

Project 6: Lemonade

Due: 12:00 PM (Noon!), May 11, 2009

Contents

| | | |
|---|-------------------------|---|
| 1 | Introduction | 1 |
| 2 | Customer Demand Model | 2 |
| 3 | Overview and Evaluation | 4 |
| 4 | What to Do | 5 |
| 5 | Notes | 6 |
| 6 | Handing In | 7 |
| 7 | Support Code | 7 |
| 8 | Resources | 8 |



1 Introduction

The summer is coming, but in this economic climate, there are no jobs to be had. But you've got an entrepreneurial spirit. So you wonder, "how can I capitalize on another crisis, like global warming?" Eureka! You decide to open a lemonade stand.

The primary challenge in running a lemonade stand is to decide how many lemons to buy each day. The difficulty is that lemons are perishable. You cannot simply buy 100 lemons on the first day of summer, with a plan to restock when those run out. On the contrary, you need a fairly accurate model of customer demand, so that you can buy the right amount every day.

To get started, you spend some time gathering data about the lemonade market, and you create a parameterized, stochastic model of customer demand for lemonade. In this project, your task is to observe the actual lemonade demanded each day, and then to use this information to infer the underlying parameters of your model. You will then propagate that information forward to make predictions about future customer demand.

In the abstract, imagine you are sampling points from a known distribution (e.g., a Gaussian) with *unknown* parameter values (e.g., the mean and the standard deviation). If you had enough samples, you would eventually be able to infer the values of the unknown parameters. Now imagine that the distribution is changing over time in some parameterizable fashion (again with unknown parameter values) so that each observed point is sampled from a slightly different distribution. In this project, your goal is to infer an underlying distribution from sample measurements, and further to predict the way in which the distribution will change over time.

2 Customer Demand Model

At a high level, your customer demand model operates as follows: each day, actual customer demand, N_i , is unknown, but is drawn from a known distribution over customer demand. This known distribution is parameterized by a variable, Q_i , whose value is also unknown. From day to day, this so-called **expected demand** parameter changes in a well-defined fashion. If your lemonade is popular on a given day, expected demand on the following day will be scaled up by some amount; on the other hand, if your lemonade is unpopular, expected demand on the following day will be scaled down by a corresponding amount. Changes in popularity are modeled by a **trend** parameter, which also changes in a well-defined fashion, based only on its past value. A Bayesian network depicting this customer demand model (for four days) appears in Figure 1.

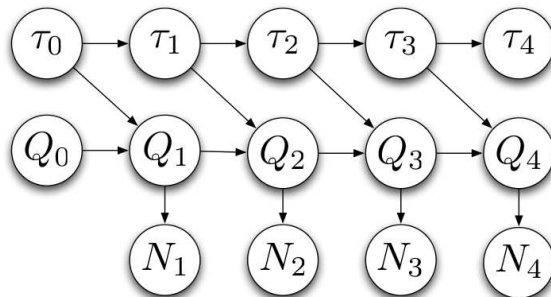


Figure 1: A Bayes net modeling four days of customer demand for lemonade. Today’s expected demand Q_i depends on both yesterday’s expected demand Q_{i-1} and yesterday’s popularity τ_{i-1} .

At the start of the summer season, the initial expected demand for lemonade, Q_0 , is relatively unpredictable. You simply assume it will fall uniformly in the interval $[Q_{min}, Q_{max}]$. After that, the expected quantity of lemonade demanded today, Q_i , depends on yesterday’s expected demand, Q_{i-1} , and on yesterday’s popularity, as expressed by a trend parameter τ_{i-1} . Formally,

$$Q_{i+1} = \min(Q_{max}, \max(Q_{min}, \tau_i Q_i)) \quad (1)$$

ht

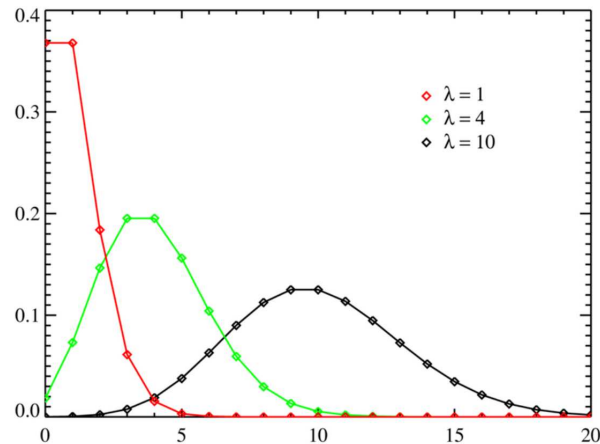


Figure 2: Three Poisson distributions with different λ parameters. Note that $\lambda = Q_i$ defines the distribution from which N_i is sampled. (Image source: Wikipedia.)

This equation reflects the fact that expected demand never falls below Q_{min} or above Q_{max} . Furthermore, if the trend parameter τ_i is greater than 1.0, then your customers are telling their friends about your delicious lemonade, and more people are expected to demand it on the following day. Similarly, if the trend parameter is less than 1.0, then your customers are telling their friends about your not-so-delicious lemonade, and fewer people are expected to demand it on the following day.

For simplicity, you assume that the popularity of your lemonade is initialized to 1.0. Each day, its popularity follows a random walk so that the today's popularity is the same as yesterday's popularity plus some random amount uniformly drawn from the interval $[\Delta\tau_{min}, \Delta\tau_{max}]$. Additionally, due to people's inherent passion and lack thereof for lemonade, there is a limit to how popular or unpopular your lemonade can become; its popularity cannot fall below τ_{min} or above τ_{max} . What we have just described is formally encoded in the following equation, which is used to update the trend parameter that governs your lemonade's popularity:

$$\tau_0 = 1 \tag{2}$$

$$\tau_{d+1} = \max(\tau_{min}, \min(\tau_{max}, \tau_d + \text{uniform}(\Delta\tau_{min}, \Delta\tau_{max}))) \tag{3}$$

Finally, the actual quantity of lemonade demanded on day i , namely N_i , is a function of the expected demand on day i , namely Q_i , as follows:

$$N_i \sim \text{poisson}(Q_i) \tag{4}$$

In words, actual demand N_i is sampled from the Poisson distribution defined by expected demand parameter $\lambda = Q_i$. Note that support code for sampling from a Poisson distribution is provided for you, but you can see Figure 2 for depictions of the distribution with different parameter values.

3 Overview and Evaluation

An outline of your set of tasks is as follows: on each day, you observe some amount of actual demand for lemonade, N_i . From this observed actual demand, your first task is to infer yesterday's underlying expected demand Q_{i-1} and popularity τ_{i-1} parameters. We call the pair (Q_i, τ_i) the underlying **demand state** on day i . Once you have inferred a distribution over possible underlying demand states for day $i - 1$, your next task is to project your beliefs forward using the expected demand and trend update equations (1 and 3) to generate distributions over future demand states $(Q_i, \tau_i), (Q_{i+1}, \tau_{i+1}), \dots$ (including today). Finally, you will compress the information contained in these demand state distributions into single point predictions (e.g., the means of these distributions) of customer demand for future days.

Your predictions will be plotted automatically by an evaluation class that has been built for you. After you run the main method in this class, a graph will appear like the one in Figure 3, which shows, for each day i , the *observed* actual customer demand N_i and the *true* expected demand Q_i . (Here, we are assuming that your model of customer demand is accurate.) In addition, every fifth day d , there is a line segment that depicts your mean predictions for parameters $Q_d, Q_{d+1}, Q_{d+2}, \dots$ up to some prediction horizon. You should observe that your initial predictions are not very good, but that they improve with time. If you run the main method in the evaluation class without writing any code first, all of your predictions will be 0; but after doing this project, you should find them to be much more accurate.

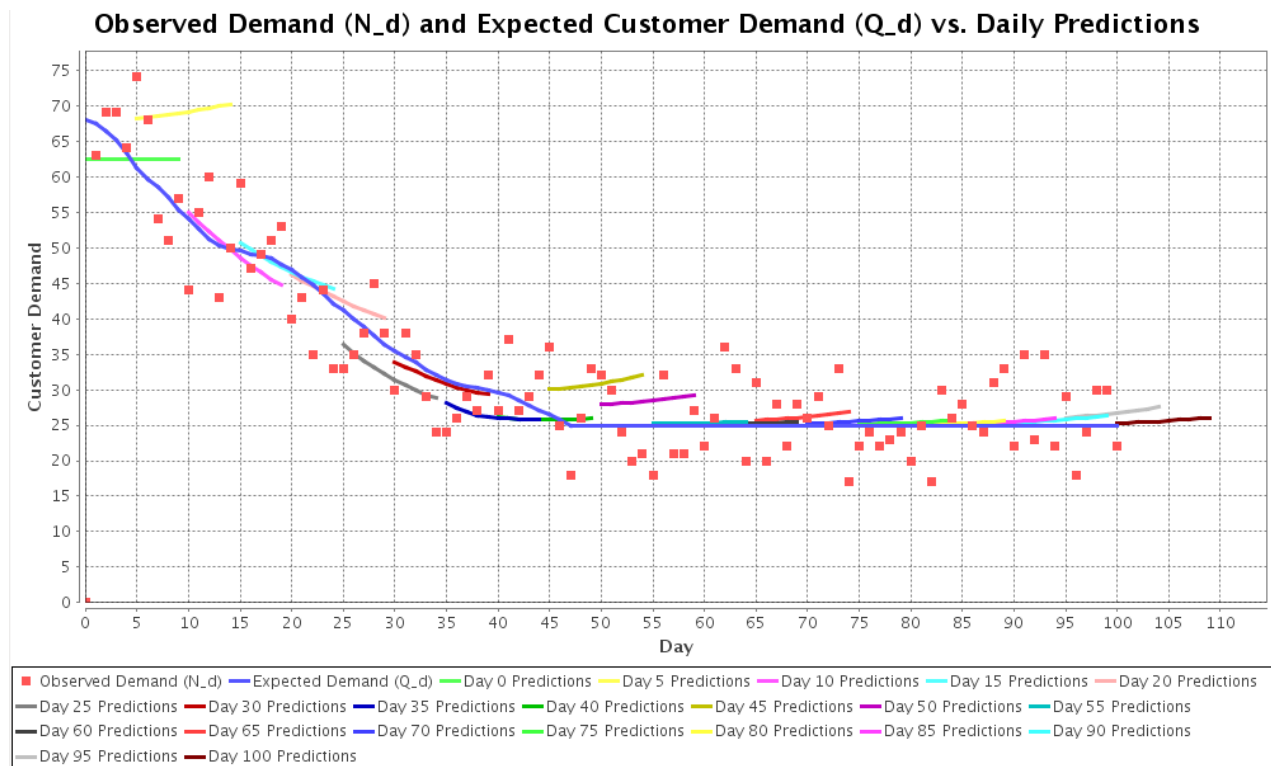


Figure 3: An example of the results you might see after you're done with the project. This plot is generated automatically when you run the main method in `Evaluate.java`.

4 What to Do

Note: Each of the following tasks will be graded independently. If you get stuck on one task, insert placeholder data and move on to the next.

1. For the first day of summer, you need to have some prior model of your lemonade's popularity τ_0 and expected demand Q_0 . Create a joint probability matrix that provides you with this information, based on the specification of the customer demand model in this writeup (from the specification you know that the initial popularity $\tau_0=1$, and the initial expected demand Q_0 is an integer between Q_{min} and Q_{max}). Implement this in the following method:

```
createInitialDemandStateDistribution(...)
```

2. As summer progresses, you observe actual customer demand from day to day. Based on this observed actual customer demand, you can infer the underlying demand state of your model. Write code that takes as input observed customer demand N_i , and updates the probability of each demand state (τ_{i-1}, Q_{i-1}) on day $i - 1$.

Hint: Use Bayes' rule.

Implement this task in the following method:

```
updateDemandStateDistribution(...)
```

A graphical representation of Task 2 can be seen in Figure 4.

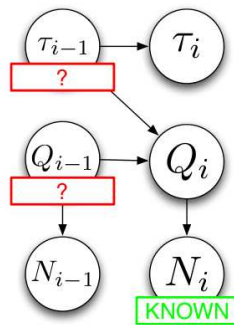


Figure 4: Graphical representation of Task 2.

3. You now have a way to update your demand state probabilities based on observed actual customer demand. However, it does you little good *today* to know a probability distribution over *yesterday's* underlying demand state. Your next task is to propagate this information forward any number of days into the future, so that you can compute the probability of being in some demand state on some future day.

Implement this task in the following method:

```
propagateDemandStateDistribution(...)
```

A graphical representation of Task 3 can be seen in Figure 5.

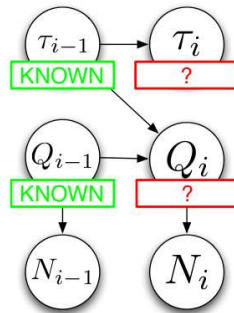


Figure 5: Graphical representation of Task 3.

4. Your next and last task is to compress the distributional information you computed in the previous task into a single point prediction. You should write code to do this in two ways:

- (a) For some future day $i + j$, compute the expected value (i.e., the mean) of the expected demand for lemonade, using your future demand state distributions: i.e., $\mathbb{E}[Q_{i+j}]$.

Implement this task in the following method:

```
computeMeanDemand(...)
```

There are actually two methods called `computeMeanDemand` that you should fill in: the first takes as input a future day, and the second takes as input a distribution over possible demand states. The second of these methods is simply a helper class for the first. We have you write this helper method because we want to be able to test the accuracy of your predicted expected demand regardless of the accuracy of the other methods you've implemented for this project.

- (b) For some future day $i + j$ and some percentile p , output the lowest level of expected demand for which you are $p\%$ certain expected demand will be at least that high.

Implement this task in the following method:

```
computePercentileDemand(...)
```

Once again, there are two methods called `computePercentileDemand` that you should fill in: the first takes a future day as input, and the second takes as input a distribution over possible demand states.

5 Notes

In order to obtain a finite encoding of the demand state space, both of the continuous variables, Q_i and τ_i , instead take on discrete values. The discretization factor for each variable is configurable, and you are welcome to experiment with the tradeoff between computational efficiency and accuracy. You will find the evaluation class useful if you undertake this kind of experimentation.

6 Handing In

Please hand in the following:

1. All the code necessary to solve the assignment (including our support code).
2. A README file that includes the following:
 - (a) Standard bug reports.
 - (b) Anything that might help us read your code: design quirks, conventions, etc.
 - (c) Comments that might help us improve and/or extend this project in the future.

Hand in by typing `cs141_handin lemonade` in the shell from the directory containing your work.

7 Support Code

Install the support code by running the install script:

```
/course/cs141/bin/cs141install lemonade
```

The support code will be installed into your `/course/cs141/projects/` directory.

Within the support code, there are three primary classes, only one of which you need to modify:

- `MyDemandPredictor.java` contains all the methods listed in the *What to Do* section, which you must implement (these methods initially do nothing).

The data structure in this class that you will be manipulating is `double[][] demandStateDistribution`. This matrix is meant to hold the joint probability of being in a particular demand state on the most recently observed day. Recall that a demand state is a (τ_i, Q_i) pair. The first dimension of `demandStateDistribution` is indexed by popularity τ_i , and the second dimension is indexed by expected demand Q_i . To get the actual τ_i and Q_i values that these indices represent, you can look them up in the `double[] possibleTrendValues` and `double[] possibleExpectedDemandValues` arrays, respectively, which are created for you based on the discretization factor.

There are also some helper methods in this class, most notably the `poisson()` method. This method takes in a λ and k parameter as input, and returns the probability that a Poisson distribution with λ expected occurrences will sample value k from the distribution.

- `DemandActual.java` is used to simulate actual customer demand. It randomly generates new popularity values τ_i , expected demand values Q_i , and actual demand values N_i , in the manner specified in this writeup. Although your predictor can only access to the N_i values up through day i on day i , the other values are provided for evaluation purposes. You do not have to edit the contents of this class.

- `Evaluator.java` is used for testing your customer demand predictions. It evaluates the performance of your predictor by comparing your predictions to the true expected demand Q_i and popularity τ_i values specified by `DemandActual.java`.

To test your predictor, simply run the `main` method found in this class.

If you'd like to change parameter values (such as the minimum and maximum Q_i values) to ensure that your predictor is flexible, you can do so the `main` method. Modulo any parameter values you'd like to change for testing, you do not have to edit the contents of this class.

There are also two JAR files that must be included, located in `lemonade/lib/`. They are:

1. `jfreechart-1.0.13.jar`
2. `jcommon-1.0.16.jar`

These JARs are used by the `Evaluator` to plot your predictions. Eclipse should automatically add these JARs, but if you have trouble, you can find help by visiting the following URL:
[http://www.wikihow.com/Add-JARs-to-Project-Build-Paths-in-Eclipse-\(Java\)](http://www.wikihow.com/Add-JARs-to-Project-Build-Paths-in-Eclipse-(Java))

8 Resources

1. Bayes' theorem on Wikipedia:
http://en.wikipedia.org/wiki/Bayes'_theorem
2. Poisson distribution on Wikipedia:
http://en.wikipedia.org/wiki/Poisson_distribution