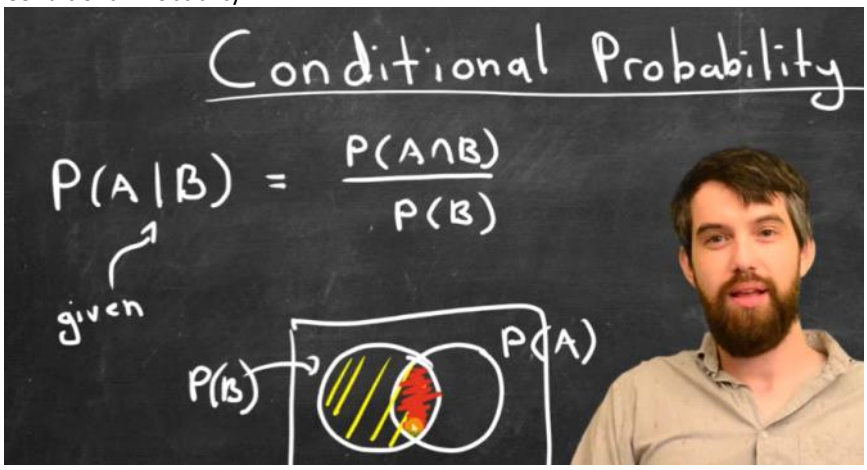
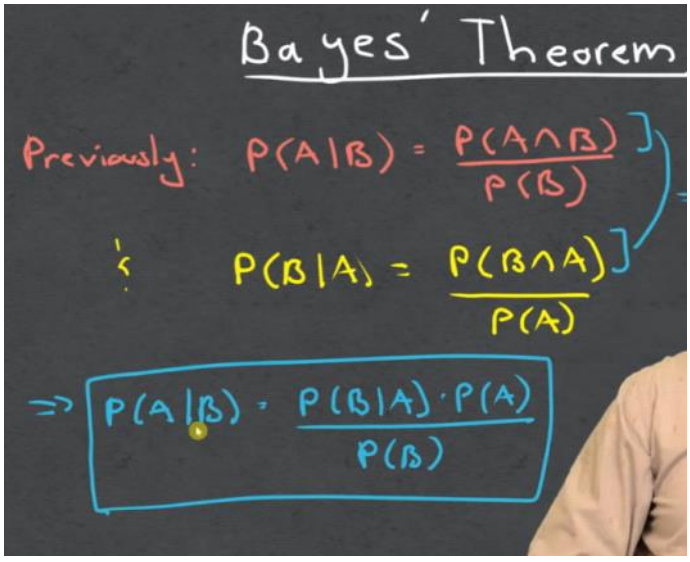
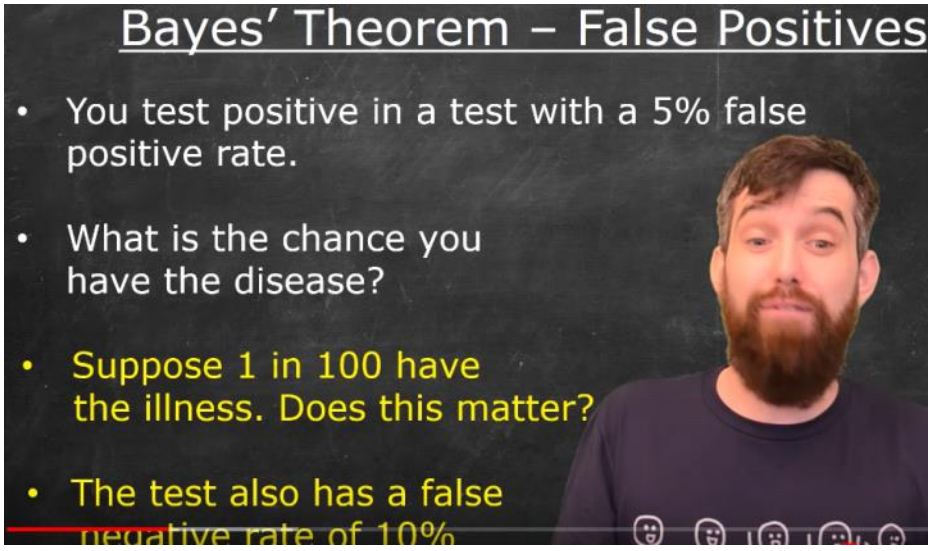
Part 2 - Probablistic Models

04 May 2023 03:47

Conditional Probablity:

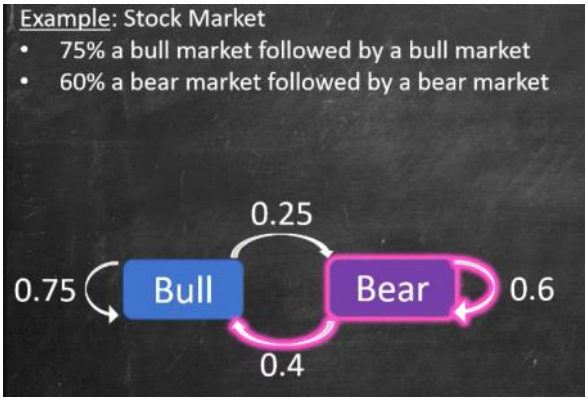


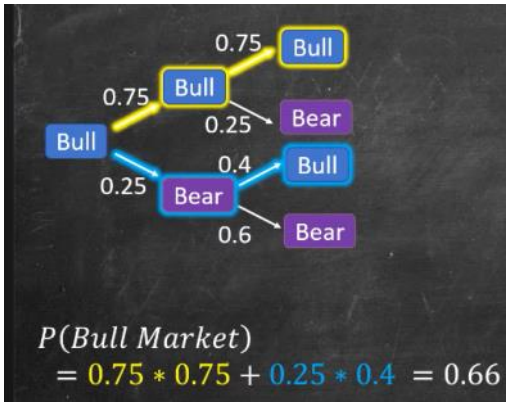


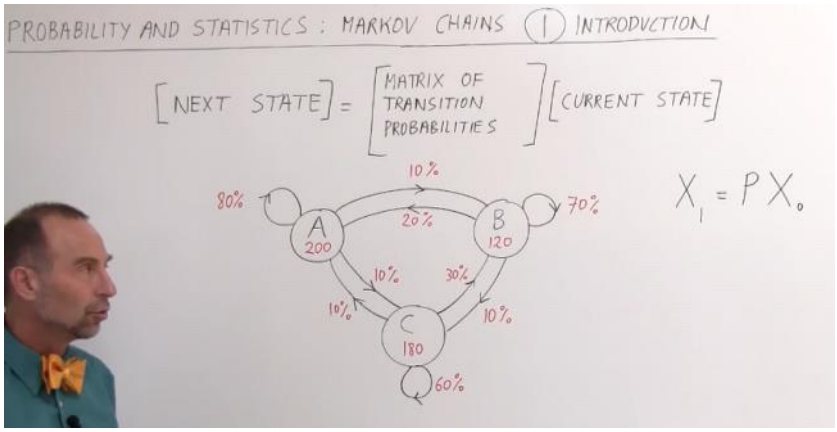


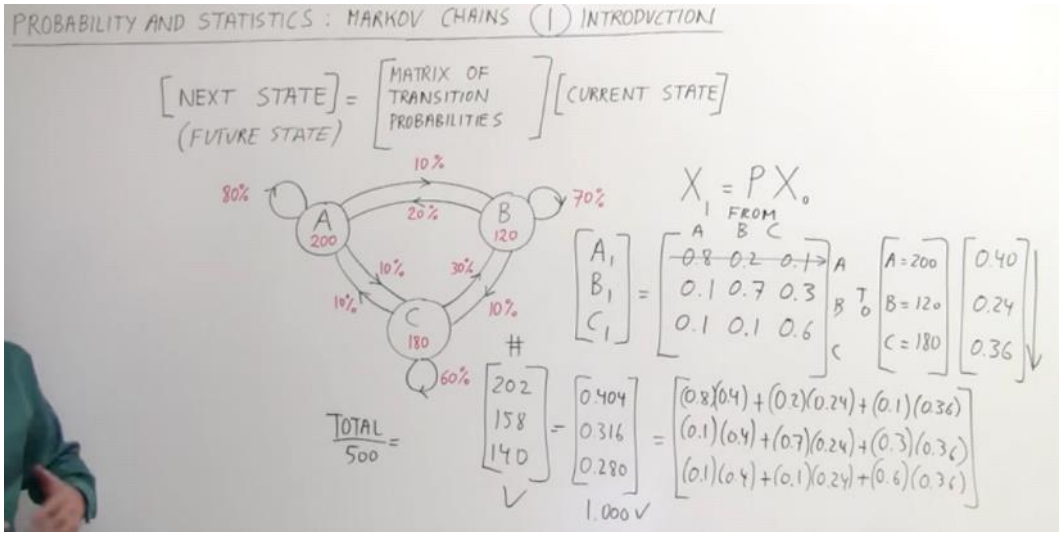
- The most important strength of bayes theorem is that , if you don't know one conditional probablity , there is still the chance you know the other conditional probablity.
- The rate at which something occur in a populace or a category, i.e rare or common affect the probability .
- As new knowledge comes, the probablity would be updated .

A *Markov Chain* is a sequence of events where the probabilities of the future ONLY depend on the present

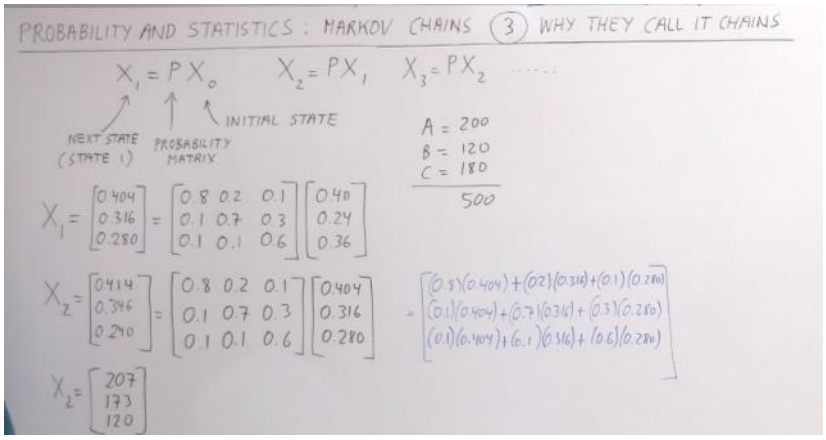


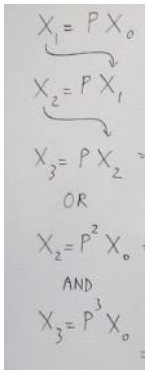


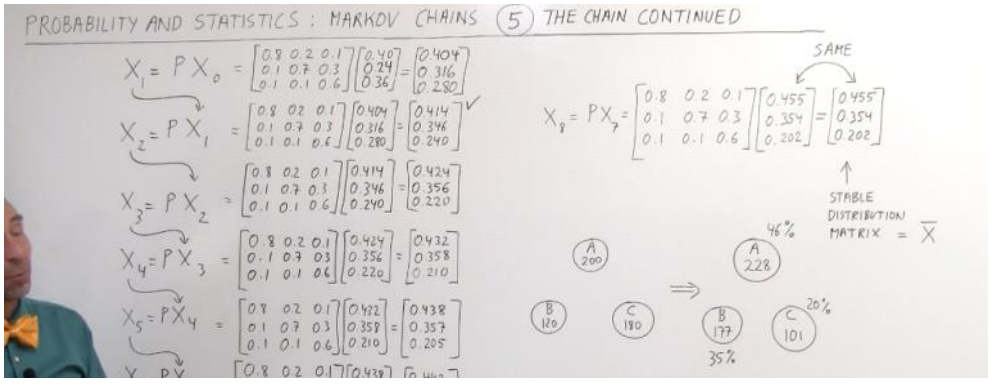




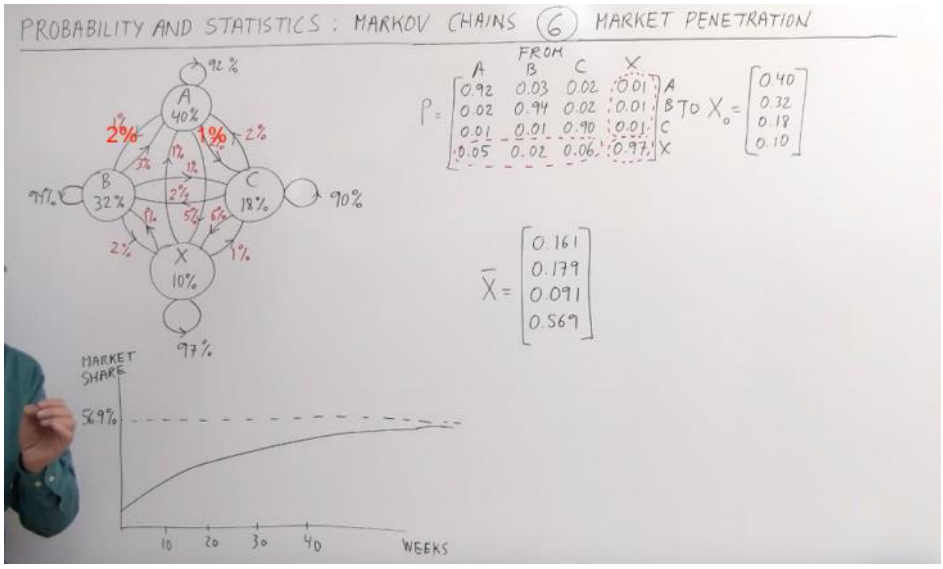
- The main idea behind these markov chain is that if matrix of transition probablities is kept constant , after a large number of iteration the next state -> value matrix will start to converge.



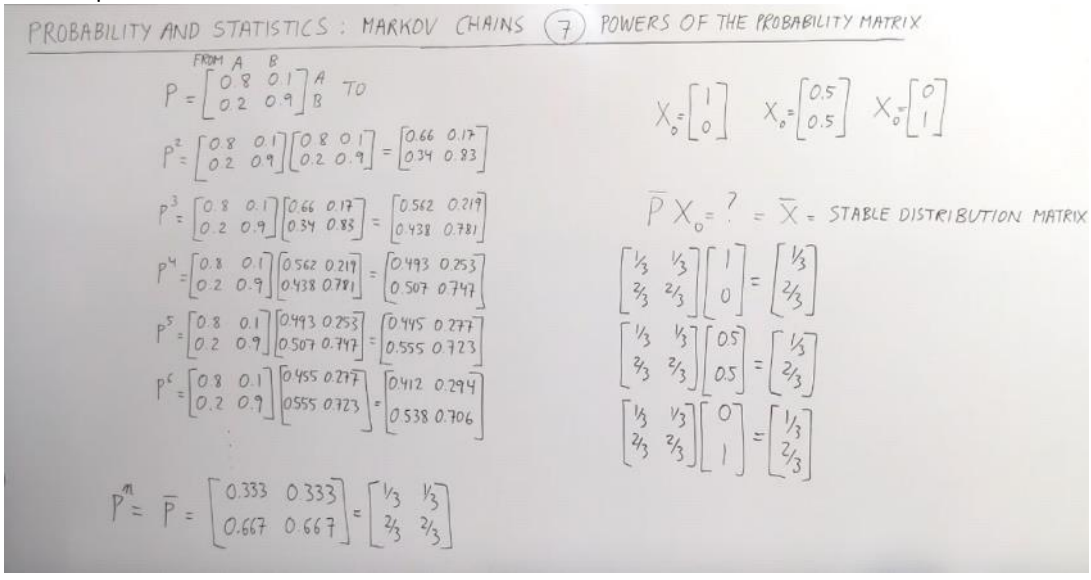


- Once the convergence is reached , we have reached the stable distribution matrix and things stop to change if probablity matrix is left unchanged.
-


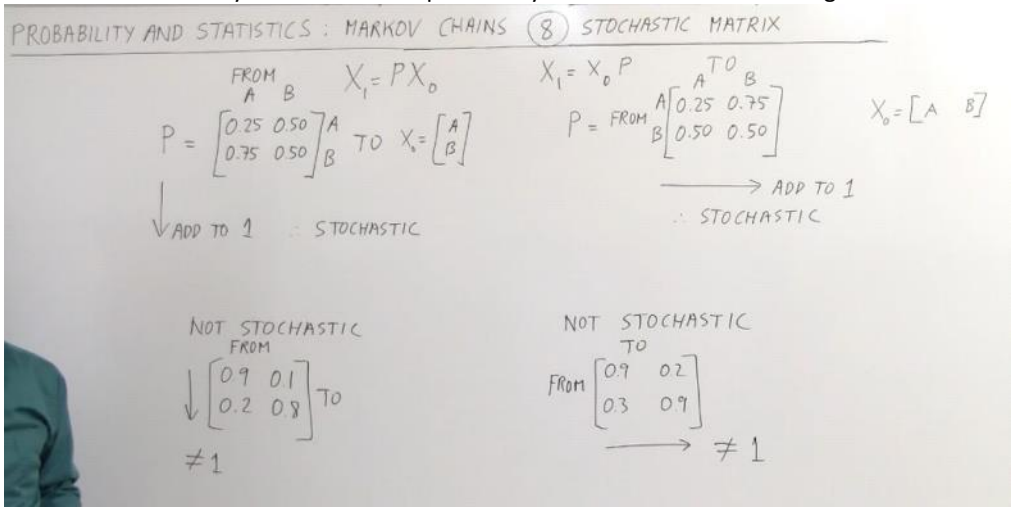
-

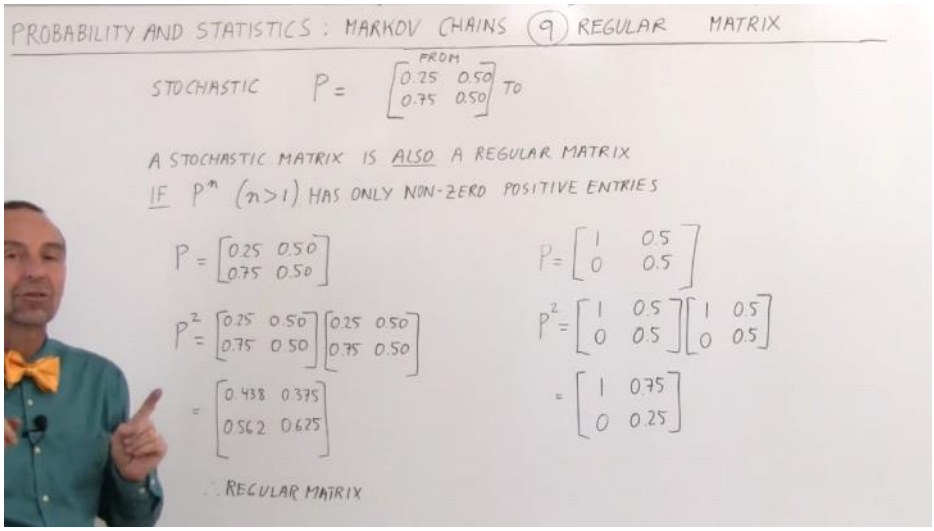


- The final state is kinda independent of initial state in this case , and this showcases the power of markov chains. If we have probablity matrix , we can start from a given initial state and irrespective of it the final state will almost be same !!!



- Markov chain only works when the probability matrix is stochastic and regular in nature.




PROBABILITY AND STATISTICS : MARKOV CHAINS (9) REGULAR MATRIX
STOCHASTIC
FROM
TO
A STOCHASTIC MATRIX IS ALSO A REGULAR MATRIX
IF P^n $(n>1)$ HAS ONLY NON-ZERO POSITIVE ENTRIES
REGULAR MATRIX

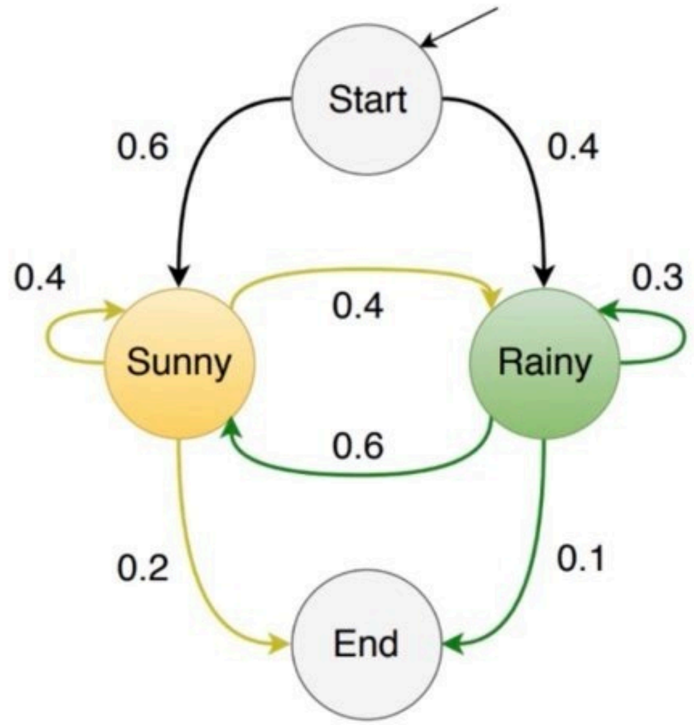
Markov Models - Section Introduction

- Markov models are everywhere!
- Finance: the basis for the famous Black-Scholes formula
- Reinforcement Learning: Markov Decision Process (MDP)
- Hidden Markov Model (speech recognition, computational biology)
- Markov Chain Monte Carlo (MCMC): numerical approximation

Another major application of the Markov model is Markov Chain Monte Carlo, also known as EMC, which

Section Outline

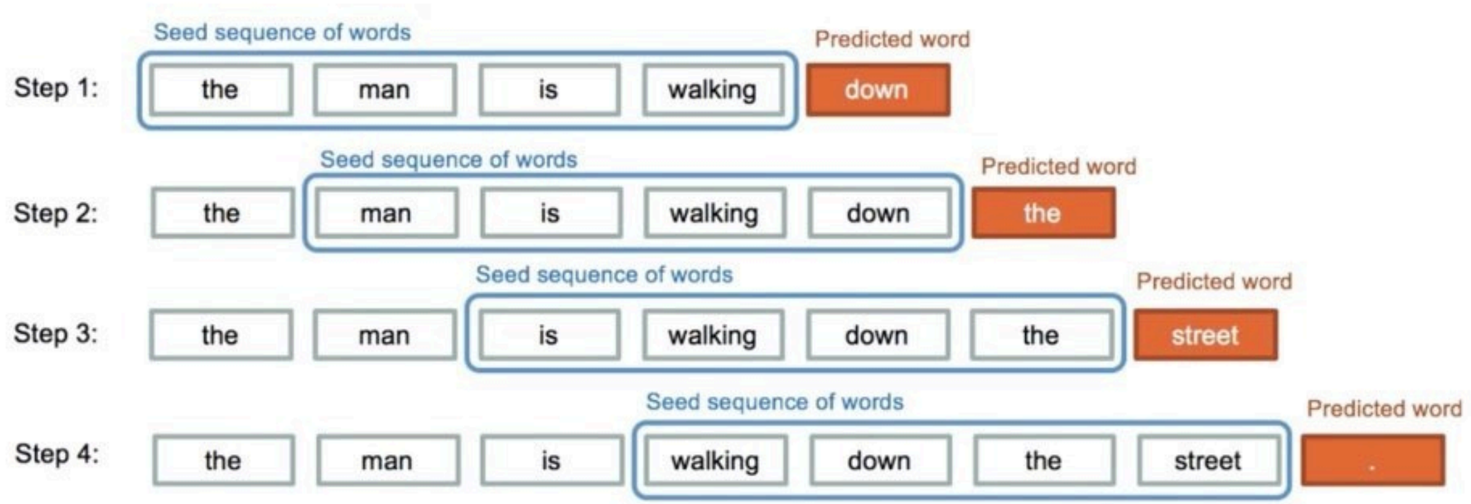
- Markov property (requires knowledge of probability)
- Markov model, state transition matrix
- "Training" a Markov model (i.e. "learning")



We'll also look at how to train a Markov model.

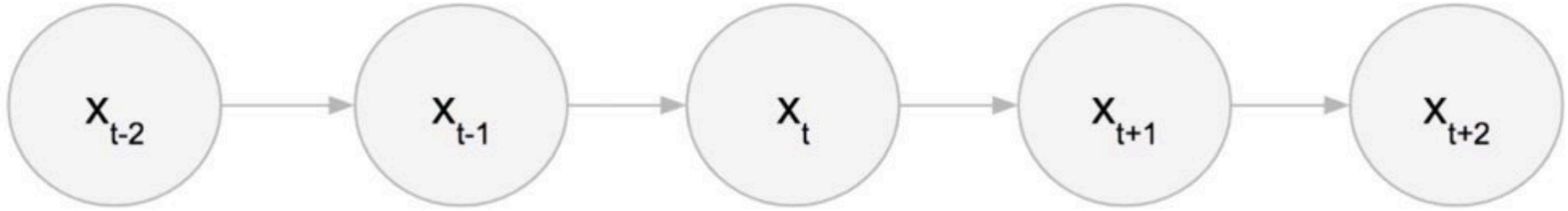
Applications

- Building a text classifier using Bayes rule + Generating poetry
- These are 2 fundamentally different ways to apply ML
- Supervised: predict a target from a dataset of inputs + labels
- Unsupervised: learn structure of data, create new examples having the same structure



The Markov Property

- The Markov property is a very restrictive assumption on the dependency structure of the joint distribution (arrows denote dependency)



- This implies that the joint distribution takes the form:

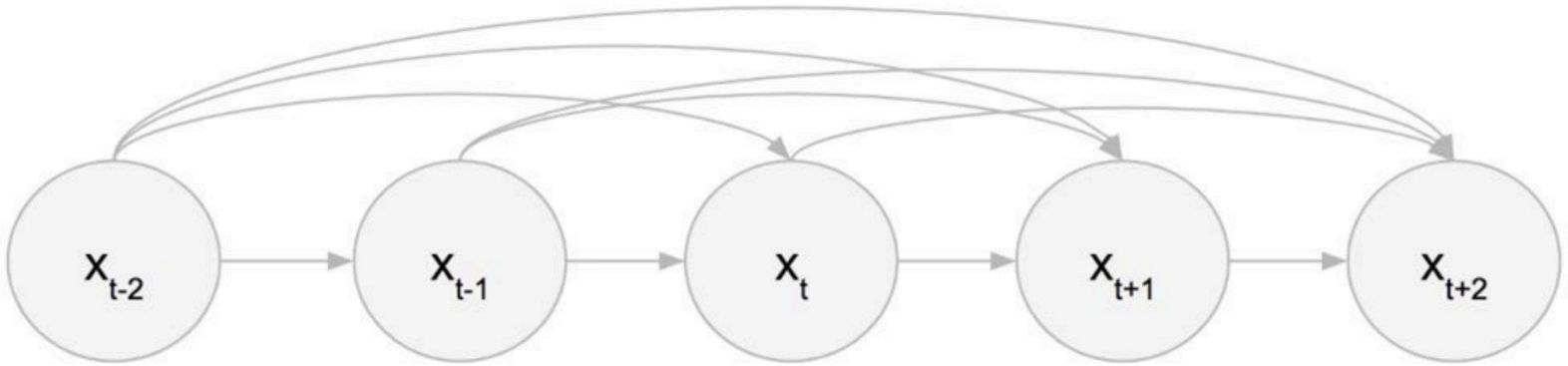
$$p(x_1, \ldots, x_T) = p(x_1) \prod_{t=2}^{T} p(x_t \mid x_{t-1})$$

an x one times p of x three given x two and so on, up to T given x t minus y.

Chain Rule of Probability

- T-variable expansion (general case)
- In general, each term depends on **all** preceding terms

$$p(x_1, ..., x_T) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2)...p(x_T \mid x_{T-1}, ..., x_1)$$



So you see, we've kind of learned about the mark of property backwards.

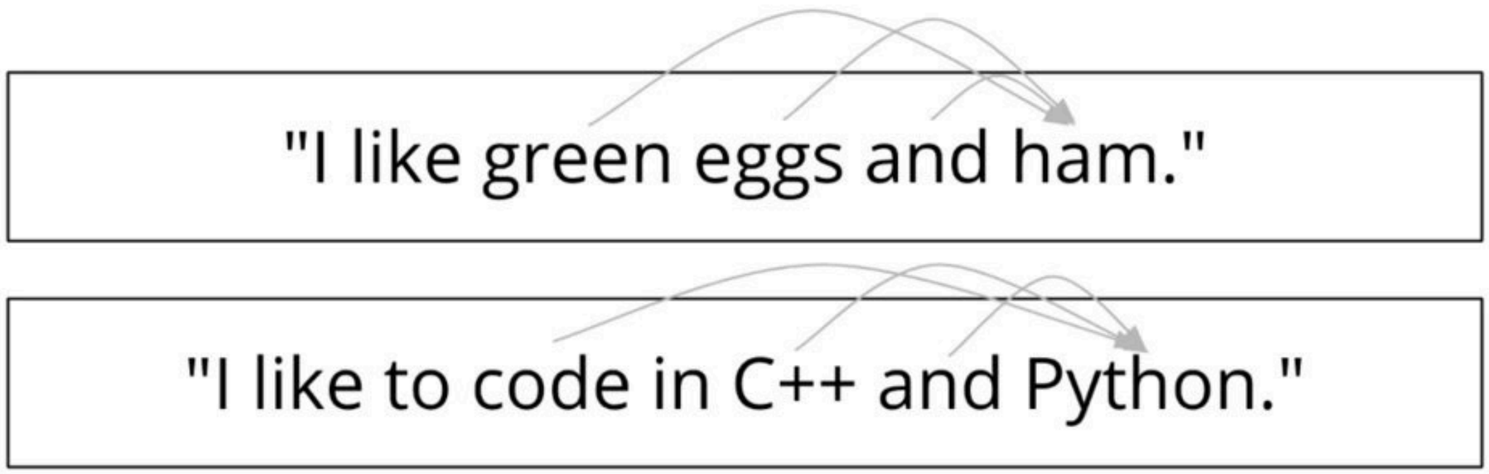
Why is the Markov Property Useful?

- Consider a Markov model of the English language
- Suppose we choose 2000 of the most common words (though some estimate that each individual uses 20k - 30k words)
- Suppose we want to model sentences of length 10 (not unreasonable)
- Our model is $p(x_{10} \mid x_9, x_8, ..., x_1)$
- Each "x" has 2000 possible values (words)
- How many probabilities to estimate? 2000x2000x2000... = 2000^{10}

In order to estimate all these numbers, we must have a sufficient amount of data to learn from as another

The Markov "Assumption"

- We assume the Markov property holds, even when it does not

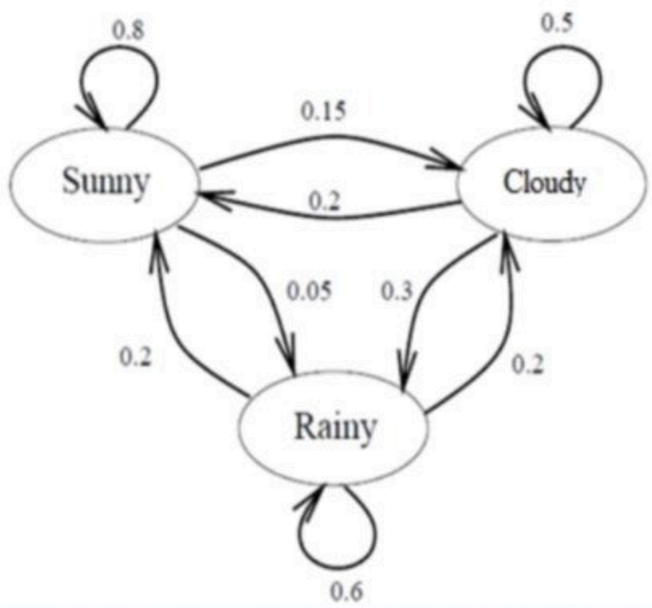


- Do "ham" and "Python" **only** depend on "and"? No!

We can also see that the word python becomes more likely because code in C++ appeared earlier in the

The Markov Model

- In this section, we'll consider sequences of **categorical symbols**, which we'll refer to as "states"
- Example weather: {sunny, rainy, cloudy}
- Example parts-of-speech tags: {noun, verb, adjective, ...}



verb, adjective, and so forth.

Notation

- We'll use *s* for "state" (other sources might use *z* or *x*)

$$s(t) = s_t = state\ at\ time\ t$$

- Time will also be discrete (t = 1, 2, ...) (e.g. no such thing as t = 1.5)
- We will **number** the states from 1, 2, ..., M
- M = total number of possible states (e.g. M = 3 for sunny/rainy/cloudy)
- We use i or j to **index** the state space

$$p(s_t = i)\ means:\ probability\ that\ state\ at\ time\ t\ is\ 'i'$$

So for example, press of T equals I translates to the probability that the state at time t is equal

State Distribution

- We have $p(s_t = 1)$, $p(s_t = 2)$, ..., $p(s_t = M)$
- This is M probability values - together they form a **distribution**
- Ex. "What is the probability that it will be rainy on Sunday?"

  Ans: $p(s_{sunday} = \text{rainy})$

$$p(s_t) = state\ distribution\ (length\ M\ vector)$$

State Transitions

$$p(s_t = j \mid s_{t-1} = i)$$

- "Probability that state at time t is j, given that the state at time t-1 was i"
- How many of these probability values exist? (for all i and j)
- Since both i and j can take any value from 1...M, there are M^2 values

Well, we must have a value for every possible ie from one up to M and every possible j from one up

State Transition Matrix

- Wait a minute... what happened to t?

$$A_{ij} = p(s_t = j \mid s_{t-1} = i), \ \forall i = 1 \ldots M, j = 1 \ldots M$$

- In general, we could have $A_{ij}(t)$
- When A doesn't depend on t: **time-homogeneous Markov process**

Probability of a Sequence

$$p(s_{1...T}) = p(s_1)p(s_2|s_1)p(s_3|s_1, s_2)...$$

$$p(s_{1...T}) = p(s_1)\prod_{t=2}^{T} p(s_t|s_{t-1})$$

Markov Property

$$p(s_{1...T}) = \pi_{s_1}\prod_{t=2}^{T} A_{s_{t-1},s_t}$$

Start thinking about how you'd implement this in Python code

Thus you should consider how you would write some code to carry out this computation.

Training a Markov Model

- We're not ready to rigorously derive this (yet), but the intuition is simple
- Suppose we flip a coin a bunch of times - how do we estimate p(heads)?

This answers the question: "How frequent is the word 'cat' in the English language?"

- That's just the binary case. What if we have M > 2?

$$p(\text{"cat"}) \approx \frac{count(\text{"cat"})}{total\ word\ count}$$

Estimating A and π ("Training")

Note: the 'hat' means 'estimate'

N = number of sequences in dataset

$$\hat{\pi}_i = \frac{count(s_1 = i)}{N}$$

$$\hat{A}_{ij} = \frac{count(i \rightarrow j)}{count(i)}$$

Example:
count("the cat") / count("the")

We would like to know what is the probability of seeing the word cat following the word the.

Markov Model Summary

- What we want to do: model a sequence of states
- State transition matrix (A), initial state distribution (π)
- Task #1: find the probability of a sequence
- Task #2: given a dataset, find A and π
- Task #2 is called "learning" or "training"
- Start considering how to implement these in computer code

Now, again, it's always useful to think about how you would implement this in computer code.

Small Modifications

- In this lecture, we will:

1) Modify how we estimate A and π
2) Compute the probability of a sequence

Previously, we used "maximum likelihood estimation"

the estimates that yield the maximum value of the likelihood given the dataset being trained on.

Probability of a Sequence

$$p(s_{1...T}) = \pi_{s_1} \prod_{t=2}^{T} A_{s_{t-1},s_t}$$

- Only involves multiplication
- What if one of the values is 0? Transition never appeared in train set
- The whole thing becomes 0!

This will make the entire expression zero, although this seems like it might be the right answer.

Add-One Smoothing

- Give a small probability to *every* possible transition
- Add a "fake count" of 1 for each (i, j) transition

$$\hat{A}_{ij} = \frac{count(i \rightarrow j) + 1}{count(i) + M}$$

Adding M to the denominator ensures that each row of A sums to 1

Since we had a one to each of the impossible values, we must also add em to the denominator as well.

Add-One Smoothing

- We can do this for the initial state distribution too

$$\hat{\pi}_i = \frac{count(s_1 = i) + 1}{N + M}$$

Add-Epsilon Smoothing

- We can make it more smooth ($\varepsilon > 1$) or less smooth ($\varepsilon < 1$)

$$\hat{A}_{ij} = \frac{count(i \rightarrow j) + \varepsilon}{count(i) + \varepsilon M}$$

$$\hat{\pi}_i = \frac{count(s_1 = i) + \varepsilon}{N + \varepsilon M}$$

denominator again by the same logic we use before we can conclude that this results in each row of AIG,

Computing the Probability of a Sequence

$$p(s_{1...T}) = \pi_{s_1} \prod_{t=2}^{T} A_{s_{t-1},s_t}$$

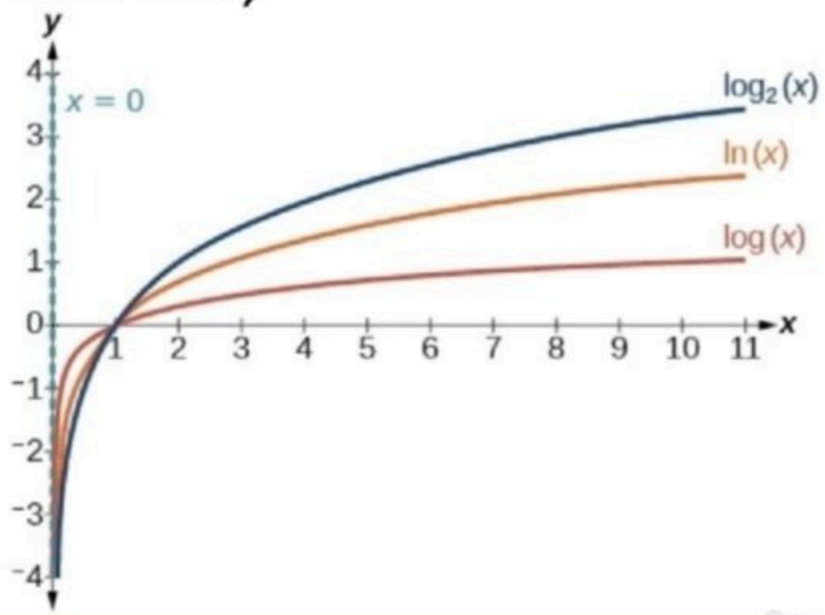
- This involves **multiplying** many small numbers together
- Common to use 20k - 50k vocabulary size in English
- As you multiply small numbers together, they approach 0! (0.1*0.1=0.01)
- Computers don't have infinite precision; eventually it just rounds to 0
- Problematic: what if we want to compare 2 sequences?

Try it yourself in wait until you encounter it to convince yourself that it can and will happen.

Working with Log Probabilities

- Solution: compute log probabilities instead
- We don't need the actual probability **value**, since what we usually want to do is **compare** (e.g. is one sequence more likely than another?)
- It works because the log is a monotonically increasing function
- If A > B, then log(A) > log(B) (i.e. the "winner" is preserved)
- Does it work? $\log_{10}(10^{-10}) = -10$
- Does it work? $\log_{10}(10^{-100}) = -100$

Now you are approaching the limit of what the computer can represent!



Working with Log Probabilities

$$\log p(s_{1...T}) = \log \pi_{s_1} + \sum_{t=2}^{T} \log A_{s_{t-1}, s_t}$$

- Since log(AB) = log(A) + log(B)
- If we don't need the probabilities, we can store only the logged values
- Note: do not compute the product and **then** take the log
- Also note: addition (+) is more efficient (faster) than multiplication (*)

computing a product, as you recall, adding as much faster than doing multiplication.

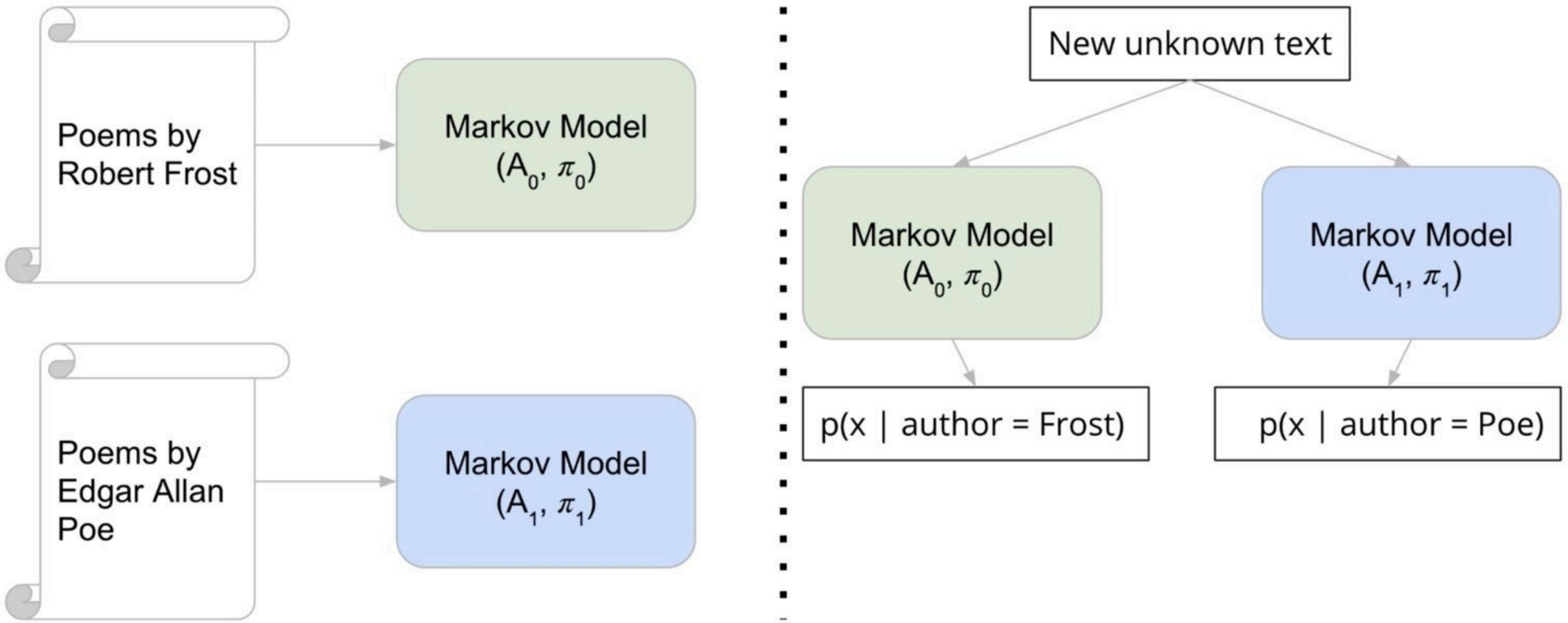
Supervised or unsupervised?

- Text classification is an example of supervised learning, but Markov models are unsupervised (training data is just text; no labels)
- Answer: We must apply Bayes' rule (and build a **Bayes classifier**)

$$p(y \mid x) = \frac{p(x \mid y)p(y)}{p(x)}$$

In fact, what we are really building in this lecture is a Bayes classifier.

Bayes Classifier



Applying Bayes' Rule

$$p(author \mid poem) = \frac{p(poem \mid author)\, p(author)}{p(poem)}$$

So this is the basic form of what we need, but there are several ways we can simplify this expression

Simplifying the decision rule

$$k^* = \arg\max_k \frac{p(poem \mid author = k)\, p(author = k)}{p(poem)}$$

$$k^* = \arg\max_k p(poem \mid author = k)\, p(author = k)$$

$$k^* = \arg\max_k \log p(poem \mid author = k) + \log p(author = k)$$

So what we end up with is that we would like the ARG mass of log of palm giving author plus
log p of

Maximum Likelihood

- We can simply take the argmax of the likelihood if the prior is uniform (i.e. we have no reason to believe, given no other info, that one author is more likely than another)

$$k^* = \arg\max_k \log p(poem \mid author = k)$$

If p(author) is uniform

In this case, our decision comes down to choosing the AMAX of Poem Given author.

Recap

- We train a separate Markov model for each class
- Each model gives us p(x | class = k) for all k
- General form of decision rule using Bayes' rule: $k^* = \text{argmax}_k\ p(\text{class} = k \mid x)$
- Posterior can be simplified since we don't need its actual value
- Maximum a posteriori (MAP): $k^* = \text{argmax}_k\ \text{logp}(x \mid \text{class} = k) + \text{logp}(\text{class} = k)$
- Maximum likelihood: $k^* = \text{argmax}_k\ \text{logp}(x \mid \text{class} = k)$

Text Classifier Exercise Prompt

- You'll be given poems by 2 authors: Edgar Allan Poe and Robert Frost
- URLs will be in the coming notebooks
    - -Copy the URLs from there, don't try to search / type by hand (yes, really)
    - -But don't look at the rest of the notebook, of course!
- Build a classifier that can distinguish between the 2 authors
- Compute train and test accuracy
- Check for class imbalance, compute F1-score if imbalanced

OK, so that's pretty much the whole exercise from a bird's-eye point of view.

Details

- Some details are specific to my implementation
- If you had other ideas, they may not apply to you (use your own judgment)

- Loop through each file, save each line to a list (one line == one sample)
- Save the labels too
- Train-test split
- Create a mapping from unique word to unique integer index
- Loop through data, tokenize each line (string split should suffice)
- Assign each unique word a unique integer index ("mapping" == "dict")
- Create a special index for unknown words (words in the test set but not the train set)

This is because there may be words in the test set that do not appear in the train set.

Details

- Convert each line of text (the samples) into integer lists
- Train a Markov model for each class (Edgar Allan Poe / Robert Frost)
- Don't forget to use smoothing (e.g. add-one smoothing)
- Consider whether you need A and π, or log(A) and log(π)
- For Bayes' rule, compute the priors: p(class = k)
- Now you have everything you need to make a prediction!

At this point, you should have everything you need to make a prediction.

Details

- Write a function to compute the posterior for each class, given an input
- Take the argmax over the posteriors to get the predicted class
- Make predictions for both train and test sets
- Compute accuracy for train/test
- Check for class imbalance
- If imbalanced, check confusion matrix and F1-score

Furthermore, you may want to compute a metric like the F1 score, which takes into account any imbalance