UNIVERSITY COLLEGE LONDON

DOCTORAL THESIS

An evaluation of machine learning methods for modeling temporal point processes

Author:
Badrul ALOM

Supervisor:
Dr. Mark HERBSTER
Advised by:
Pedro Mediano (Emotech Ltd.)

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Computer Science

August 16, 2017

Declaration of Authorship

I, Badrul ALOM, declare that this thesis titled, "An evaluation of machine learning methods for modeling temporal point processes" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

University College London

Abstract

Faculty Name
Department of Computer Science

Master of Science

An evaluation of machine learning methods for modeling temporal point processes

by Badrul ALOM

tbc

Chapter 1

Introduction

The modeling of event sequences is useful across industries. For instance the periods in which a customer makes an online purchase can help determine the optimal periods for target marketing. The times at which public transport users tend to travel can help better manage resources to meet demand. The times at which a medical illness re-occurs can help predict future episodes.

In all these cases modeling the temporal behaviour of the system is important in predicting the next event. At an aggregate level, behaviour may appear to be deterministic, such as the times at which peak rush-hour occurs, but such behaviour is often composed of thousands or millions of individual stochastic processes such as the decisions made by individuals as to whether to leave work at 5pm or continue working a little longer.

While the area of product recommendation has received extensive attention in recent years, the area of recommendation timing less so. This research looks at how we can model the temporal behaviour of individuals, and whether deep learning is better able to learn these patterns than other methods.

1.1 Context

We take as our context for this research, the goal of estimating the probability that a user of a home-audio device would like to listen to music right now, based on their listening-event history. One such application of this research would be to allow home audio devices to suggest music toi a user at an ooportune time.

The goal will be evaluate the effectiveness of several different methods, including an LSTM-RNN. The research was guided by Emotech Ltd., a home audio hardware and software company and the creators of Olly [8].

1.2 Data

The dataset being used in this analysis is the LastFM1k dataset, which is freely available oneline and containings the listening history of a thousand LastFM listeners. It consists of a series of timestamps denoting when user started played a song. We wish to learn the temporal patterns of a users behaviour in order to predict the next item in the next item in the sequence - a play or non-play event.

The dataset contains the timestamp, user Id, and track Id of users listening habits over a number of years (2005-2009).

1.3 Structure of the report

tbc

Chapter 2

Literature Review

Event prediction is alternatively referred to in literature as sequence predictions and temporal point processes. The problem can be defined mathematically as determining:

$$p(x_t|x_h)$$

where h is the event history sequence, $t-n...t-1$.

2.1 Point Processes

This is one of the most widely used method for modeling event sequences. A temporal point process [2] is a way of modeling events data with t being a sequence of a fixed period interval with $t_i \epsilon R+$ and $i \epsilon Z+$.

It can be modelled as a series of inter-event times (time until next event) or the number of events occurring in the interval. Examples of point processes are the times between financial transactions [4], and the times at which a customer makes a purchase from an online retailer *** insert ref***.

At its simplest a point process can be represented as:

$$\xi = \sum_{i=1}^{n} \delta_{X_i},$$

where δ denotes the Dirac measure, a probability measure of whether a set contains point x or not.

Point processe models seek to estimate the probability of an event happening at time t, based on an event history upto, but not including, time t.

There are different ways of representing point process data as shown in figure 2.1, with the inter-event time being the most common.

2.1.1 Conditional Intensity Function in Point Processes

A conditional intensity function is the key part of modeling point processes [3]. In this method the probability of an event $\lambda(t)$ is derived from a stochastic model such as the Poisson process.

Conditional intensity functions can be inhomogenous such as with a Gaussian Kernel $\lambda(t) = \sum_{i=1}^{k} \alpha(2\pi\sigma_i^2)^{-1/2} exp(-(t-c_i)^2/\sigma_i^2)$, for $t \epsilon [0, T]$ where c_i and σ are fixed center and standard deviations, respectively, and α_i is the weight for kernel i.

Or they can vary in intensity, such as with the self-exciting (Hawkes) process where the intensity is determined by previous events through the parametric form $\lambda(t) = \mu + \beta \sum_{t_i<t} g(t-t_i)$ and where g is some non-negative kernel function.

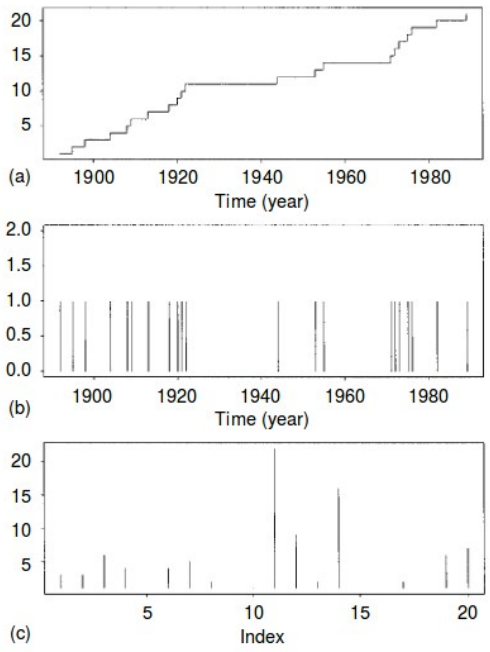

FIGURE 2.1: Three different representations of the same point-process a) cumulative count b) date of occurence c) interval time between floods

As noted by Wass et. al [15], conventional point process models often make unrealistic assumptions about the generative processes of the event sequences. The conditional intensity function make various parametric assumptions about the latent dynamics governing the generation of the observed point patterns. As a consequence, model misspecification can cause significantly degraded performance using point process models.

2.2 Deep RNN Point Process Models

In recent years deep learning has demonstrated the power to learn hierarchical nonlinear patterns on large-scale datasets [6] through multiple layers of abstraction (e.g. multi-layer feedforward neural networks). It has achieved state-of-the-art performances on a wide range of applications, such as computer vision [5], natural language processing [13], and protein structure prediction [7].

However it has not been applied to temporal point processes until recently with Xiao et. al [15] applying Generative Adversarial Networks (GANS) to the problem. GANs consist of two neural network models - a generator tasked with generating (i.e. predicting) a future sequence of events based on the history, and a discriminator tasked with detecting the true (fround truth) sequence amongst the generated ones.

For measuring the loss between a generated and true sequence, the authors found the Wassertein-Distance [9] performed better than Maximum Likihood Estimate (MLE) which they remarked "may suffer from mode dropping or get stuck in an inferior local minimum".

Their findings showed that where as parametric point process models work better with problems where a parametetric form exists, with real world data a GAN model with Wasserterin-Distance outperformed all other models (including an RNN

model using MLE). This signals a promising new direction for temporal point process research.

Chapter 3

Experimental Design

In this chapter we describe the different methods that were assessed for the task of predicting whether a user is interested in listening to music at time t given their play history h. The methods, in order of gradually increasing sophistication, are as follows:

1. Baseline model
2. Bayesian Inference
3. Binary Logistic Regression
4. Linear SVM Classifier
5. Non-Linear SVM Classifier
6. Recurrent Neural Networks

All methods, except for the Bayesian Inference method, require the data to be structured as a time-series. The Bayesian method adopts a different approach that requires the data to be aggregated into half-hourly buckets of a week.

The data preparation that was performed is described first in the next section, followed by an explanation each method, and the evaluaion criteria for assessing the methods.

3.1 Data preparation

3.1.1 Data transformation

The analysis was carried out in Python (via Jupyter notebooks) running on Ubuntu. The raw data consisted of timestamps of when a song was played and a user ID. These were loaded as-is into a SqlLite3 database in order to reduce the need to repeat data preparation steps.. The methods themselves utilized Scikit-learn for all models, save for the RNN model which used Tensorflow.

UserIDs were converted to integer (e.g. 'User0005' became '5') and a period lookup table was created at n minute intervals, to which all timestamps in our main dataset could be mapped to. n was chosen to be 30 although it is possible to re-run the analysis for other levels of granularity.

More significantly the data, which contained entries for the times at which each user listened to music, was supplemented with all the times they did *not* listen to music, between their date of their first and last play. This was required in order to generate a sequence of play and non-play events. It was noted however that in cases where a user stops listening to music for a long period, then listens to it sporadically, we would get an even greater imbalance of data (see later).

3.1.2 Time-series fetures

The approach to feature selection for this research was to devise an initial of features based on the preliminary analysis, and use that list across all models.

The time series features that were derived from the raw data were time-lags: $t, t-1, t-2, t-3, t-4, t-5, t-12hrs, t-23.5hrs, t-24hrs, t-24.5hrs, t-1wk, t-2wks, t-3wks, t-4wks$ where t is the event being predicted.

$t-1$ to $t-5$ represent user activity in the previous 2.5 hours. The remaining time-lags were chosen to represent half-day, daily, and weekly cycles, with additional empahsis around the -24 hour mark due to the daily patterns observed in the preliminary analsysis (see next chapter).

In addition a few non-time lag features were selected. One of which was the number of hours away from 5pm in either direction (so a timestamp at 4pm and 6pm would both equal 1), based on the observations in the preliminary analysis of 5pm being a peak hour. The others were binary features representing the day of the week (isMon, isTue etc.). Again these were based on the patterns observed in the preliminary analysis.

3.2 Validation and Test dataset selection

Our working dataset was a subset of the full 1000 user dataset, and comprised of 4,217,228 rows of training data across 97 users. Of this a random sample of 100,000 rows was taken on which 5-fold cross-validation.

Due to the time it takes for training cross validation was not performed on the RNN model, instead 10 randomly selected users were held back from the main training dataset and a random sample of 10,000 rows were taken from each user.

Data Imbalance

Data imbalance is when the training data is when one or more of the classes is underrepresented in the dataset. Of the 4,217,228 rows in our data set, 361,081 (8.6%) were play events and 91.4% were non-play events.

This could lead to models that achieve a high accuracy score by following either or both of the following two heuristcs: * Predict non-event for everything * Predict t will be the same as $t-1$

There are several methods for dealing with data imbalance [1] including restricting input data, having a weighted loss function, and using recall as an evaluation measure.

In this research class weights were employed to automatically adjust weights inversely proportional to class frequencies in the input data $n_samples/(n_classes * np.bincount(y))$.

This would have the impact of encourgaing the model to predict play events more frequently. However our goal is to encourage the model to predict play events, while minimizing the number of false postivies on the predicitions (i.e. we attach less cost to false-positives for non-event prediction). Encouraging this sort of behaviour requires not only class weighting, but also sample weighting to give a higher penalty to false positives for play-predictions. This strategyh was also employed in the models as shown in the following Python snippet (1 represents a Play event):

$$sampleWeights = 1 + (y[:] == 1) * (x[:, 1] == 0)$$

3.3 Methods)

3.3.1 Baseline Model

Our baseline model will be to assume $t = t - 1$, that is to say a person will listen to music at period t if, and only if, they listened to music in the period immediately priod. As music listening events tend to be clustered (people listen to music in batches) the accuracy of the baseline model is expected to be fairly high. However it will not be able to predict the first play of a listening session, or where the listening session duration lasts no longer than a single periodm, which in the experiments is defined as a 30 minute period.

3.3.2 Bayesian Inference

We employ a simple Bayesian Inference approach by utilizing the Beta-Binomial model. The Beta-Binomial model is built upon the weekly patterns observed in the preliminary analysis. Conceptually it seeks to build up a users personalized weekly listening profile as observations come in, using a Beta-Binomial probability distribution, with the priors based on the population as a whole. We then assess how effective this profile is at predicting listening events for that user.

We divide a week up into timeslots of half hourly intervals (24*2*7=336 timeslots). The liklihood function which represents the probability of a user listening to music, is defined as a binomial distribution, where k is the number of plays in a given period, n is the sum of plays and non-plays, and θ is the unknown probability parameter for the binomial distribution..

$$\binom{n}{k} p^k (1-p)^{n-k}$$

Our prior is the Beta distribution of θ:

$$p(\theta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$$

where

$$\mathrm{B}(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

As the Beta distribution is a conjugate prior to the Binomial the formula can be reduced to: $Beta(\alpha + P, \beta + Q)$ where P is the count of plays and Q is the count of non-plays.

Our parameters for the beta distribution, α and β, are derived from the training set, with an estimate for each half-hourly time period.

Finally we convert the probabilites into a binary outcome by optimizing for a threshold λ at which we predict a play event.

3.3.3 Binary Logistic Regression

This method (and the subsequent methods) adopts a more classicial time-series approach to the prediction problem by constructing a dataset as a sequence of play events at time t $Y_t = 1$ and non-play events $T_t = 0$ with t representing a time period of a fixed interval, which for our research will be 30 minute chunks. We seek

to calculate the probability of an event in the current time period t, given the history of events: $p(Y_t = 1 \mid Y_h)$, for each individual user together with additional non-time-series features (see .

Our binary logistic model is therefore defined as:

$$p(Y_t = 1|Y_h) = \sigma(w^T x + b)$$

$$p(Y_t = 1|Y_h) = 1 - p(Y_t = 1|Y_h)$$

with σ being the sigmoid function defined as:

$$\sigma(x) = \frac{e^x}{1 + e^x}$$

Determining the optimal weights and constant can be determined by maximization of the log-liklihood or the minimization of the negative log-liklihood [**NegLog**]. The latter is given as:

$$Csum_{i=1}^{n} log(exp(-y_i(X_i^T w + b)) + 1)$$

for class $C = 1$.

In addition we will be comparing results with and withour the L2 regularizer term: $+1/2w^T w$

Regularization is technique for encouraging the model to converge on a smaller set of strong features rather than a wide set of weak features and is imporant in helping us determine which time-lags and additional features are important and which are not.

3.3.4 Linear SVM Classifier

SVM models seek to determine a separation plane between classes based on the support vectors, the data points closes to the decision boundary.

A linear SVM regression model performs this through the Epsilon Intesive loss function [14]. The objective becomes to *minimize:*

$$max(0, \|(y_i - w_i x_i - b) - \epsilon\|)$$

In other words we ignore cost functions that are within a certain margin ϵ. In our case this may be of importance in cases where the probability of user listening to music is close to the decision boundary, which may be the case for the very first song played at the start of a session.

3.3.5 Non-Linear SVM Classifier

Here we extend a regression model to include a Gaussian RBF kernel. A Gaussian kernel is a popular method for modeling non-linear decision boundaries, so for example if the decision to play music is based on some non-linear combination of day of the week, time of the day, and time since last play. Our model becomes:

$$p(E = 1) = b + \sum_{i=1}^{N} w_i RBF(x, x_i)$$

where the RBF kernel is defined as:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

Note that our actual implementation in Tensorflow makes use of the more computationally efficient algorithm as defined by McClure [10]. This restated method has a parameter γ that takes the place of $2\sigma^2$ and which requires tuning.

Mini-batching was also performed with a batch size of 2000 and 5 iterations of the dataset, as it was found that rows of more than 5000 caused memory errors, and iterations beyond 5 showed minimal or no decrease in loss).

3.3.6 Recurrent Neural Networks

Recurrent nets are a type of artificial neural network designed to recognize patterns in sequences of data, such as text, genomes, handwriting, the spoken word, or numerical times series data emanating from sensors, stock markets and government agencies. Whilst a traditional Feed-Forward network [12] has input nodes, hidden layers, and an output layer, with data flowing in one direction only, RNNs allow for the hidden state from one timestep of the neural net to be an input into the next (see fig. 3.1).

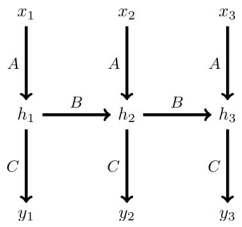

FIGURE 3.1: RNN with 3 timesteps

RNN models can employ different methods the propogration of the hidden state over time. One such well known method is Long Short-Term memory (LSTM) [11]. Here the hidden state is the product of a further four layers that interact in a way as to learn what information to retain and what information to throw away.

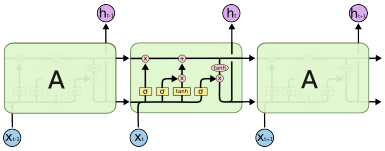

FIGURE 3.2: LSTM

We could therefore provide the model with a sequence of historical data at times $t-1..t-n$ where $n > 1$) which would be fed into the RNN as separate time steps in forward temporal order. The hidden state would propogate information from t_n forward at each time step, until it was used to predict the outcoem a time t.

Practically speaking this mean feeding in the data with input shape (batch rows, time-steps, features). Some points of interest to fellow researchers are:

1. The features dimension here does not need to contain all time-lags as we do in our other time-series models. However it must contain t-1. In this research we experiment with both t1 only and all our timelag features in this research.

2. The time-steps dimension is where the features list in previous time steps will be stored. These are then 'unrolled' within Tensorflow and fed into the LSTM. In this research we experiment with different lengths of time-steps.

3. When the data is unrolled, time step t for all rows in the batche are processed together as one block, before moving onto the next time-step.

4. Constructing the 3-d shape often requires building them up in slices. A significant speed up was observed in Python when using a pre-allocated array vs. appending to it.

3.4 Evaluation critera

Deciding on an approporiate evaluation measure requires careful consideration to the costs attached to different predictions. In our case we assign a lower cost to predicting a non-play event, and being wrong, than predicting a play-event and being right (as the cost of suggesting music when the user is likely not interested is higher than not playing music when they likely not interested).

We also wish to rate models higher if they manage to predict the first play of a listening session, as this is seen be harder than simply predicting a play event, when the last n events were also a play event.

With these in mind a number of possible evaluation criteria were looked at. The most-straight forward of which was *accuracy*. This computes the count of correct predictions as a fraction of the total number of predictions. While this is an intuitive measure, it would not distinguish between models that had a high accuracy on the play-events vs. a high accuracy on non-play events.

To do this we look at precision. Precision (P) is defined as the number of true positives over the number of true positives plus the number of false positives. A positive in this case is a play-event so our precision equation becomes:

$$Precision = \frac{CorrectPlayPredictions}{TotalPlayPredictions}$$

.

This will be measuring our models on how many of their play event guesses were correct which takes care of one of our considerations outlined above. However it does not distinguish between models that predict a play event only when the previos event was a play (a 'safe bet' and so a high precision score) vs. models that predict correctly in cases where the previous event was not a play and are threfore harder to get right. We turn to another measure to aid with this: recall.

Recall is defined as the number of true positives over the number of true positives plus the number of false negatives. A false negative is where a model predicts a non-event, when in fact there was an event. For example if we predicted 100 plays correctly but there were in fact 110 plays in the dataset, then recall would be 100/(100 + 10)= 91

$$Recall = \frac{CorrectPlayPredictions}{TotalPlayPrectionsInDataset}$$

Precision and recall in combination was therefore our chosen evaluation method. In addition tests were carried out examine how well the models performed on predicting the first play event in a series.

3.5 Summary

We have described how our data was transformed to make it useful for our experiments. In particular how we had to convert our list of play events into a list of play and non-play events for every period. By doing this we are faced with an imbalance of data as non-play events make up 91% of the data and so we employ a weighting strategy to counteract this.

We then described the methods we will be utilizing in our experiments, consisting of a Baseline model which is simply to assume $t = t - 1$, a Bayesian model which build up a weekly profile of listening habits for each user, Logistic and SVM models that apply classical machine learning time-series prediction techniques to our problem, and finally a deep learning model in the form of an RNN-LSTM where we seek to utlize the capabilities for it to learn temporal patterns through a hidden memory state.

Finally we looked at different ways for measuring the success of our problem, with precision and recall being the preferred choice.

In the next chapter we shall present the results of our experiements and a discussion of what this tells us about learning user-level temporal patterns and performing event prediction.

Chapter 4

Results

We begin our discussion of the results with preliminary analysis of the data that helped shape the experimental design. Here we seek to understand what are some of the overarching patterns in music listening habits and how these look at an individual level.

After this we present a summary of our results followed by a discussion of the performance of each individual method.

4.1 Preliminary analysis

4.1.1 Daily play patterns

By grouping track plays into 30 min intervals and aggregating by periods within a day, we see a clear daily pattern with music listening hitting a peak at around 5pm and a trough at around 6am.

FIGURE 4.1: 5-5.30pm is peak listening time

Zooming out to view the pattern across an entire week in figure 4.2, we see that the daily pattern occurs across every day of the week with weekends having a lower total number of plays.

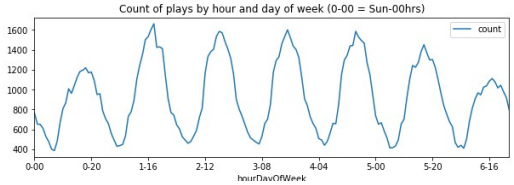

FIGURE 4.2: Most popular times to listen to music across all users

At a high level therefore one can get good accuracy by simply anticipating music demand to peak at 5pm. However if we select two users at random, we see (see

fig. 4.3) that these daily patters are not as strongly discernable. This demonstrates why models modeling the high levels patterns is not enough for individual user prediction.

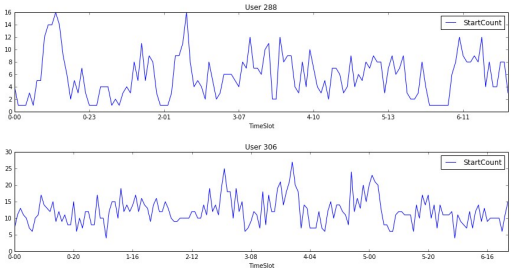

FIGURE 4.3: Most popular times to listen to music by individual user

4.1.2 Inter-event times

The dataset contains a timestamp associated with each user. This does not necessarily mean the user played a song in its entirety. Analysis shows plenty of cases where the interval time between tracks was a few seconds suggesting the user skipped tracks.

Figure 4.6 shows a frequency plot of intervals. Intervals beyond 30 minutes continue the exponential decrease and are not shown. We see that while the mode is on par with a typical song length, there is a significant number of plays that lasted under 5 minutes.

4.1.3 Time-series analysis

Here we examine our data once it has been transformed in a binary sequence of events (1) and non-events(0). We seek to understand better how an optimizer may perform based on traits of the data.

We begin with assessing how well our baseline model may perform based on assuming $t = t - 1$. Fig 4.4 shows that the 76% of Plays, also had a play in t-1. However simply using this as a rule would also capture 2.2% of non-plays.

	Count	%
Plays	361,081	
of which t-1 is a play	274,985	76.2%
of which t-1 is a non-play	86,096	23.8%
Non-Plays	3,856,147	
of which t-1 is a play	86,088	2.2%
of which t-1 is a non-play	3,770,059	97.8%
Total	**4,217,228**	

FIGURE 4.4

Furthermore the 23.8% of Plays that did not have a Play in the prior period are harder to predict yet of more interest as they representthe beginning of the listneing period and therefore more useful to a music recommender system.

Given the daily patterns we have seen, it might be reasonable to assume that looking at the same period 24 hours prior may be a good indicator of whether t is a play event. However as we see from fig. 4.5 this is not a reliable indicator either.

	Count	%
Plays	361,081	
of which t-1 is a play	274,985	
of which t-24hrs is a play	102,522	37.3%
of which t-24hrs is a non-play	172,463	62.7%
of which t-1 is a non-play	86,096	
of which t-24hrs is a play	23,478	27.3%
of which t-24hrs is a non-play	62,618	72.7%

FIGURE 4.5

What both of these results tell us is that fairly high precision score of around 76% ought ot be possible purely based on t-1 but going above this while having a good precision score on the non-play events will be harder.

4.1.4 Outliers

The data was checked for any unusual outliers that may impinge upon the goal of developing a model to predict user behaviour. An analysis of plays by user reveals a high amount of variance between users on how many tracks are played.

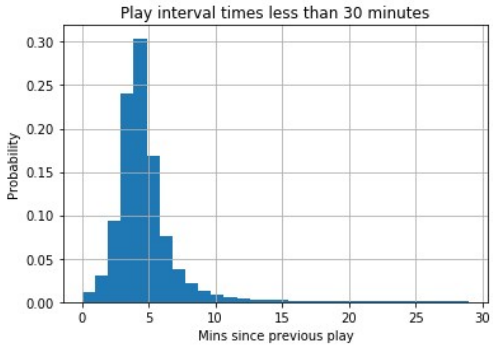

FIGURE 4.6

For our purposes these are include as evidence that the user was interested in playing music at time t.

We can also assume that the song plays are not independent of one another, in that the probability of a play event at time t+1 is significantly higher if there was an event at time t.

4.1.5 Outliers

The data was checked for any unusual outliers that may impinge upon the goal of developing a model to predict user behaviour. An analysis of plays by user reveals a high amount of variance between users on how many tracks are played.

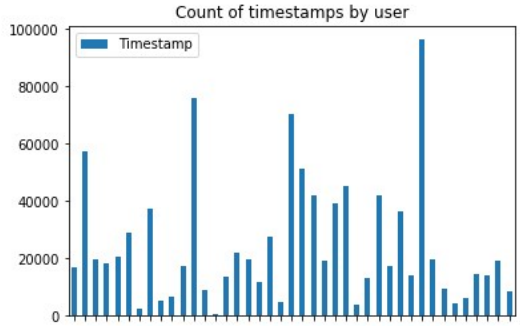

FIGURE 4.7: Total play count by user

Further analysis showed one user in particular with very high amount of plays, with very low durations, suggesting it was likely to have been generated by a bot, possibly a LastFM test. This was excluded from the dataset.

4.2 Main results

We will now present the results of the experiments conducted. We start with a summary of the results obtained through 5-fold cross validation then discuss our results in more depth.

Fig. 4.8 shows the overall results from 5-fold cross validation. From this it would appear that almost all models outperformed the baseline model. However these results disguise the true picture. The precision and recall are the average across both plays and non-plays. We can instead look at 'first play' events only (fig. 4.9) by defining 'first play' as any plays where there were no plays in the previous 4 perids (i.e.2 hours). Here we see a more mixed pattern with SVM models scoring an average 81% on precision, logistic regression an average of 68% on recall, and the Baseline and Bayesian models performing significantly worse.

	Play & Non-Plays			
Model	**Precision**	**(+/-)**	**Recall**	**(+/-)**
Baseline	84.0%		80.0%	
Bayesian Inference	84.0%	0.5%	89.0%	0.5%
Logistic Regression	78.6%	0.2%	87.6%	0.1%
Linear SVM Regression	87.4%	0.5%	87.3%	0.5%
RBF Regression	87.4%	0.5%	87.3%	0.5%
RNN T1 only				
RNN-All Features				

FIGURE 4.8

	Play events only			
Model	**Precision**	**(+/-)**	**Recall**	**(+/-)**
Baseline	8.0%		13.0%	
Bayesian Inference	41.0%		13.0%	
Logistic Regression	78.0%	0.4%	71.0%	0.8%
Linear SVM Regression	81.0%	0.0%	68.0%	0.1%
RBF Regression				
RNN T1 only				
RNN-All Features				

FIGURE 4.9

4.3 Bayesian Inference

Prior probabilities were calculated or each half-hourly period in a weekl. Fig 4.10 shows the calculations for the first 2.5 hours of a Sunday (d-hour-hh format). Our parameter of interest is the mean for the Beta distribution representing the probability of play in that time period. We estimate this by calculating probability of a play (total plays in period / count of plays and non-plays) per user, then taking the mean and variance across users. a and b are then determined as: $a = (\frac{(1-\mu)}{\sigma} - \frac{1}{\mu})\mu^2$ and $\beta = \alpha\left(\frac{1}{\mu} - 1\right)$.

Timeslot	mean	var	a	b
1-00-1	0.098577	0.010807	0.711953	6.510370
1-00-2	0.092327	0.011242	0.595911	5.858451
1-01-1	0.090256	0.011784	0.538632	5.429205
1-01-2	0.089523	0.011741	0.531937	5.409996
1-02-1	0.087637	0.011772	0.507577	5.284259

FIGURE 4.10

This allows us to visualize the prior probability for any given time period as shown in 4.11.

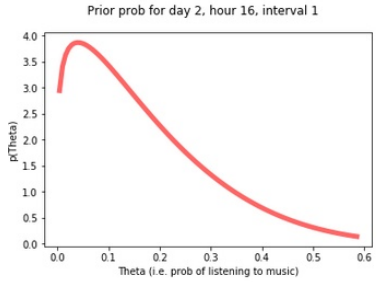

FIGURE 4.11

Once established we can calcualte the liklihood of an individual user listening to music in a given time period by passing their observations as they come in for a specific timeslot s: $(H_{t,s})$, into a Beta-Binomial formula to determine the probability of listening at the next occurence of that time slot as shown in <TBC>

Finally the threshold at which we determine that a probability constitutes a Play event is determined by comparing the false positive rates with the true positive rates using a ROC curve (4.12). This tells us that 0.4 is the optimal threshold at which to determine a Play event.

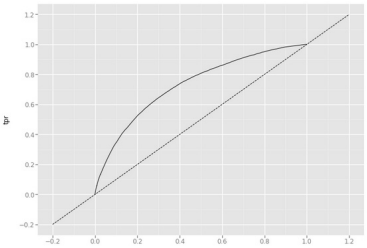

FIGURE 4.12: ROC curve showing 0.4 as the optimal threshold

The results from the model are shown in figure 4.13. We see that the recall and precision of play events (as denoted by 1) is very low suggesting that relying on a

Bayesian approach centered around a weekly profile of each users habits is not an effective method for predicting a play event for a new time period.

	precision	recall	f1-score	support
0	0.93	0.98	0.96	227823
1	0.40	0.13	0.20	19763
avg / total	0.89	0.91	0.89	247586

FIGURE 4.13: Beta-Bionmial Model Results

4.4 Logistic Regression

4.5 RNN-LSTM

The RNN model required a lot more effort to set up and get meaningful results from. While some of this was down to how Tensorflow operates, some of it is also a feature of deep leanting itself. In the quest to find a model that worked the following tweak were made to the starting model.

Computational efficiency

RNN's require the construction of a 3-dimensional tensor in which the second dimension is unstacked to create a list of timesteps. Using pre-allocated arrays than appending to arrays was found to provide a significan speed boost in shaping the data in theis way.

Data Sampling

It is well known that deep learning requires a large amount of data from which to learn from. In order to provide it within enough varied data the following hierarchy of iteration was set up up:

1. Sampling iteration (s)
2. Period iteration (p)
3. Batch iteration (n) and Batch size (z)

The highest level of iteration was a sampling iteration, S. Each sampling iteration selects a user at random. For each random user p random periods were selected. For each period selected z preceding periods were selected to form a single batch of training data. This model was then fed this batch n times.

Further traning parameters was the number of hidden notes, h, in the RNN; and the number of timesteps, selected for each row of the batch.

It was oberved that a variery of training data, particularly from different users, was more beneficial than high amount of iteraions or a deeper network.

Figure <TBC> shows the performance across different parameter settings, with the best result across all models shown in <TBC>

	precision	recall	f1-score	support
0.0	0.93	0.97	0.95	2645
1.0	0.66	0.46	0.54	355
avg / total	0.90	0.91	0.90	3000

FIGURE 4.14

4.6 Overall results summary

The best scores per model are shown in figure 4.15. Looking at the F1-score we see that the Logistic regression model performs the best. It was also found to be consistent across different runs of the model.

	Non-Play events			Play events		
Model	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Beta-Binomial	93.00%	98.00%	96.00%	40.00%	13.00%	20.00%
Logistic Regression	97.00%	97.00%	97.00%	79.00%	75.00%	77.00%
Linear SVM Regression	95.00%	98.00%	97.00%	79.00%	63.00%	70.00%
RBF Regression	86.00%	59.00%	70.00%	14.00%	41.00%	20.00%
RNN T1	91.00%	94.00%	92.00%	24.00%	14.00%	18.00%
RNN-All Features	93.00%	97.00%	95.00%	66.00%	46.00%	54.00%

FIGURE 4.15: Summary of best scores

TO BE COMPLETED

4.7 Adaptability to new users

One of our areas of interest was how accurate such models are when it comes to learning the patterns of new users. Tie figures below show how the accuracy improved over time for each of the 10 test users. The periods indicate half-hourly intervals with 2-3,000 periods (1.5-2 months) appearing to be the time it takses to learn the habits of a typical user.

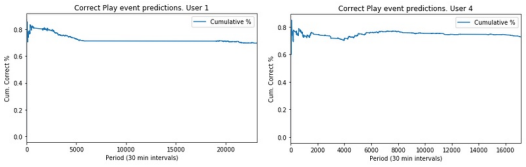

FIGURE 4.16

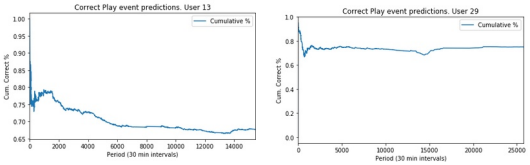

FIGURE 4.17

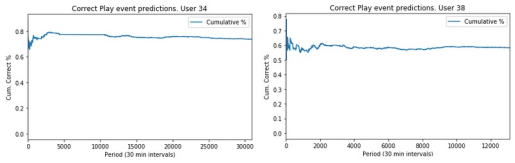

FIGURE 4.18

From these charts it appears that two months worth of data is required to for the model to rget trained up on a new user.

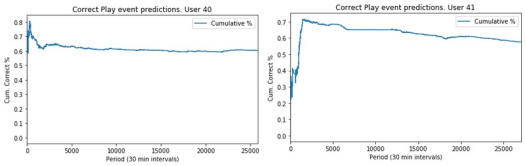

FIGURE 4.19

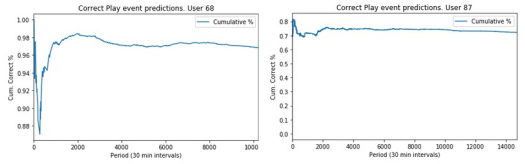

FIGURE 4.20

Chapter 5

Summary

5.1 Conclusion

The research set out to evaluate the effectiveness of Neural Networks to learn temporal patterns, partciularly without explicit encoding of time-lagged features.

Appendix A

Content not used

Intro: While the modeling of aggregate patterns is well understood, these models ofen breakdown when applied to customizing results for individual users. At this level the temporal patterns of an individual combined with the behaviour of the population may be a better predictor of event timing. For instance, sticking with the example above, the times at which a person has lunch during the day may help predict that they will finish work a little later.

Bibliography

[1] Jason Brownlee. *8 tactics to combat imbalanced classes in your machine learning dataset*. 2015. URL: http://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/.

[2] DJ Daley and D Vere-Jones. *An introduction to the theory of point processes*. 2003.

[3] Nan Du et al. "Time-sensitive recommendations from recurrent user activities". In: *Advances in Neural Information Processing*. NIPS. 2015.

[4] Robert Engle and Jeffrey R. Russell. "Autoregressive Conditional Duration: A New Model for Irregularly Spaced Transaction Data". PhD thesis. University of California, 1996.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing*. NIPS. 2012.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521 (May 2015).

[7] Pietro Di Lena, Ken Nagata, and Pierre Baldi. "Deep architectures for protein contact map prediction". In: *Bioinformatics* 28 (19 Oct. 2012).

[8] Emotech Ltd. *Your robot with personality*. URL: https://www.heyolly.com/.

[9] Léon Bottou Martin Arjovsky Soumith Chintala. "Wasserstein GAN". PhD thesis. Facebook AI Research, 2017.

[10] Nick Mclure. *Tensorflow Cookbook*. 2016. URL: https://github.com/nfmcclure/tensorflow_cookbook/blob/master/04_Support_Vector_Machines/04_Working_with_Kernels/04_svm_kernels.ipynb.

[11] Christopher Olah. *Understanding LSTMs*. 2017. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning internal representations by error propagation". In: *Parallel distributed processing: explorations in the microstructure of cognition*. Cambridge, MA: Association for Computational Linguistics, 1986, pp. 1631–1642.

[13] Richard Socher et al. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, WA: Association for Computational Linguistics, 2013, pp. 1631–1642.

[14] Vladimir N. Vapnik. *The nature of statistical learning theory*. 1995.

[15] Shuai Xiao, Mehrdad Farajtabar, and Xiaojing Ye. "Wasserstein Learning of Deep Generative Point Process Models". PhD thesis. Georgia Institute of Technology, 2017.