6.867 Machine Learning

Problem set 1 — Solutions

Thursday, September 19

What and how to turn in?

Turn in short written answers to the questions explicitly stated, and when requested to explain or prove. Do **not** turn in answers when requested to "think", "consider", "try" or "experiment" (except when specifically instructed). When answering question of the form *What is X* or *Write down X* show how exactly you arrived at the answer, without missing crucial steps in reasoning. You may turn in answers to questions marked "optional"— they will be read and corrected, but a grade will not be recorded for them.

Turn in all MATLAB code explicitly requested, or that you used to calculate requested values. It should be clear exactly what command was used to get the answer to each question. Do **not** turn in code supplied to you with the problem set.

To help the graders (including yourself...), please be neat, answer the questions briefly, and in the order they are stated. Staple each "Problem" separately, and be sure to write your name on the top of every page.

Please address questions and comments to `6.867-staff@ai.mit.edu`.

Problem 1: regression

Reference: Lecture two; Chapter five.

Here we will be using a regression method to predict mileage per gallon (MPG) of gas for different vehicles. You'll find the data in the file "mpg.dat". Information about the data, including the column interpretation can be found in the file "mpg.names". These files, like many other data files in the course, are taken from the UCI Machine Learning Repository `http://www.ics.uci.edu/~mlearn/MLSummary.html`.

First, we will use a linear regression model to predict the MPG values, using squared-error as the criterion to minimize. In other words $y = f(\mathbf{x}; \hat{\mathbf{w}}) = \hat{w}_0 + \sum_{i=1}^{7} \hat{w}_i x_i$, where $\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{t=1}^{n} (\mathbf{y_t} - f(\mathbf{x_t}; \mathbf{w}))^2$; here y_t is the MPG of a car t, x_t is the

7-component input vector representing the car t, and n is the number of training examples.

To allow experiments with linear regression, the following Matlab functions have been provided.

```
w = train(trainX,trainY)
```

given the set of training vectors $\mathbf{x}_t$, $t = 1, \ldots, n$ (rows of `trainX`) and corresponding output values y_t, $t = 1, \ldots, n$ (elements of `trainY`), returns in `w` the "optimal" weight vector $\hat{\mathbf{w}}$.

```
Y = predict(X,w)
```

given a set of input vectors (rows of `X`) and a weight vector $\hat{\mathbf{w}}$, returns in elements of `Y` output values $\hat{y}$ predicted for `X` using $\hat{\mathbf{w}}$.

```
[trainE, testE] = trainNtest(trainX,trainY,testX,testY)
```

given the input and output values in a training set and a test set, trains and evaluates linear regression model, and returns in `trainE` and `testE` the training and test errors, respectively.

```
[trainE,testE] = testLinear(x,y,numtrain)
```

takes as input the complete data set (both input vectors and output values), and the desired training set size, and returns the mean training and test errors.

To test our linear regression model, we will partition the available data into disjoint training and test sets. This will be done, for a training set size N, by using the first N lines in the data file for training, and the remaining lines for testing.

The easiest way to read in the data in Matlab, and

```
data = load('mpg.dat');
x = data(:,1:7);
y = data(:,8);
```

The training and test errors of linear regression are plotted in Figure 1 as a function of an increasing size of the training set. Make sure you get the same values.

1. [**8pt**] How would you expect the graphs on Figure 1 to behave with an unlimited supply of examples? In this case we estimate the regression model with only a small training set as before but the size of the test set would be infinite. In your answer refer to the values of the training and test errors as well as to their behavior with respect to each other.

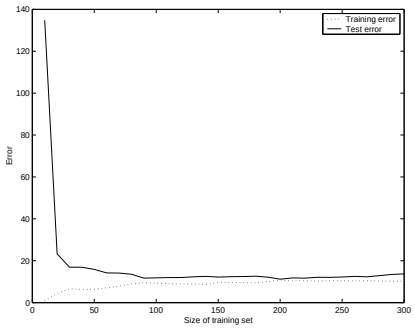

Figure 1: Training / test error of linear regression as a function of training set size

Answer: Note that both the training error and the test error can be viewed as estimates of the true generalization error. Both are consistent estimates; as the size of the data increases, their value approaches generalization error. In this case, the test error will approach that limit. Its value is likely to be somewhat larger than the training error, since the training error on data set of fixed (small) size is "optimistic" - the model parameters are optimized to minimize the training error. If the gap between the test error and the training error is large we say that the learning algorithm is overfitting.

2. [**5pt**] In all but very unlikely scenarios the value of the training error on a training set of size 5 in MPG should not depend on the actual data. Explain why, and write down that value. Do you get this answer by running `testLinear` on the training set of size 5? If no, explain why.

Answer: The linear regression model for MPG has 8 parameters $w_0, \ldots, w_7$. Fitting the model to 5 data points can be expressed as the system of 5 linear equations and 8 unknowns:

$$\begin{cases} y1 = w_0 + w_1 x_{1,1} + \ldots + w_7 x_{1,7} \\ \vdots \\ y5 = w_0 + w_1 x_{5,1} + \ldots + w_7 x_{5,7} \end{cases}$$

where $x_{i,j}$ denotes the j-th component of the i-th input vector. This system will usually have an exact solution - in fact, if the points are in general position

(not collinear), there will be an infinite number of solutions. The training error corresponding to such a solution is zero.

The "unlikely scenario" to which we refer in the question is the scenario in which the MPG measurements taken for two (or more) cars with identical parameters produce different values. In that case, the system above would not be consistent, and parameters of the linear regression model would result in a non-zero training error. As we get more training points the system of equations would typically be "inconsistent" (more equations than unknowns).

In practice, finding the parameters involves a number of calculation steps, such as finding a matrix pseudo-inverse, that inevitably introduce numerical error (due to finite precision). Therefore, our Matlab program reports a very small, but non-zero training error; we got an error of the order $1o^{-28}$.

We will now move on to polynomial regression. We will predict the MPG values using a function of the form:

$$f(\mathbf{x};\mathbf{w}) = w_0 + \sum_{i=1}^{7}\sum_{d=1}^{m} w_{i,d}x_i^d,$$

where again, the weights $\mathbf{w}$ are chosen so as to minimize the mean squared error on the training set. Think about why we also include all lower order polynomial terms up to the highest order rather than just the highest ones [do not turn in an answer].

Note that we only use features which are powers of a single input feature. We do so mostly in order to simplify the problem. In most cases, it is more beneficial to use features which are products of different input features, and perhaps also their powers. Think of why such features are usually more powerful [do not turn in an answer].

You can use the provided Matlab function

```
xx = degexpand(x,m)
```

that for given set of input vectors `x` and a degree `m` returns the set of expanded vectors (containing all the powers of the original inputs).

You will have to slightly modify `testLinear` in order to experiment with polynomial regression. You do not have to turn in the modified code.

NOTE: When the degree is high, some of the features will have extremely high values, while others will have very low values. This causes severe numeric precision problems with matrix inversion, and yields wrong answers. To overcome this problem, one must appropriately scale each feature x_i^d included in the regression model, to bring all features to roughly the same magnitude. Our version of `degexpand`, for instance, normalizes each feature by the maximum absolute value of that feature.

Of course, this scaling of features can be safely done only if it does not change the regression predictions. That is, given training feature vectors and output values

$\{\mathbf{x}_t, y_t\}_{t=1,\ldots,n}$ and a test input vector $\mathbf{x}_{\text{test}}$, and scaling factors $\{\alpha_i\}_{i=1,\ldots,7}$, we would like to make sure that the prediction of the test output value would be the same if we trained a linear regression on $\{\tilde{\mathbf{x}}_t, y_t\}_{t=1,\ldots,n}$, where $\tilde{x}_{t,i} = \alpha_i x_{t,i}$, and predicted on $\tilde{\mathbf{x}}_{\text{test}}$, where $\tilde{x}_{\text{test},i} = \alpha_i x_{\text{test},i}$. In fact, it is enough to prove this for the linear model (maximum degree one). The result extends to scaling each "extended" feature x_i^d (which is what we actually do), since this is just linear regression using these "extended" features. Now, let's assume $\hat{w}_i$ is the "optimal" weight corresponding to the i^{th} feature when we use the original non-scaled inputs. Then, $\tilde{w}_i = \hat{w}_i/\alpha_i$ gives the "optimal" weights for the scaled features since $\sum_{i=1}^{7} \tilde{w}_i \tilde{x}_i = \sum_{i=1}^{7} \hat{w}_i x_i$. In other words, we make the same predictions in both cases. More generally, since $\tilde{\mathbf{x}}$ is an (invertible) linear function of $\mathbf{x}$, the set A of linear function of $\mathbf{x}$ is exactly equal to the set $\tilde{A}$ of linear functions of $\tilde{\mathbf{x}}$. Therefore, the optimal predictor for the original problem, $\text{argmin}_{w \in A}$(training error) is exactly the same as the optimal predictor $\text{argmin}_{w \in \tilde{A}}$(training error) for the scaled problem.

Advice: In order to understand the expansion that `degexpand`*, apply it on a small data set, for example, on*

```
x=[1,   2,   3;
   2,   0.5, 1.5;
   0,   1,   2];
```

and make sure you understand the results of `degexpnad(x)`*.*

3. [**7pt**] For a training set size of 300, turn in the mean squared training and test errors for maximal degrees of zero through eight.

   "Sanity check": for maximal degree two, you should get a training error of 6.8164 and a test error of 9.4656).

   Answer:

   The training and test errors we got are listed in the table below:

Degree	Training error	Test error
0	63.887201	50.599153
1	10.222106	13.702433
2	6.816417	9.465568
3	6.383646	**7.042212**
4	6.200593	11.294545
5	6.055586	105.603196
6	5.973245	13714127.434883
7	5.598525	418342920955.738220
8	5.464701	254846135855.677948

4. [**15pt**] Explain the qualitative behavior of the test error as a function of the polynomial degree. Which degree seems to be the best choice?

Answer: The higher the maximal degree in the polynomial regression model, the better it can fit complex patterns in the data. The model classes indexed by the maximal degree are also nested in the sense that the models with higher maximal degree include those with lower maximal degree as special cases (we can simply set the higher order coefficients to zero). This means that by increasing the maximal degree, we never loose any ability to fit the data, only increase it. This accounts for the monotonically decreasing value of the training error as we increase the degree. Initially, this also leads to an improved generalization performance (as measured by the test error) - the model fits better the true underlying distribution of the problem. However, the more complex model also has more parameters, and consequently requires more data points to adequately estimate those parameters. Without providing additional data, we are bound to start overfitting the complex model to the data - that is, instead of fitting the true behavior, the model fits "artifacts" that are present in the specific data set but are not typical for the data in general. As a result, the generalization error (and its estimate, the test error) start deteriorating.

The best choice of a degree here is to pick the degree with minimal test error, which is 3. From our experiments, it appears that the model corresponding to maximal degree 3 complex enough to fit the data well (on the test set, better than other models), but not "too complex" so it does not grossly overfit the training set.

Although this experiment suggest 3 as the best choice, in practice a more careful evaluation would be required. What we really want to minimize is the generalization error, of which test error is an estimate; this estimate, though unbiased, is a random variable, and has nonzero variance. It is conceivable that with a different test set, another model say of degree 2, would show better test performance. A typical methodology in practice consists of training different models on the same training set, choosing one based on a separate validation set *(the role played in our experiments by the "test set") and finally evaluating the chosen model on a separate test set (the last step is missing in our experiments).*

5. [**15pt**] In real life, the measured values of the target function (MPG in this case) will be affected by noise. That is, if the "true" value of the function is $y_{true} = f(\mathbf{x}, \mathbf{w}_{true})$, then the value we record (and the one included in the data set) is $y = y_{true} + \epsilon$, where ϵ is a random noise with distribution $p(\epsilon)$. The weights $\hat{\mathbf{w}}$, found with linear regression from $\mathbf{x}$ and y, are an estimate of the true weights $\mathbf{w}_{true}$. We would like to find how *biased* is this estimate (more about the bias of an estimator in the Problem 2). Assuming $\mathbf{E}_{p(\epsilon)}[\epsilon] = 0$, what is the bias $\mathbf{E}_{p(\epsilon)}[\hat{\mathbf{w}} - \mathbf{w}_{true}]$?

Answer: First, our assumption about the underlying noiseless model here can

be written as $\mathbf{y}_{true} = \mathbf{X}\mathbf{w}_{true}$. Multiplying both sides by $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T$, we get

$$\mathbf{w}_{true} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}_{true}.$$

Now,

$$\begin{aligned}
\mathbf{E}_{p(\epsilon)}\left[\hat{\mathbf{w}}\right] =& \mathbf{E}_{p(\epsilon)}\left[\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}\right] \\
=& \mathbf{E}_{p(\epsilon)}\left[\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T(\mathbf{y}_{true} + \epsilon)\right] \\
=& \mathbf{E}_{p(\epsilon)}\left[\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}_{true}\right] + \mathbf{E}_{p(\epsilon)}\left[\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\epsilon\right] \\
=& \mathbf{w}_{true} + \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{E}_{p(\epsilon)}\left[\epsilon\right] = \mathbf{w}_{true}.
\end{aligned}$$

That is, as was stated in the lecture, $\hat{\mathbf{w}}$ is unbiased.

Problem 2: estimation

Reference: Lecture 1-2; Recitations; Chapter 4.

The distribution of random variables we deal with often belongs to a particular *parametric family* , $P(x\ ;\ \theta)$. For instance, the number h of heads in a series of n coin tosses has binomial distribution $P(h;p) = \binom{n}{h}p^h(1-p)^h$ with parameter p, the probability of a head in a single toss. A Gaussian univariate distribution has two parameters - the mean μ and the variance σ^2. The *likelihood* $L(x;\theta)$ of the data x given the value of the parameter θ is the value of $P(x|\theta)$; the different terminology and notation here is used to emphasize that we are concerned with the function of θ, having observed the data, rather than predicting the value of the data.

In this problem, we will derive estimators for the parameters of the *multinomial distribution*. Consider a sequence of n independent random trials, each having one of N possible outcomes $1,\ldots,N$. In each trial, the outcome k occurs with probability π_k. Of course, since one of $1,\ldots,N$ necessarily occurs, $\sum_k \pi_k = 1$. The random vector $X_1,\ldots,X_N$, where X_k is the count of times k was the outcome of a trial, has multinomial distribution

$$P(X_1 = x_1,\ldots,X_N = x_N) = \frac{n!}{\prod_{k=1}^N x_k!}\prod_{j=1}^N \pi_j^{x_j}.$$

An example of such distribution, for the case of $N = 2$, is the perhaps more familiar binomial distribution - for instance, the counts of heads and tails in n coin tosses. The distribution of the counts of 1 through 6 in a series of n rolls of a die is multinomial as well. Remember that the values drawn from the multinomial distribution are the **counts** of outcomes $1,\ldots,N$ in n trials, and not the outcomes themselves.

Maximum Likelihood Estimation We'd like to find the parameter setting $\hat{\theta}$ that best "explains the data". More precisely, we need a measure of how well the distribution $p(\cdot|\theta)$ with different values of θ fits the observed data x. Since $p(x|\theta)$ is the probability of reproducing the data x as samples from the distribution, it seems natural to try to maximize this probability. This gives the *Maximum Likelihood* estimation criterion for θ:

$$\hat{\theta}_{ML} = \operatorname{argmax}_\theta L(x;\theta).$$

Note that likelihood of a set of independently identically distributed (i.i.d.) random variables is the product of their likelihoods (by definition of independence): $L(x_1,\ldots,x_n;\theta) = \prod_{i=1}^n L(x_i;\theta)$. This expression can be quite complicated, and make maximization technically challenging. However, any monotonically increasing function of the likelihood will have the same maxima. One such function is *log-likelihood* $\log L(x;\theta)$; taking the log turns the product into a sum, making derivatives significantly simpler. We will follow this practice of maximizing the log-likelihood instead of likelihood.

1. [**20pt**] Write down the log-likelihood of a set of observed values from multinomial distribution, and derive the ML estimator for the parameters $\pi_i,\ i = 1, \ldots, N$.

Advice: Note that the effective number of parameters ("degrees of freedom") of this distribution is $N-1$ rather than N (make sure you understand why).

Answer: We start with writing down the log-likelihood of a set of observations $X = (x_1, \ldots, x_n)$, given the multinomial parameter vector $\pi = [\pi_1, \ldots, \pi_N]$:

$$\log L(X\ ;\ \pi) = \log n! - \log \prod_{k=1}^{N} x_k! + \sum_{j=1}^{N} x_j \log \pi_j$$

Note that also we refer to N parameters in the above, there are only $N-1$ degrees of freedom, due to the constraint $\sum_k \pi_k = 1$; we can express one of the parameters in terms of the other, say $\pi_N = 1 - \sum_{k=1}^{N-1} \pi_k$.

Now, we are interested in maximizing log-likelihood as a function of π, so we can ignore the terms independent of π:

$$\begin{aligned}\hat{\pi}_{ML} =& \operatorname*{argmax}_{\pi} \log L(X\ ;\ \pi) = \operatorname*{argmax}_{\pi} \sum_{j=1}^{N} x_j \log \pi_j \\ =& \arg\max_{\pi_1, \pi_{N-1}} \sum_{j=1}^{N-1} x_j \log \pi_j + x_N \log \left(1 - \sum_{i=1}^{N-1} \pi_i\right),\end{aligned}$$

i.e. $\hat{\pi}_{ML}$ is the solution of the following system of equations:

$$\begin{cases} \frac{\partial}{\partial \pi_1} \log L(X\ ; \pi) = \frac{x_1}{\pi_1} - \frac{x_N}{1 - \sum_{i=1}^{N-1} \pi_i} = \frac{x_1}{\pi_1} - \frac{x_N}{\pi_N} = 0 \\ \vdots \\ \frac{\partial}{\partial \pi_{N-1}} \log L(X\ ; \pi) = \frac{x_{N-1}}{\pi_{N-1}} - \frac{x_N}{1 - \sum_{i=1}^{N-1} \pi_i} = \frac{x_{N-1}}{\pi_{N-1}} - \frac{x_N}{\pi_N} = 0 \end{cases}$$

From this we can infer that

$$\frac{x_i}{\hat{\pi}_i} = c \quad \text{(constant)}$$

or, alternatively, $\hat{\pi}_i = x_i / c$, for all $i = 1, \ldots, N$. Since $\hat{\pi}_i$ has to sum to one, we see that c has to be

$$c = \sum_{j=1}^{N} x_j$$

This finally gives the result

$$\hat{\pi}_k = x_k / \sum_{j=1}^{N} x_j.$$

That is, the ML estimate of π_k is the observed frequency x_k/n.

2. [**5pt**] Let $\hat{\theta}(X_1, \ldots, X_n)$ be an estimator based on data $X_1, \ldots, X_n$, where the data are viewed as random samples from $p(\cdot|\theta)$. We can try to assess how well the estimator can recover the parameters θ. One useful measure is the *bias* of the estimator. This is defined as the expectation $\mathbf{E}\left[\hat{\theta}(X_1, \ldots, X_n) - \theta\right]$, taken with respect to the true distribution of the data $X_1, \ldots, X_n$, which in this case is $p(\cdot|\theta)$. The bias measures whether the estimator systematically deviates from the true parameters θ that were used to generate the data. An estimator is called *unbiased* if its bias is zero. Show that the ML estimator $\hat{\pi_{kML}}$ is indeed unbiased in this sense.

   Answer: To show this, we need to find the expected value of the ML estimator, with respect to the distribution of the observations. Let o_k^i, $i = 1, \ldots, n$, $k = 1, \ldots, N$ be random variables

$$o_k^i = \begin{cases} 1 & \text{if observation } x_i = k, \\ 0 & \text{otherwise.} \end{cases}$$

   Note that $\mathbf{E}\left[o_k^i\right] = \pi_k$. Now, $\hat{\pi}_k = \sum_{i=1}^n o_k^i/n$, therefore,

$$\mathbf{E}\left[\hat{\pi}_k\right] = \sum_{i=1}^n \mathbf{E}\left[o_k^i\right]/n = \sum_{i=1}^n \pi_k/n = \pi_k,$$

   which proves the lack of bias.

Maximum A-Posteriori (MAP) Estimation ML estimator treats the parameter θ as an unknown fixed value, and tries to find a value which is most likely, based only on the observed data. A more sophisticated approach could be to treat θ as a random variable itself, having a distribution $p(\theta)$. We can then seek the value of θ that is the best fit both to our prior belief about θ, expressed by $p(\theta)$, and to the observed data x. For instance, consider a biased coin, and a series of 20 tosses that all happen to be heads. ML estimator of the parameter p of binomial distribution will tell us that $p = 1$, i.e. tails will never occur. This seems unreasonable, however, since we believe that even the very biased coin would at least rarely produce a tail.

The *Maximum A-Posteriori* (MAP) estimator allows us to incorporate such knowledge into the estimate. In MAP, we explicitly maximize the posterior probability of θ given x:

$$\hat{\theta}_{MAP} = \operatorname{argmax}_{\theta} p(\theta|x).$$

Now, from the definition of conditional probability, and using Bayes rule, we have

$$p(\theta|x) = \frac{p(\theta, x)}{p(x)} = \frac{p(x|\theta)p(\theta)}{p(x)}.$$

The denominator $p(x)$ is independent of θ; it is, in fact, equal to $\int_\theta p(x,\theta)p(\theta)d\theta$ (why?). Therefore, it can be ignored in the maximization, and thus the MAP estimator is

$$\hat{\theta}_{MAP} = \operatorname*{argmax}_\theta p(x|\theta)p(\theta).$$

Note that as opposed to ML, MAP estimator is somewhat "subjective"; its value depends on the particular prior $p(\theta)$ that one believes is appropriate. A commonly used prior for the parameters of the multinomial distribution is the *Dirichlet* distribution. This is a distribution over vectors π such that all elements $\pi_k > 0$ and $\sum \pi_k = 1$. The Dirichlet distribution can be written as

$$D(\pi_{\mathbf{1}}, \ldots, \pi_{\mathbf{N}}) = \frac{1}{Z}\prod_{k=1}^{N} \pi_k^{\alpha_k - 1}$$

where α_k are the *hyperparameters* (parameters of the prior), and Z is a normalization constant needed to make this a valid distribution.

3. [**20pt**] Derive the MAP estimate of the parameters of multinomial distribution, using the Dirichlet prior.

   Hint: write down the term maximized by MAP as the product of likelihood and prior, take the log, and follow steps similar to the derivation of the ML estimator.

   Answer: The MAP estimate, by definition, is

   $$\hat{\pi}_{MAP} = \operatorname*{argmax}_\pi \log p(X|\pi)p(\pi) = \operatorname*{argmax}_\pi \left(\sum_{j=1}^{N} x_j \log \pi_j + \sum_{j=1}^{N} (\alpha_j - 1) \log \pi_j \right),$$

   and the corresponding optimization constraints are expressed by the system

   $$\frac{x_k + \alpha_k - 1}{\pi_k} = \frac{x_N + \alpha_N - 1}{\pi_N}, \quad k = 1, \ldots, N-1.$$

   Solving this system in a way very similar to our solution for the ML estimate, we get

   $$\hat{\pi}_k = \frac{x_k + \alpha_k - 1}{\sum_{j=1}^{N}(x_j + \alpha_j - 1)}.$$

   Note that this is equivalent to ML estimate, but taking into account, in addition to the observations X, a hypothetical additional set of observations, in which the outcome K occurs $\alpha_K - 1$ times.

4. (*Optional*) We would like the influence of the prior to be in some sense proportional to our confidence in it. Suggest a way of achieving this with the Dirichlet prior.

Answer: We might believe a priori that the parameters π_j should be something like p_j, where $p_j \geq 0$, and $\sum_j p_j = 1$. We'd like to express this prior belief in a Dirichlet distribution and also specify the strength of our belief. We can do this easily by setting $\alpha_k = \alpha p_k$, where larger values of α serve to concentrate the prior distribution around p_j (we would need more data to overcome such a prior). You can see this from the previous result by letting $\alpha \to \infty$. In this case, $\hat{\pi}_j = p_j$ as expected.

5. [**5pt**] Show that the influence of any fixed prior on the estimate of multinomial parameters becomes less significant as the amount of data increases. That is, show that as $K = \sum_k x_k$ increases, $\hat{\pi}_{kMAP}$ approaches a limit which is independent of the prior. This is a sensible conclusion since with sufficient data it shouldn't matter what we thought about the parameter values prior to seeing the data.

Answer: First of all, we notice that

$$\lim_{n\to\infty} \frac{\alpha_k - 1}{n + \sum_j \alpha_j - 1} = 0.$$

Therefore, $\lim_{n\to\infty} \hat{\pi}_{kMAP} = \lim_{n\to\infty} \frac{x_k}{n+\sum_j \alpha_j - 1}$. Now,

$$\lim_{n\to\infty} \hat{\pi}_{kML}/\hat{\pi}_{kMAP} = \lim_{n\to\infty} \frac{n + \sum_j \alpha_j - 1}{n} = 1,$$

that is, as the number of available observations increases, the MAP estimate approaches the ML estimate, which is of course independent of the fixed values of α_j.