67750: Advanced Practical Machine Learning	2019/20

Exercise 2 — Structured Prediction for PoS Tagging

Dr. Omri Abend — *TA: Omri Bloch*

1 Theoretical Questions (10 Points)

1.1 MEMM "contains" HMM

Show that the log linear model we saw in class is able to express any HMM of the type that was learned in class in terms of inference (given a HMM with parameters t and e, find a relevant feature mapping and weight vector such that the MEMM and HMM give the same results when running Viterbi).

1.2 Higher Order Markov Model

In a second-order Markov model, a state at time i is independent of previous states given the last two states:

$$\mathbb{P}(x_{1:N}, y_{1:N}) = \prod_{i=1}^{N} \mathbb{P}(y_i|y_{i-2}, y_{i-1})\mathbb{P}(x_i|y_i)$$

Can this model also be written as log linear model? why not? Or if it is possible - how many parameters will the model have?

1.3 Energy Based Model Gradient

The MEMM, along with the MST parser, can be viewed as particular kinds of **energy based models** (EBM). These models are inspired by statistical mechanics and define an energy function, $E(x;\theta)$, such that the modeled distribution of the data follows the Boltzmann distribution:

$$p(x) = \frac{1}{Z} e^{-E(x;\theta)}$$

$$Z = \sum_{x} e^{-E(x;\theta)}$$

1. What is the energy function of the MEMM for a single transition, $E(y_t|y_{t-1}, x_{1:T};\theta)$?
2. Calculate the expected gradient of the log probability of a general energy based model and show that it is the difference between the gradient of the energy according to the model distribution and the true distribution. Formally, show that:

$$\mathop{\mathbb{E}}_{x\sim\mathcal{D}}\Big[\frac{\partial log(\mathbb{P}_\theta(x))}{\partial\theta}\Big] = \mathop{\mathbb{E}}_{x\sim\mathbb{P}_\theta}\Big[\frac{\partial E(x;\theta)}{\partial\theta}\Big] - \mathop{\mathbb{E}}_{x\sim\mathcal{D}}\Big[\frac{\partial E(x;\theta)}{\partial\theta}\Big]$$

In words, the gradient decreases the energy (increases the probability) of inputs which exist in the real distribution and increases the energy (decreases the probability) of inputs that exist in the model distribution.

1.4 MLE for HMM

In class, we saw the log likelihood function for the HMM:

$$\ell(S,\theta) = \sum_{i=1}^{N}\sum_{t=1}^{T^{(i)}} log(t_{y_{t-1}^{(i)},y_t^{(i)}}) + \sum_{i=1}^{N}\sum_{t=1}^{T^{(i)}} log(e_{y_t^{(i)},x_t^{(i)}})$$

Derive the MLE in class for $t_{i,j}$:

$$\hat{t}_{i,j} = \frac{\#(y_i \to y_j)}{\sum_k \#(y_i \to y_k)}$$

2 Practical Exercise (90 Points)

Please submit a single tar file named "ex2_<YOUR_ID>". This file should contain your code, and an "Answers.pdf" file in which you should write your answers and provide all figures/data to support your answers (and write your ID(s) in there as well, just in case). Your code will be checked manually so please write readable, well documented code.

If you have any constructive remarks regarding the exercise (e.g. question X wasn't clear enough, we lacked the theoretical background to complete question Y, the exercise was too long and question Z felt like a lot of work with little knowledge gained...) we'll be happy to read them in your Answers file.

2.1 Exercise Requirements

1. Implement a baseline model which assumes the PoS tags are independent from one another and only depend on the observed word (10 points).
2. Write a function that calculates the MLE for the transition and emission probabilities in a HMM (10 points).
3. Write a sampling function from a given HMM (5 points).
4. Write a Viterbi function that calculates the MAP assignment of the hidden variables of a HMM, given a sequence of observed variables and a model (20 points).
5. Write a Viterbi function that calculates the MAP assignment of the hidden variables of a MEMM (10 points).
6. Write the Perceptron algorithm for learning MEMM parameters (15 points).
7. Design feature mapping for your MEMM and demonstrate which features are important/-constructive when tagging parts-of-speech. Compare the model to the others (20 points).

Note that you are given a suggested API, but you are allowed to change the supplied function signatures and anything else you want as long as you explain the changes you've made in your answers pdf.

2.2 Data

In this exercise you are given a large dataset of sentences along with their corresponding Parts-of-Speech. You should split this data into a training and test set to evaluate your models during the exercise.

The dataset is comprised of three pickled files:

1. PoS_data.pickle - sentences and their corresponding parts-of-speech.
2. all_words.pickle - a list of all the words in the data set.
3. all_PoS.pickle - a list of all the parts-of-speech in the data set.

An example of loading and using the data set can be seen in the "data_example" function.

2.2.1 Processing the Data

Note that the data hasn't been fully processed for you. We did not add "START" and "END" hidden states (PoS tags), so you will have to add them in if you intend to use them in your model. Also, we did not deal with "rare" words as discussed in the recitation, so you will have to do that for yourself.

2.3 Baseline Model

To see how the HMM model improves upon the most naive model, you need to implement a naive baseline model. In this model, we will assume that the hidden states are actually independent - $\mathbb{P}(y_t|y_{1:t-1}) = \mathbb{P}(y_t)$. This means you only need to learn the emission probabilities and multinomial probabilities over the hidden states.

So in our baseline model, the probability of a sequence will be:

$$\mathbb{P}(x_{1:T}, y_{1:T}) = \prod_t \mathbb{P}(y_t)\mathbb{P}(x_t|y_t) = \prod_t \pi_{y_t} e_{y_t,x_t}$$

MAP inference in our baseline model will be very simple. Since the hidden variables are independent from one another, we can simply calculate the MAP for every PoS in the sentence separately:

$$\underset{y_t}{argmax}\mathbb{P}(y_t|x_t) = \underset{y_t}{argmax}\pi_{y_t} e_{y_t,x_t}$$

Learning the model is also very simple and can be derived using the MLE principle.

Implement the baseline model as described in the supplied code.

2.4 HMM

2.4.1 MLE Estimation

In this part you are required to create a Maximum Likelihood estimator for the standard multinomial HMM learned in class. Given training data, your estimator should return the transition and emission probabilities that you will later be able to use for prediction and other tasks. See the function header for more details.

2.4.2 Sampling from a Generative Model

One of the benefits of a generative model is that we can easily obtain data that is actually sampled from the probability distribution described by the model. This is useful for testing our model and, for stronger generative models, for generating data in general.

Implement a function for sampling from a given HMM (see function header for more details) and show results from your sampling function (after you've learned the model with MLE). Do the samples resemble English at least in a PoS sequence sense?

2.4.3 Inference

Write a Viterbi function for HMMs, which calculates the MAP assignment of PoS tags for a given sentence. Look at the function header for more details.

Make sure to use log probabilities to avoid numerical issues...

2.5 MEMM

2.5.1 Inference

Write a Viterbi function for MEMMs, which calculates the MAP assignment of PoS tags for a given sentence. Look at the function header for more details.

2.5.2 Learning w

Use your Viterbi implementation to learn the log-linear model parameter w using the Structured Perceptron algorithm learned in class. See function header for more details.

2.5.3 Make Your Own Model

Now that everything is ready, we can start experimenting with discriminative models. Create a feature mapping that uses features in addition to the transitions and emissions (word suffixes, letter capitalization, pairs of words or any other feature you can think of).

2.6 Model Comparison

Perform a comparison between the different models (the baseline model, the HMM and the MEMM with your feature mapping). Use increasing proportions of the data (10%, 25%, 90%) to learn your model, and evaluate the learned model on the remaining 10%. How do the models compare? Did the features in the MEMM improved the performance?

Provide figures/data as necessary.

2.7 Additional Advice

2.7.1 Working in Log Space

The same numerical problems we dealt with in the image denoising exercise are coming back to haunt us here. Using normal probabilities for a sentence that can contain over 10 words, where each transition and emission has a very small probability of occurring, will cause all of your probabilities to be zero.

In order to deal with this issue, we recommend you work in log-space when calculating the MAP estimate using the Viterbi algorithm for your HMM and MEMM, as explained in the recitation.

2.7.2 Runtime

We recommend starting early with this exercise since training your models, especially the log-linear models using the Perceptron algorithm, can take some time. As usual, make sure you write your code in vector notation to allow NumPy to optimize the different operations.

We also recommend training with the Perceptron algorithm code on a small subset of the training set, and moving to 10% of the data when you are confident your code works. you don't need to train it on larger parts, it might take too much time.

Using the CS computers could also improve your runtime if your personal computer is slow.

2.7.3 Evaluating the Models

In order to evaluate your models on the test set, we need to define a measure of success. We will use the standard measure of success for PoS tagging, which will be the percentage of successful PoS tags in the test set. This means you should measure each PoS tag separately (and not whether or not the entire sentence was tagged correctly).

With this evaluation method you should expect your HMM to reach ˜95% success.

2.7.4 Long Feature Vectors

In MEMMs we can generally treat the feature mapping ϕ as mapping a transition and an emission to a general vector in $\mathbb{R}^d$. But since the feature vectors are possibly rather long (if we give a specific

index to each possible transition and emission), it can help us to treat the feature mapping as a mapping to $\{0,1\}^d$ (as seen in class).

This means that each element in the feature space is binary, and signifies whether or not a certain feature is present (does the suffix "ly" appear? is there a transition from a noun to a verb?). This makes the Viterbi algorithms faster, since **we don't really need to calculate dot products of large vectors to get the probabilities** - we just need to sum the relevant indices of the weight vector:

$$w^T \phi = \sum_{i:\phi_i=1} w_i$$

Same goes for the Perceptron, since we don't need to subtract the full vector of weights - just subtract the learning rate from the relevant indices of the weight vector.

2.7.5 MEMM vs HMM

Don't be alarmed if your MEMM performs worse than the HMM. While it is indeed more expressive than HMMs (as seen in the theoretical part), the training makes it harder to reach the right weights. There are intelligent ways to adapt the Perceptron algorithm so as to improve the convergence and performance, but you are not required to come up with such ways (you are of course welcome to try). These methods could be a smart initialization of w, using batches of sentences instead of single sentences and so on.

While we do not expect your MEMM to beat your HMM, you are still expected to reach a high accuracy with your MEMM (and explore the effects of additional features to your model's performance).