CSCI 567, Summer '18
Michael Shindler

Programming Set #2

Due: 10 days after assigned

General instructions

Due date and time This would be a 10-day assignment if it were assigned and collected for grade.

Starting point Please find GitHub Classroom link posted on Piazza to get access to your individual repository. Please do not change the name of this repository or the names of any files we have added to it. Please perform a `git pull` to retrieve these files.

High-level descriptions

0.1 Tasks

0.1.1 Logistic regression

- Finish implementing the functions: `binary_train`, `binary_predict`, `OVR_train`, `OVR_predict`, `multinomial_train`, and `multinomial_predict`. Refer to `logistic.py` and Sect. 1 for more information.
- Run the scripts `logistic_binary.sh` and `logistic_multiclass.sh` after you finish your implementation. This will output: `logistic_binary.out` and `logistic_multiclass.out`.
- Add, commit, and push `logistic.py`, `logistic_binary.out`, and `logistic_multiclass.out`.

0.1.2 Hidden Markov model

- Finish implementing the functions: `forward`, `backward`, `seqprob_forward`, `seqprob_backward`, and `viterbi`. Refer to `hmm.py` for more information and Sect. 2 for more information.
- Run the script `hmm.sh`, after you finish your implementation. `hmm.sh` will output `hmm.txt`.
- Add, commit, and push `hmm.py` and `hmm.txt`.

0.2 Dataset

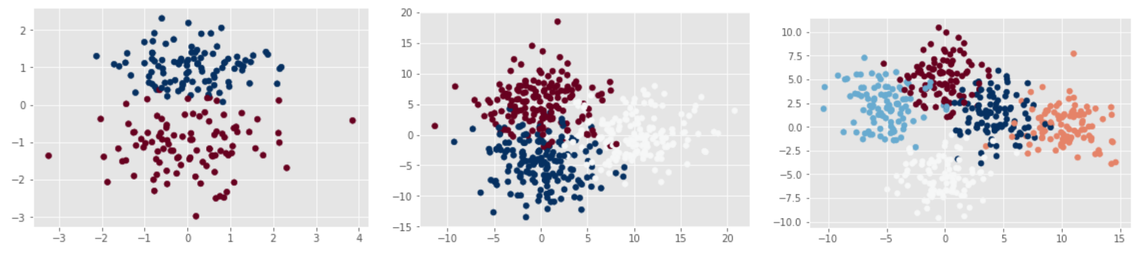
0.2.1 Logistic regression

For logistic regression in Sect. 1, you will be using synthetic datasets with two, three and five classes. Synthetic data that you will use is illustrated in Figure 1.

0.2.2 Hidden Markov model

For HMM, you will experiment with a small hidden Markov model. The parameters of the model are given in `hmm_model.json`.

Figure 1: Synthetic datasets



0.3 Warnings Please **DO NOT** import packages that are not listed in the provided code. Follow the instructions in each section strictly to code up your solutions. **Do not change the output format**. **Do not modify the code unless we instruct you to do so**. A homework solution that does not match the provided setup, such as format, name, initializations, etc., **will not** be graded. It is your responsibility to **make sure that your code runs with the provided commands and scripts on the VM**. Finally, make sure that you **git add, commit, and push all the required files**, including your code and generated output files.

0.4 Programming advices

- We are extensively using `softmax` and `sigmoid` functions in this homework. To avoid numerical issues such as overflow and underflow caused by *numpy.exp()* and *numpy.log()*, please follow these implementation techniques:
  - Let x be an input vector (one data sample) to the `softmax` function. Use $\tilde{x} = x - \max_j(x_j)$ instead of using x directly for the `softmax` function f, i.e. $f(\tilde{x}_i) = \frac{\exp(\tilde{x}_i)}{\sum_{j=1}^{D} \exp(\tilde{x}_j)}$
  - If you are using *numpy.log()*, make sure the input to the log function is positive. Also, there may be chances that one of the outputs of softmax, e.g. $f(\tilde{x}_i)$, is extremely small but you need the value $\ln(f(\tilde{x}_i))$, you can convert the computation into $\tilde{x}_i - \ln(\sum_{j=1}^{D} \exp(\tilde{x}_j))$.

  We have implemented and run the code ourselves without problems, so if you follow the instructions and settings provided in the python files, you should not encounter overflow or underflow.
- In Python, for-loops are slow. If you find that your codes run for longer than 20 minutes, consider using *numpy*'s matrix operations instead.

0.5 Final submission Add, commit, and push `hmm.py`, `hmm.txt` that you have generated before the deadline. **Please DO NOT commit and push extraneous files, including any dataset *not* already added by teaching staff. Failure to follow this instruction will result in a penalty**.

Problem 1 Logistic Regression

For this assignment you are asked to implement Logistic Regression for binary and multiclass classification. Recall that Logistic Regression does not have a closed-form solution, so for training you will need to implement gradient descent. We ask that you implement batch gradient descent for this assignment. You can set a threshold on the size of the gradients or the changes in weights to do early-stopping; however, make sure you don't stop prematurely.

1.1 Binary classification Recall from lecture 15 and 16, that binary classification with Logistic regression is performed by the following rule:

$$p(y == 1|x) = \sigma(w^T x + b) \tag{1}$$

$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{2}$$

Given a training set $\mathcal{D} = \{(x_n, y_n)_{n=1}^N\}$, where $y_i \in \{0, 1\}\ \forall i = 1...N$, you will need to write a function that will find the optimal parameters w.

You will need to implement functions `binary_train` and `binary_predict` in `logistic.py`.

After you finished implementation, please run `logistic_binary.sh` script, which will produce `logistic_binary.out`.

What to submit: `logistic.py` and `logistic_binary.out`.

1.2 Multiclass classification: One-Versus-Rest In the lectures you learned several methods to perform multiclass classification. One of them was one-versus-rest approach.

For one-versus-rest classification in a problem with K classes, we need to train K classifiers. Each classifier is trained on a binary problem, where belonging to the class corresponding to the classifier is a positive outcome, and belonging to any other class is a negative outcome. After that, the multiclass prediction is made based on the combination of all predictions from K binary classifiers. [*Hint*: use numpy argmax function for combination.] You can assume that the class labels are from the set $\{0, 1, ..., K - 1\}$.

You will need to complete functions `OVR_train` and `OVR_predict` to perform one-versus-rest classification. If needed, you can make calls to the binary logistic regression train and predict functions that you implemented before.

After you finished implementation, please run `logistic_multiclass.sh` script, which will produce `logistic_multiclass.out`.

What to submit: `logistic.py` and `logistic_multiclass.out`.

1.3 Multiclass classification: Multinomial Yet another multiclass classification method you learned was multinomial logistic regression. Here, for the conditional probability we replaced the *sigmoid* function from binary classification with a *softmax* function:

$$p(y == k|x) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{k'} e^{\mathbf{w}_{k'}^T \mathbf{x}}} \tag{3}$$

In this setup we again need to train K binary classifiers, and each point is assigned to the class, that maximizes the conditional probability from Eq 3.

Complete the functions `multinomial_train` and `multinomial_predict` to perform multinomial classification.

After you finished implementation, please run `logistic_multiclass.sh` script, which will produce `logistic_multiclass.out`.

What to submit: `logistic.py` and `logistic_multiclass.out`.

Problem 2 Hidden Markov Models

In this problem, you are given parameters of a small hidden Markov model and you will implement three inference procedures, similar to what you will or have seen in Problem Set 2. In `hmm_model.json`, you will find the following model parameters:

- π: the initial probabilities, $\pi_i = P(Z_1 = s_i)$;
- A: the transition probabilities, with $a_{ij} = P(Z_t = s_j | Z_{t-1} = s_i)$;
- B: the observation probabilities, with $b_{ik} = P(X_t = o_k | Z_t = s_i)$.

Now we observe a sequence O of length L. Your task is to write code to compute probabilities and infer about the possible hidden states given this observation. In particular, first in **1.1** and **1.2**, we want to compute $P(x_1, \dots, x_L = O)$, the probability of observing the sequence. You should use the forward algorithm and the backward algorithm to achieve that. Then in **1.3**, we infer the most likely state path. Note that your code might be tested against different parameter sets/observation sequences at grading time.

N.B. Since `backward()` was not covered in class this semester. You will have to look it up yourself (i.e. Introduction to Research 101) or just ignore pertaining parts.

2.1 Please finish the implementation of the functions `forward()` and `backward()` in `hmm.py`:

- `forward(`π, A, B, O`)` takes the model parameters π, A, B and the observation sequence O as input and output a numpy array δ, where $\delta[j, t] = \delta_t(j) = P(Z_t = s_j, x_{1:t})$.
- `backward(`π, A, B, O`)` takes the model parameters π, A, B and the observation sequence O as input and output a numpy array γ, where $\gamma[j, t] = \gamma_t(j) = P(Z_t = s_j \mid x_{t+1:T})$.

2.2 Now we can calculate $P(x_1, \dots, x_L = O)$ from the output of your forward and backward algorithms. Please finish the implementation of the function `seqprob_forward()` and `seqprob_backward()` in `hmm.py`. Both of them should return $P(x_1, \dots, x_L = O)$.

2.3 We want to compute the most likely state path that corresponds to the observation O. Namely,

$$\boldsymbol{k}^* = (k_1^*, k_2^*, \cdots, k_L^*) = \arg\max_{\boldsymbol{k}} P(s_{k_1}, s_{k_2}, \cdots, s_{k_L} | x_1, x_2, \cdots, x_L = O).$$

Please implement the Viterbi algorithm in `viterbi()` in `hmm.py`. The function `viterbi(`π, A, B, O`)` takes the model parameters π, A, B and the observation sequence O as input and output a list `path` which contains the most likely state path $\boldsymbol{k}^*$ (in terms of the state index) you have found.

What to do and submit: After you finish each task in 2.1, 2.2 and 2.3 in `hmm.py`, run the script `hmm.sh`. It will generate `hmm.txt`. Add, commit, and push both `hmm.py` and `hmm.txt` before the due date.