":"Sun Jun 02 02:49:12 +0000
2019","id":1135015480010452993,"id_str":"1135015480010452993","text":"but
any , back to the backwood dilema","source":"\u003ca
href=\"http:\/\/twitter.com\/download\/iphone\" rel=\"nofollow\"\u003eTwitter
for iPhone\u003c\/a\
u003e","truncated":false,"in_reply_to_status_id":null,"in_reply_to_status_id_s
tr":null,"in_reply_to_user_id":null,"in_reply_to_user_id_str":null,"in_reply_t
o_screen_name":null,"user":{"id":1636148010,"id_str":"1636148010","name":"Ben
Dover","screen_name":"flexthegod_","location":"the
trap","url":null,"description":"i cant hear you , im high. #LE\u00d8 RIP
Anthony \u2018Tony\u2019 Edward Stark\ud83d\ude4f\ud83c\
udffe","translator_type":"regular","protected":false,"verified":false,"followe
rs_count":425,"friends_count":397,"listed_count":6,"favourites_count":3862,"st
atuses_count":15241,"created_at":"Wed Jul 31 19:05:54 +0000
2013","utc_offset":null,"time_zone":null,"geo_enabled":true,"lang":null,"contr
ibutors_enabled":false,"is_translator":false,"profile_background_color":"C0DEE
D","profile_background_image_url":"http:\/\/abs.twimg.com\/images\/themes\/
theme1\/bg.png","profile_background_image_url_https":"https:\/\/
abs.twimg.com\/images\/themes\/theme1\/
bg.png","profile_background_tile":true,"profile_link_color":"0084B4","profile_
sidebar_border_color":"000000","profile_sidebar_fill_color":"DDEEF6","profile_
text_color":"333333","profile_use_background_image":true,"profile_image_url":"
http:\/\/pbs.twimg.com\/profile_images\/1129952119413497856\/
KRLhae1F_normal.jpg","profile_image_url_https":"https:\/\/pbs.twimg.com\/
profile_images\/1129952119413497856\/
KRLhae1F_normal.jpg","profile_banner_url":"https:\/\/pbs.twimg.com\/
profile_banners\/
1636148010\/1558226759","default_profile":false,"default_profile_image":false,
"following":null,"follow_request_sent":null,"notifications":null},"geo":null,"
coordinates":null,"place":null,"contributors":null,"is_quote_status":false,"qu
ote_count":0,"reply_count":0,"retweet_count":0,"favorite_count":0,"entities":
{"hashtags":[],"urls":[],"user_mentions":[],"symbols":
[]},"favorited":false,"retweeted":false,"filter_level":"low","lang":"en","time
stamp_ms":"1559443752659"}
```


That's the record of a single tweet. The text itself is limited by design to 280 characters, and long tweets even have their text truncated in the data stream, but with all the metadata included, the tweet takes up almost a whole page.

Initially I wasn't sure which fields would be of interest, and I decided to collect whole records, but to make the data more manageable I eliminated tweets that weren't relevant to the project.

Specifically, I eliminated deletes (records merely indicating deletion of a previous tweet) and retweets (repetition of a previous tweet by a different user), both indicated by the presence of a specific field in the record; I included only tweets that had "`lang`", "`user`", and "`text`" fields, and for which the language indicated was English; I eliminated tweets with no "`name`" subfield or no "`id`" subfield under "`user`"; and I eliminated tweets for which the gender indicated by the "`name`" subfield was inconclusive. Overall, this process eliminated about 95% of the original records.

I used a package called `gender-guesser` to categorize tweets as male or female based on the display name (the "`name`" subfield of the "`user`" field).

I collected files of up to 3000 tweets each. (Sometimes the stream is broken, so the number of tweets that can be collected consecutively is not reliable. However, the majority had the full 3000.) The code for collecting a file of tweets is in `get_tweets.ipynb`. A quick initial look at some of the tweets is in `process_tweets.ipynb`.

I decided to index the tweets by timestamp and user ID, to facilitate validation in a way that avoids leakage. (This information was in the "`created_at`" field and the "`id`" subfield of the "`user`" field.) Since I had already eliminated irrelevant records in the data collection process, I had only to select relevant fields from the remaining records. Since the project was defined as predicting gender from text, the only relevant fields (aside from the user ID and timestamp to be used as record identifiers) were the "`text`" field (the tweet itself) and the "`name`" subfield of "`user`" (used to ascertain the gender label).

To clean the text, I used a function copied from Timothy Renner on Github to eliminate non-text elements (links, hashtags, mentions, media, and symbols). The code to read the tweet files, aggregate them, extract the relevant data, clean the text, and supply gender labels is in `aggregate_tweets.ipynb` and `utils.py`.

After getting an initial corpus for the primary analysis, I continued to collect files of tweets for use in a subsequent online learning procedure, but (as of 2019-07-12) these files have not yet been processed.

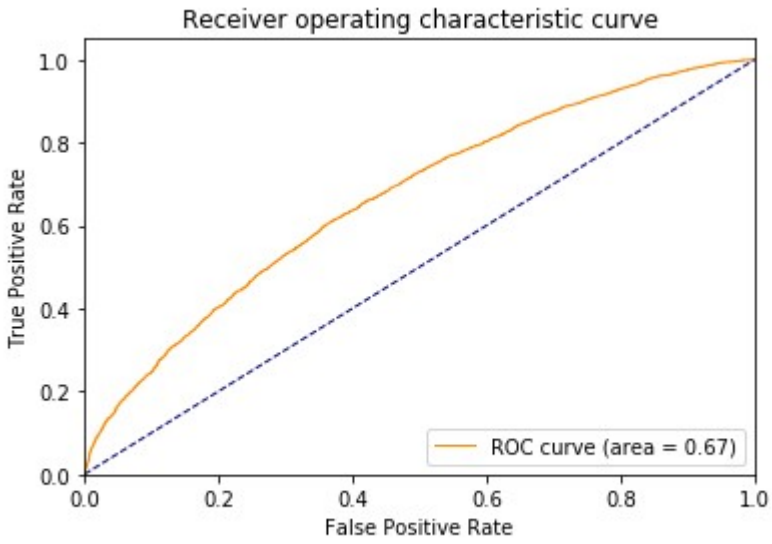
To minimize leakage, I decided to split the corpus into train, validation, and test sets in a way that entirely avoided overlap by time period or by user ID and where the time sequence from train to validation to test was monotonically forward. Also, to avoid class imbalance issues and to facilitate interpretation of the results, I decided to downsample the modal gender (male) in all three sets to create sets with equally balanced labels. The code to split and downsample the data is in `utils.py` and `split_initial_tweet_corpus.ipynb`.

III. Initial Analysis

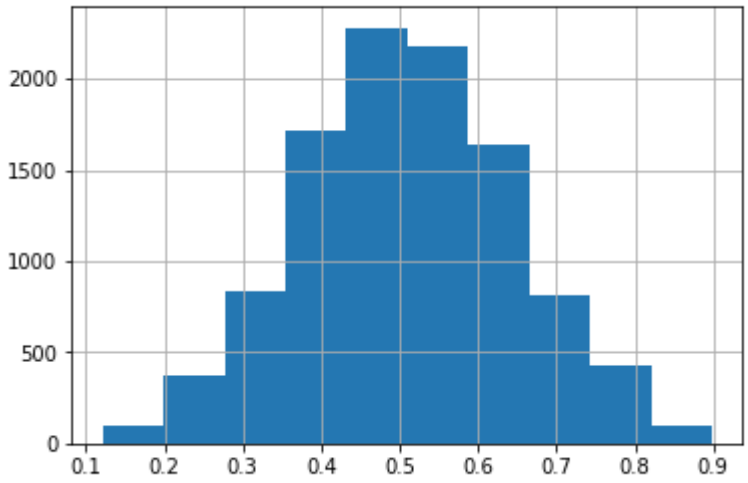
I began by using Google's Universal Sentence Encoder to create sentence embeddings. Though my initial attempts used the standard version (linked above), which uses a deep averaging network to create sentence representations, I soon settled on using the "Large" version, which applies a transformer approach. The Large version (henceforth "USE-L") gets better results and turns out to be only a little bit more resource intensive. For each tweet ("sentence"), the output of the encoder is a 512-dimensional embedding vector. The encoder can be called with its pre-trained embedding weights to produce such an output, or it can be used as part of a deep learning model with user-specified layers on top, and the weights can be fine-tuned for the specific application. The fine-tuned model can also then be used to produce embeddings as features for input to a different model.

I tried several different types of prediction models (multi-layer perceptrons, gradient boosted decision trees, and logistic regression with various kinds of preprocessing) using the embeddings from USE-L. In general the results did not differ greatly from one another in overall quality. In the initial analysis, the best results seemed to be from a regularized logistic regression of quadratic features generated from principal components of the embeddings. This is the model I have used for most of the analysis. As a demonstration, I made a Kaggle kernel that applies this model using the pre-trained embeddings. As a baseline, the test set accuracy is around 61.8 percent. (Hyperparameters—namely the number of principal components and the regularization parameter—were tuned using the validation set.)

Here's what the ROC curve for the test set looks like:



Here is the distribution of predictions:



And here is how well the predictions performed:

		count	accuracy
minprob	maxprob		
0.0	0.1	0	NaN
0.1	0.2	105	0.885714
0.2	0.3	534	0.788390
0.3	0.4	1646	0.664034
0.4	0.5	2743	0.555231
0.5	0.6	2766	0.548445
0.6	0.7	1759	0.627629
0.7	0.8	707	0.772277
0.8	0.9	190	0.842105
0.9	1.0	0	NaN

Essentially, for most of the tweets, the model realizes it doesn't have enough information to make a confident prediction, but for some of them, it is appropriately confident. It's encouraging that the actual accuracy within each prediction range corresponds to the predicted probability within that range. The model seems to be doing a good job of predicting probabilities, even if most of those probabilities do not contain much information.

The next step was to fine-tune the embeddings. This process required adding an additional layer or layers on top of the embeddings to produce a complete deep learning model that could be optimized. The procedure was then first to fit the model with the embedding weights frozen and then to make the embedding weights trainable and fit the model again starting from the

optimized weights for the upper layer(s). The structure that seemed to work best was a single 512-unit hidden layer on top of the embeddings with a large dropout (drop=0.8) going into the final prediction. Given the large number of parameters (over 200 million) and the small amount of data (34146 training samples), I found that the model always overfit on the second unfrozen epoch, even with such extensive dropout and even with very slow learning. The final model therefore trains for a single epoch to fine-tune the embeddings after training for several epochs to generate initial weights for the upper layer.

The fine tuning model (`twitgen_use_large_best.ipynb`) has a test set accuracy of about 62.8 percent. The performance can change from one run to another, but in general it is comparable to the best performance I have obtained using the fine-tuned embeddings as features in more comprehensive models. However, the process of tuning the embeddings is costly, so for purposes of online learning it makes sense to use the already fine-tuned embedding weights rather than re-fitting the whole model. It's an open question whether the diversity allowed by including features generated by other embedding models will make the online learning process more robust.

Using the fine-tuned embeddings in my preferred logistic model (using quadratic features of principal components, see `lr_poly_corpus_tweets.ipynb`) produces a test set accuracy of only 62.0 percent, better than with the pretrained embeddings but not as good as the full deep learning model. However, the advantages of the logistic model are that it allows me to include additional features conveniently and it is somewhat easier to interpret.

I made some attempt at interpretation in another notebook (`analyze_lr_poly_results.ipynb`, which uses an earlier version of the model). Sentence embeddings do not submit easily to human interpretation, and they are not reliable from one run of the model to another, but there may be some insight that can be gained from looking at the principal components. Under the heading of "Feature Importance", I found the components that had the greatest individual impact in producing predictions near 1 or near 0 ("most masculine components" and "most feminine components") and then looked at the tweets that scored most highly on those components. Subjectively, in this particular analysis, some important "masculine" components seem to represent "argumentativeness", "inspirational talk", and "male celebrities", while some important "feminine" components seem to represent "negative interpersonal emotions", "geography", "tearfulness", and "parenthood and making a living." Subsequent runs, however, show that your mileage may vary quite a bit.