Université Jean Monnet

Course: Machine Learning

Practical Session 2

Hidden Markov Models

Jorge Chong, Oussama Bouldjedri

March 26, 2017

1 Exercise 2.1: Analysis of the previous code

We analyse the code provided, and use the exercise in class about 3 hidden States: Cloudy, Rainy, Sunny and 2 observable states: Museum, Beach.

1. ```
   states = ["Cloudy", "Rainy", "Sunny"]
   n_states = len(states)
   observations = ["Museum", "Beach"]
   n_observations = len(observations)
   ```

   The hmm model expect the number of hidden states. The states are represented by numbers from 0 to n_states - 1.

2. ```
   model = hmm.MultinomialHMM(n_components=n_states,
   init_params="",
   n_iter=10,
   algorithm='viterbi',
   tol=0.00001)
   model.startprob_ = np.array([1, 0, 0])
   model.transmat_ = np.array([[0.2, 0.6, 0.2],[0.3, 0.2, 0.5],[0.1,
   0.1, 0.8]])
   model.emissionprob_ = np.array([[0.7, 0.3],[1, 0],[0.1, 0.9]])
   ```

   This configures the model using the number of states specified and the matrix A, B and initial probabilities

3. ```
   gen1 = model.sample(3)
   print(gen1)
   ```
   Using the model, generates random samples (3)

4. ```
   seqgen2, stat2 = model.sample(5)
   print(seqgen2)
   ```

```
print(stat2)
```

Generates 5 random samples using the model. and return a tuple of the sequences generated and their respective hidden states

5.
```
gen3 = model.sample(2)
print(gen3)
```

Generates 2 samples using the model

6.
```
sequence1 = np.array([[0, 1]]).T
logproba = model.score(sequence1)
print(logproba)
```

Given a sequence of observations computes the log probability of observing this sequence. In the example provided in the classroom was Museum, Beach. We got the log probability of -1.783, which gives us the probability of 0.0165, which is close to the problem solve in the classroom

7.
```
logproba_extend =
model.score_samples(sequence1)
print(logproba_extend)
```

Computes the log probability of a sequence and their respective posteriors

8.
```
p = model.predict(sequence1)
print(p)
p_extend = model.score_samples(sequence1)
print(p_extend)
```

Find the most probable states that corresponds to the observed sequence

9.
```
sequence3 = np.array([[0, 1, 0, 1]]).T sequence4 = np.array([[1,
1, 0, 1, 1]]).T
sample = np.concatenate([sequence3, sequence4])
lengths = [len(sequence3), len(sequence4)]
model.fit(sample, lengths)
```
Try with many sequences and fits the model according to the observed frequencies

10.
```
sequence = np.array([[1, 1, 0, 1]]).T
logprob, state_seq = model.decode(sequence,
algorithm="viterbi")
print("Observations", ", ".join(map(lambda x:
```

```
observations[int(x)], sequence)))
print("Associated states:", ", ".join(map(lambda x:
states[x], state_seq)))
```

Prints neatly the names of the states and uses the Viterbi algorithm to find the most likely hidden states that explain the observed symbols

2 Exercise 2.2: HMM Learning

1. The log probability of getting abbaa is -3.58406908976, probability = 0.000026
2. Applying the Baum Welch algorithm for just 1 iteration we get a log probability of -2.93888405787. Probability = 0.001151
3. After 15 iterations we reduce the log probability to -1.39440241596. Probability = 0.04
4. Training at convergence after 200 iterations we get a log probability of -1.3863770878, probability = 0.041
5. Changing the HMM of 5 states, and training at convergence, we got a log probability of 0, meaning that the model "explains" with absolutely certainty the sequence of observations, which in this case is only 1 sequence

3 Exercise 2.3: HMM Learning exercise 2

1. For this exercise we create two models with the same initial parameters, but trained with different sequences L1 and L2
2. The probabilities of L1 and L2 for model A are almost 0 and 0.00023 and for model B are 0.000015 and almost 0. This can be explained if we observe that the training data for L1 had more patterns with a's and the training data for L2 more patterns with b's, therefore the model trained with L1 is going to perform poorly on L2 and the same for the other model

4 Exercise 2.4: Handwritten digit recognition

For this problem we tried to fit models for different values of hidden states between 15 and 20, 25 and 50 and finally for 200 states. After many trials we selected parameters between 15 and 19 states, due to the fact that training for more than 25 hidden states takes more than 20 min per model. Empirically we found something acceptable between 15 and 19. We tried to select the best classifier running cross validation, finally we evaluated the best

model for each class, comparing over testing set, against the models for the other classes. Our final project has the next files that has to be run in this order:

1. `Digitreader.py`
   Reads the original files and prepares the data for the algorithms. Creates a random split for training, validating and testing in each class. The final product is stored in a file.

2. `HMM.py`
   Reads the file of the previous step and execute the training for each class. We store the results in a file for the next phase.

3. `HMM_validator.py`
   Using the split for validation, for each class, and for each hyperparameter (number of hidden states), we compute the score on validation data, selecting the model that gets the higher log probability.

4. `HMM_test.py`
   Executes the model performance assessment on testing data. To do that, for each target class, and for each sequence, get the score over testing data, and compare with the best score of the other classes over that testing data sequence

5. `HMM_Results.py`
   Shows the final results

6. `HMM_Predict.py`
   Given a sequence and the model, predicts the class that gets the highest log probability

```
For Class 0: Best Model with 19 hidden states: Performance = 0.9131736526946108
For Class 1: Best Model with 19 hidden states: Performance = 0.88268156424581
For Class 2: Best Model with 18 hidden states: Performance = 0.8903225806451613
For Class 3: Best Model with 19 hidden states: Performance = 0.9376854599406528
For Class 4: Best Model with 17 hidden states: Performance = 0.9348534201954397
For Class 5: Best Model with 18 hidden states: Performance = 0.94140625
For Class 6: Best Model with 19 hidden states: Performance = 0.9437086092715232
For Class 7: Best Model with 19 hidden states: Performance = 0.8413173652694611
For Class 8: Best Model with 18 hidden states: Performance = 0.9456869009584664
For Class 9: Best Model with 19 hidden states: Performance = 0.9088050314465409
```

Figure 1: Final HMM for each Class

5 Exercise 2.5: Personal Problem

Computer virus are complex pieces of code that are difficult to detect precisely. Network administrators use mechanisms like firewalls and code inspection to detect and prevent early the spreading of malicious code inside their networks. The problem with this mechanisms is that sometimes, hard rules result in a high rate of false positives, breaking the functionality of internal and internet applications. We would like to estimate the probability of malicious bytes inside small size traffic packets, only by inspecting a few bytes. Using markov hidden models, we model the problem as inspecting a few bytes of a stream of data. These are the symbols in our model. By historical data, and virus signature databases we know somehow the classification of observables stream of bytes as suspicious or not, that can be helpful for testing and as a rough estimate of initial probabilities. The hidden states, as in the case of hand written digit recognition, don't have any apparent meaning, but we assume that there is some hidden state that explains the emission of bytes.