Wilson's FM - Deep Playlist Generator

Alistair Wilson Gillespie
April 19, 2020

Machine Learning Engineer Capstone Report

Background

Wilson's Morning Wake Up (WMW) is a Spotify playlist I curate each month; the playlist is designed to
assist listeners in starting their day with no more than 15 tracks; and explores a range of genres including house, classical, funk and jazz, to name a few. WMW is structured in a way as to gently build in tempo and intensity - commencing with classical and minimalistic tracks - then introducing dance, house and electronic tracks later in the playlist, culminating in an hour or so of blissful listening.

Define

Quality playlists take time and effort to curate. I thought it would be cool to train a model to generate beautiful playlists each day. This project is an attempt to generate playlists of the same quality and structure as WMW.

To be effective, the project needed to perform the following tasks:

1. Extract metadata about all of the WMW playlists to date.
2. Transform track metadata.
3. Train a sequence estimator that learns the traits and relationships of each playlist.
4. Create a playlist using the estimator and Spotify's recommendation API.
5. Post a playlist to Spotify at Wilson's FM.

Analyze

The WMW dataset comprises 35 volumes each containing no more than 15 tracks. There is a total of 554 tracks in the dataset. The dataset was extracted using the Spotify for Developers API and includes a set of audio features engineered by the Spotify team. Furthermore, I've included an additional attribute for track position for guiding the sequence models discussed later in this paper.

The following features were selected to represent each track:

- *acousticness*: confidence score of whether the track is acoustic (float)
- *danceability*: how suitable the track is for dancing based on a combination of musical elements (float)
- *energy*: measure of the intensity and activity of the track (float)
- *instrumentalness*: confidence score of whether the track has vocals or not (float)
- *liveness*: detects presence of an audience in the recording (float)
- *loudness*: overall loudness of a track in decibels (float)
- *speechiness*: detects presence of spoken words in a track (float)
- *tempo*: estimated tempo of a track in beats per minute (BPM) (float)
- *popularity*: popularity of the track represented by a value between 0 and 100, with 100 being the

most popular (integer)

- *genres*: a list of genres used to classify the album in which the track features (array of strings)
- *song position*: position of the track in the respective WMW playlist (integer)

Figure 1 illustrates the distribution of the above features against each track position for all WMW volumes. In general, each feature tends to trend upwards as the playlist progresses, as depicted below.

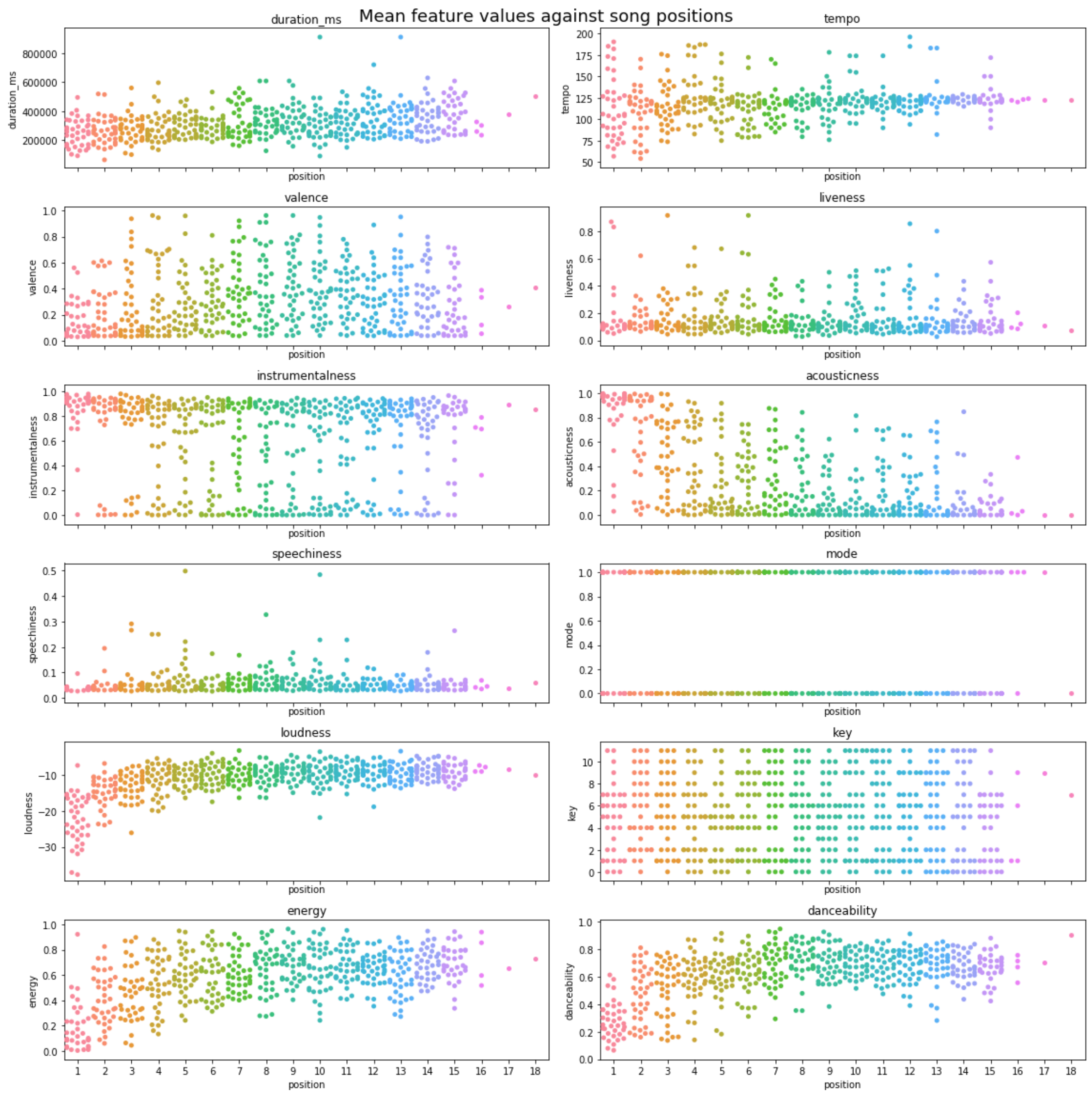


Fig. 1: Mean feature values against each track position for all WMW volumes

Further information regarding the audio features can be found here.

Principle Component Analysis (PCA) was applied to the dataset to reduce the number of dimensions for analysis and track comparisons. The data was transformed and projected into a 3-dimensioanl space to give a visual representation of tracks at each position. This view (in Figure X) provides insights about the relative variance of tracks selected at each position. Track positions 7, 8 , 11 and 12 tend to cluster closely together and resemble higher degrees of feature similarity. These tracks tend to be house and dance tracks of higher tempo. Conversely, track positions 1, 2 and 3 show higher variance which is interesting given position 1 is always a classical track and positions 2 and 3 are generally minimal electronic tracks.

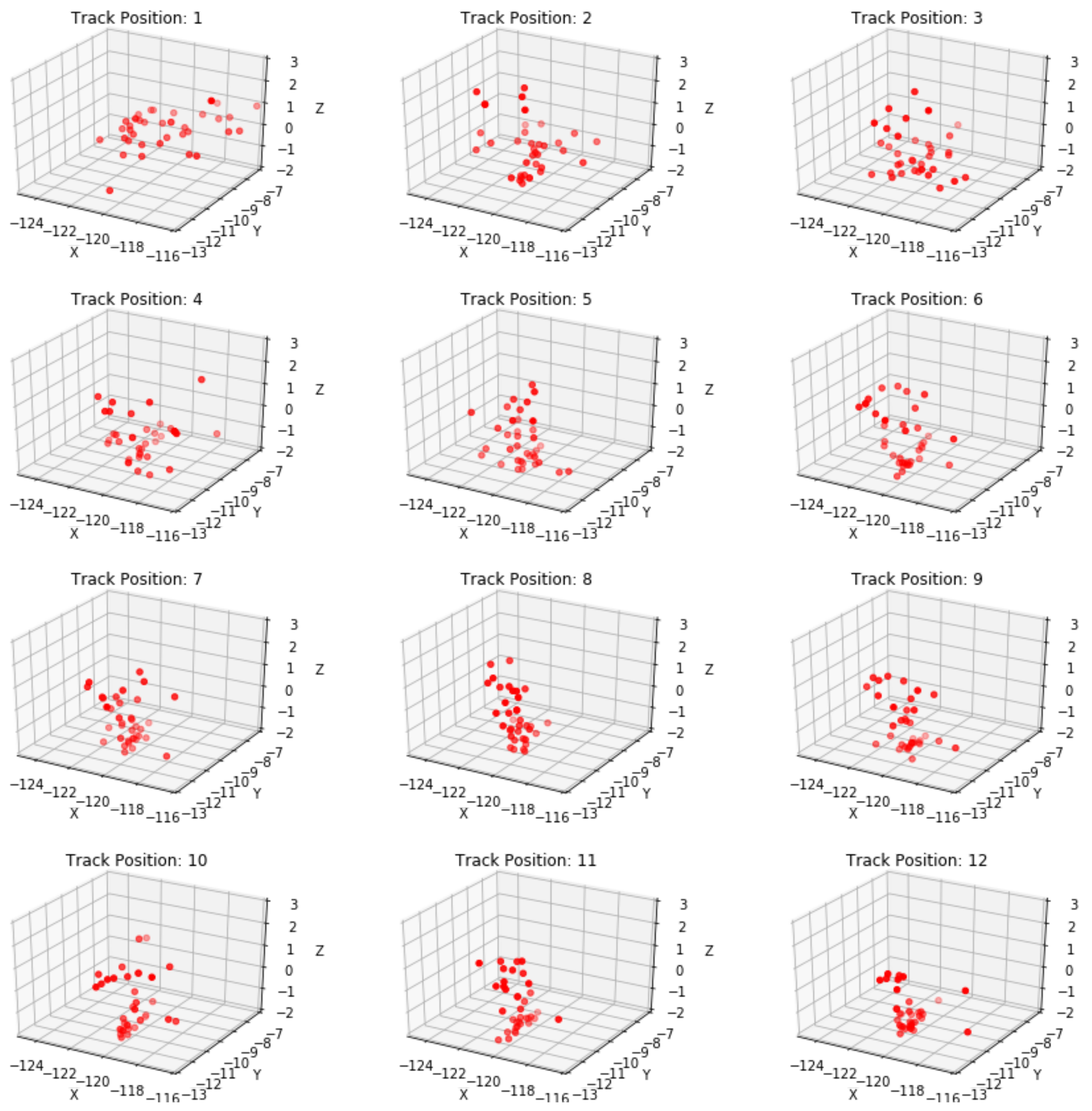


Fig. X: PCA analysis at each track position for all WMW playlists

Next, a view of each WMW volume was produced, pictured in the following figure. This view shows how each playlist moves through feature space over its duration. Interestingly, when I started this project, I anticipated that each playlist would resemble similar trends and behaviour. This view clearly shows that WMW volumes can vary significantly in feature composition and traits. For example, Volume 13 traverse in multiple directions as it plays out, however, Volume 27 tends to traverse laterally.

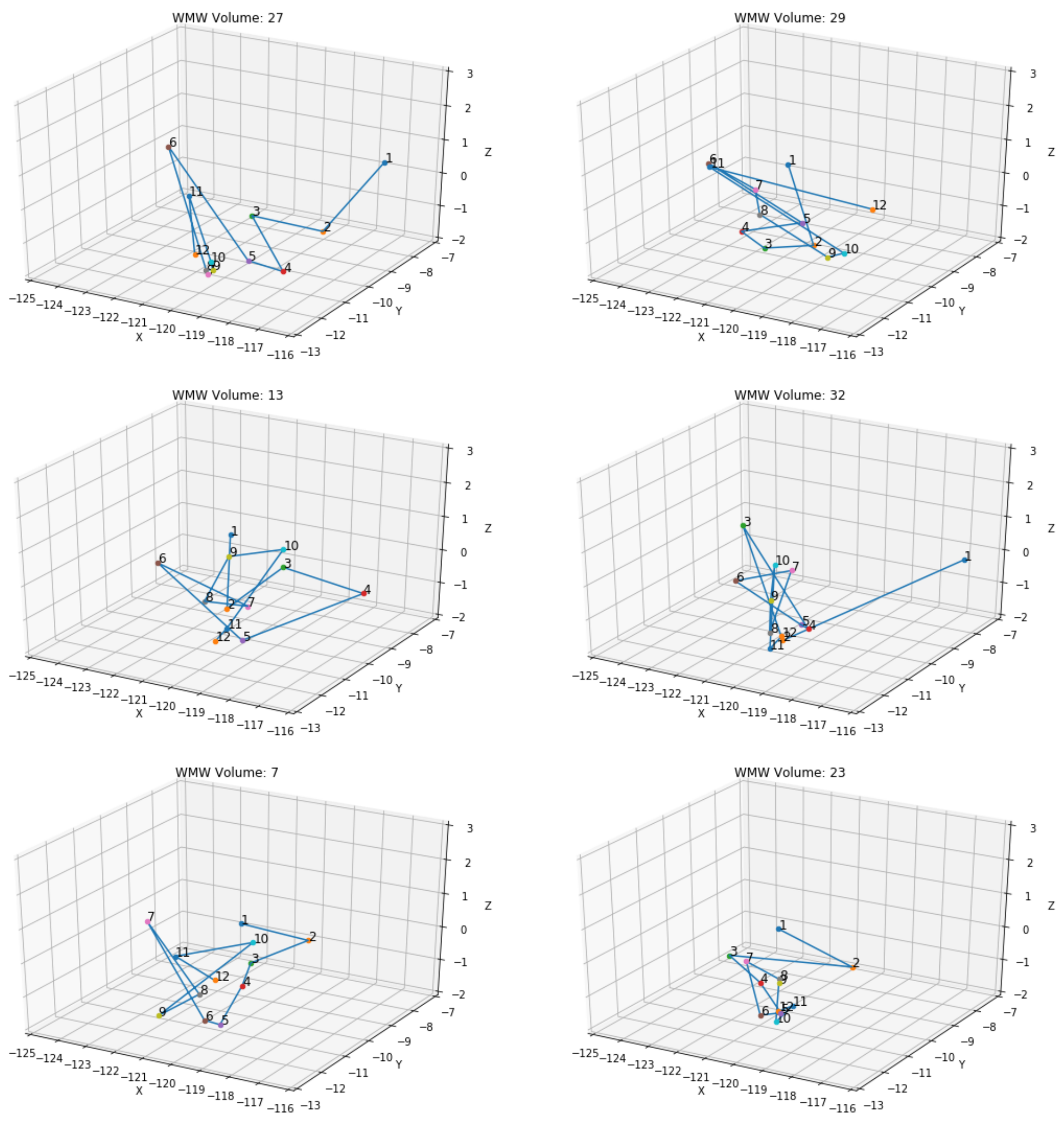


Fig. X: PCA analysis of a sample of volumes

Finally, I decided to look at all of the WMW volumes in a single plot to try and discern any relationships or clusters of tracks (Figure X). I wanted to test whether tracks at a given position were situated close to one another in feature space. Simply eyeballing the below chart shows little signs of distinct groups of tracks at a particular position. Rather there is quite a lot of variance across the whole sequence in features space. The only trait I can see is that a number of tracks at position 1 are positioned to the right as clear outliers.

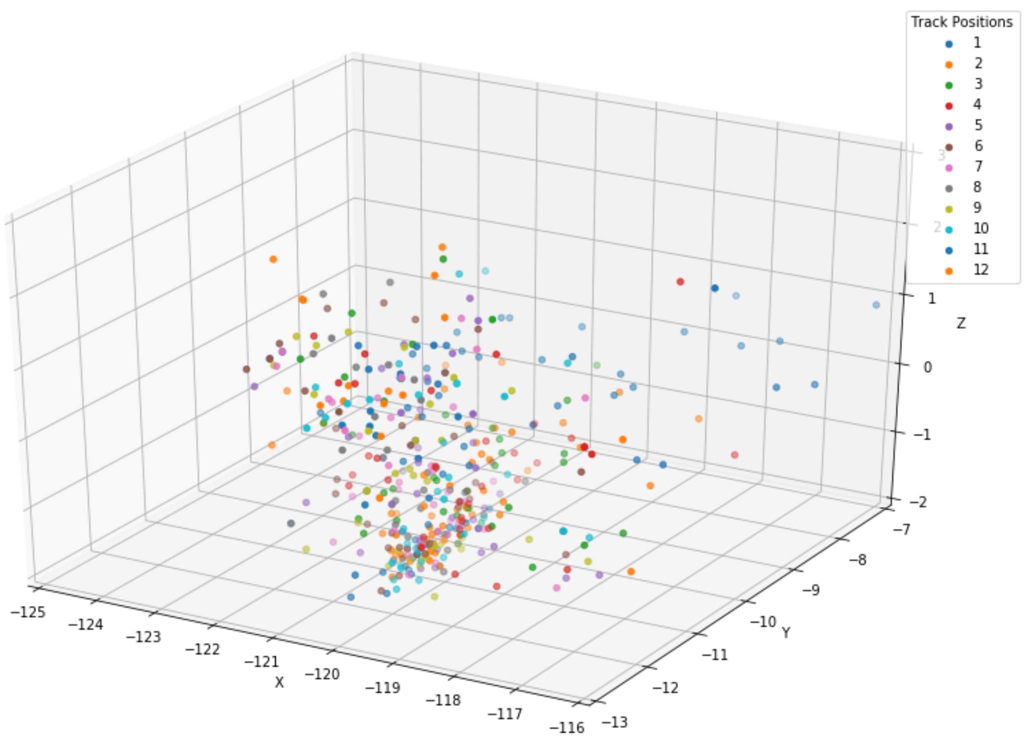


Fig. X: PCA analysis of all volumes

TODO: Analysis of linear combinations of features in each component

Implement

Python was used for this project along with famous tools such as PyTorch, Pandas and Numpy.

The project was structured like so:

```
.
|-- artefacts/ # Save models and artefacts here
    |-- dim_red.pkl # Principal Component Analysis
    |-- lstm_model.pth # Long Short-Term Memory Neural Network
    |-- rnn_model.pth # Vanilla Recurrent Neural Network
    |-- standard_features.pkl # Standard Scaler
|-- data/
    |-- tensor_train.csv # Training dataset
    |-- wmw.csv # Pool of Wilson's Morning Wake Up tracks to date
|-- model/
    |-- LSTMEstimator.py # LSTM Model with initialisation and feed-forward
    |-- LSTMTrain.py # Code for training the LSTM on AWS SageMaker
    |-- PlaylistDataset.py # Dataset Class
    |-- Predict.py # Code for predictions on AWS SageMaker
    |-- RNNEstimator.py # RNN Model with initialisation and feed-forward
    |-- RNNTrain.py # Code for training the RNN on AWS SageMaker
|-- img/
    |-- ...
|-- .gitignore # ...
|-- 0_Setup_Database.ipynb # Databased Setup for future use
|-- 1_Explore.ipynb # Initial data ingestion and analaysis
|-- 2_Feature_Engineering.ipynb # Feature preparation and further analysis
|-- 3_Train_Deploy_LOCAL.ipynb # Pipeline for training each model locally
|-- 4_Train_Deploy_AWS.ipynb # Pipeline for training each model on AWS SageMaker
|-- 5_Generate.ipynb # Generates a playlist and posts to Spotify
|-- LICENSE # MIT License
|-- local_env.yml # Environment details
|-- main.py # Pipeline that generates a playlist and posts to Spotify via CLI
|-- playlist.py # Playlist class
|-- PROPOSAL.md # Project Proposal
|-- README.md # ...
|-- REPORT.md # Project Report
```

The input data was shifted by 1 position and mirrored as the target so each input was mapped to the next track in the playlist. A PlaylistDataset class was built to serve as an iterable over the dataset. A DataLoader was then used to fetch batches - equivalent to the size of a playlist - from the PlaylistDataset class during training time.

The objective was to learn the common traits and relationships of each of the previous WMW playlists. Thus, it was decided a many-to-many recurrent neural network approach would satisfy this problem. The idea is to learn how to use the current track to predict the next track whilst maintaining a state of information about the previous tracks in the playlist. A Vanilla Recurrent Neural Network (RNN) was chosen as the baseline model and a Long Short-Term Memory Model (LSTM) was chosen as the alternative model.

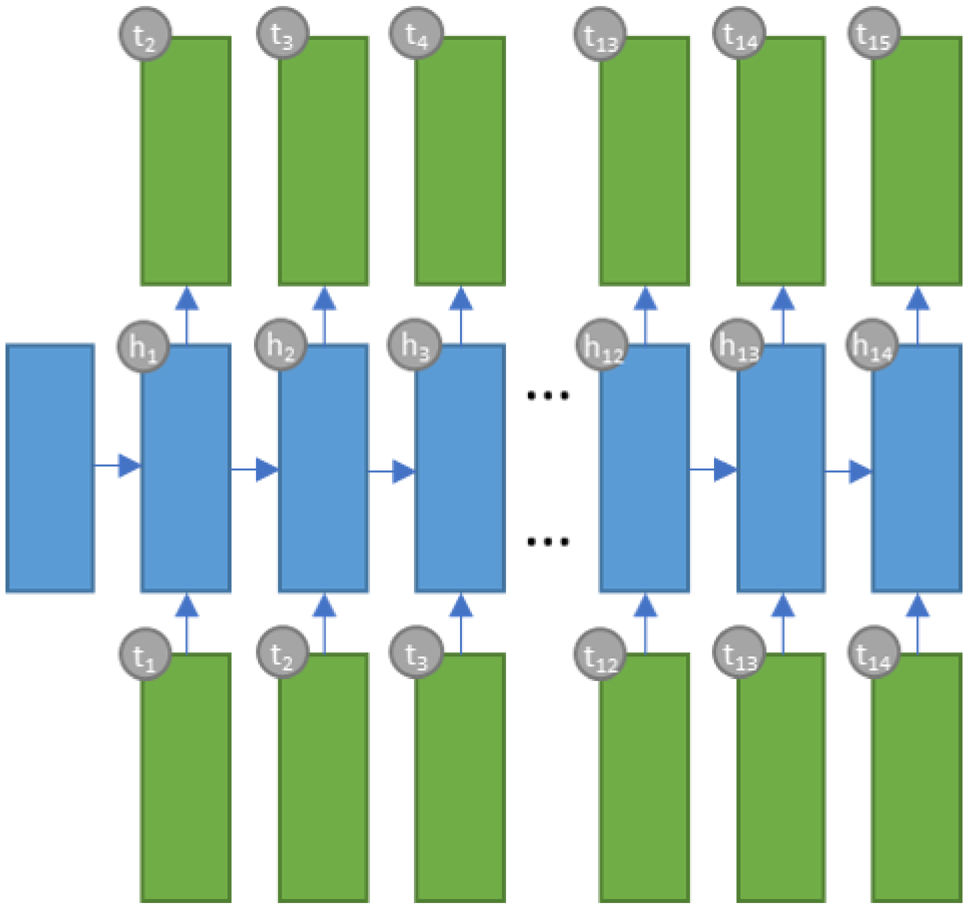


Fig. 2: Many-to-many: the model's hidden state (h) is initiated then each following track (t+1) is predicted using the current hidden state and current track (t) as input until the last track position.

The Mean Absolute Error - alternatively called the L1Loss in PyTorch - was selected to evaluate the performance of each model during training time. For test time, the resulting playlist was evaluated by listening to it on Spotify. MAE served as an indicator of the model fit to the data. It is worth noting that, ideally, a little bit of error is tolerable given it could enable "creativity" or new and unique track selections during playlist creation. However, the true indicator of performance is the listening experience, as a result, each playlist was "tested" in the living room of our house by my housemates and I. This was simply a qualitative analysis.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$$

Fig. 5: Notation for Mean Average Error

Results

It is expected that the solution will pick tracks, conditioned by the context in which they appear, to deliver harmonic track sequences that closely rese mble the manually crafted volumes to date. In theory, Vanilla RNNs can persist such information across sequences of input data but in practice commonly fall short. Then came along Long Short-Term Memory networks capable of persisting longer-term contexts of information. Christoper Olah's insightful images pictured below display the differences in how Vanilla RNNs and LSTMs persist information across sequences of inputs and outputs:

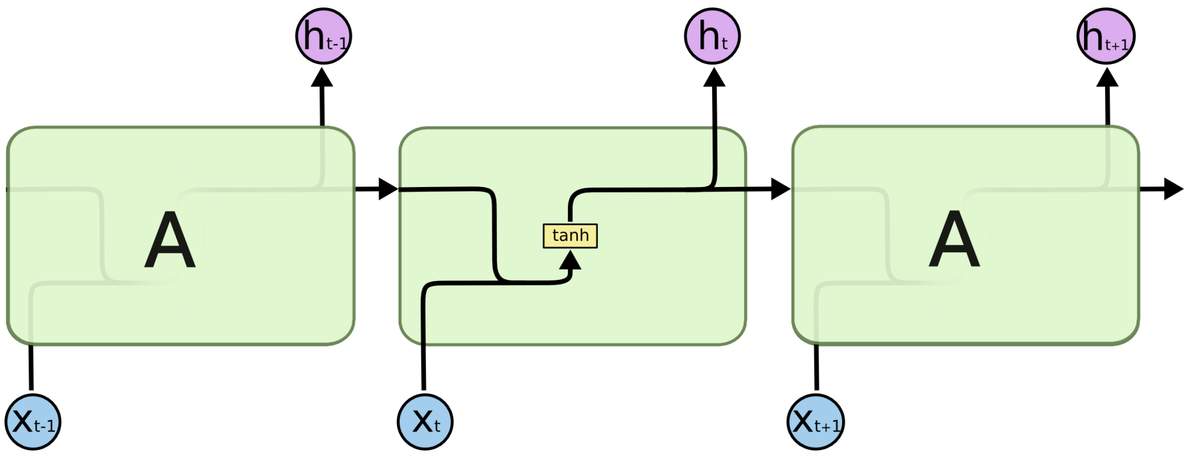


Fig. 3: Example of a Vanilla Recurrent Neural Network (Christopher Olah 2015)

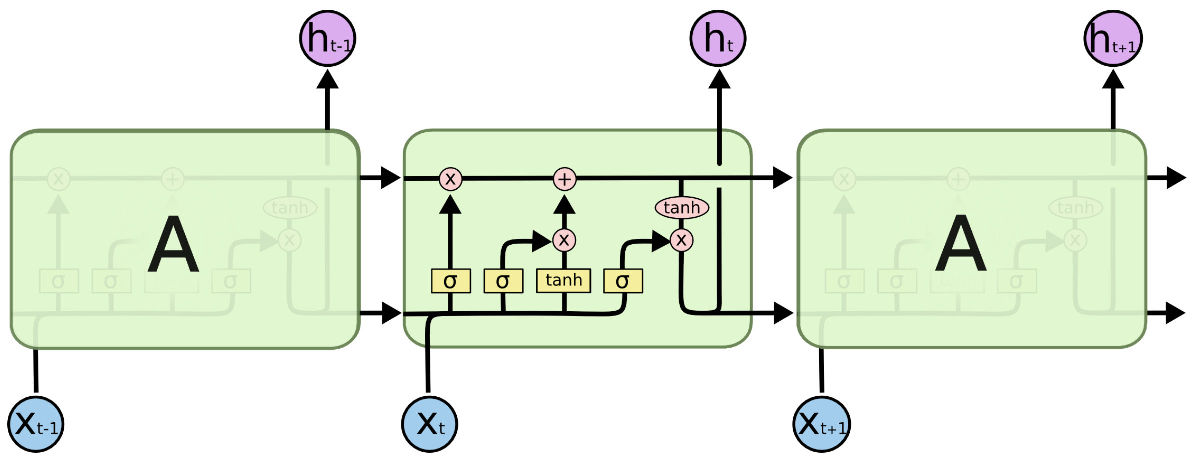


Fig. 4: Example of a Long Short Term Memory Network (Christopher Olah 2015)

For benchmarking, a Vanilla RNN architecture will be used to assess overall performance against a LSTM candidate model. The models will be evaluated upon one of the loss functions detailed in the next section.

Conclusion

The Mean Squared Error (MSE) and Mean Absolute Error (MAE) metrics will be considered as loss functions for the evaluation of the candidate models. MSE and MAE are defined as follows (Peltarion 2020):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$$

Fig. 5: Notation for the Mean Squared Error and Mean Average Error Loss Functions

It is important to note that MAE is not as sensitive to outliers compared with MSE, thus for examples with the same input features, the result will be the median target values. In contrast, MSE is suited to scenarios where the target values - conditioned on the input features - are normally distributed.

Project Design

The workflow will be structured in a way that is closely aligned to the Team Data Science Process defined by Microsoft whilst also inheriting elements from Udacity's standard project structure. As a result, an iterative data science methodology will be followed to deliver the project. The key steps are pictured below and are explained in the subsequent sections:

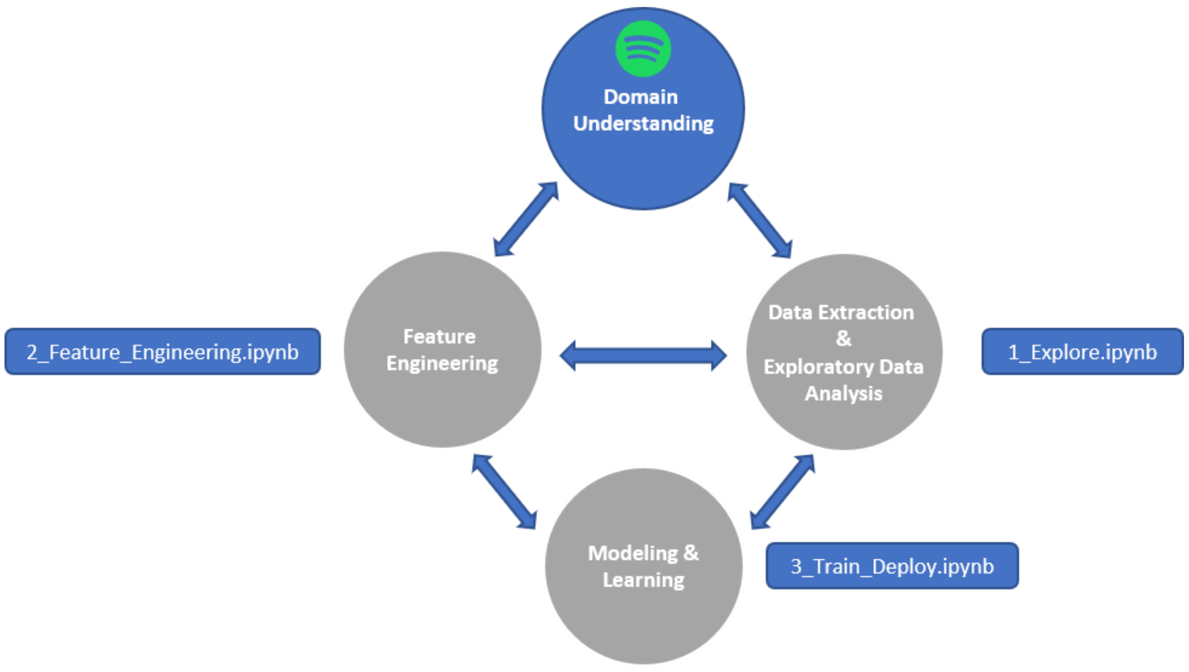


Fig. 6: Project Data Science Process with Key Project Notebooks

The project code will be structured - subject to change - as follows:

```
.
|-- artefacts/ # Save models and artefacts here
    |-- dim_red.pkl # Principal Component Analysis
    |-- lstm_model.pth # Long Short-Term Memory Neural Network
    |-- rnn_model.pth # Vanilla Recurrent Neural Network
    |-- standard_features.pkl # Standard Scaler
|-- data/
    |-- tensor_train.csv # Training dataset
    |-- wmw.csv # Pool of Wilson's Morning Wake Up tracks to date
|-- model/
    |-- LSTMEstimator.py # LSTM Model with initialisation and feed-forward
    |-- LSTMTrain.py # Code for training the LSTM on AWS SageMaker
    |-- PlaylistDataset.py # Dataset Class
    |-- Predict.py # Code for predictions on AWS SageMaker
    |-- RNNEstimator.py # RNN Model with initialisation and feed-forward
    |-- RNNTrain.py # Code for training the RNN on AWS SageMaker
|-- img/
    |-- ...
|-- .gitignore # ...
|-- 0_Setup_Database.ipynb # Databased Setup for future use
|-- 1_Explore.ipynb # Initial data ingestion and analaysis
|-- 2_Feature_Engineering.ipynb # Feature preparation and further analysis
|-- 3_Train_Deploy_LOCAL.ipynb # Pipeline for training each model locally
|-- 4_Train_Deploy_AWS.ipynb # Pipeline for training each model on AWS SageMaker
|-- 5_Generate.ipynb # Generates a playlist and posts to Spotify
|-- LICENSE # MIT License
|-- local_env.yml # Environment details
|-- main.py # Pipeline that generates a playlist and posts to Spotify via CLI
|-- playlist.py # Playlist class
|-- PROPOSAL.md # Project Proposal
|-- README.md # ...
|-- REPORT.md # Project Report
```

Data Extraction and Exploratory Data Analysis

To extract the data, I will use Spotipy - which leverages the Spotify for Developers API - to query my account and return all track metadata for each of the WMW volumes into a Pandas DataFrame. All data extraction related code will be defined within the 'utilities.py' file for use across the code base and notebooks.

Next, the data will be analyzed using descriptive statistics and visualizations within the '1_Explore' notebook. This phase of work will consider the distribution and nature of the feature set such as data types, outliers and feature correlation. The resulting findings will then inform any required feature engineering tasks in the following phase of development. The tools used to conduct this analysis will include - but not limited to - Seaborn, Matplotlib and Pandas.

Feature Engineering

An iterative approach will be used - by moving back and forth - between explorative data analysis, feature engineering and modeling to optimize the solution. The '2_Feature_Engineering' notebook will comprise any preliminary data preprocessing tasks, such as the following activities:

- Standardization of numerical features
- Normalization of numerical features
- Encoding of categorical features
- Feature Reduction and Aggregation
- Harmonic feature engineering such as 'mixing in key'
- Outlier exclusion
- Cross-validation (i.e. train/test splitting)
- Storage of preprocessed dataset on AWS for downstream training

Modeling & Learning

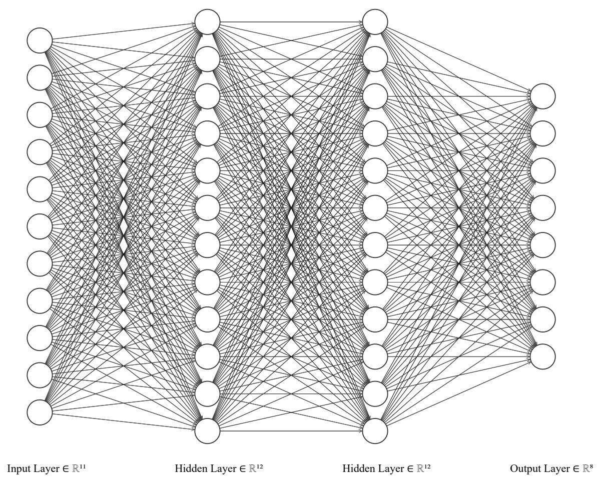
PyTorch will provide the necessary tools to define and train the benchmark and candidate models.

For modeling and training, the following tasks will be performed:

1. Load training and validation datasets into memory
2. Define the network architectures for each of the RNN models
3. Define the loss function
4. Train the networks

5. Monitor validation and training loss
6. Evaluate and select the winning model

The primary focus of this phase will be to consider the problem domain at hand, and any key findings, to configure the network appropriately, in particular, the input vector dimensions (feature set), number of hidden layers, output vector dimensions and regularization (Figure 7).



Input Layer $\in \mathbb{R}^{11}$
Hidden Layer $\in \mathbb{R}^{12}$
Hidden Layer $\in \mathbb{R}^{12}$
Output Layer $\in \mathbb{R}^{8}$

Fig. 7: Example Recurrent Neural Network Unit for Track Estimating

Estimation and Playlist Generation

Lastly, the chosen model will be used to select tracks at each position of the target playlist by:

- Extracting a bunch of candidate tracks using track, artist and genre seeds via Spotipy
- Selecting tracks for each position according to the model's predicted output vectors (audio features) using selection thresholds

Furthermore, a Python Playlist class will be defined to construct objects that will facilitate the creation of playlists.

Future Considerations

Future iterations of this solution may consider deployment options such as a web interface to allow users to train their own deep playlist model based on any defined playlists. Furthermore, collaborative filtering is another exciting option for sourcing more meaningful track recommendations for playlist selection.

References

Thank you to the following authors for providing excellent insights and inspiration for this project:

The Unreasonable Effectiveness of Recurrent Neural Networks 2015, Andrej Karpathy, accessed 1 March 2020, http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

RRN Music Recommender 2019, Taylor Hawks, accessed 26 February 2020, https://github.com/taylorhawks/RNN-music-recommender.

Understanding LSTM Networks 2015, Christopher Olah, accessed 26 February 2020, https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Regression loss metrics 2020, Peltarion, accessed 1 March 2020, https://peltarion.com/knowledge-center/documentation/evaluation-view/regression-loss-metrics.

Team Data Science Process 2020, Microsoft, accessed 1 March 2020, https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/overview.