CSCI547 Machine Learning
Homework 4

Zachary Falkner
Department of Computer Science
University of Montana

May 4, 2018

1 Bayesian Networks

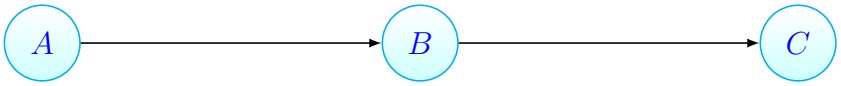
1A

$P(a,b) \neq P(a)P(b)$
$0.048 \neq 0.192 * 0.048$
$0.048 \neq 0.009216$

$P(a,b|c) = P(a,c)P(b,c) for c \in 0,1$
$\frac{P(a,b,c)}{P(c)} = P(a,c)P(b,c)$
$\frac{0.096}{0.144} = 0.064 * 0.216$
$0.013824 = 0.013824$
qed

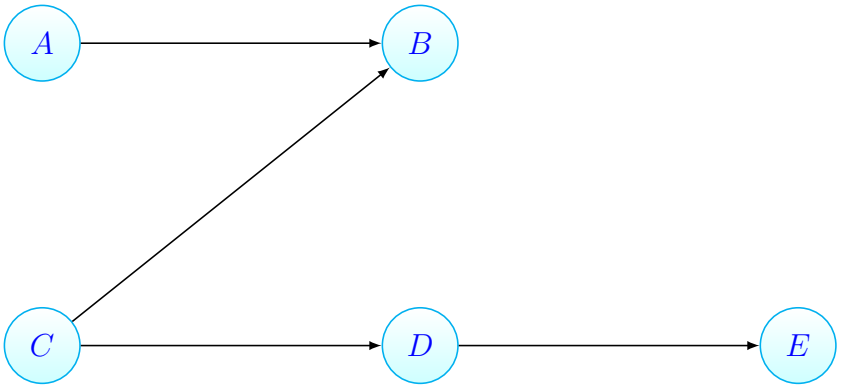
1B

$P(a,b,c) = P(a) * P(c|a) * P(b|c)$
$P(a,b,c) = P(a) * \frac{P(a,c)}{P(a)} * \frac{P(b,c)}{P(c)}$
$0.096 = 0.192 * \frac{0.064}{0.192} * \frac{0.216}{0.144}$
$0.096 = 0.096$
qed

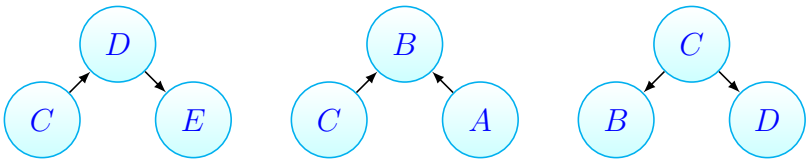


1C*

$P(A, B, C, D, E) = P(A)P(C)P(B|A, C)P(D|C)P(E|D)$



$P(A = 1|E = 1, C = 1)$
Consider the following relationships:



...

2 Markov Models: Gene sequence clustering

2A

```
import pickle
import numpy as np

from markov_models import FirstOrderMarkovModel
```

```
DATASET_TRAINING = "genes_training.p"

def new_sequence(class_id, models):
        return models[class_id].generate_phrase()


if __name__ == "__main__":
        training = pickle.load(open(DATASET_TRAINING, "rb"))

        training_data = np.array(training[0])
        training_lables = np.array(training[1])

        training_0 = training_data[training_lables[:] == 0]
        training_1 = training_data[training_lables[:] == 1]


        sequences_0 = training_0[0]
        seq_0 = ''.join(str(seq) for seq in sequences_0)

        sequences_1 = training_1[0]
        seq_1 = ''.join(str(seq) for seq in sequences_1)

        sequence_mm_model_0 = FirstOrderMarkovModel(seq_0)
        sequence_mm_model_0.build_transition_matrices()

        sequence_mm_model_1 = FirstOrderMarkovModel(seq_1)
        sequence_mm_model_1.build_transition_matrices()

        models = [sequence_mm_model_0, sequence_mm_model_1]
        for i in range(0,2):
                print(new_sequence(i, models))
```

Modification to markov_models.py

```
def generate_phrase(self, length=20):
        phrase = ''
        for i in range(0,length):
                w_minus_1 = np.random.choice(list(self.transitions[0].keys()
                    ),replace=True,p=list(self.transitions[0].values()))
                phrase += w_minus_1
        return phrase
```

```
>python 2a.py
TTTCCATTGTCGGATAAATT
AACCGGTGAGACATGCAGCA
```

2B

```
import pickle
import numpy as np

from markov_models import FirstOrderMarkovModel
```

```
DATASET_TRAINING = "genes_training.p"
DATASET_TEST = "genes_test.p"

if __name__ == "__main__":
        training = pickle.load(open(DATASET_TRAINING, "rb"))
        test = pickle.load(open(DATASET_TEST, "rb"))

        training_data = np.array(training[0])
        training_lables = np.array(training[1])
        test_data = np.array(test[0])
        test_labels = np.array(test[1])

        sequences_0 = training_data[training_lables == 0]
        sequences_1 = training_data[training_lables == 1]

        seq_0 = ''.join(str(seq) for seq in sequences_0)
        seq_1 = ''.join(str(seq) for seq in sequences_1)

        sequence_mm_model_0 = FirstOrderMarkovModel(seq_0)
        sequence_mm_model_0.build_transition_matrices()

        sequence_mm_model_1 = FirstOrderMarkovModel(seq_1)
        sequence_mm_model_1.build_transition_matrices()

        predictions = []

        for sequence in test_data:
                sequence = ' '.join(sequence)
                scores = []
                scores.append(sequence_mm_model_0.compute_log_likelihood(
                    sequence))
                scores.append(sequence_mm_model_1.compute_log_likelihood(
                    sequence))
                predictions.append(np.argmax(scores))

        total = test_labels.size
        correct = np.sum(predictions == test_labels)
        accuracy = correct/total
        print("Accuracy: {}".format(accuracy))
```

```
>python 2b.py
Accuracy: 0.985
```

2C*

```
import pickle
import numpy as np

from markov_models import NaiveBayesModel

DATASET_TRAINING = "genes_training.p"
DATASET_TEST = "genes_test.p"
```

```
if __name__ == "__main__":
        training = pickle.load(open(DATASET_TRAINING, "rb"))
        test = pickle.load(open(DATASET_TEST, "rb"))

        training_data = np.array(training[0])
        training_lables = np.array(training[1])
        test_data = np.array(test[0])
        test_labels = np.array(test[1])

        sequences_0 = training_data[training_lables == 0]
        sequences_1 = training_data[training_lables == 1]

        seq_0 = ''.join(str(seq) for seq in sequences_0)
        seq_1 = ''.join(str(seq) for seq in sequences_1)

        sequence_nb_model_0 = NaiveBayesModel(seq_0)
        sequence_nb_model_0.build_transition_matrices()

        sequence_nb_model_1 = NaiveBayesModel(seq_1)
        sequence_nb_model_1.build_transition_matrices()

        predictions = []

        for sequence in test_data:
                sequence = ' '.join(sequence)
                scores = []
                scores.append(sequence_nb_model_0.compute_log_likelihood(
                    sequence))
                scores.append(sequence_nb_model_1.compute_log_likelihood(
                    sequence))
                predictions.append(np.argmax(scores))

        total = test_labels.size
        correct = np.sum(predictions == test_labels)
        accuracy = correct/total
        print("Accuracy: {}".format(accuracy))
```

```
>python 2c.py
Accuracy: 0.899
```

Simply put, naive Bayes' is naive It takes no consideration as to the ordering of the nucleobases, only their frequency. While yes this produces a random string made up of ATGC, it does not necessarily mean that it will look anything like a real snippet of genetic code. The Markov model takes into account the likelihood of characters following each other which is why it produces a significantly better result when used as the model.