Middle East Technical University Department of Computer Engineering

CENG 499

Introduction to Machine Learning

Spring 2020-2021

Homework 4 - Naive Bayes, Hidden Markov Models

version 1

Due date: July 9, 2021, Friday, 23:59

1 Introduction

In this assignment, you have 2 independent tasks: sentiment analysis using Naive Bayes and evaluation and decoding tasks of Hidden Markov Models. No report is required in this homework. All related files can be downloaded from `http://user.ceng.metu.edu.tr/~artun/ceng499/hw4_files.zip`.

2 Sentiment Analysis using Naive Bayes

2.1 Dataset

The dataset is created using Twitter US Airline Sentiment dataset. You need to use the version we provided. The examples are some tweets that mention some airline firms in US. They are labeled as negative, neutral, and positive and given as plain text files in hw4_data/sentiment.

2.2 Background

2.2.1 Naive Bayes

Let h be our hypothesis which is defined as

$$h(x) = \underset{y}{\operatorname{argmax}}\, P(y|x),$$

where y is the label and x is the feature. It basically selects the label that has the highest probability given x. Using the Bayes' Rule, we can obtain

$$\begin{aligned} h(x) &= \underset{y}{\operatorname{argmax}}\, P(y|x) \\ &= \underset{y}{\operatorname{argmax}}\, \frac{P(x|y)P(y)}{P(x)} && \text{Bayes' Rule} \\ &= \underset{y}{\operatorname{argmax}}\, P(x|y)P(y) && P(x) \text{ does not depend on y;} \end{aligned}$$

however, estimating $P(x|y)$ is still hard since we need to see the exact example in out dataset. In our case, it is highly unlikely that we see the exact same test tweet in our training dataset. To achieve more feasible solution, Naive Bayes assumes conditional independence between the dimensions of the features given label. Although this independence does not hold in our case in reality, it still works well in practice. Assuming the independence, we achieve

$$h(x) = \underset{y}{\operatorname{argmax}} P(y) \prod_{j=1}^{d} P(x_j|y)$$

where x_j is the j^{th} dimension of the feature x and d is the number of dimensions of x. In general, it is not a good idea to multiply many probabilities because our precision may not be enough to represent it and the operation may result with a 0. Instead, we can take its logarithm (which does not change the results since it is a monotonically increasing function) and get

$$\begin{aligned} h(x) &= \underset{y}{\operatorname{argmax}}\, log(P(y) \prod_{j=1}^{d} P(x_j|y)) \\ &= \underset{y}{\operatorname{argmax}}\, log(P(y)) + \sum_{j=1}^{d} log(P(x_j|y)). \end{aligned}$$

2.2.2 Our case

We are going to assume that the positions of the words do not matter and simply count the occurrences of the words in an example. This kind of modelling is called bag-of-words. To do that, first, we are going to create our vocabulary which will contain every word in the training set. (We could also use every word in the English dictionary but in this homework, we are going to use the training data we already have.) Secondly, for a specific example (a tweet), we will count the occurrences of every word in the vocabulary.

To give an example let's say our tweet is "the ball is on the floor". Then, our representation will be something like in Table 1 assuming that our vocabulary is {a, amazing, ball, floor, is, on, the, zoom}.

a	0
amazing	0
ball	1
floor	1
is	1
on	1
the	2
zoom	0

Table 1: The representation of "the ball is on the floor".

We have two probabilities to estimate: $P(x|y)$ and $P(y)$. $P(y)$ is easy to estimate. For class c we will count the examples labeled as class c and divide it by the number of all examples. Let $\pi_c = P(y = c)$ be the probability of a tweet being in class c. Then, it can be estimated as

$$\hat{\pi}_c = \frac{\sum_{i=1}^{n} I(y^{(i)} = c)}{n}$$

where I is the indicator function which returns 1 when the condition holds and 0 otherwise, $y^{(i)}$ is the label of i^{th} example in the dataset, and n is the number of examples in the training dataset.

In our model, we will think like while writing a tweet, every word is selected by rolling a die with d sides where d is the number of words in our vocabulary. Let θ_{jc} be the probability of rolling the word j in our vocabulary given that our class label is c. Then, the probability of generating a tweet with length m is Multinomial Distribution (`https://en.wikipedia.org/wiki/Multinomial_distribution`) and can be expressed as

$$P(x|m, y = c) = \frac{m!}{x_1!x_2!...x_d!} \prod_{j=1}^{d} \theta_{jc}^{x_j}.$$

Since we are taking argmax, we don't actually need to calculate the term $\frac{m!}{x_1!x_2!...x_d!}$ since it will be the same for every class. Therefore, finally our hypothesis function will be

$$h(x) = \underset{c}{\operatorname{argmax}}\, log(\hat{\pi}_c) + \sum_{j=1}^{d} x_j log(\hat{\theta}_{jc})$$

where $\hat{\theta}_{jc}$ is the estimation of θ_{jc}. We will see how to estimate that in the next section.

2.2.3 Additive (Laplace) Smoothing

Calculation of $\hat{\theta}_{jc}$'s will be very similar to the calculation of $\hat{\pi}_c$'s. We will simply divide the number of occurrences of the j^{th} word in all examples labeled with class c by the number of total words in the examples labeled with c. More formally, it can be described as

$$\hat{\theta}_{jc} = \frac{\sum_{i=1}^{n} I(y^{(i)} = c)x_j^{(i)}}{\sum_{i=1}^{n} \left[I(y^{(i)} = c) \sum_{j'=1}^{d} x_{j'}^{(i)} \right]}$$

where $y^{(i)}$ is the label of the i^{th} example, $x_j^{(i)}$ is the number of occurrences of the j^{th} word in our vocabulary in the i^{th} example. The term $\sum_{j'=1}^{d} x_{j'}^{(i)}$ simply calculates the number of words in the i^{th} example.

Let's say the word "cake" appears in some negative tweets but never appears in a positive tweet in our training set. Then, if we estimate θ's like we did in above, the estimated probability of the word "cake" appearing in a positive tweet would be 0. Since we multiply the probabilities, the overall probability will be 0. (Since we are taking the logarithm of them, we can get a domain error.) However, we may get a test example where the word "cake" appears in it while being a positive tweet. So we want to somehow smooth the probabilities so that even it does not occur in our training set, the probability will be some small value, but not 0.

In additive smoothing with smoothing parameter 1 (`https://en.wikipedia.org/wiki/Additive_smoothing`), we are going to behave like every word appears at least once in the training dataset. If we update our estimation formula above, the number of every word will increase by one so the denominator will increase by the number of words in our vocabulary (d) and the numerator will increase by 1 and we will get

$$\hat{\theta}_{jc} = \frac{1 + \sum_{i=1}^{n} I(y^{(i)} = c)x_j^{(i)}}{d + \sum_{i=1}^{n} \left[I(y^{(i)} = c) \sum_{j'=1}^{d} x_{j'}^{(i)} \right]}.$$

2.3 Implementation

You could do your implementations by directly coding the formulas above, it would be a very inefficient since it contains many unnecessary calculations. This homework does **not** require you to implement very efficient code; however, your code still needs to finish in a reasonable amount of time. (for example in 10 minutes). If you don't remove the redundant things, the calculations may far exceed this time limitation.

Notice that although the equations seem very complicated, all it does is counting and dividing the results to the total number of what it counts. To estimate π_c, we can count how many times a specific label c occurs in the training data and then divide it by the the length of our the training set. Similarly, to estimate $\theta_{:c}$, we can count how many times every word occurs in the training data labeled with specific class c by only traversing it once and divide the values by the total number of words for class c. Additive smoothing can also be easily adapted to this kind of implementation. If you get rid of the redundancies in your code in this way, your implementation should be able to finish its job under 1 second.

2.4 Task

You are expected to train a Naive Bayes classifier, test it using the test set and report the accuracy. The function templates are given in the **nb.py** and some tests are given in the **nb_mini_test.py**. Passing all these tests does not mean you will get full points.

- Extract the words in the sentences. Since these tweets are not preprocess, you can clean up your data by removing the special characters. However, some of them may be important to the classification; for example, the emojis. This part of the homework will not be graded however, you still need to convert the sentences into words so that you can use them in your naive bayes algorithm. You can simply divide your sentences from whitespaces if you want to.
- Implement the functions in nb.py according to their descriptions. During test time, you can skip the words that are not in the vocabulary created using the train set. The scores of the test function should be calculated using (as mentioned above)

$$h(x) = \underset{c}{\operatorname{argmax}}\, log(\hat{\pi}_c) + \sum_{j=1}^{d} x_j log(\hat{\theta}_{jc}).$$

- Calculate the accuracy of your model using the functions in the nb.py on the test set and print it.

3 Hidden Markov Models

In this part, you are going to work on **evaluation** and **decoding** tasks of Hidden Markov Models (HMM). To do that, you are going to implement **forward** and **Viterbi** algorithms. The template of the functions are given in "hmm.py". **You can check the recitation slides for the detailed explanations of forward and Viterbi algorithms and understand the notation used here.**

3.1 Data

Only filling the **forward** and **viterbi** functions is enough for this homework. Arguments will be directly fed into those functions. Therefore, there is no need to explicitly parse the input or printing the output. You only need to return the expected outputs.

Although the outputs of the tasks are different their inputs are the same. The inputs are as mentioned in Table 2.

To test your implementation, you can use hmm_mini_test.py which uses the example in the recitation slide. Final grading will not be done using only those inputs; therefore, passing the given examples may not mean you will get 100 points.

The data will be given in numpy arrays. $A[i, j]$ is the state transition probability from state i to state j. $B[i, j]$ is the probability of observing observation j in state i. $pi[i]$ is the probability of initial state being i. $O[t]$ is the t^{th} observation, which is an index between 0 and M-1 (inclusive).

Input Symbol	Input Name	Input Size
A	State Transition Matrix	NxN
B	Observation Probability Matrix	NxM
pi	Initial State Probabilities	N
O	Observation Sequence	T

Table 2: Explanation of the input symbols in tasks. N, M, T represent number of states, number of possible observations, length of the observation sequence, respectively. ($2 \leq N \leq 10$, $2 \leq M \leq 10$, $1 \leq T \leq 30$)

3.2 Evaluation Task

For this task, you are going to implement **forward** algorithm by filling the forward function in "hmm.py". The output should be the **probability** of an observation sequence given A, B, pi and **the calculated alpha values** in a numpy array. This can be done by calculating this probability for every possible state sequence and summing them up; however, this takes exponential time. Therefore, in this homework, **you are asked to implement the forward algorithm**, which runs in N^2T time. If your algorithm runs in exponential time, you won't be able to get full points.

3.3 Decoding Task

Similar to the evaluation task, you are going to implement **Viterbi** algorithm by filling the viterbi function in "hmm.py". The output should be the numpy array of **most likely observation sequence** given A, B, pi and **the calculated deltas** in a numpy array. This can be done by calculating this probability for every possible state sequence then selecting the maximum; however, this takes exponential time. Therefore, in this homework, **you are asked to implement the Viterbi algorithm**, which runs in N^2T time. If your algorithm runs in exponential time, you won't be able to get full point.

4 Specifications

- The codes must be in Python3 and use only numpy. Any other programming language or library will not be accepted.
- Falsifying results is strictly forbidden and you will receive 0 if this is the case. Your programs will be examined to see if you have actually reached the results and if it is working correctly.
- The late policy given in the syllabus applies here. You have total of 6 late days for **all** your homeworks but you can spend at most 3 for a specific homework. The late submission penalty will be calculated using $5n^2$, that is, 1 day late submission will cost you 5 points, 2 days will cost you 20 points and 3 days will cost you 45 points. No late submission is accepted after reaching a total of 3 late days.
- Using any piece of code that is not your own is strictly forbidden and constitutes as cheating. This includes friends, previous homeworks, or the internet. The violators will be punished according to the department regulations.
- Follow the course page on ODTUCLASS for any updates and clarifications. Please ask your questions on Homework 4 Discussion section of ODTUCLASS instead of e-mailing if the question does not contain code or solution.

5 Submission

Submission will be done via ODTUCLASS. If you do not have access to ODTUCLASS send send an email to "artun@ceng.metu.edu.tr" as soon as possible. You will submit a zip file called "hw4.zip" that contains all the source code, README file.