Take-home Exam

Deadline: January 12, 2025, 23:59

This document contains the instructions for the take-home exam in the "Advanced Probabilistic Machine Learning" course (SSY316). The exam is divided into four parts, and the corresponding datasets for each section can be accessed on the course website.
Each part requires the submission of a project report, and all components of the exam must be completed in groups of two individuals, as per the previously registered group list.
The total exam score is 40 points, with 10 points allocated to each part.

P1. Clustering Using K-Means and MCMC Methods (10 points)

Description

In this problem, you will explore clustering techniques for a dataset generated from a Gaussian Mixture Model (GMM). You will implement **K-Means**, **Gibbs Sampling**, and **Metropolis-Hastings (MH)** to cluster the data and compare their performance. Additionally, you will evaluate the clustering results using the `Adjusted Rand Index (ARI)` function from `scikit-learn`.
The dataset will be generated using a predefined Gaussian Mixture Model. A Jupyter Notebook is provided on the course website to help you generate the dataset: P1.ipynb. Open the notebook, ensure that you can run it successfully, and visualize the generated data along with the true clusters. The dataset consists of 10 clusters with 1,000 data points per cluster (10,000 points in total).
Figure 1 shows an example visualization of the dataset.
Your tasks are:

1. Cluster the data using K-Means, Gibbs Sampling, and Metropolis-Hastings.
2. Compare the clustering results of the three methods using `ARI`.
3. Discuss the strengths and weaknesses of each approach.

Tasks

Q1. K-Means Clustering (2 points)

- Use the K-Means algorithm to cluster the data into $K = 10$ clusters.

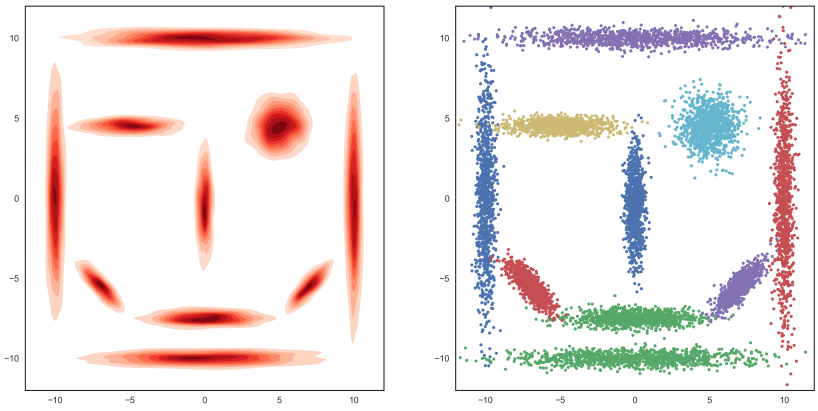

Figure 1: Example dataset generated from the specified GMM.

- Visualize the resulting clusters by coloring the data points according to their assigned cluster and marking the cluster centers.
- Report:
  - Cluster centers,
  - The `ARI` score comparing K-Means results to the true labels,
  - A brief analysis of the K-Means clustering results.

Q2. Gibbs Sampling for Clustering (3 points)

- Implement the Gibbs Sampling algorithm for clustering:
  - Assume z_i is the cluster assignment for data point x_i.
  - Iteratively sample z_i for each point x_i from its conditional distribution:

$$P(z_i = k \mid x_i, \text{rest}) \propto \pi_k \mathcal{N}(x_i; \mu_k, \Sigma_k).$$

  - Update the parameters π_k, μ_k, Σ_k based on the current cluster assignments.
- Use $n_{\text{burn-in}} = 200$ and collect 500 samples after burn-in.
- Visualize the final clustering results by coloring the data points according to their most probable cluster assignment.
- Report:
  - The `ARI` score comparing Gibbs Sampling results to the true labels,
  - A brief analysis of the Gibbs Sampling results.

Q3. Metropolis-Hastings for Clustering (4 points)

- Implement the Metropolis-Hastings algorithm for clustering:
  - Use a Gaussian proposal distribution:

$$q(z_i' \mid z_i) = \mathcal{N}(z_i'; z_i, \sigma^2),$$

    where σ^2 is the proposal variance.
  - Proposal variances to explore: $\sigma^2 = \{0.01, 0.1, 1, 10\}$.
  - Compute the acceptance ratio:

$$\alpha = \min\left(1, \frac{P(z_i' \mid x_i)}{P(z_i \mid x_i)}\right),$$

    where z_i is the current assignment and z_i' is the proposed assignment.
  - Accept or reject the proposal based on α.
- Use $n_{\text{burn-in}} = 200$ and collect 500 samples after burn-in.
- Visualize the final clustering results by coloring the data points according to their most probable cluster assignment.
- Report:
  - The `ARI` score comparing Metropolis-Hastings results to the true labels,
  - A brief analysis of the Metropolis-Hastings results.

Q4. Comparison and Analysis (1 points)

- Compare the results of the three clustering methods:
  - Provide a table summarizing:
    - `ARI` scores,
    - Computational efficiency (e.g., runtime),
    - Strengths and weaknesses of each method.
- Discuss:
  - Which method performed best and under what conditions,
  - Any challenges faced during implementation.

Deliverables

- **Report:** Clustering visualizations for K-Means, Gibbs Sampling, and Metropolis-Hastings.
- **Code:** Submit well-documented Python code for all three methods.
- **Analysis:** Compare the results and provide insights into the performance of the methods.

P2. Naive Bayes and Logistic Regression Classifiers for MNIST (10 points)

Description

In this problem, you will implement and analyze two classifiers for handwritten digit recognition on the MNIST dataset:

- **Naive Bayes Classifier**: Assumes that pixels are conditionally independent given the class label.
- **Logistic Regression Classifier**: Uses the entire flattened image as a feature vector to model the relationship between all pixels and the class label.

Both classifiers will be trained and evaluated on the MNIST dataset. By comparing their performances, you will explore the impact of using generative versus discriminative models in image classification.

Tasks

Preparation

Open the P2.ipynb notebook and make sure you can run the code to load the MNIST dataset and visualize some sample digits. This will help you understand the data format and structure.

Q1. Naive Bayes Classifier (4 points)

- Implement the Naive Bayes Classifier from scratch (you are not allowed to use any external machine learning packages for this task).
- Training:
  - Estimate the prior probabilities $P(y)$ for each digit class $y \in \{0, \ldots, 9\}$:

$$P(y) = \frac{\text{Count of images with label } y}{\text{Total number of images}}.$$

  - Estimate the conditional probabilities $P(x_i \mid y)$ for each pixel x_i using Laplace smoothing:

$$P(x_i \mid y) = \frac{n_{iy} + 1}{n_y + 2},$$

    where n_{iy} is the number of images of class y where pixel x_i is active (1), and n_y is the total number of images of class y.

- Prediction:

$$\hat{y} = \arg\max_y \left[\log P(y) + \sum_{i=1}^{784} \Big(x_i \log P(x_i \mid y) + (1 - x_i) \log(1 - P(x_i \mid y)) \Big) \right].$$

- Evaluate the classifier on the test dataset and compute its accuracy.

Q2. Logistic Regression Classifier (4 points)

- Train a multinomial logistic regression classifier using the entire flattened image as the feature vector (size 784).
- Use any machine learning package you are comfortable with for this task (e.g., `scikit-learn`).
- Example using `scikit-learn`:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=1000, multi_class='multinomial', solver='lb
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

- Evaluate the classifier on the test dataset and compute its accuracy.

Q3. Comparison and Analysis (2 points)

- Compare the accuracy of the Naive Bayes and Logistic Regression classifiers on the test set.
- Discuss the differences between the two classifiers:
  - How does the assumption of pixel independence in Naive Bayes affect its performance?
  - How does Logistic Regression leverage the entire feature vector to achieve better accuracy?
- Provide insights into when generative models like Naive Bayes may outperform discriminative models like Logistic Regression.

Deliverables

- **Code:** Submit your Python implementation of the Naive Bayes Classifier and the Logistic Regression Classifier.
- **Plots:**
  - Visualizations of sample images from the training dataset (from `Q2.ipynb`).
  - Visualizations of the learned conditional probabilities $P(x_i \mid y)$ for Naive Bayes.
- **Analysis:** Include answers to all discussion questions and a summary of your findings.

P3. Skill Estimation in Competitive Games with TrueSkill (10 points)

In this task, you will design and analyze a Bayesian model for estimating the skills of players based on match outcomes, using the TrueSkill ranking system developed by Microsoft Research. TrueSkill assigns a Gaussian-distributed skill to each player and updates these skills using Bayesian inference based on observed match results.
You will analyze match outcomes from the Italian 2018/2019 Serie A football league to explore how skill distributions evolve with data. The dataset, `SerieA.csv`, available on Canvas (SerieA.csv), contains the match results from this league. Additionally, you will extend the analysis by testing the method on a dataset of your choice.
The probabilistic model you will define represents all quantities as random variables. In this model:

- Each player's skill is a Gaussian random variable.
- When two players compete in a match, the outcome is modeled as a Gaussian random variable with a mean equal to the difference in the two players' skills.
- The result of the match is determined as follows:
- A result of 1 indicates the victory of Player 1 if the outcome is greater than zero.
- A result of -1 indicates the victory of Player 2 if the outcome is less than zero.

Your task is to use this model to estimate skills and analyze the performance of the ranking system based on the provided and selected datasets.

Tasks

Q1. Model Formulation (1 points)

1. Define the Bayesian model for a single match:
   - **Skills:** Represent the skills of two players as Gaussian random variables $s_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $s_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$.
   - **Outcome:** Model the outcome of the match as a Gaussian random variable $t \sim \mathcal{N}(s_1 - s_2, \sigma_t^2)$.
   - **Result:** Define a discrete variable y where $y = 1$ if $t > 0$ (Player 1 wins) and $y = -1$ if $t \leq 0$ (Player 2 wins).
2. Specify the joint distribution of all variables and identify the five hyperparameters in the model.

Q2. Bayesian Network and Factor Graph (1 points)

1. Draw the Bayesian network for the model.
2. Convert the network into a factor graph with:
   - Variable nodes for s_1, s_2, t, y.
   - Factor nodes for priors, skill differences, and the outcome likelihood.

Q3. Conditional Probabilities and Posterior (1 points)

1. Derive:
   - $p(s_1, s_2 \mid t, y)$: the posterior distribution of player skills given the outcome.
   - $p(t \mid s_1, s_2, y)$: the conditional distribution of the outcome (Truncated Gaussian).
   - $p(y = 1)$: the probability of Player 1 winning.
2. Implement a Python script to compute these distributions for a given set of hyperparameters and outcomes, and validate your derivations numerically.

Q4. Gibbs Sampling (1 points)

1. Implement a Gibbs sampler for $p(s_1, s_2 \mid y)$ to estimate player skills:
   - Plot the samples and identify an appropriate burn-in period.
   - Compare the histogram of samples with the Gaussian posterior approximation.
2. Report computational trade-offs:

- Evaluate accuracy vs. time using different numbers of samples post-burn-in.
- Justify the optimal number of samples.

Q5. Streaming Data with Assumed Density Filtering (ADF) (1 points)

Assumed Density Filtering (ADF) is an online Bayesian inference method where the posterior distribution of one match is used as the prior for the next match. This approach is particularly useful for processing a sequence of data in real-time or iterative updates.

1. Use ADF with Gibbs sampling to process the matches in the `SerieA.csv` dataset:
   - Skip matches that ended in draws and update the skill distributions after each non-draw match.
   - Rank teams based on their final skill estimates and interpret the variance of the skill distributions.
2. Shuffle the order of matches randomly and re-run ADF. Does the result change? Why or why not?

Q6. One-Step-Ahead Predictions (1 points)

One-step-ahead predictions involve using the model to predict the outcome of each match based on the skills updated from prior matches. This is done iteratively for the entire dataset.

1. Implement a prediction function to decide whether Player 1 or Player 2 will win:
   - Use the current skill distributions to compute the probability $p(y = 1)$.
   - Return $+1$ if Player 1 is predicted to win and -1 otherwise.
2. Compute one-step-ahead predictions for all matches in the `SerieA.csv` dataset:
   - Calculate the prediction rate:
   $$r = \frac{\text{Number of correct predictions}}{\text{Total number of matches}}$$
   - Compare your model's prediction rate to random guessing.

Q7. Message Passing Algorithm (1 points)

1. Implement a message-passing algorithm on the factor graph:
   - Use moment matching to approximate truncated Gaussians.
   - Compute the posterior skill distributions after one match.
2. Compare the message-passing results with Gibbs sampling:
   - Overlay the results (histograms and Gaussian approximations) in a single plot.

Q8. Custom Dataset (1 points)

1. Test your TrueSkill methods on a dataset of your choice. Possible datasets include:
   - Team sports (e.g., basketball, rugby).
   - Two-player sports (e.g., tennis, chess).
   - Online game data (e.g., multiplayer rankings).
2. Describe the source of your dataset and any preprocessing steps.
3. Report your findings, including a ranking of players or teams.

Q9. Model Extensions (2 points)

1. During your implementation of TrueSkill, you may have observed certain limitations in both the model and the inference algorithms. In this part of the project, you are tasked with defining and implementing your own extension of the model or method. Your extension should address a specific limitation or enhance the applicability of the model. Below are some suggestions:
   - Modify the model to account for draws or incorporate score differences in matches.
   - Introduce an a-priori advantage for certain players (e.g., white in chess).
   - Integrate external data or additional datasets to improve prediction accuracy.
   - Refine the representation of match outcomes by using detailed match scores or performance metrics.
   - Develop an improved prediction function that leverages additional insights from the data.

   Implement at least one extension to the model and test its performance. Analyze whether your extension results in:
   - Improved skill estimates.
   - Better prediction accuracy.
   - Reduced computational complexity or increased scalability.

   Discuss and justify your choice of extension, and provide insights on its impact.
2. Evaluate your extension on both the `SerieA.csv` dataset and your custom dataset:
   - Compare the results of your extended model with those from the baseline TrueSkill model.
   - Highlight any improvements in player rankings, prediction accuracy, or computational efficiency.
   - Provide a summary of your findings, supported by relevant plots or tables.

Deliverables

- **Report:** Include derivations, plots, and explanations for each question. Be concise but thorough.
- **Code:** Provide a well-documented Python implementation with a README.
- **Results:** Include player rankings, prediction rates, and insights from model extensions.

P4. QWOP (10 points)

Description

In 2010, the QWOP game gained popularity for its challenging gameplay. Your task is to write a computer program to control the QWOP avatar by simulating its thigh and knee angles to maximize the distance it runs in the 100-meter event at the Olympic Games.
The QWOP physics engine has been implemented in Python and is available on the course website (P4.ipynb). The provided function accepts a list of 40 floating-point numbers (corresponding to 20 instructions for the angle of the thighs and 20 for the angle of the knees) as input and outputs the final x-position of the avatar's head. Your goal is to find an input vector that maximizes this output.

Simulation Setup

Ensure that you can call the Python function `sim(plan)` with a `plan` of 40 floating-point numbers in the range $[-1, 1]$. Use the provided visualization tool to observe the avatar's movements and understand how the input angles affect its motion. Comment on any challenges you face in controlling the avatar.

Tasks

Q1. Optimization (5 points)

Optimize QWOP to maximize the distance d the avatar travels. Design your own optimization methods to achieve this goal; do not use pre-existing online optimization code. Experiment with at least two of the following techniques:

- Markov Chain Monte Carlo (MCMC),
- Simulated annealing,
- Variants of gradient-based optimization,
- Evolutionary algorithms,

- Any other methods of your choice.

Q2. Evaluation (5 points)

Evaluate the effectiveness of your methods:

- Report the best x-distance achieved by the avatar.
- Provide the length-40 input plan that resulted in the best performance.
- Compare the performance of the different methods you tried. Discuss their relative strengths and weaknesses.

Deliverables

- For Q1: Submit your Python code, clearly labeled with the optimization methods implemented.
- For Q2: Include a discussion of the best distance achieved, the length-40 input plan, and an analysis of the methods used.

Instructions for Writing Your Report and Submission

- Submit your solutions to all the questions in this document as a single PDF file. Your report should include:
  - Figures and plots for visualization.
  - Important derivations and results.
  - Design choices and explanations of your implementation.

  Bibliography, simulation code, and additional material can be added as an appendix.
- For each question, submit a single Python script or Jupyter notebook file named `Pi.py` or `Pi.ipynb`, where $i = 1, 2, 3, 4$. Ensure that each file corresponds to the relevant problem.
- Create a `requirements.txt` file listing all necessary Python packages required to execute your scripts.
- Place all your submission files (scripts, notebooks, and `requirements.txt`) in a zip file. Do not include the dataset in the zip file; assume the dataset is placed in the same directory as your script when executed.
- Ensure that your script or notebook can be executed without errors in a clean Python virtual environment (≤ 3.5). It should work with the following commands:

```
pip install -r requirements.txt
python Pi.py  # For Python scripts
jupyter notebook Pi.ipynb  # For Jupyter notebooks
```

- Your solution should be well-documented, with clear explanations and comments in the code.