TSPbayes

Jiakai Xu ax2155, Amy Qi xq2224

November 2023

Idea 1: Traveling Salesman Problem

Overview

The Traveling Salesman Problem (TSP) is a classic optimization problem in computer science and mathematics. In the TSP, a salesman is tasked with visiting a set of cities exactly once and returning to the starting city while minimizing the total distance traveled. This seemingly simple problem becomes computationally challenging as the number of cities increases, leading to an exponential growth in possible solutions.

Methodology

Current solution

The traditional sequential solution for TSP employs a brute-force approach. It first represents cities as coordinates and calculates the Euclidean distance between them. The total distance of a tour is computed by summing the distances between consecutive cities. The algorithm generates all possible tours through permutations and selects the tour with the minimum total distance. While conceptually simple, the brute-force method becomes computationally expensive as the number of cities increases, leading to exponential growth in the solution space.

Possible solution

Given that traditional algorithms simply try to compute the distance sum for various combinations of cities in sequence, and the computing order between combinations does not interfere with each other, taking full advantage of Haskell's parallelism allows us to compute the distance sum of different combinations of routes in parallel and obtain the smallest of all possible values at the end.
Furthermore, we can make use of Regular Parallel Arrays (Repa) in Haskell. In the TSP, the representation of tours and their distances can be naturally modeled as arrays, namely we just need a pair of coordinates to represent the location at a minimum. Operations on these arrays should be able to be paralleled automatically by Repa, exploiting data parallelism to perform computations concurrently on different parts of the array.

Challenges

1. **IO problem**. Although the algorithm requires a large number of repetitive computations for each piece of data, i.e., the coordinates of the cities, we inevitably have to read the coordinate information of all the cities from an external data file, which poses a potential IO problem. We disagree on whether the IO issue disqualify this idea for the course project and are unsure if the IO problem will significantly affect the paralleling effectiveness of our algorithm.

2. **Performance**. We have the concern that although we believe our parallel algorithm can be much faster than our chosen sequential algorithm, we must realize that such a classical problem has been optimized by many scholars from the algorithm level. We are not sure whether the parallelized computation based on a traditional algorithm can outperform other heavily optimized sequential algorithms. Or, maybe we do not need to worry about this problem, as long as we can make good implementation and parallelization of our chosen traditional sequential algorithm?

Idea 2: Naive Bayes Classifier

Overview

Another idea we have is a machine learning project. In this project, we plan to implement a naive Bayes classifier that classifies a set of synthesized data generated from a statistical distribution. In addition, we plan to implement feature selection based on cross validation.
Several pieces of background information are listed below.

1. **Classification**. Classification in machine learning involves assigning predefined categories or labels to unseen data based on patterns learned from labeled training data. The goal is to accurately predict the class or category to which new data belongs

2. **Naive Bayes classifier**. A Naive Bayes classifier is a probabilistic machine learning model based on Bayes' theorem. It calculates the probability of a data point belonging to a certain class based on the presence of certain features.

$$\hat{y} = \arg\max_{c \in C} P(C = c) \prod_{i=1}^{n} P(X_i = x_i | C = c) \tag{1}$$

   where:
   - $\hat{y}$ is the predicted class.
   - C represents the set of possible classes.
   - x_i are the feature values.
   - $P(C = c)$ is the prior probability of class c.
   - $P(X_i = x_i | C = c)$ is the conditional probability of feature i having value x_i given class c.
   - n is the number of features.

3. **Feature selection with cross validation**. Feature selection with cross-validation is about picking the most useful features from data. This is usually done with cross validation involves splitting data into multiple subsets, using each subset for both training and validation.

Methodology

In this project, we will focus on single-variate normal generative models, i.e., features are independent of each other. If we have extra time, we will explore bi-variate or multi-variate normal generative model where more complex computation is needed.
Several tasks we have identified are listed below.

1. **Generate data**. To avoid an IO-bound project, we plan to synthesize data based on a normal distribution. This approach aims to replicate the phenomenon observed in nature, where many occurrences also adhere to a normal distribution, like the petal length of Iris flowers.

2. **Feature selection**. Assuming independence for each feature in a single-variate model, we'll determine the most representative feature by individually assessing each of the features. Suppose there are 10 features, then this process involves 10 iterations. This process can be run in parallel.

3. **Train-test split**. We will conduct k-fold validation, where k represents a number between 1 and n, with n being the number of sample data points available. Assuming we choose k as 5, we'll divide the data in a 4:1 ratio, using 1 portion for validation in each of the 5 iterations. This process can be run in parallel.

4. **Calculate model parameters**. Using the training data, we estimate model parameters such as mean and variance for single-variate models or covariance matrices for more complex models. Then we construct the model and evaluate its performance using the validation data. The calculation of parameters can be parallelized.

5. **Model evaluation**. From the preceding steps, we've identified the optimal feature and model. With this information, we employ the test data, generated from the same distribution, to create a confusion matrix and assess the accuracy of our classification.

Challenge

Is this project interesting ☺