

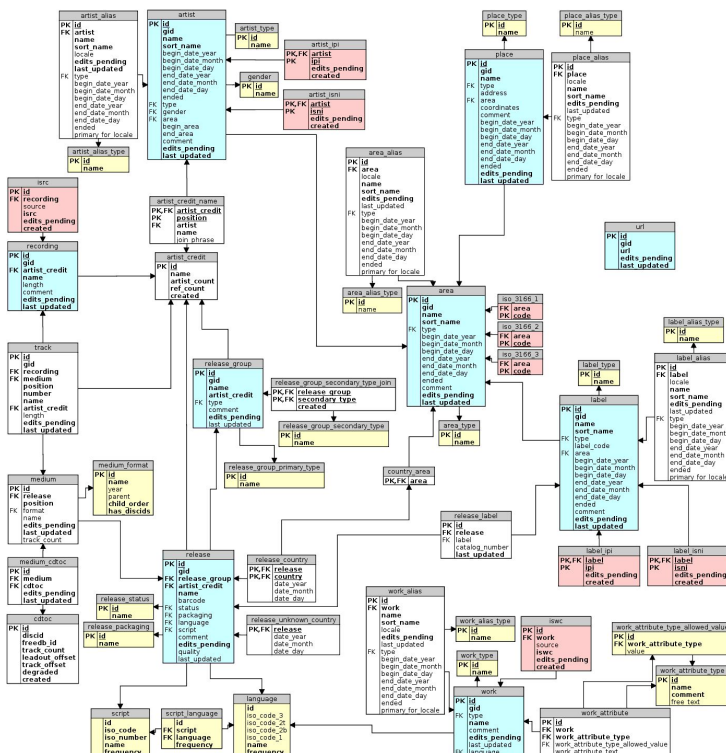
INTRODUCTION

The purpose of our application is to allow users to directly interact with the database and select a subset of training data to train a neural network. The neural networks is set to predict the year in which a song was written using information such as tempo, time signature, lyrics, artist, etc. Our application utilizes Tensorboard as our frontend interface to help visualize the performance of the neural networks with respect to the individual inputs. The Python backend parses user input to construct corresponding queries to request data from the SQL database.

IMPLEMENTATION

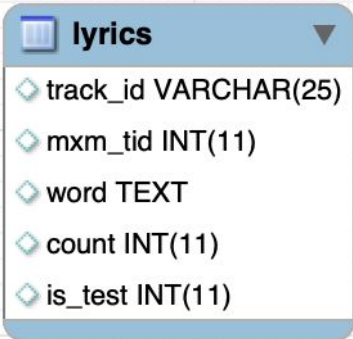
The Dataset

Our database combined the use of three separate databases with song metadata, which we could join across using song IDs. The first was the MusicBrainz dataset, which came pre-normalized, and contained 11 GB of song metadata. Getting the dataset required cloning the mbzdb package (<https://github.com/elliottchance/mbzdb>) which contained a Perl script to download the data and load it into SQL. Indices were created on the appropriate data fields. The following is the schema of the dataset:



The second dataset was the Million Song Dataset, which was downloaded from The Laboratory for the Recognition and Organization of Speech and Audio (LabROSA). We used the 10,000 song subset of the data. Importing the data into MySQL required converting the information from the hdf5 format. We wrote a shell script and Python script to automate this process, and the Python script used numpy and pandas to convert the data into a form that could be loaded into a SQL table. Normalization of the database required moving artist information out of the megarelation. The largest challenge in normalization was that about half of the artist metadata was functionally dependent on the artist ID, but half of it was not, including artist name and other information that would be expected to. This was due to the fact that different songs might feature the same artist, but also have other collaborators. In order to solve this problem, we created two separate artist relations and our own artist ID which would uniquely determine all the artist metadata.

The final dataset was bag of lyrics information about each song, collected from musixmatch by LabROSA, which was over 1 GB of information. The musixmatch dataset is primarily a single table with five attributes, so no additional normalization was required. An index was created on the secondary ID (mxm_tid).



The image shows a screenshot of a database table named 'lyrics'. The table has five columns: track_id (VARCHAR(25)), mxm_tid (INT(11)), word (TEXT), count (INT(11)), and is_test (INT(11)). Each column is preceded by a small blue diamond icon. The table is displayed in a light blue box with a dropdown arrow on the right side of the header.

lyrics	
track_id	VARCHAR(25)
mxm_tid	INT(11)
word	TEXT
count	INT(11)
is_test	INT(11)

(mxm)

megarelation	song	artist_details	artist
track_id VARCHAR(40)	track_id VARCHAR(40)	artist_full_id INT(11)	artist_full_id INT(11)
danceability DOUBLE	artist_full_id INT(11)	artist_name VARCHAR(255)	artist_id VARCHAR(20)
song_id TEXT	danceability DOUBLE	artist_hottnesss DOUBLE	artist_7digitalid INT(11)
release_attr TEXT	song_id TEXT	artist_familiarity DOUBLE	artist_playmeid INT(11)
artist_hottnesss DOUBLE	release_attr TEXT	artist_location TEXT	artist_mbid TEXT
title TEXT	title TEXT	artist_longitude DOUBLE	
artist_longitude DOUBLE	end_of_fade_in DOUBLE		
end_of_fade_in DOUBLE	time_signature INT(11)		
time_signature INT(11)	mode INT(11)		
artist_id TEXT	loudness DOUBLE		
mode INT(11)	mode_confidence DOUBLE		
loudness DOUBLE	song_hottnesss DOUBLE		
artist_7digitalid INT(11)	time_signature_confidence DOUBLE		
mode_confidence DOUBLE	key_attr INT(11)		
artist_familiarity DOUBLE	analysis_sample_rate INT(11)		
song_hottnesss DOUBLE	year INT(11)		
time_signature_confidence DOUBLE	key_confidence DOUBLE		
artist_name TEXT	audio_md5 TEXT		
key_attr INT(11)	track_7digitalid INT(11)		
artist_playmeid INT(11)	energy DOUBLE		
analysis_sample_rate INT(11)	duration DOUBLE		
year INT(11)	release_7digitalid INT(11)		
key_confidence DOUBLE	tempo DOUBLE		
artist_location TEXT	start_of_fade_out DOUBLE		
audio_md5 TEXT	artist_latitude DOUBLE		
artist_mbid TEXT			
track_7digitalid INT(11)			
energy DOUBLE			
duration DOUBLE			
release_7digitalid INT(11)			
3 more...			

(msd)

The Neural Network

The neural network is coded using the TensorFlow deep learning framework developed by the Google Brain Team. We decided to use this particular framework because it works in conjunction with Tensorboard -- an open source front end template which allows developers to visualize the effect of each component of a neural network. The neural network we implemented is a simple fully connected dense model -- nothing fancy like recurrent or convolutional networks. It consists of two hidden layers, each

followed by a ReLU non-linear activation function. The last output layer uses a softmax function, and we utilize a “mean-squared error loss function” and an Adam optimizer.

```
def model(feature_size):  
    model = keras.Sequential()  
    model.add(keras.layers.Dense(units=feature_size, activation='relu'))  
    model.add(keras.layers.Dense(units=10, activation='relu'))  
    model.add(keras.layers.Dense(units=6, activation='relu'))  
    model.add(keras.layers.Dense(units=1, activation='softmax'))  
    model.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])  
  
    return model
```

The way we incorporated song lyrics as inputs is by “vectorizing” words through neural word embeddings. We used a pre-trained GloVe dataset (essentially a map of “word-vector” pairs) to assign each distinct word a vector. The GloVe dataset was trained on 6 billion tokens and has 400,000 words in its vocabulary. We then compressed the “word vectors” into a tensor along with other numerical inputs to feed the neural network.

```
# takes in list of words, outputs list of word vectors  
def vectorize_bag_of_words(bag_of_words):  
    matrix_len = len(bag_of_words)  
    weights_matrix = np.zeros((matrix_len, 50)) # dim 50  
    words_found = 0  
  
    for i, word in enumerate(bag_of_words):  
        try:  
            weights_matrix[i] = glove[word]  
            words_found += 1  
        except KeyError:  
            weights_matrix[i] = np.random.normal(scale=0.6, size=50)  
    return weights_matrix
```

WALKTHROUGH

The following steps are required to setup and run the project (and are included in the README):

- Download glove dataset for neural word embedding <http://nlp.stanford.edu>
- Edit settings in settings.py.
- Edit glove_path in word2vec.py
- Create a database in MySQL server. You do not need to create a table.

- mbzdb: clone from <https://github.com/elliotchance/mbzdb> and run `./init.pl` after editing settings
- msd: run `./getMSD.sh` after editing `settings.py`
- mxm: run `./getMXM.sh`
- install bazel 0.20.0
- `git submodule update --init --recursive`
- `bazel run //greeter_tensorboard`
`--incompatible_remove_native_http_archive=false -- --logdir=/tmp/greeter_demo`

The output of the final command should include a URL to access the TensorBoard dashboard, which should automatically include visualizations for any of the runs performed.

SUMMARY

One of the biggest challenges we faced throughout this project was dealing with the package dependencies and figuring out version compatibility between different scripts. Since we were working with three separate datasets, there were a lot of moving parts and even just getting the sql files to load was an issue. It was a lot of trial and error before we could have three proper schemas working on our sql workbenches. To overcome those hurdles, we hacked it out by reading documentation and online resources to understand what links were causing us those issues. There was also an added layer of complexity since both of us use different operating systems (and thus different package managers).

Another challenge was just simply dealing with the huge amount of data. Since a neural network requires a lot of training data for it to be at least somewhat of a decent classifier, the results of our neural networks were largely limited by the computational power we had. We did play around with dummy data (<500 data points), and it unsurprisingly resulted in terrible classification performance (<5% accuracy) despite being run for 1000+ epochs.

Overall, it was a great experience learning how to build non-trivial applications using pre-existing open-source frameworks. We collaborated mostly using git, and learned a

lot about the importance of having a good workflow and using standardized software design patterns. Matthew worked primarily with the normalization of data and polishing the frontend. Thomas worked primarily on the python backend and developing the neural network model. We would say that it was an enjoyable collaborative experience and we learned from each other as we progressed through the project.