Listen to Your Weather - Music Recommendation Based on Forecast Weather

Di Mu
dm3686
dm3686@columbia.edu

Yunchen Yao
yy2949
yy2949@columbia.edu

Wen Zhan
wz2539
wz2539@columbia.edu

Abstract

We constructed an hourly playlist generator system based on the real-time weather forecast using the 'OpenWeather' API, and 'Spotify' API. We performed experiments for temperature model selection by varying response encoding method, window size and model type. We finally deployed a real-time LSTM model in predicting both weather category and temperature. Due to privacy issues and access limits, we are not able to fetch global music playing data and construct any recommendation models, thus we recommend songs based on the user's playing history and top 50 music playlists provided by API. Our final product is a website where users could login, view 7 hours ahead temperature prediction results and play with the recommended playlists. This is the first hourly weather-based music playlist generator we have seen so far.

1. Introduction

For the powerful impact of music, modern music-psychologists believe that music is a "magic cure" for pressure and bad mood relief. On the other hand, some psychological researchers for a long period of time keep studying the common factors of people's emotion and found that weather is actually a very common factor on an individual's emotions[1]. Thus, a hypothesis naturally raises: Weather is associated to one's mood, and music could cure bad moods, thus if we could monitor temperature changes, and recommend songs according to it, we could in part release partial of those bad emotions at an very early stage. Put it further, this might also be used as a new approach to prevent depression and many other mental problems.

In fact, as modern society condemned humans to be lonely, people tend to spend more and more time listening to music. According to edison research[3], the average time spent listening weekly in 2020 is 15 hours and 12 minutes. This accounts for a large fraction of time when they are on their own. This further confirms the necessity of building a temperature-based music playlists recommendation system.

Actually, building a music recommendation system is not a new topic now, but current recommendation systems are mostly constructed based on the user's playing history and his/her music taste. But under our assumption described above, weather could greatly affect people's mood, which in turn affects their choice of music at a specific moment. For instance, it is commonly believed that higher temperatures could bring a depressed person up, thus they might be more willing to listen to rock music; On the contrary, lower temperature always cools people down, so a soft piano music might fit their mood better. Although it is admitted that currently there are a few music recommendations already take temperature into consideration. Yet their recommendations are mostly daily-based, which could not fit people's fast changing mood well, and the range of temperature in a day is quite large in general. This drives us to our project: a real-time hourly-weather-forecast-based music playlist generator.

We obtained historic and real-time weather data from the 'OpenWeather' API, and fed them into our real-time model. It might be confusing why we utilized new models rather than directly getting the predicted results from the official resources. This is because the current available models used in the public forecasting institute are all climate models which require a large computational power and resources, and cost a lot. They simulate a virtual Earth's climate system, and track every tiny change in every single second and perform extremely expensive computations to get accurate predictions. Clearly, our music recommendation system does not need extremely accurate data. Thus, we decide to use machine learning models and auto-regression time series models to perform the predictions. In fact, the API also provides a model which integrates machine learning models and climate models. However since the prediction models in the API are not free and they have set up a lot of restrictions, we finally decide to rely on our own models.

In the modern climate data science world, the most commonly used models are statistical ARIMA model, deep learning-based LSTM model and transformer model. We

decide to implement all these three models and compare the performance of them. Then choose the best model to predict the weather and make real time music recommendations. More details are described in the experiments section.

After model selection, we used 40 years of historic data to train the deployed model, and used the recent 48 hours data as the test set to predict the next 1 hours weather, and used the predicted 1 hour data with 47 most recent hours data to predict the next 1 hour again... This process repeats 7 times until we get the prediction of the next 7 hours. Next, we chose the most recent hour prediction as the predicted weather category to make a recommended playlist for users. More specifically, we assign valence weight, instrumentalness weight, energy weight, danceability weight and acousticness weight to different weathers. Since we also form song databases for each user, which includes current user's playing history and daily global top 50 popular song playlist auto-generated by spotify, we could use the assigned weight to calculate the scores for all songs in the user database for different weathers. We then sort all weights in descending order, and select the top songs to generate a recommended playlist. This whole is scheduled to run in the first second of every hour, and the data is acquired and stored in a temporary text file to ensure our website loads fast enough.

We finally built a website, the original page displays our predicted temperature and weather category for the next 7 hours. Users could then click the bottom button to login, and give authorization to add playlists in their account. The website then would direct them into the page displaying all recommended songs.

2. Related work

Generating playlists of songs based on weather conditions is basically one of implementations of context ware recommendation. As recommendation and retrieval system and weather forecast have both been mature and widely applied study fields with significant amount of work, context aware recommendation is a relatively new topic, which is introduced in 2006. Our literature review of this study focuses on two aspects, one is predictions on weather data and the other is context aware recommendation in music domain.

2.1. Weather forecast

Traditional weather forecasts are usually based on the numerical methods, which utilize computers to simulate the natural meteorological processes. The method is based on the real physical processes, so every step is traceable and the result is explainable. For example, National Oceanic and Atmospheric Administration's (NOAA) National Weather Service[2] uses Doppler radar, weather satellites and other equipment to collect meteorological data and applies Numerical Weather Prediction (NWP) method to predict weather. However, the traditional methods have drawbacks that are hard to eliminate. One of the main tasks for NWP is to solve a large amount of partial differential equations describing the meteorological system by numerical models. Thus many approximations are involved to convert derivative and matrix manipulation into the format that computers could make calculations. Also, computational simulation could hardly take all physical processes into consideration and there are still unresolved processes that require advanced physical parameterization. These limitations make it difficult for numerical methods to reach higher accuracy. Moreover, the process are computationally demanding of using numerical method to solve partial differential equations and considering all the physical states in each small steps. And computers need to deal with a large amount of data from different observation instruments, which will also take much time.

Considering the limitations of numerical method itself, the high requirements for computing resources such as calculation speed and capacity and the difficulties of collecting data from observational system, machine learning and deep learning methods have been introduced to weather forecast to accelerate the prediction process and increase the accuracy. In 2019, Scher and Messor[9] used general circulation models(GCMs) for weather forecast and climate date's generation and initially obtained some results in the meteorological area. In 2020, NOAA[4] has announced to build a new partnership with Google, in order to make better use of environmental data using artificial intelligence. Although nowadays NWP models still have a more stable and accurate performance than deep learning based models[5], deep learning methods are hopefully to become a great stride in the research of weather forecast in the future.

Nowadays, many weather APIs have provided functions of predicting weather based on the NWP method enhanced by machine learning. However, most of APIs need subscriptions and have limited quota for requests, which is not convenient if we want to update the predicted weather frequently. And also we are thinking about putting aside the numerical method and predicting the weather totally based on statistical and deep learning models to explore its feasibility and effects. For the above considerations, we decided to make weather forecast by ourselves rather than get predicted data from APIs directly.

2.2. Music recommendation

As recommendation system has been studied deeply and extensively, how the problem of music recommendation differs from other recommendation system is that preferences for music are complicated and influenced by many factors

including genres, artists, albums and geographical location. Besides, since we intuitively imagine that people's preference for music change according to external elements like mood, weather and surroundings, context awareness is introduced into the field of music recommendation[6]. In a word, music is subjective and people's tastes for music varies from person to person, which make recommending music a tough task.

Although reducing by recommending tracks with homogeneous metadata, like similar artist or albums, can be an efficient strategy for music recommenders[8], it ignores that tracks created by one artist may be varied and listeners just enjoy some of them. In the meantime, some recommenders prefer to base on listeners' historical behaviours and usage patterns rather than focus on attributes of tracks. Following such strategy, collaborative filtering performs generally better in music recommendation domain[10]. However, such approach is subject to several aspects. First, collection of usage data is sometimes challenging and limited because it is difficult to discover similarity between listening patterns and users do not always tend to give their feedback or ratings. Additionally, due to the lack of usage data, fresh tracks inevitably tend to be hidden. Such issues matter especially when recommending to non-mainstream music listeners. Currently, combination of two strategies seems to predict much more accurately since playlist is created by utilizing both acoustic features and attribute data of songs[11].

Context aware system applied on music recommendation seems to be reasonable because perception of listeners change with the situation in which they are listening, while it is still at early stage with high potential. Considering what similar users listened in same context, music recommender is able to be more accurate[7]. Hence our study try to focus on music recommendation in the context of weather.

3. Data

3.1. Weather Data

Our weather data is acquired from the OpenWeather API. Openweather API is a weather API that could help us instantly access real-time and historical data. It also has some built in models that could complete the forecasting task. However, since it is not a free platform, and the predicted model is a black box weather model, we built and deployed model by ourselves. The historic data and real-time weather data are acquired directly from this API. The weather data has 25 features, includes time-related features, location related features and weather-related covariates.

For the historic data, we acquired and stored them as a csv file. It contains hourly observations from 1979-01-01 00:00:00 to 2021-12-11 23:00:00. Since there are too many features, we are forced to perform feature selection to reduce training time, even though it's recommended to include as many features as possible to get a high-accuracy model. Typically, we might consider ranking Information gain, performing Chi-square tests, calculating Fisher's Score or correlation Coefficients to do the feature selection. However, since temperature is our target variable, we have some prior knowledge in this circumstance.

Self-defined Weather categories	weather type in original dataset
sunny	Clear
cloudy	Clouds
windy	Squall, Tornado
foggy	Mist, Haze, Dust, Fog, Smoke
rainy	Drizzle, Rain, Thunderstorm
snowy	Snow

Table 1. Types of weather we defined

Excepting for temperature, we also select pressure and wind speed as the weather-related features. Regarding time-related features, by plotting the feature trends, we found that the annual cycles looks sinusoidal. Thus, we considered sinusoidal functions. Since we would like to capture a wide range of seasonal effects, we used four sinusoidal basis functions to model hour frequency ($\frac{1}{365.25yrs \times 24hrs \times 60mins \times 60secs}$, one cycle every 365.25 days in seconds expression) and month frequency($\frac{1}{24hrs}$, one cycle every 24 hours in hours expression). The below expression also shows that a simple linear combination is good enough to capture the seasonal effect, thus we could actually use them as linear covariates.

$$\beta_1 sin(\frac{2\pi X_{hour}}{24}) + \beta_2 cos(\frac{2\pi X_{hour}}{24})$$

$$= \sqrt{\beta_1 + \beta_2} cos\{\frac{2\pi X_{hour}}{24} + arctan(-\frac{\beta_1}{\beta_2})\}$$

Due to 'spotify' API limited data access issue, we can't get public music playing data from all users, so we are not able to construct a recommendation model by ourselves. This forces us to assign scores to songs, and recommends songs according to it. Thus, we decide to assign weather types to our data. Among 25 features provided by the API, there is a column names 'weathermain', which classify weathers into 13 categories. We further map them to 6 main types. The detailed mapping could be seen from Table 1. Certainly, when fitting models we treat them as categories and encoded them as numerical value.

So the final features include:

- Response variable: X_{temp}, $X_{weather\ type}$

- time-related variables:
  $X_{day_{sin}} = sin(\frac{2\pi X_{hour}}{24})$,
  $X_{day_{cos}} = cos(\frac{2\pi X_{hour}}{24})$,
  $X_{month_{sin}} = sin(\frac{2\pi X_{timestamp}}{365.25\times24\times60\times60})$,
  $X_{month_{cos}} = cos(\frac{2\pi X_{timestamp}}{365.25\times24\times60\times60})$
- weather-related variables: $X_{pressure}$, $X_{wind_{speed}}$

For the real-time weather data, we get the json format data directly from the open-weather API using developer API key. Then we pre-processed and cleaned them using the similar approach as what we did for the historic data. One potential issue is that the API limits the query times, which is easy to exceed. We create several developer accounts to temporarily solve this issue when writing scripts, but certainly this would become a big problem if we at the end of the day deploy our app. One possible way to solve this problem is do web scrapping on the National Weather Service website.

3.2. Playlist Data

Our playlist data is acquired from the spotify API. When users login their spotify account on our website, we get access to the user's playing history. We would first examine whether they have a historic favorite playlist, if yes then all of them are stored to a temporary database, if no then we used the global daily top popular songs to construct this database. Later on, after we get the predicted weather types, we would immediately get the assigned scores of alence weight, instrumentalness weight, energy weight, danceability weight and acousticness weight. Then we calculate the scores for each song in that database as well as their linear combination as the final score. Next, we sort the final scores in descending order, and then select the top songs as the baseline songs and feed them to the recommendation model provided by the spotify API to generate recommended songs and form a recommended playlist. The format of data we acquired is in json form, we cleaned and used it using basis python operations.

4. Methods

Our project mainly has two tasks. One is predicting weather types and temperature for the next 7 hours, the other one is generating new playlists for current user every hour based on the predicted weather types for the next hour using user's current playlist data and daily global top popular playlist data.

For the first task, we require a real-time temperature prediction. However, the available models that are obtainable from the open sources are mostly climate-based models, which require high computational power. We decided to implement our own models. We performed experiments shown in the experiments section, and then select the LSTM model to deploy.

For the second task, our initial thought is to construct our own recommendation system. However, since we have no access to the public music playing data, we are not able to build our own model. Instead, we could only use the model provided by the spotify API. The way that spotify API provided to generate recommended playlist for user requires 5 songs as the input baseline songs. We should find a way to get that 5 songs every hour based on the predicted weather types.

We first form a database to get that 5 songs. If the user has at least one playlist, then we use all songs in their playlists to form that database. If the user has no playlist in their account now, we use daily top popular songs playlist as the database.

We then come up with a way to get 5 songs every hour from the database based on the predicted temperature type. Spotify API assigns each song some metrics: valence weight, instrumentalness weight, energy weight, danceability weight and acousticness weight. We could assign weights for each metric according to the predicted weather types, then take the linear combination of all metrics and sort them in descending order. The top five songs would then become the baseline songs that we could feed to the recommendation model provided by the Spotify API, used to generate recommended playlist.

A notable issue is how to assign weights to each metric for different weather types. According to psychological researches, in higher temperature, people tend to listen to more energetic songs, so the danceability, energy, valence might be higher. For each temperature category, we assign weights according to the current research results. This is a way to get songs, but clearly this is a potential weakness of our method.

4.1. ARIMA

Our baseline model is the ARIMA model, which is the most commonly-used methods when dealing with time series data. It is generally used to describe the autocorrelations in the data. But it has a strong assumption on data: Data should be stationary (white noise). If this is not the case of the original data, we should transform it to be suitable for the model assumption.

We first did a exploratory analysis on our target feature: temperature. As seen from the Figure 1, the original plot of temperature and the first difference does not satisfy the white noise assumption, while the second difference of temperature data looks much better, since the variance is somewhat stabilized. The other typical methods used to stabilize

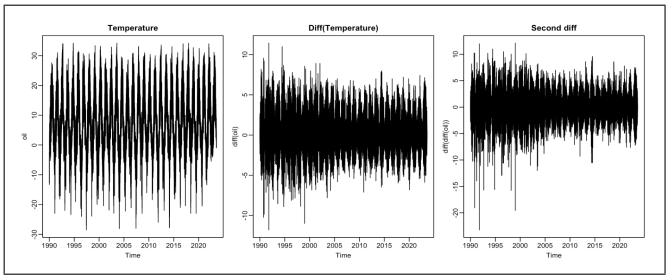

Figure 1. Plot of temperature, the first difference and the second difference of temperature

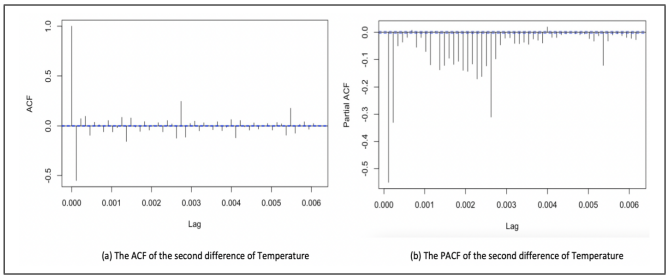

Figure 2. Plot of ACF, PACF of the second difference temperature

variance is taking log, but it also works bad here.

The figure 2 plots the ACF and PACF of the second difference temperature. This identifyies the dependence orders of the model. From these plots, we propose two ARIMA models: ARIMA(1, 2, 0) and ARIMA(1, 2, 1). The figure 3 shows the residual analysis for the better model ARIMA(1, 2, 1) among the two models. The model seems white noise. However, the ACF of the standardized residuals shows an departure from the model assumptions in particular when lag is small. The QQ plot of the residuals shows that the assumption of normality is reasonable, with the exception of some outliers. Moreover, the p-values for Ljung-box statistics all below the threshold dotted line, so we reject the null hypothesis that the residuals are independent. Therefore, the model assumption is violated, and the models are not valid model for prediction.

The reason that the model is not white noise might because each observation we got is hourly temperature, which has large variance. The first way we found to resolve this issue is to change the hourly data into monthly average data. This works as could seen from the the figure 4. The green lines are predicted value, while the orange lines are actual value. We could see that the forecasting using daily data looks like a horizontal line with no trend and pattern. Yet the temperature forecasted using monthly average data looks quite accurate. At least the actual and predicted data have same trend.

Though this works, we still hope to get the hourly forecasting temperature as accurate as possible. This points

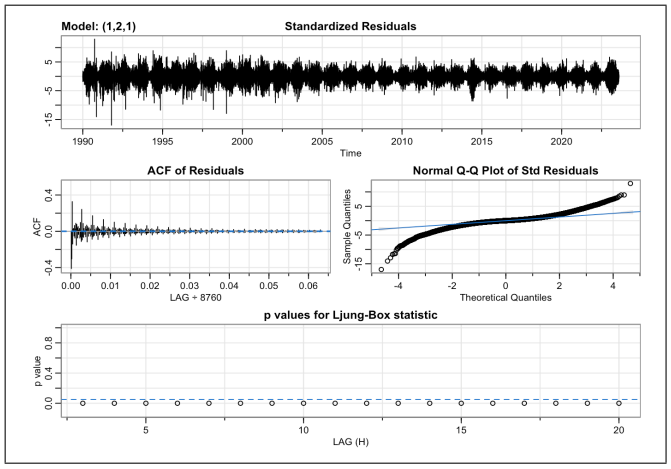

Figure 3. Residual diagnostics plot for ARIMA(1, 2, 1)

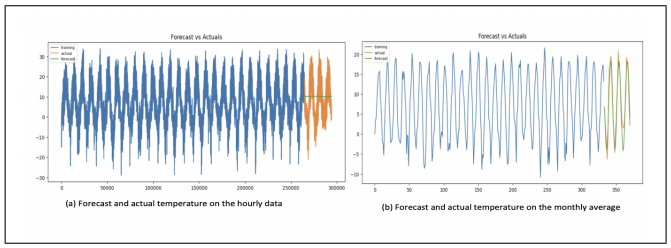

Figure 4. The predicted and actual temperature on hourly and monthly average data using ARIMA model

to the fact that the traditional auto-regressive model is not suitable for predicting the hourly temperature, which leads us to the deep learning models.

4.2. LSTM

As discussed above, the baseline model ARIMA's model assumption is violated, so it's not appropriate to deploy that model. Then, we come to the deep learning world to seek for improvement of results. The first deep learning model we tried is LSTM model, it follows the artificial recurrent neural network (RNN) architecture. We used 48 hours window as number of lags (hours back) to predict one step ahead(one hour ahead) data. In the training process, we have 40 years historic data. Each observation takes $X = X_i, X_{i+1}, ...X_{i+47}$ to predict X_{i+48}, where each X_i includes values for temp, daycos, daysin, monthsin, monthcos, pressure, wind speed – 7 covariates. We have in total 294227 training samples at the beginning.

The architecture of the LSTM model is shown in figure 5. It has four gates: Forget Gate, Learn Gate, Remember Gate and Use Gate in each layer. These gates together are used to model both Long Term Memory and Short Term Memory. More specifically, it takes the current and previous

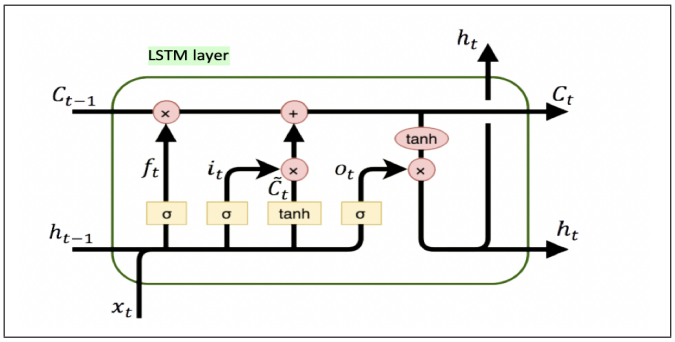

Figure 5. Architecture of LSTM model

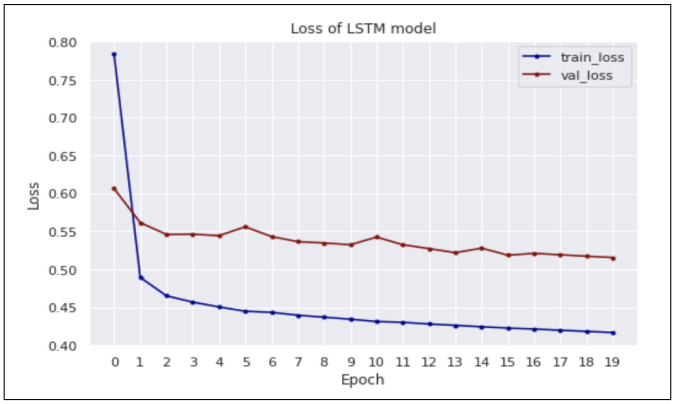

Figure 6. Loss curve of LSTM model

layer's short term memory as input in the learn rate to make prediction; it takes previous long term memory as input in the forget gate to determine which information to forget; In the Remember Gate, it combines both previous short term memory and current data to produce output; finally in the use gate, it combines previous long term memory and previous short term memory to produce output for the current data.

We used mean absolute error $MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$ as the loss function to train this model. Comparing to mean square error, mean absolute error(MAE) is more robust to data with outliers. Since we have large dataset, it's easy to get some outlier points, thus we use mean absolute error here. The lower value of it implies higher accuracy of the model. The loss we achieve on the test set is 0.0591, which is small enough to indicate the high accuracy. The entire training and testing loss curve is shown in Figure 6.

Since our goal is to predict 7 hours ahead. We treat the newly predicted data as known, and use the current most recent 48 hours data to predict next 1 hour data again. We repeat this process 7 times in total to get 7 hours prediction. More details of why we choose this setting is shown in the experiments part.

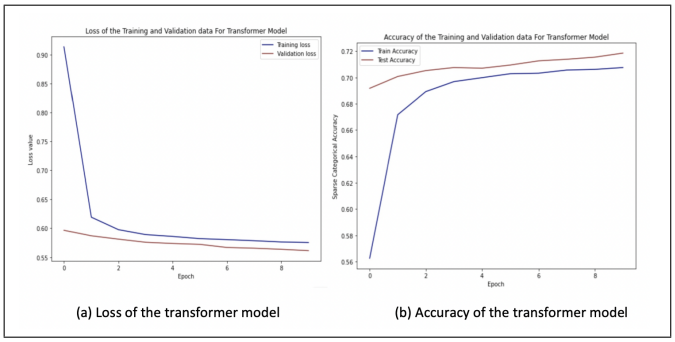

Figure 7. Loss and accuracy curves of the transformer model

4.3. Transformer

The other deep learning model we tried is transformer model. Unlike LSTM, it adopts the self-attention method, and weight the significance of different part of the input data differentially. It is also designed to deal with sequential input data like the temperature data. Normally it achieves worse result than LSTM model, and it is the case here. We converted this problem to a classification problem here where increasing temperature is denoted as 1 and decreasing is denoted as 0. The loss we use here is sparse categorical cross-entropy. The loss we get is 0.533, and the accuracy we achieved is 0.7386. The entire loss and accuracy curves are shown in Figure 7.

5. Experiments

We performed experiments by varying model types, the way to predict next 7 hours data and how to encode target variables. We have three model choices, including arima model, transformer model and LSTM model. We have two ways to predict next 7 hours data: One is letting our target variable be in window of size 7, so that we predict 7 hours ahead in each time; the other one is letting our target variable be in the window of size 1, so that we predict 1 hour ahead in each time, and then treat this newly predicted data as known and then keep using the most current 48 hours data to predict the next 1 hour data. Then repeats this process again and again until we get 7 hours predicted data.

Since we have two tasks in modeling, one is forecasting next 7 hours temperature, the other one is predicting next 7 hours weather types, we have two target variables: temperature and weather type. For weather type, we treat it as the classification task, so we could only encode it as numerical integers. For temperature, we have two ways to encode our target variables: one is treat it as a numerical variable so that we could perform regression analysis and predict the direct result; the other one is treat it as binary variable so that we could predict whether the temperature is increasing

Model	Target Variable	Times ahead	Loss Type	Loss on Test Set
LSTM	Temp	1	MAE	0.0553
LSTM	Temp	7	MAE	0.1298
LSTM	Type	1	MAE	0.4148
LSTM	Type	7	MAE	0.6538
Transformer	Temp	1	MAE	1.2037
Transformer	Temp	7	MAE	0.2629
Transformer	Type	1	MAE	1.1706
Transformer	Type	7	MAE	1.2511

Table 2. Experiment of target variable, model, times ahead

or decreasing.

As seen from the table 2, when all using maximum absolute error, the LSTM model achieves lower loss than transformer model in general. When predicting temperatures, LSTM is more accurate in predicting one steps ahead than predicting seven steps ahead. On the contrary, the transformer model tends to have lower loss (higher accuracy) when predicting seven steps ahead than one step ahead. This might because of the forgetting mechanism of the LSTM model. When predicting temperature with seven steps per time, transformer model does not have very large loss comparing to LSTM model. However, when predicting one hour ahead, it achieves significantly worse result. This suggests that we should deploy one step ahead LSTM model to predict temperature.

Among weather types prediction models, LSTM model also outperforms transformer models. But the advantage of LSTM model is more obvious in this case, and the MAE loss of either one step ahead model or seven step ahead model all significantly smaller than that obtained by the transformer model. Also, among LSTM models, predicting one step ahead weather type still achieves better performance than predicting seven steps ahead weather types. Among Transformer models, one step ahead also achieves slightly better results than seven steps model. This suggests that we should deploy one step ahead LSTM model to predict weather types.

A trivial thing is that predicting temperature in general achieves better results than predicting weather types. This suggest that if possible, deploying step ahead LSTM temperature model is the best choice. However, recommending songs based on weather types makes more sense than based only on temperatures. Since it's extremely hard to say what's the relationship between temperature and preferred songs. Comparing to it, we at least know the direction of the correlation between weather types and song metrics. Therefore, we finally decide to deploy the one step ahead LSTM weather types model.

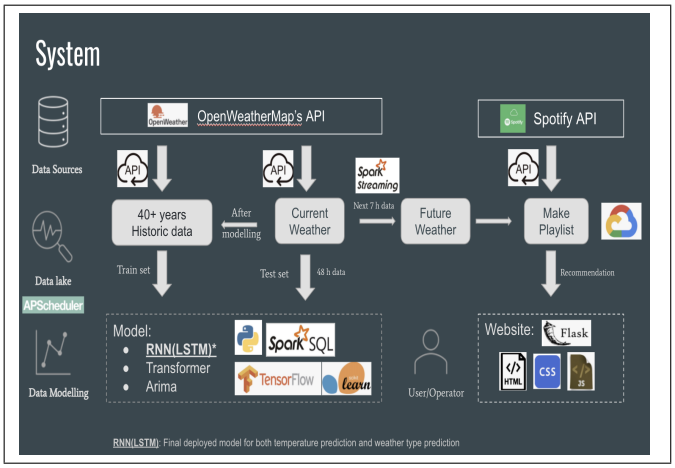

Figure 8. System of our project

6. System Overview

6.1. Software architecture and tech stacks

The system of our project is shown in Figure 8. It includes a backend to make weather prediction and music recommendation and a frontend for user interface.

The work of backend can be mainly divided into two parts. One is for weather prediction. We acquire 40 years historic data from the Openweather API, with Apache Spark to do the data pre-processing. We train our LSTM models written in keras with processed data and saved our models in an h5 file. Then we schedule to predict the weather in the next 7 hours at the beginning of every hour. Every hour, we use 48 hours window to predict the next 1 hour data. The process repeats for 7 times until we get 7 hours ahead predictions. The predictions (temperature, and weather type) along with the current weather are saved into a file for later display on the website.

Another part of backend is music recommendation. We use one hour ahead weather type prediction for the recommendation. There are six weather categories, we assign each of them weights to songs' audio features: valence weight, instrumentalness weight, energy weight, danceability weight and acousticness weight. First, we get users top 50 tracks in the recent 4 weeks from Spotify API after users grant our access to their data on Spotify. If there is no top track for users, we instead get tracks in the global TOP 50 playlist provided by Spotify. Second, according to the predicted weather type, we take the linear combination of the corresponding weights to score the tracks, sort the tracks and choose the top 5 as seed tracks for recommendation. Then we feeds Spotify's recommendation system with seed tracks and get the recommended playlist every hour.

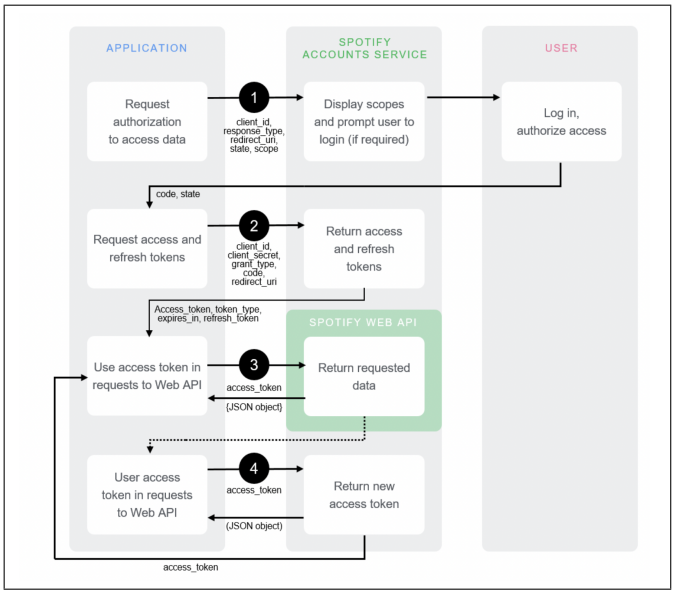

Figure 9. Spotify authorization flow

For the frontend, a user interface built with css and html is designed to display the predicted weather and the recommended playlist. We use Flask to develop our web applications. On the first page, we read current and 7 hours ahead predicted weather from the file we mentioned above, and display the data on the page. To show to the icons of weather types, we refer to: https://erikflowers.github.io/weather-icons/

To provide the recommended playlist, our application need to get access permissions to users' data on Spotify. When users click the button at the bottom of the first page, our app request authorization to Spotify's data. Then web page is redirected to the login page of Spotify and Spotify prompt users to login with their Spotify account. After users' authorization, our app request tokens from Spotify and then use access tokens to get users' top tracks from Spotify API by "Get User's Top Items" request. The authorization flow is shown in figure 9.

After we obtain the recommended playlist based on users' top tracks or global top tracks, the second page will show up with the weather type in the next hour and a window including all recommended tracks. The user can play the whole playlist from the beginning or click every track they want to listen to. The playlist will be saved to users' Spotify account automatically.

6.2. Potential bottlenecks and improvements

6.2.1 Real-time Weather prediction vs. slow UI

Since it takes amount of time to run the predicted model and forecast weather, initially our UI runs quiet slow because of it. We used several approaches to solve this issue. We first fixed stored our models fit on the train set as h5 using keras model save function. Every hour when we fetch new data in 48-hours-window, we treat them as the test set and evaluate it on our fixed models. We don't update model and include the newly-observed data in the train set as planned before. This will not affect our model much, since our dataset has large size, one more observation is trivial to affect modeling results.

However, since we deployed deep learning LSTM model, which is quite complex with a lot of parameters, even loading the fixed model and evaluating on the test set is time expensive. Thus, we decide to schedule events that forecast 7 hours ahead temperature and weather types at the first second each hour and stores them as a text file. Every time we load the UI, only stored results in the text file are read. This reduces the running time of our UI quite a lot. We used BlockingScheduler from apscheduler package to schedule the events.

6.2.2 Recommended playlist vs. self-defined song metrics weights

The slow UI problem is the one we could solve by ourselves, but this self-defined song metrics weather-specified weights is hard to solve at this time. Since spotify only supports query on the login account, we can't get song playing data from other users. This makes it impossible for us to build a recommendation system by ourselves. So we could use the one provided by the spotify API, which takes 5 songs as baseline songs to generate a recommended playlist. In order to connect it with weather, we defined six weather types, and assigned each of them scores to the song metrics: valence weight, instrumentalness weight, energy weight, danceability weight and acousticness weight. With them, we could calculate the total score of each song, and select the 5 songs with highest total score to make the recommended playlist. The issue is obvious: First, the recommendation system is a black box to us, we could not combine user-based recommendation and content-based recommendation together as planned. Second, the scores we assigned to song metrics based on weather types are not accurate enough to relate weather and songs.

Based on public research papers, we basically know the direction of correlation between song metrics and weather. However, we don't know the exact numeric value that could describe this relationship well. Thus, we assume a non-informative uniform distribution on them as prior, and update them several time based on how we feel about the playlist it recommended to us. Our feeling to it could be regarded as the likelihood, so we could calculate the posterior scores and treat it as the new prior again. Finally, we get the scores that could produce good

Goal	Package
pre-processing	pyspark, numpy, pandas, datetime, pickle, string
modelling	tensorflow, sklearn, pmdarima, tatsmodels
get real-time data	pyowm, secret, requests, urllib, os, base64, json
Scheduling	apscheduler
App structure	flask

Table 3. Packages we used and our goal

enough recommended playlist. Though this bayes-based methods seems a lot more scientific than blindly making a simple assumption on scores, it's still not accurate enough to describe the relationship of songs and weathers. Further researches on how to model this relationship could be done in the future. But obviously, a much better improvement method is to build a recommendation by ourselves. We could do more research on how to get public songs playing data.

6.3. Software packages

The table 3 shows the main packages we used. Some auxiliary packages are not shown. The second column shows the main packages we used, while the first column shows our goal to use these packages.

6.4. How to use our application

Figure 10 shows an overview of our application. The blue page is the initial website. At the top, it shows the location, current temperature and weather type; at the middle, it shows the seven hours ahead predictions on temperatures and weather types; at the button, there is a button where user could click to login and give authorization to our application. After authorize us the access to their account, the user would be redirect to the green page, where user could play the recommended songs, share it to their friends, and save the whole playlist to their account.

7. Conclusion

In this project, we explored a possible way to connect hourly predicted weather and recommended songs. It might turn out to be very helpful for people to obtain a song that could represent their mood, and might further be able to prevent depression at an early stage. We trained three types of models with some variations on model settings in predicting temperatures and weather categories. We generate recommended playlists for users based on their current playlists

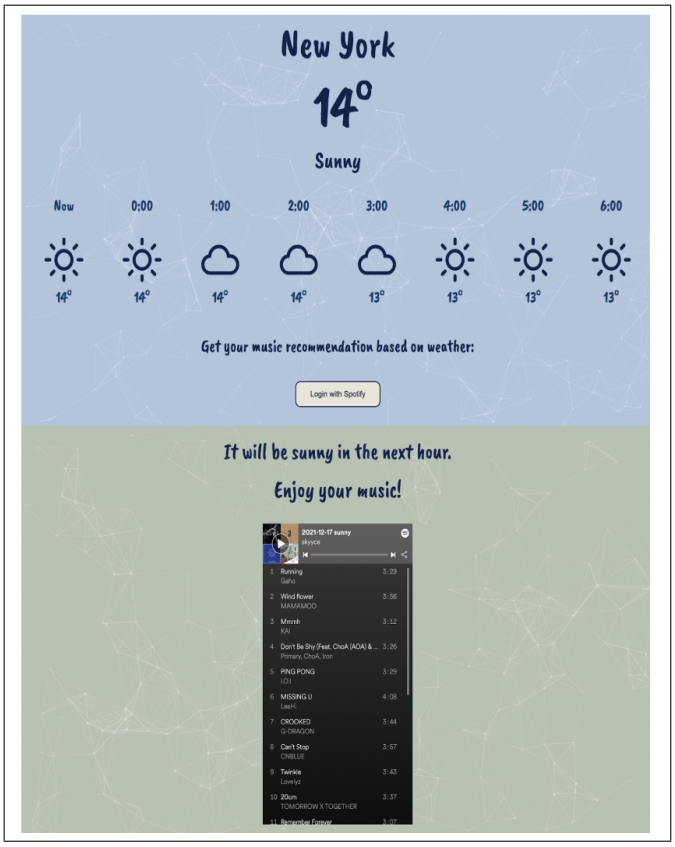

Figure 10. Appearence of our application

using spotify recommendation system. We built a website to get users interacting with an hourly recommended playlist.

We become more familiar with the packages and API we used and be more confident to build websites. We gained precious experience in working with very large dataset, and being able to use big data tools such as pyspark to clean it. We learnt how to probe big problems into small parts and solve them one by one. The biggest challenge we met is we cannot get public data of song playing so it's not possible to build a recommendation system by ourselves. But we still figured out a solution that partially solved this problem, though not accurate. In the future, we might consider using web-scraping to get a public song playing dataset and weather dataset, and try to build our own recommendation model. Also, we would add more interactive features in our website. For example, users could enter their current mood in English or numeric values, and we could also recommend songs to them. This might become a very supportive feature that could help users find a song that could represent their mood and erase their bad emotions.

References

[1] The effect of weather on mood, productivity, and frequency of emotional crisis in a temperate continental climate. In *International Journal of Biometeorology volume*, volume 32, page 134–143, 1988. 1

[2] 6 tools our meteorologists use to forecast the weather. https://www.noaa.gov/stories/6-tools-our-meteorologists-use-to-forecast-weather, 2017. 2

[3] Numbers and facts you need to know about audio content in 2021. https://www.business2community.com/digital-marketing/numbers-and-facts-you-need-to-know-about-audio-content-in-2021-02398919, 2021. 1

[4] John Bateman. Ai agreement to enhance environmental monitoring, weather prediction. https://www.noaa.gov/media-release/ai-agreement-to-enhance-environmental-monitoring-weather-prediction, 2020. 2

[5] Alex Bihlo. A generative adversarial network approach to (ensemble) weather prediction. *Neural Networks*, 139:1–16, 2021. 2

[6] Marius Kaminskas and Francesco Ricci. Contextual music information retrieval and recommendation: State of the art and challenges. *Computer Science Review*, 6(2-3):89–119, 2012. 3

[7] Jae Sik Lee and Jin Chun Lee. Context awareness by case-based reasoning in a music recommendation system. In *International symposium on ubiquitious computing systems*, pages 45–58. Springer, 2007. 3

[8] Aäron Van Den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Neural Information Processing Systems Conference (NIPS 2013)*, volume 26. Neural Information Processing Systems Foundation (NIPS), 2013. 3

[9] Sebastian Scher and Gabriele Messori. Weather and climate forecasting with neural networks: using general circulation models (gcms) with different complexity as a study ground. *Geoscientific Model Development*, 12(7):2797–2809, 2019. 2

[10] Malcolm Slaney. Web-scale multimedia analysis: Does content matter? *IEEE MultiMedia*, 18(2):12–15, 2011. 3

[11] Yading Song, Simon Dixon, and Marcus Pearce. A survey of music recommendation systems and future perspectives. In *9th International Symposium on Computer Music Modeling and Retrieval*, volume 4, pages 395–410. Citeseer, 2012. 3