Data visualization

COSC 480B

Reyan Ahmed

rahmed1@colgate.edu

Lecture 21

Hidden Markov models

Overview

- Defining interpretive models
- Using Markov chains to model data
- Inferring hidden state using a hidden Markov model

Overview

Exercise 1

What makes a model interpretable may be slightly subjective. What’s your criteria for an interpretable model?

Overview

Exercise 1

What makes a model interpretable may be slightly subjective. What’s your criteria for an interpretable model?

ANSWER

We like to refer to mathematical proofs as the de facto explanation technique. If one were to convince another about the truth of a mathematical theorem, then a proof that irrefutably traces the steps of reasoning is sufficient.

Example of a not-so-interpretable model

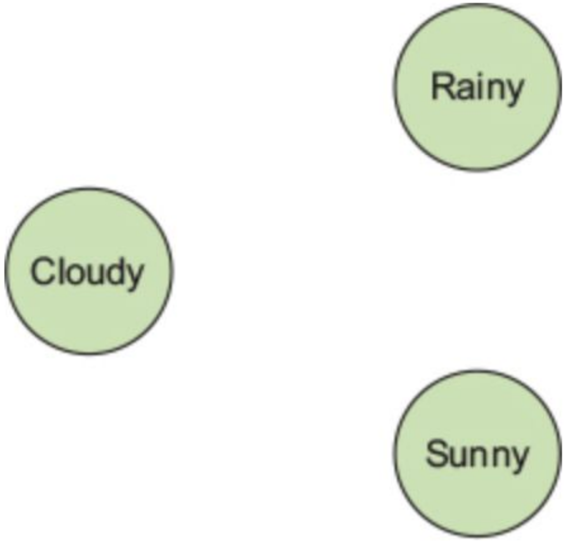
- One classic example of a black-box machine-learning algorithm that’s difficult to interpret is image classification.
- You’ll learn how to solve the problem of classifying images in next lectures
- It’s difficult to ask an image classifier why it made the decision that it did.
- Machine learning sometimes gets the notoriety of being a black-box tool that solves a specific problem without revealing how it arrives at its conclusion.
- The purpose of this chapter is to unveil an area of machine learning with an interpretable model.
- Specifically, you’ll learn about the HMM and use TensorFlow to implement it.

Markov model

- Andrey Markov was a Russian mathematician who studied the ways systems change over time in the presence of randomness.
- For example, maybe a gas particle in Europe has barely any effect on a particle in the United States. So why not ignore it?
- The mathematics is simplified when you look only at a nearby neighborhood instead of the entire system.
- This notion is now referred to as the Markov property.

Markov model

Weather conditions (states) represented as nodes in a graph



Markov model

Exercise 2

A robot that decides which action to perform based on only its current state is said to follow the Markov property. What are the advantages and disadvantages of such a decision-making process?

Markov model

Exercise 2

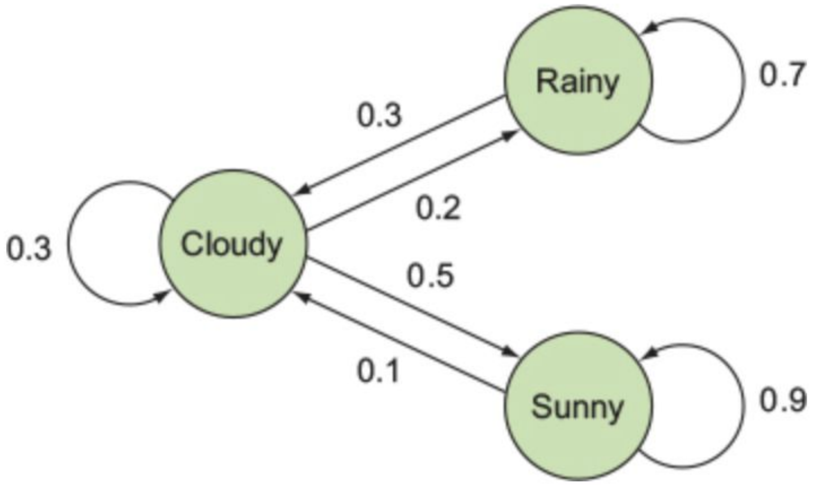
A robot that decides which action to perform based on only its current state is said to follow the Markov property. What are the advantages and disadvantages of such a decision-making process?

ANSWER

The Markov property is computationally easy to work with. But these models aren’t able to generalize to situations that require accumulating a history of knowledge. Examples of these are models in which a trend over time is important, or in which knowledge of more than one past state gives a better idea of what to expect next.

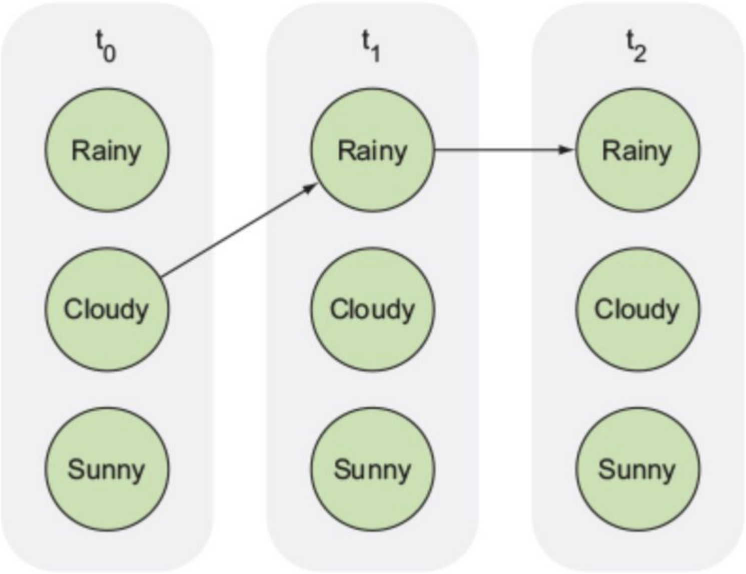
Markov model

Transition probabilities between weather conditions are represented as directed edges.



Markov model

A trellis representation of the Markov system changing states over time



Markov model

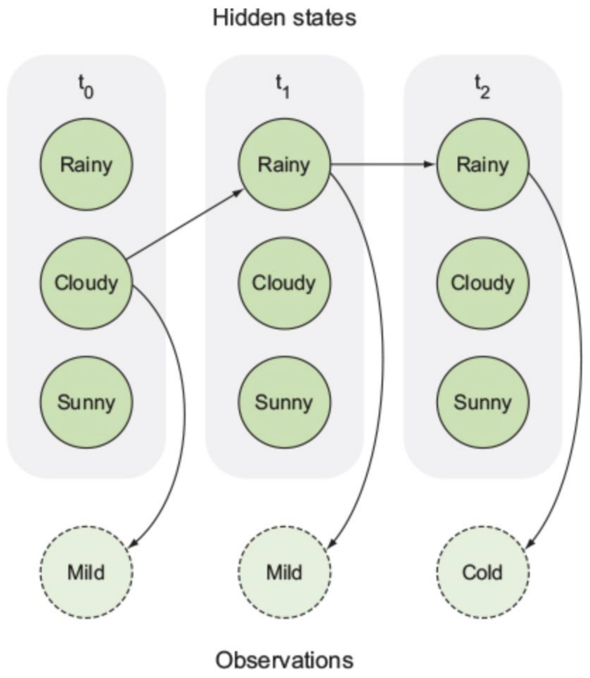
A transition matrix conveys the probabilities of a state from the left (rows) transitioning to a state at the top (columns).

	Cloudy	Rainy	Sunny
Cloudy	0.3	0.2	0.5
Rainy	0.3	0.7	0.0
Sunny	0.1	0.0	0.9

3 × 3 transition matrix

Hidden Markov model

A hidden Markov model trellis showing how weather conditions might produce temperature readings



Forward algorithm

Defining the HMM class

```
import numpy as np                                    1
import tensorflow as tf                              1

class HMM(object):
    def __init__(self, initial_prob, trans_prob, obs_prob):
        self.N = np.size(initial_prob)                   2
        self.initial_prob = initial_prob                2
        self.trans_prob = trans_prob                      2
        self.emission = tf.constant(obs_prob)             2

        assert self.initial_prob.shape == (self.N, 1)       3
        assert self.trans_prob.shape == (self.N, self.N)
3
        assert obs_prob.shape[0] == self.N                3

        self.obs_idx = tf.placeholder(tf.int32)           4
        self.fwd = tf.placeholder(tf.float64)           4
```

1 Imports the required libraries
2 Stores the parameters as method variables
3 Double-checks that the shapes of all the matrices make sense
4 Defines the placeholders used for the forward algorithm

Forward algorithm

Creating a helper function to access emission probability of an observation

```
def get_emission(self, obs_idx):
        slice_location = [0, obs_idx]                                    1
        num_rows = tf.shape(self.emission)[0]
        slice_shape = [num_rows, 1]                                      2
        return tf.slice(self.emission, slice_location, slice_shape)  3
```

1 The location of where to slice the emission matrix
2 The shape of the slice
3 Performs the slicing operation

Forward algorithm

Initializing the cache

```
def forward_init_op(self):
    obs_prob = self.get_emission(self.obs_idx)
    fwd = tf.multiply(self.initial_prob, obs_prob)
    return fwd
```

Forward algorithm

Updating the cache

```
def forward_op(self):
    transitions = tf.matmul(self.fwd,
 tf.transpose(self.get_emission(self.obs_idx)))
    weighted_transitions = transitions * self.trans_prob
    fwd = tf.reduce_sum(weighted_transitions, 0)
    return tf.reshape(fwd, tf.shape(self.fwd))
```

Forward algorithm

Defining the forward algorithm given an HMM

```
def forward_algorithm(sess, hmm, observations):
    fwd = sess.run(hmm.forward_init_op(),
feed_dict={hmm.obs_idx:
     observations[0]})
    for t in range(1, len(observations)):
        fwd = sess.run(hmm.forward_op(),
feed_dict={hmm.obs_idx:
     observations[t], hmm.fwd: fwd})
    prob = sess.run(tf.reduce_sum(fwd))
    return prob
```

Forward algorithm

Screenshot of HMM example scenario from Wikipedia

```
states = ('Rainy', 'Sunny')

observations = ('walk', 'shop', 'clean')

start_probability = {'Rainy': 0.6, 'Sunny': 0.4}

transition_probability = {
   'Rainy  : {'Rainy': 0.7, 'Sunny': 0.3},
   'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6},
}

emission_probability = {
   'Rainy  : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},
   'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1},
}
```

Forward algorithm

Defining the HMM and calling the forward algorithm

```
if __name__ == '__main__':
    initial_prob = np.array([[0.6],
                             [0.4]])

    trans_prob = np.array([[0.7, 0.3],
                           [0.4, 0.6]])

    obs_prob = np.array([[0.1, 0.4, 0.5],
                         [0.6, 0.3, 0.1]])

    hmm = HMM(initial_prob=initial_prob, trans_prob=trans_prob,
     obs_prob=obs_prob)

    observations = [0, 1, 1, 2, 1]
    with tf.Session() as sess:
        prob = forward_algorithm(sess, hmm, observations)
        print('Probability of observing {} is {}'.format(observations, prob))
```

Viterbi decoding

- Many permutations may cause a particular observation
- So enumerating all possibilities the naive way will take an exponentially long time to compute
- Instead, you can solve the problem by using dynamic programming
- The Viterbi decoding algorithm finds the most likely sequence of hidden states, given a sequence of observations.

Viterbi decoding

Adding the Viterbi cache as a member variable

```
def __init__(self, initial_prob, trans_prob,
obs_prob):
  ...
  ...
  ...
  self.viterbi = tf.placeholder(tf.float64)
```

Viterbi decoding

Defining an op to update the forward cache

```
def decode_op(self):
        transitions = tf.matmul(self.viterbi,
      tf.transpose(self.get_emission(self.obs_idx)))
        weighted_transitions = transitions * self.trans_prob
        viterbi = tf.reduce_max(weighted_transitions, 0)
        return tf.reshape(viterbi, tf.shape(self.viterbi))
```

Viterbi decoding

Defining an op to update the back pointers

```
def backpt_op(self):
    back_transitions = tf.matmul(self.viterbi, np.ones((1, self.N)))
    weighted_back_transitions = back_transitions * self.trans_prob
    return tf.argmax(weighted_back_transitions, 0)
```

Viterbi decoding

Defining the Viterbi decoding algorithm

```
def viterbi_decode(sess, hmm, observations):
    viterbi = sess.run(hmm.forward_init_op(), feed_dict={hmm.obs:
     observations[0]})
    backpts = np.ones((hmm.N, len(observations)), 'int32') * -1
    for t in range(1, len(observations)):
        viterbi, backpt = sess.run([hmm.decode_op(),
hmm.backpt_op()], feed_dict={hmm.obs: observations[t],
hmm.viterbi: viterbi})
        backpts[:, t] = backpt
    tokens = [viterbi[:, -1].argmax()]
    for i in range(len(observations) - 1, 0, -1):
        tokens.append(backpts[tokens[-1], i])
    return tokens[::-1]
```

Viterbi decoding

Running the Viterbi decode

```
seq = viterbi_decode(sess, hmm, observations)
print('Most likely hidden states are {}'.format(seq))
```

Uses of hidden Markov models

- Modeling a video
- Modeling DNA
- Modeling an image
- Sequencing words in a sentence
- Recognizing part of speech