

Homework 3 – Markov Models and Time Series Forecasting

Due: May 6, 2025

Overview

Part 1: 2nd-Order Markov Chains

You will be building a 2nd-order Markov chain to generate song lyrics. In some ways, this assignment will be reminiscent of the novel generator you created in RAIK 184, with two key distinctions: we are using a 2nd-order chain (i.e., a sequence of two words will be used to predict the next word), and the predicted word will be dictated by the Markov chain's transition probabilities, which constitute a weighted probability distribution.

Part 2: Time Series Forecasting

Time Series forecasting has no one-size-fits-all solution, with several different approaches and algorithms. In this section, you will work with field-based meteorological observations to predict wind power generation from turbines in the future. The objective to estimate wind power generation with this data can be tackled by many implementations, so the goal of this part is for you to experiment with feature engineering, lag features, model selection, or optimization techniques in order to most accurately predict wind power generation.

Part 1: 2nd-Order Markov Chains

Complete the tasks below. Your completed code file will be your submission for this part.

1. Either download the Taylor Swift lyrics corpus [here](#) or select your own artist and locate and download a corpus of their songs.
 - a. As a non-Swiftie, I would understand anyone choosing a different dataset, so you are welcome to do so!
 - b. You don't need a comprehensive collection, just one that includes a multitude of songs over which you can train your data.
 - c. We'll be training and generating by line, so ensure whatever data you use has reasonable newlines and is not just a combined jumble of text.
2. Using Python, open the data and prepare to process it line-by-line.
3. For each line, you should remove all punctuation, apply lower-case, and split the line into a sequence of words.
4. Next, we'll need to iterate over each word on the line in order to learn our probabilities.
 - a. First, the Markov chain needs starting probabilities to select a first word for each line. I recommend creating a dictionary to map starting words to their frequency. This dictionary will define a probability distribution!
 - i. To do this step, you can make a string to integer map which simply counts how often a word begins a line.
 - ii. That is, if "the" starts 10 lines, you might have an entry in your dictionary like: {'the': 10}.
 - b. Second, we need state transition probabilities. Because this is a 2nd-order model, you'll want to map 2-tuples representing a sequence of words to their subsequent words.
 - i. Consider the following example lyric:

When you think Tim McGraw
I hope you think my favorite song

- ii. Using a nested, dictionary for word count with the outer dictionary keys being 2-tuples, the following might be part of our dictionary:

```
{ ("", when) : {'you' : 1}, ('when', 'you') : {'think' : 1}, ('you', 'think') : {'tim' : 1, 'my' : 1}, ('think', 'tim') : {'mcgraw' : 1}, ('tim', 'mcgraw') : {'\n' : 1}, ("", 'I') : {'hope' : 1} ...
```

- iii. You can probably already see why we need at least a fairly sizeable corpus since the 2nd-order model needs to observe a lot of word sequences to build its probability list.
- c. Note that you'll need to handle the last two words specially. Mark the next word somehow as the end of the line (e.g., \n or "end" or similar) and move on to the next line.
- d. Additionally, you'll need to account for the second word of a line not having a first word in the 2-tuple. I recommend either having a separate dictionary for just the second word that doesn't use tuples as keys or just use the tuple where the first word is an empty string: ("", <second word>).

5. After processing all the lines into counts of transitions, convert them to probabilities.
 - a. For the start words, you can just divide every count in the dictionary by the sum of all counts. This computes the likelihood of that word starting a line out of all line starts.
 - i. As an example, if you have a dictionary mapping start words to their count, the following code creates a probability-based dictionary called "norm_firsts" using dictionary comprehension:

```
starts = {k:v / sum(starts.values( )) for k, v in starts.items( )}
```

- b. For the transition probabilities, you'll do a similar step, but you normalize within each word tuple. Thus, we visit all word tuples and scale to the probabilities of the words that have transitioned from that tuple.
 - i. For example, using the dictionary snippet above,

('when' , 'you') : { 'think' : 1 } remains unchanged if that is the only time that tuple appears.

- ii. However, the key-value pair for tuple ('you' , 'think') becomes

```
( 'you' , 'think' ) : { 'tim' : 0.5 , 'my' : 0.5 }
```

6. Now we're ready to generate text!
 - a. Generate 10 lines with 12 words max in each line.
 - i. I just selected these values based on what would reasonably match the existing lyrical structure, but it is clearly imperfect since there's no accounting for rhythm in this model.
 - ii. You are welcome to use adjusted values so long as you generate multiple lines and avoid extremely long outputs!
 - b. For each line you are generating, select a start word from the starting distribution.
 - i. For this and the transition probabilities, you need to make a weighted selection, not just a totally random chance. Words that are more often at the start are more likely to be a start.
 - ii. Numpy and Python's random library both provide helpful choosing function for sampling based on a probability distribution. For example, the following could be code to use the random library to select a start word

```
word0 = random.choices( list(starts.keys( )) , weights=starts.values( ) , k=1 ) [0]
```

- c. Select a second word based on the selected start word (either from your second list or feed the tuple (" , <start word>) into your transition probabilities) and repeat for all subsequent words.
 - d. Note all the conditions for ending a line:
 - i. If the 2-tuple does not exist in your transition probabilities, end the line.
 - ii. If the 2-tuple selects your end-of-line flag as the next word (e.g., '\n' or 'end' or similar), end the line.
 - iii. If 12 words have been selected, end the line. This will be important to prevent potential infinite loops if a 2-tuple leads back to a sequence of words that causes recurring selections.

7. Lastly, print your lyrical lines as text. For example, here is an output using the Taylor Swift song data:

all youre ever gonna hear in the pouring rain
 with drops of jupiter in his office
 then whyd you have knocked me off
 our song it brings
 so heres to silence
 baby im just gonna shake shake
 when we go crashing down we almost broke up the porch lights
 the bitter sting of what love gave me a nice little
 and i am not your typical
 you break my heart

From this example, we can readily observe why more context is generally needed for text generation than just two words! Modern transformers encode vast amounts of contextual information into their models. In this case, we even see some original lyrics make it through the model since there were not enough 2-tuples for variety. That said, with enough data and longer sequences, Markov chains can capture some useful information, like conversational autocomplete suggestions.

Part 2: Time Series Forecasting

Background

In this part, you'll work with time series data with meteorological variables and their impact on wind power generation from a turbine. Your goal is to forecast wind power generation by using a time series approach discussed in class.

The dataset represents a detailed hourly record, starting from January 2, 2017, and going till December 31, 2021. Features of the dataset include:

- Time - Hour of the day when readings occurred (time series index column)
- temperature_2m - Temperature in degrees Fahrenheit at 2 meters above the surface
- relativehumidity_2m - Relative humidity (as a percentage) at 2 meters above the surface
- dewpoint_2m - Dew point in degrees Fahrenheit at 2 meters above the surface
- windspeed_10m - Wind speed in meters per second at 10 meters above the surface
- windspeed_100m - Wind speed in meters per second at 100 meters above the surface
- winddirection_10m - Wind direction in degrees (0-360) at 10 meters above the surface*
- winddirection_100m - Wind direction in degrees (0-360) at 100 meters above the surface*
- windgusts_10m - Wind gusts in meters per second at 100 meters above the surface

- **Power** - Turbine output, normalized to be between 0 and 1 (i.e., a percentage of maximum potential output)

**Degrees are measured from 0 to 360. Since 0 and 360 represent the same spot on a circle, consider transforming these using sine and/or cosine. Also consider converting them to radians, instead of degrees.*

To Complete

The time series forecasting implementation is completely up to you! What this means is you have the freedom to choose your:

- Preprocessing and Feature Engineering steps (i.e. - feature selection, scaling, imputation, creation of features (e.g., rolling average features, lag features, time-based features), etc.)
- Model Selection (i.e. - autoregressive models (e.g., ARIMA, SARIMA), boosting algorithms (e.g., XGBoost, LightGBM), tree-based models (e.g., Random Forest), neural networks, or hybrid models that combine time series and machine learning approaches)
 - o I would encourage some exploratory data analysis work beforehand to strengthen your model selection decision!
- Training strategy (train/test splits, evaluation metrics, and visualization of results/residuals)

You will train your time series model with data from 2017 to 2020. Once you are satisfied with your trained model, you will create predictions for wind power generation on the test dataset. You will convert these predictions into a .csv file to upload to Gradescope. Upon uploading your .csv file, your predictions will be evaluated by the true 2021 values, and you will see how well your model performs.

Further, you can see how your model stacks up compared to your classmates' models on the leaderboard! Feel free to tweak your model and predictions based on your model's performance to climb the leaderboard!

CSV File Upload Specification

When you run your trained model on the test dataset, you will generate predictions for the power generation at each hour of 2021.

For autoregressive models, you will need to generate 8760 predictions in the future. For non-autoregressive models that rely on meteorological features for prediction, running the test data through your preprocessing pipeline and through your trained model should yield 8760 predictions for 2021.

You will then save these predictions to a .csv file to upload to Gradescope for scoring and your leaderboard placement. For the auto-scoring capability, the .csv file should follow this format:

1. It should contain **only 8760 rows and 1 column.**
2. The 1 column title must be **"predictions"**.
3. The .csv file should be titled **"predictions.csv"**.

Here is a sample .csv file with valid formatting. For simplicity, you may copy this .csv file in your code directory and rewrite the 'predictions' column with predictions for your trained model.

Use your name on your leaderboard submission name.

Example predictions.csv file: [predictions.csv](#)

Reflection

Write a brief reflection (250 word minimum) outlining and justifying the decisions you made during the exercise. Consider the following questions to guide your reflection:

- What steps did you initially take to explore the data?
- If applicable, what guided your decisions to augment, add, or handpick features? If you chose only to train on the target variable, what motivated this decision?
- Why did you choose the model you chose?
- Why did you choose the evaluation metric / visualization you chose?
- What did you learn broadly about time series forecasting from this exercise?

What to Submit

Turn in your code file for Part 1, code file for Part 2, and your **correctly formatted** predictions.csv file to Gradescope.