# 10-708 PGM (Spring 2019): Homework 2 v1.1


Andrew ID: [your Andrew ID]
Name: [your first and last name]
Collaborators: [Andrew IDs of all collaborators, if any]


## 1 Sequential models for POS tagging [30 pts] (Maruan)

Conditional Random Field (CRF) [1] is a popular probabilistic model used for structured prediction or modeling conditional probability distributions over structured spaces. In Homework 1, you have implemented an algorithm for learning parameters of an (unsupervised) Hidden Markov Model (HMM). In this problem, we will apply a (supervised) HMM and a CRF to part-of-speech (POS) tagging and compare the two models.

In part-of-speech tagging, we are given a sentence and would like to categorize each word (or assign it a tag) to a particular part of speech. Here is an example:

```
Sentence: The  reaction  in   the  newsroom  was   emotional  .
POS tags: DET  NOUN      ADP  DET  NOUN      VERB  ADJ        .
```

In this problem, we will use the universal tagset [2] (which consists of 12 tags), and train and evaluate our algorithms on the Penn Treebank (PTB) corpus. Download the data from: `https://sailinglab.github.io/pgm-spring-2019/assets/assignments/hw2/pos-data.zip`.

The data folder must contain the following files: `train_set.npy`, `test_set.npy`, `vocab.txt`, `tagset.txt`, that correspond to the training sentences, testing sentences, the vocabulary, and the tagset, respectively.

### 1.1 POS tagging with HMM [10 pts]

Since assigning POS tags to words outside of context can be very ambiguous (e.g., try to tag "*the sailor dogs the hatch*"), we need a model that can take into account the whole sequence of words and infer the most likely sequence of corresponding tags.

1. **[1 point]** Denoting the sequence of $T$ words as $\mathbf{x} := (x_1, \ldots, x_T)$ and the corresponding sequence of POS tags as $\mathbf{y} := (y_1, \ldots, y_T)$, define an HMM for modeling these sequences: (a) specify how you represent observations and latent states, and (b) specify the parameters of your model.
2. **[3 points]** Given a set of tagged training sentences, describe and implement the learning algorithm for your HMM. Note that due to sparsity, estimation of emission probabilities requires smoothing (use $\lambda = 0.01$ for smoothing). Report the following:
   - The obtained state transition matrix $A$ (round or truncate the values in the matrix to a reasonable precision for visualization purposes).
   - The *negative log likelihood* for your HMM on the train and test sets.[1]

[1] You can report either $-\log P_\theta(\mathbf{x})$ or $-\log P_\theta(\mathbf{x}, \mathbf{y})$, but specify which one you report.

3. **[6 pts]** Now having a trained model, we can answer queries such as, *what is the most likely sequence of tags for the given sequence of words?* Mathematically, we answer such queries by solving the following optimization problem:

$$\mathbf{y}^\star = \arg\max_{\mathbf{y}} \mathbb{P}_{\text{HMM}}(\mathbf{y} \mid \mathbf{x}) \tag{1}$$

Describe and implement an efficient algorithm for solving (1) (*hint*: you can use some primitives from the Baum-Welch algorithm that you implemented in HW1). Infer the most likely sequences of tags for the test set sentences. Report the per-word accuracy and visualize the confusion matrix between the tags.

## 1.2 POS tagging with a linear-chain CRF [20 pts]

POS tagging is a supervised learning problem with sequential inputs and sequential outputs. Instead of using HMM (a generative model), we could try to directly model the conditional distribution $\mathbb{P}(\mathbf{y} \mid \mathbf{x})$ using the following form:

$$\mathbb{P}(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \Psi(y_t, y_{t-1}, \mathbf{x}), \tag{2}$$

where $\Psi(y_t, y_{t-1}, \mathbf{x})$ is a non-negative potential function that inputs a pair of tags $y_t, y_{t-1}$ and a sequence of words $\mathbf{x}$, and $Z$ is the normalizing constant.

1. **[2 pts]** Show that the conditional distribution, $\mathbb{P}(\mathbf{y} \mid \mathbf{x})$, that results from an HMM is a particular kind of a linear chain CRF (*hint*: write down the complete likelihood for the HMM as a product of exponents).
2. **[2 pts]** Directly modeling $\mathbb{P}(\mathbf{y} \mid \mathbf{x})$ with a CRF allows us to additionally "featurize" the model, or in other words, define additional potentials $\Psi(y_t, y_{t-1}, \mathbf{x})$ that may depend on non-trivial features extracted from the input sequence. Mathematically define a linear chain CRF model that has potentials of the form $\Psi(y_t, y_{t-1}, \mathbf{x})$ which are *linear functions* of the following features:
   (a) identity of the given word, $x_t$, and the current tag, $y_t$,
   (b) identity for whether $x_t$ is the first word in the sentence, and the current tag, $y_t$,
   (c) identity for whether $x_t$ is the last word in the sentence, and the current tag, $y_t$,
   (d) identity of the previous word, $x_{t-1}$, and the current tag, $y_t$,
   (e) identity of the next word, $x_{t+1}$, and the current tag, $y_t$,
   (f) identity for whether $x_t$ contains a capital letter, and the current tag, $y_t$,
   (g) identity of the previous tag, $y_t$, and the current tag, $y_t$,
3. **[2 pts]** Derive an inference algorithm for determining the most likely sequence of POS tags under your CRF model (*hint*: the algorithm should be very similar to the one you designed for HMM in 1.1).
4. **[2 pts]** Derive a maximum likelihood learning algorithm for your linear chain CRF.
5. **[10 pts]** Implement the learning and inference procedures for CRF that you derived in the previous steps.
   (a) First, pre-process the training and test data to incorporate the additional features we defined for the potential functions.
   (b) Implement the learning and inference algorithms as you described above. External libraries that provide complete implementations of CRF models are prohibited (you would get 0/10 credit for

this part of the problem). However, you can use any external packages for solving optimization problems as building blocks for your algorithms (including PyTorch or TensorFlow).

(c) Finally, train your model on the `train_set` and run inference on the `test_set`. Report the following:

i. The negative log conditional likelihood (i.e., $-\log \mathbb{P}(\mathbf{y} \mid \mathbf{x})$) of your trained CRF model on both training and test sets.

ii. Per-word accuracy and the confusion matrix for your model predictions on the test set.

iii. Values of the parameters of your linear chain CRF after learning.

6. **[2 pts]** As we have seen, HMM and CRF are respectively generative and discriminative approaches to solving the problem of sequence prediction. Discuss advantages/disadvantages of each model. What if we have limited labeled data?

# 2 Variational Inference (VI) [40 pts] (Hao)

## 2.1 Warm-up: VI vs. MCMC [10 pts]

We will compare Variational Inference (VI) with Markov Chain Monte Carlo (MCMC), the two most popular methods for approximate inference. Below are a list of algorithmic properties, or problem cases that you want to choose one of them to solve. For each item, link to either VI or MCMC, and reason briefly with 1-2 sentences.

1. **[1 point]** Inference results are generally closer to target distributions.
2. **[1 point]** Non-parametric.
3. **[1 point]** Amendable to batched computation using GPUs.
4. **[1 point]** Transform inference into optimization problems.
5. **[1 point]** Easier to integrate with back-propagation.
6. **[1 point]** Involves more stochasticity.
7. **[1 point]** Easier to set the termination condition for the computational loop.
8. **[1 point]** Higher variance under limited computational resources.
9. **[1 point]** Problem case: Estimating a topic model with online streaming text data.
10. **[1 point]** Problem case: Estimating a topic model from a very small text corpus.
11. **[Bonus, 1 point]** Problem case: Posterior inference with non-conjugate distributions.
12. **[Bonus, 1 point]** More naturally fit with stick-breaking process.

## 2.2 VI in Practice [30 pts]

### 2.2.1 A Review of the Vanilla LDA [9 pts]

We have covered the Latent Dirichlet Allocation (LDA) [3], one of the most successful topic models in class. Given a collection of text documents $\mathcal{D} = \{\boldsymbol{w}_d\}_{d=1}^{D}$,we denote $\boldsymbol{w}_d$ as the $d$th document in the corpus, and $\boldsymbol{w}_d = \{w_{di}\}_{i=1}^{N_d}$ with $w_{di}$ being the $i$th word in $\boldsymbol{w}_d$, and $N_d$ as the number of words in $\boldsymbol{w}_d$. LDA assumes each document is generated from a mixture of $K$ topics, with the mixture proportion $\boldsymbol{\theta}_d$, where each topic is a multinomial distribution parametrized by $\boldsymbol{\beta}_k$ over all $V$ words existed in the vocabulary. The generative process of LDA is therefore as follows:

- For $k = 1 \to K$:
  - Draw topic $\boldsymbol{\beta}_k \sim \text{Dirichlet}(\boldsymbol{\eta})$.
- For each document $\boldsymbol{w}_d \in \mathcal{D}$:
  - Draw per-document mixture proportion $\boldsymbol{\theta}_d \sim \text{Dirichlet}(\boldsymbol{\alpha})$.
  - For each word $w_{di} \in \boldsymbol{w}_d$:
    - Sample a topic indicator $z_{di} \sim \text{Multinomial}(\boldsymbol{\theta}_d)$
    - Sample the word $w_{di} \sim \text{Multinomial}(\boldsymbol{\beta}_{z_{di}})$

where we have used $\boldsymbol{\eta}$ and $\boldsymbol{\alpha}$ as the parameters of the two prior Dirichlet distributions for $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$, respectively. The graphical model diagram of LDA is showed in Figure 1. Answer the following questions about LDA (no need to explain).

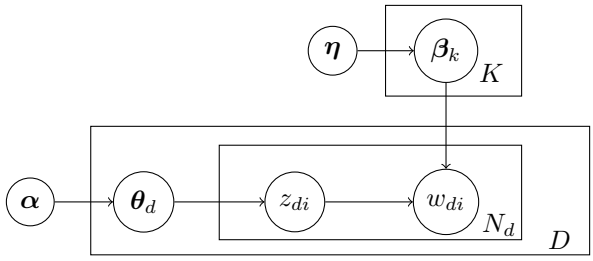


Figure 1: The graphical model diagram for LDA.

1. **[1 point]** True or False: Solving LDA involves posterior inference with non-conjugate distributions.
2. **[1 point]** True or False: Solving LDA using Variational EM involves non-convex optimization at E-step.
3. **[2 points]** Count the number of **model parameters** of the LDA using notations defined above.
4. **[2 points]** Write down the computational complexity of mean-field VI for LDA using Big O notation.
5. **[2 points]** Write down the memory complexity of mean-field VI for LDA using Big O notation.
6. **[1 point]** True or False: The vanilla LDA can be used to extract word embeddings.

### 2.2.2 More HMMs [21 pts]

The vanilla LDA generally assumes each word is independently generated given the topic vectors. In this question, we will try to enhance it by further considering the orders of the words during the document generation process, as we believe in many natural languages, the topic of the word at index $i$ might be dependent on the topic of its previous word (at index $i-1$).

A natural and powerful approach to modeling orders is to (once again) use the Hidden Markov Model (HMM). To incorporate HMM into LDA, we assume a transit matrix $\boldsymbol{T}$ among $K$ topics where the entry at the $m$th row and $n$th column of $\boldsymbol{T}$, denoted as $\boldsymbol{T}_{mn}$, represents the probability of the $m$th topic transiting to the $n$th topic. With $\boldsymbol{T}$ introduced, it is convenient for us to model sequential dependencies in the hidden space: instead of sampling each word independently, we let the word $w_{di} (i > 1)$, with probability $p (0 < p < 1)$, sampled from the topic $z_{di}$, while with probability $1 - p$, transited from the topic $z_{d(i-1)}$, which is the topic indicator of the word $w_{d(i-1)}$. We therefore can alter the generative process as follows:

- For each document $\boldsymbol{w}_d$:
  - Sample $p_d \sim \text{Beta}(\gamma)$.
  - Sample $\boldsymbol{\theta}_d \sim \text{Dirichlet}(\boldsymbol{\alpha})$.
  - For the first word (the head of HMM), sample $z_{d1} \sim \text{Multinomial}(\boldsymbol{\theta}_d),\ w_{d1} \sim \text{Multimomial}(\beta_{z_{d1}})$.
  - For each word $w_{di}$ $(i > 1)$:
    - Sample $\delta_{di} \sim \text{Bernoulli}(p_d)$.
    - If $\delta_{di} = 1$: sample $z_{di} \sim \text{Multinomial}(\boldsymbol{\theta}_d)$, then $w_{di} \sim \text{Multimomial}(\beta_{z_{di}})$.
    - If $\delta_{di} = 0$: sample $z_{di} \sim \text{Multinomial}(\boldsymbol{T}_{z_{d(i-1)},:})$, then $w_i \sim \text{Multimomial}(\beta_{z_{di}})$.

Where we use $\boldsymbol{T}_{m,:}$ to denote the $m$th row of $\boldsymbol{T}$. The posteriors of this model are still intractable, and we will use variational EM to perform posterior inference and estimate model parameters.

1. **[3 points]** Identify (a) the set of observed variables, (b) the set of hidden variables, (c) the set of model parameters to be estimated.
2. **[3 points]** Based on the altered generative process and Figure 1, draw the new graphical model for HMM-enhanced LDA (HMM-LDA).
3. **[2 points]** Write down the joint distribution defined by this generative process.
4. **[3 points]** Define variational distributions over hidden variables using proper distributions and variational parameters.
5. **[2 points]** Derive the ELBO using variational distributions.
6. **[4 points]** Derive the update rules for all variational parameters, which will be used in the E step.
7. **[4 points]** Derive the update rules for all model parameters, which will be applied in the M step.
8. **[Bonus, 6 points]** Implement the derived variational EM algorithm for HMM-LDA using Python ($\geq 3.5$). Train the model on the provided Wikipedia corpus (more details about the dataset can be found in the bundled `readme.txt`). Set the number of topics $K = 10$.
9. **[Bonus, 4 points]** Visualize each of the learned 10 topics by printing its top 10 words with highest probabilities, both in your script and in your write-up.

**If you choose to do question 8 and 9, please read the implementation notes below carefully before submitting your code**:

1. Only submit a single python script named `YourAndrewID.py` and a `requirements.txt` where you put all your needed python requirements. Assuming the dataset file is placed on the same directory path with your script, make sure your script is executable (without bugs) under a clean python ($\geq 3.5$) virtual environment with the following two commands:

```
pip install -r requirements.txt
python YourAndrewID.py
```

2. Tune your hyper-parameters properly to make sure the model could converge correctly. Print the values of ELBO after each iteration of E-step and M-step in stdout during the training process. Report the iteration of convergence. Print the value of ELBO after convergence in stdout.
3. Visualize the learned $K$ topics after convergence in both stdout and the your write-up in a way similar to the upper part of Figure 8 in [3].

# 3 Importance Sampling [10 pts] (Paul)

Given a random distribution $p(x)$ on $x = [x_1, ..., x_D]^\top \in \mathbb{R}^D$, suppose we want to perform inference $\mathbb{E}_{p(x)}[f(x)]$ using importance sampling, with $q(x)$ as the proposal distribution. According to importance sampling, we draw $L$ i.i.d. samples $x^{(1)}, ..., x^{(L)}$ from $q(x)$, and we have

$$\mathbb{E}_{p(x)}[f(x)] \approx \frac{1}{\sum_{i=1}^{L} u_i} \sum_{i=1}^{L} f(x^{(i)}) u_i \tag{3}$$

where the (unnormalized) importance weights $u_i = \frac{p(x^{(i)})}{q(x^{(i)})}$.

1. **[2 points]** Find the mean and variance of the unnormalized importance weights $\mathbb{E}_{q(x)}[u_i]$ and $\mathrm{Var}_{q(x)}[u_i]$. You may leave your answer in terms of $\mathbb{E}_{p(x)}$.
2. **[3 points]** Prove the following lemma: $\mathbb{E}_{p(x)}\left[\frac{p(x)}{q(x)}\right] \geq 1$ with equality if and only if $p = q$.
3. **[4 points]** A measure of the variability of two components in vector $u = [u_1, ..., u_L]^\top$ is given by $\mathbb{E}_{q(x)}[(u_i - u_j)^2]$. Assume that $p \neq q$ and that both $p$ and $q$ can be factorized, i.e. $p(x) = \prod_{i=1}^{D} p_i(x_i)$, and $q(x) = \prod_{i=1}^{D} q_i(x_i)$. Show that $\mathbb{E}_{q(x)}[(u_i - u_j)^2]$ has exponential growth with respect to $D$.
4. **[1 points]** Use the conclusion in (c) to explain why the standard importance sampling does not scale well with dimensionality and would blow up in high-dimensional cases.

# 4 Markov Chain Monte Carlo [20 pts] (Paul)

Nowadays, statistical modelling of sport data has become an important part of sports analytics and is often a critical reference for the managers in their decision-making process. In this part, we will work on a real world example in professional sports. Specifically, we are going to use the data from the 2013-2014 Premier League, the top-flight English professional league for men's football clubs, and build a predictive model on the number of goals scored in a single game by the two opponents. Bayesian hierarchical model is a good candidate for this kind of modeling task. We model each team's strength (both attacking and defending) as latent variables. Then in each game, the goals scored by the home team is a random variable conditioned on the attacking strength of the home team and the defending strength of the away team. Similarly, the goals scored by the away team is a random variable conditioned on the attack strength of the away team and the defense strength of the home team. Therefore, the distribution of the scoreline of a specific game is dependent on the relative strength between the home team A and the away team B, which also depends on the relative strength between those teams with their other opponents.

Table 1: 2013-2014 Premier League Teams

| | | | | | |
|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 |
| Team | Arsenal | Aston Villa | Cardiff City | Chelsea | Crystal Palace |
| Index | 5 | 6 | 7 | 8 | 9 |
| Team | Everton | Fulham | Hull City | Liverpool | Manchester City |
| Index | 10 | 11 | 12 | 13 | 14 |
| Team | Manchester United | Newcastle United | Norwich City | Southampton | Stoke City |
| Index | 15 | 16 | 17 | 18 | 19 |
| Team | Sunderland | Swansea City | Tottenham Hotspurs | West Bromwich Albion | West Ham United |

Here we consider using the same model as described by Baio and Blangiardo (2010) [4]. The Premier League has 20 teams, and we index them as in Table 1. Each team would play 38 matches every season (playing each of the other 19 teams home and away), which totals 380 games in the entire season. For the $g$-th game, assume that the index of home team is $h(g)$ and the index of the away team is $a(g)$. The observed number of goals $(y_{g0}, y_{g1})$ of home and away team is modeled as independent Poisson random variables:

$$y_{gj}|\theta_{gj} \sim \text{Poisson}(\theta_{gj}), \quad j = 0, 1 \tag{4}$$

where $\theta = (\theta_{g0}, \theta_{g1})$ represents the scoring intensity in the $g$-th game for the team playing at home ($j = 0$) and away ($j = 1$), respectively. We put a log-linear model for the $\theta$s:

$$\log \theta_{g0} = home + att_{h(g)} - def_{a(g)} \tag{5}$$

$$\log \theta_{g1} = att_{a(g)} - def_{h(g)} \tag{6}$$

Note that team strength is broken into attacking and defending strength. And home represents home-team advantage, and in this model is assumed to be constant across teams. The prior on the home is a normal distribution:

$$home \sim \mathcal{N}(0, \tau_0^{-1}) \tag{7}$$

where we set the precision $\tau_0 = 0.0001$.

The team-specific attacking and defending effects are modeled as:

$$att_t \sim \mathcal{N}(\mu_{att}, \tau_{att}^{-1}) \tag{8}$$

$$def_t \sim \mathcal{N}(\mu_{def}, \tau_{def}^{-1}) \tag{9}$$

We use conjugate priors as the hyper-priors on the attack and defense means and precisions:

$$\mu_{att} \sim \mathcal{N}(0, \tau_1^{-1}) \tag{10}$$

$$\mu_{def} \sim \mathcal{N}(0, \tau_1^{-1}) \tag{11}$$

$$\tau_{att} \sim \text{Gamma}(\alpha, \beta) \tag{12}$$

$$\tau_{def} \sim \text{Gamma}(\alpha, \beta) \tag{13}$$

where the precision $\tau_1 = 0.0001$, and we set parameters $\alpha = \beta = 0.1$.

This hierarchical Bayesian model can be represented using a directed acyclic graph as shown in Figure 2.

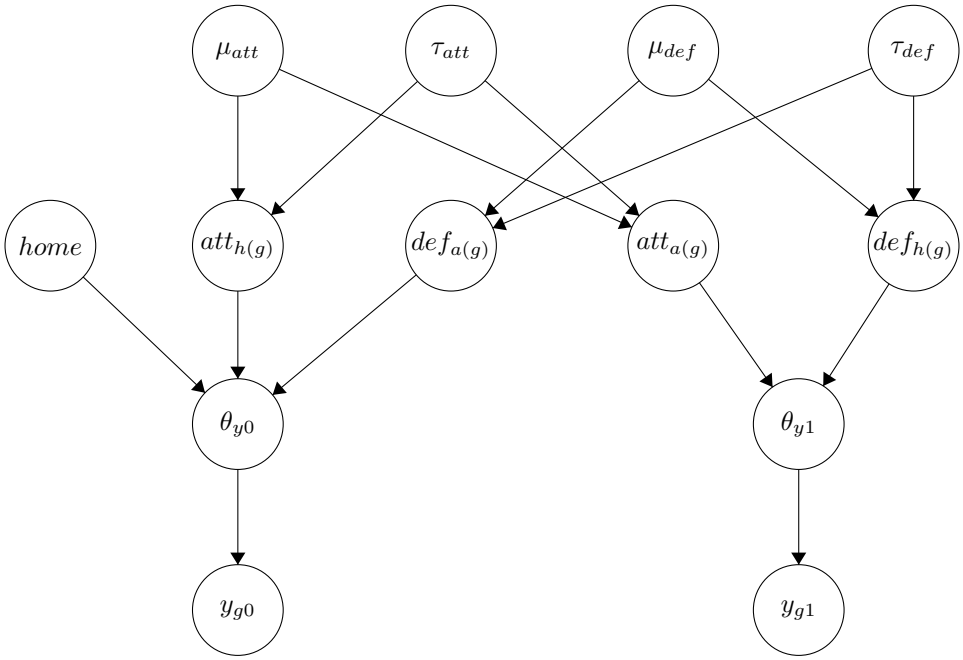


Figure 2: The DAG representation of the hierarchical Bayesian model. Figure adapted from [4].

The goals of each game are $\mathbf{y} = \{y_{gj} | g = 0, 1, ..., 379, j = 0, 1\}$ are the observed variables, and parameters $\boldsymbol{\theta} = \{home, att_0, def_0, ..., att_{19}, def_{19}\}$ and hyper-parameters $\boldsymbol{\eta} = (\mu_{att}, \mu_{def}, \tau_{att}, \tau_{def})$ are unobserved variables that we need to make inference on. To ensure identifiability, we enforce a corner constraint on the parameters (pinning one team's parameters to 0,0). Here we use the first team as reference and assign its attacking and defending strength to be 0:

$$att_0 = def_0 = 0 \tag{14}$$

In this question, we want to estimate the posterior mean of the attacking and defending strength for each team, i.e. $\mathbb{E}_{p(\boldsymbol{\theta},\boldsymbol{\eta}|\mathbf{y})}[att_i]$, $\mathbb{E}_{p(\boldsymbol{\theta},\boldsymbol{\eta}|\mathbf{y})}[def_i]$, and $\mathbb{E}_{p(\boldsymbol{\theta},\boldsymbol{\eta}|\mathbf{y})}[home]$.

1. **[4 points]** Find the joint likelihood $p(\mathbf{y}, \boldsymbol{\theta}, \boldsymbol{\eta})$.

2. **[4 points]** Write down the Metropolis-Hastings algorithm for sampling from posterior $p(\boldsymbol{\theta}, \boldsymbol{\eta}|\mathbf{y})$, and derive the acceptance function for a proposal distribution of your choice (e.g. isotropic Gaussian).

3. **[12 points]** Implement the Metropolis-Hastings algorithm to inference the posterior distribution. The data can be found from `https://sailinglab.github.io/pgm-spring-2019/assets/assignments/hw2/premier_league_2013_2014.dat`, which contains a $380 \times 4$ matrix. The first column is the number of goals $y_{g0}$ scored by the home team, the second column is the number of goals $y_{g1}$ scored by the away team, the third column is the index for the home team $h(g)$, and the fourth column is the index for the away team $a(g)$.

   - Use an isotropic Gaussian proposal distribution $\mathcal{N}(0, \sigma^2 I)$ and use 0.1 as the starting point.
   - Run the MCMC chain for 5000 steps to burn in and then collect 5000 samples with $t$ steps in between (i.e., run M-H for $5000t$ steps and collect only each $t$-th sample). This is called thinning, which reduces the autocorrelation of the MCMC samples introduced by the Markovian process. The parameter sets are $\sigma = 0.005, 0.05, 0.5$, and $t = 1, 5, 20, 50$.
   - Plot the trace plot of the burn in phase and the MCMC samples for the latent variable *home* using proposal distributions with different $\sigma$ and $t$.
   - Estimate the rejection ratio for each parameter setting, report your results in a table.
   - Comment on the results. Which parameter setting worked the best for the algorithm?
   - Use the results from the optimal parameter setting:
     (a) plot the posterior histogram of variable *home* from the MCMC samples.
     (b) plot the estimated attacking strength $\mathbb{E}_{p(\boldsymbol{\theta},\boldsymbol{\eta}|\mathbf{y})}[att_i]$ against the estimated defending strength $\mathbb{E}_{p(\boldsymbol{\theta},\boldsymbol{\eta}|\mathbf{y})}[def_i]$ for each the team in one scatter plot. Please make sure to identify the team index of each point on your scatter plot using the index to team mapping in Table 1.

4. **[0 points]** Despite what the data says, conclude that Manchester United is the best team in the Premier League!

**Note: You are free to use Python or MATLAB for your implementation. You are NOT allowed to use any existing implementations of Metropolis-Hastings in this problem. Please include all the required results (figures + tables) in your writeup PDF submission, as well as submit your code to Gradescope separately.**

# References


[1] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[2] Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*, 2011.

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[4] Gianluca Baio and Marta Blangiardo. Bayesian hierarchical model for the prediction of football results. *Journal of Applied Statistics*, 37(2):253–264, 2010.