Prof. David Draper
University of California, Santa Cruz
Department of Statistics
Baskin School of Engineering

STAT 206 (Applied Bayesian Statistics)

Take-Home Test 3: Final Version

Absolute due date: uploaded to `canvas.ucsc.edu` by 11.59pm on **22 Mar 2020**

Problem 2(B) on this test is required for the graduate students in the class but extra-credit for the undergraduates.

Here are the ground rules: this test is open-book and open-notes, and consists of two problems (true/false and calculation); **each of the 7 true/false questions is worth 10 points, and the calculation problem is worth xxx total points, for a total of xxx points, plus the possibility of up to xx extra-credit points**.

The right answer with no reasoning to support it, or incorrect reasoning, will get **half credit**, so try to make a serious effort on each part of each problem (this will ensure you at least half credit). In an AMS graduate class I taught in 2012, on a take-home test like this one there were 15 true/false questions, worth a total of 150 points; one student got a score of 92 out of 150 (61%, a D−, in a graduate class where B− is the lowest passing grade) on that part of the test, for repeatedly answering just "true" or "false" with no explanation. Don't let that happen to you.

On non-extra-credit problems, I mentally start everybody out at −0 (i.e., with a perfect score), and then you accumulate negative points for incorrect answers and/or reasoning, or parts of problems left blank. On extra-credit problems, the usual outcome is that you go forward (in the sense that your overall score goes up) or you at least stay level, but please note that it's also possible to go backwards on such problems (e.g., if you accumulate +3 for part of an extra-credit problem but −4 for the rest of it, for saying or doing something egregiously wrong).

This test is to be entirely your own efforts; do not collaborate with anyone or get help from anyone but me or our TAs (Peter Trubey and Xingchen (Joe) Yu). The intent is that the course lecture notes and readings should be sufficient to provide you with all the guidance you need to solve the problems posed below, but you may use other written materials (e.g., the web, journal articles, and books other than those already mentioned in the readings), **provided that you cite your sources thoroughly and accurately**; you will lose (substantial) credit for, e.g., lifting blocks of text directly from `wikipedia` and inserting them into your solutions without full attribution.

If it's clear that (for example) two people have worked together on a part of a problem that's worth 20 points, and each answer would have earned 16 points if it had not arisen from a collaboration, then each person will receive 8 of the 16 points collectively earned (for a total score of 8 out of 20), and I reserve the right to impose additional penalties at my discretion. If you solve a problem on your own and then share your solution with anyone else (because people from your cultural background routinely do this, or out of pity, or kindness, or whatever motive you may believe you have; it doesn't matter), you're just as guilty of illegal collaboration as the person who took your solution from you, and both of you will receive the same penalty. This sort of thing is necessary on behalf of the many people who do not cheat, to ensure that their scores are meaningfully earned. In the AMS graduate class in 2012 mentioned above, five people failed the class because of illegal collaboration; don't let that happen to you.

In class I've demonstrated numerical work in `R`; you can (of course) make the calculations and plots requested in the problems below in any environment you prefer (e.g., `Matlab`, `Python`, ...).

Please collect {all of the code you used in answering the questions below} into an Appendix at the end of your document, so that (if you do something wrong) the grader can better give you part credit. To avoid plagiarism, if you end up using any of the code I post on the course web page or generate during office hours, at the beginning of your Appendix you can say something like the following:

> *I used some of Professor Draper's `R` code in this assignment, adapting it as needed.*

1 True/False

[70 total points: 10 points each] For each statement below, say whether it's true or false; if true without further assumptions, briefly explain why it's true (and — *extra credit* — what its implications are for statistical inference); if it's sometimes true, give the extra conditions necessary to make it true; if it's false, briefly explain how to change it so that it's true and/or give an example of why it's false. If the statement consists of two or more sub-statements and two or more of them are false, you need to explicitly address all of the false sub-statements in your answer.

In answering these questions you may find it helpful to consult the following references, available on the course web page: DS (Degroot and Schervish (2012)) sections 3.10, 12.5, 12.6; Gelman et al. (2014) Chapter 11.

(A) If You can figure out how to do IID sampling from the posterior distribution of interest to You, this will often be more Monte-Carlo efficient than MCMC sampling from the same posterior.

(B) A (first-order) Markov chain is a particularly simple stochastic process: to simulate where the chain goes next, You only need to know (i) where it is now and (ii) where it was one iteration ago.

(C) The bootstrap is a frequentist simulation-based computational method (with ties to Bayesian nonparametrics) that can be used to create approximate confidence intervals for population summaries even when the population distribution of the outcome variable y of interest is not known; for example, if all You know from problem context is that Your observations $\boldsymbol{y} = (y_1, \ldots, y_n)$ are IID from *some* distribution with finite mean μ and finite SD σ, You can use the bootstrap to build an approximate confidence interval for μ even though You don't know what the population distribution is.

(D) In MCMC sampling from a posterior distribution, You have to be really careful to use a monitoring period of just the right length, because if the monitoring goes on for too long the Markov chain may drift out of equilibrium.

(E) Simulation-based computational methods are needed in Bayesian data science (inference, prediction and decision-making) because conjugate priors don't always exist and high-dimensional probability distributions are difficult to summarize algebraically.

(F) In MCMC sampling from a posterior distribution, You have to be really careful to use a burn-in period of just the right length, because if the burn-in goes on for too long the Markov chain will have missed its chance to find the equilibrium distribution.

(G) You're MCMC sampling from a posterior distribution for a vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$. During the monitoring period, the column in the MCMC data set for a component of θ (θ_j, say) behaves like an autoregressive time series of order 1 (AR_1) with estimated first-order autocorrelation $\hat{\rho}_j = 0.992$. As usual, You'll use the sample mean $\bar{\theta}_j^*$ of the monitored draws θ_{ij}^* as Your Monte Carlo estimate of the posterior mean of θ_j. To achieve the same estimated Monte Carlo standard error for $\bar{\theta}_j^*$ that You would have been able to attain if You could have done IID sampling, Your MCMC monitoring sample size would have to be about 250 times bigger than the length of the IID monitoring run.

2 Calculation

Parts (A) and (B) of this problem are similar to problem 2(B) in Take-Home Test 2, in that they look really long but don't actually have that much for You to do: just read the problems carefully, run my code (sometimes You'll need to modify it a bit), and figure how to interpret the output.

(A) *[115 total points for this problem, plus up to 20 extra credit points]* As I'm sure You know, if You encounter a wild mushroom in a forest there's no guarantee that it's edible; every year several people die in the U.S. from wild mushroom poisoning. Two questions come to mind, in this age of cell phone apps: (1) Can the edible/poisonous status of a wild mushroom be accurately predicted from characteristics such as its appearance and odor? and (2) If You were building an app to give people advice about whether a wild mushroom they've found is edible, (to make the app easy to use) what's the minimum number of variables necessary to get highly accurate predictions?

The *U.C. Irvine Machine Learning Repository* has a data set – a copy of which is now on the STAT 206 web page, along with a text file containing important contextual information – consisting of $n = 8{,}124$

> hypothetical samples corresponding to 23 species of gilled mushrooms in the *Agaricus* and *Lepiota* Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The *Audubon Society Field Guide to North American Mushrooms* (1981) clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like "leaflets three, let it be" for Poisonous Oak and Ivy.

As You'll see when You begin looking at the data set, there are $k = 22$ predictor variables $(x_1, \ldots, x_k)$ available, ranging from aspects of the mushroom's cap to its habitat, and the outcome variable y is coded 1 for poisonous and 0 for edible. The goals of this problem, corresponding to the two questions above, are (1) to build linear regression models, using these predictors, to produce estimated probabilities $\hat{p}$ that $(y = 1)$ as a function of a given mushroom's characteristics, (2) to identify the smallest subset of the x_j (for inclusion in the app) that still produces highly accurate $\hat{p}$ values, and (3) to decide whether the predictive system is accurate enough to release the app to the general public without poisoning a lot of people in the process.

We're going to do a maximum-likelihood analysis of this data set, because (I'm assuming that) you and I know so little about 'gilled mushrooms in the *Agaricus* and *Lepiota* Family' that a Bayesian analysis here would just reproduce the likelihood story. I'm also doing something a bit unusual in having you fit *linear* regression models to a binary outcome variable, but the more usual choice of *logistic* regression models (like those in problem 2(B)) would give essentially the same results.

When You examine the set of predictor variables, You'll see that they're all categorical (`R` calls such variables *factors*), taking on a number of possible values (*levels*) ranging from 1 to 12. **Important: all of the levels of all of the predictors in the data set have been abbreviated to a single letter in the standard English alphabet; the context file contains a dictionary that translates those abbreviations to their actual meanings.** Obviously any predictor variable that takes on only 1 possible value is useless for predicting anything; there is such a variable, so early on in the analysis we'll drop it (`veil.type`) and reset k to 21. One variable – `stalk.root` – has a lot of missing values (2,480 out of 8,124), but one nice thing about categorical predictors is that *missingness can be treated as just another level of the factor*, so that no cases are lost by having to omit rows in the data set with missing values in them (an undesirable action that's not needed with factor predictors).

As discussed in class, the basic frequentist linear regression model is of the form (for $i = 1, \ldots, n$)

$$y_i = \beta_0 + \sum_{j=1}^{k} x_{ij}\,\beta_j + e_i\,, \tag{1}$$

in which the $(e_i \,|\, \sigma\,\mathcal{N}\,\mathcal{B})$ are IID $N(0, \sigma^2)$; here $\mathcal{N}$ denotes the Normality assumption, which is not part of problem context. In class we also saw that this model can be written in matrix form as

$$\boldsymbol{y} = X\,\boldsymbol{\beta} + \boldsymbol{e}\,, \tag{2}$$

where $\boldsymbol{y}$ is the $(n \times 1)$ vector whose transpose is $(y_1, \ldots, y_n)$, X is the $[n \times (k+1)]$ matrix whose first column is a vector of 1s (to account for the intercept term β_0) and whose ith row is $(1, x_{i1}, \ldots, x_{ik})$, $\boldsymbol{\beta}$ is the $[(k+1) \times 1]$ vector whose transpose is $(\beta_0, \beta_1, \ldots, \beta_k)$ and $\boldsymbol{e}$ is the $(n \times 1)$ vector whose transpose is $(e_1, \ldots, e_n)$.

In applying this model to the mushroom data, a new question immediately arises: how can You bring a categorical predictor – such as `ring.number`, with the 3 levels "n" (none), "o" (one) and "t" (two) – into a regression model? The answer is with a set of *indicator*, also known as *dummy*, variables: with x_{16} = `ring.number` as an example, having $\ell = 3$ levels, You create a new variable z_1 that's 1 if x_{16} = "n" and 0 otherwise, and another new variable z_2 that's 1 if x_{16} = "o" and 0 otherwise, and a third new variable z_3 that's 1 if x_{16} = "t" and 0 otherwise.

If You now include all $\ell = 3$ of the z_j in the set of predictors, in place of x_{16}, You will have created what's called a *collinearity* problem: by the nature of how the z_j were defined, for every mushroom i in the data set it's a fact that $z_{i1} + z_{i2} + z_{i3} = 1$. This makes the X matrix in equation (2) non-invertible, meaning that the computation of the maximum-likelihood estimate of $\boldsymbol{\beta}$, namely $\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \boldsymbol{y}$, would be more difficult to carry out. The (simple) solution is to omit one of the z variables in the set of z_j You include in the modeling: after all, in the `ring.number` example, if You knew z_{i1} and z_{i2}, $z_{i3} = (1 - z_{i1} - z_{i2})$ would be redundant (in the jargon of regression modeling, the category whose z dummy has been left out is called the *omitted group*). Letting ℓ_j be the number of levels of categorical predictor x_j and setting $L = \sum_{j=1}^{k} \ell_j$, the new linear regression model, expressed in terms of the dummy variables z, is

$$\begin{aligned} y_i \;=\; & \beta_0 + [\beta_1\, z_{i1} + \cdots + \beta_{\ell_1 - 1}\, z_{i,\ell_1 - 1}] + [\beta_{\ell_1}\, z_{i2} + \cdots + \beta_{\ell_1 + \ell_2 - 2}\, z_{i,\ell_1 + \ell_2 - 2}] \\ & + \cdots + \left[\beta_{L-K-(\ell_k - 2)}\, z_{i,L-K-(\ell_k-2)} + \cdots + \beta_{L-k}\, z_{i,L-k}\right] + e_i\,. \end{aligned} \tag{3}$$

This looks nasty but isn't: original categorical variable (factor) x_1 is replaced by $(\ell_1 - 1)$ dummy variables, original factor x_2 is replaced by $(\ell_2 - 1)$ dummies, and so on up to original factor x_k being replaced by $(\ell_k - 1)$ dummies, for a total of $k^* = (L - k)$ dummy variables replacing the original k

factors. In the mushroom data set there are $k = 21$ non-trivial factors as predictor variables, and the total number of dummies needed to carry this information is $[(6 + 4 + 10 + 2 + 9 + 2 + 2 + 2 + 12 + 2 + 5 + 4 + 4 + 9 + 9 + 4 + 3 + 5 + 9 + 6 + 7) - 21] = 95$. (Where did I get the numbers $(6 + 4 + \cdots + 7)$?)

(a) *[5 total points for this sub-problem]* Create a new directory for this case study and download into this directory all of the files on the course web page that have 'mushroom' in their names. I've written some `R` code for You, to start You on the analysis of this data set; it's in the file you just downloaded called

`stat-206-mushroom-partial-data-analysis.txt`

There's a block of code at the top of the file that begins `'the first block of code starts here'` and ends `'the first block of code ends here'`; run this code block and study the output. The function `tab.sum` in this code block provides diagnostic information on whether a factor x will turn out to be predictively useful in the modeling; briefly explain in what sense `tab.sum` provides such information (*Hint:* the function estimates the conditional mean (and SD, not useful here) of what variable given what other variable?) *[5 points]*.

(b) *[10 total points for this sub-problem]* Run the second code block, in which a linear regression model is fit with the dichotomous outcome `poisonous` regressed on the factor `cap.shape`, and study the output. When the predictions $\hat{y}$ from equation (3) are specialized to this regression, they looks like

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 \, z_{i1} + \cdots + \hat{\beta}_5 \, z_{i5} \, , \tag{4}$$

in which the $\hat{\beta}_j$ are the maximum-likelihood estimates of the regression coefficients and where $\{z_{i1} = 1$ if `cap.shape` = 'c' and 0 otherwise$\}$, $\{z_{i2} = 1$ if `cap.shape` = 'f' and 0 otherwise$\}$, and so on down to $\{z_{i5} = 1$ if `cap.shape` = 'x' and 0 otherwise$\}$. Now examine the extracts from the `tab.sum` and regression fitting in Table 1. Explicitly identify $(\hat{\beta}_0, \ldots, \hat{\beta}_5)$ in the output in the table *[5 points]*, and – by thinking about the form of equation (4) for each of the levels of `cap.shape` – explicitly relate the numbers in the `mean` column of the `tab.sum` output in Table 1 to the $\hat{\beta}_j$ *[5 points]*.

(c) *[20 total points for this sub-problem]* Toward the end of the second code block, the code computes predicted $\hat{p} = P(y = 1 \,|\, x_1 \, \mathcal{L} \, \mathcal{B})$ values (in which $\mathcal{L}$ signifies the linear regression modeling assumption, which is not part of $\mathcal{B}$), makes a diagnostic plot and computes a numerical diagnostic – the *Predictive Separation Index (PSI)* – measuring the predictive strength of the factor `cap.shape`.

(i) *[10 total points for this sub-problem]* The diagnostic plot is in two parts: the top panel is a histogram of the $\hat{p}$ values for the mushrooms for which $y = 0$, and the bottom panel shows the same thing except for the cases in which $y = 1$. What would the ideal shape of these histograms be, if a factor x has extremely strong information for predicting a dichotomous outcome y? Explain briefly *[5 points]*. Do the histograms achieve that goal with the predictor `cap.shape`? Explain briefly *[5 points]*.

(ii) *[10 total points for this sub-problem]* The PSI, which is a numerical index that goes hand-in-hand with the diagnostic plot, is defined as follows:

$$PSI(x) = [\text{mean } (\hat{p} \text{ given } x) \text{ when } y = 1] - [\text{mean } (\hat{p} \text{ given } x) \text{ when } y = 0] \, . \tag{5}$$

What's the ideal value of the PSI, if a factor x is highly predictive of y? Explain briefly *[5 points]*. Does the PSI come close to achieving that goal with `cap.shape`? Explain briefly *[5 points]*.

Table 1: *Extracts from the output of the second code block.*

```
#      cap.shape n    mean      sd

# [1,] 1         452  0.1061947 0.308428
# [2,] 2         4    1         0
# [3,] 3         3152 0.4936548 0.5000391      output of tab.sum
# [4,] 4         828  0.7246377 0.4469667
# [5,] 5         32   0         0
# [6,] 6         3656 0.4671772 0.4989898

# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  0.10619    0.02279   4.659 3.22e-06 ***
# cap.shapec   0.89381    0.24335   3.673 0.000241 ***   output of
# cap.shapef   0.38746    0.02437  15.898  < 2e-16 ***   linear
# cap.shapek   0.61844    0.02834  21.824  < 2e-16 ***   regression
# cap.shapes  -0.10619    0.08864  -1.198 0.230926
# cap.shapex   0.36098    0.02416  14.942  < 2e-16 ***
```

Table 2: *Predictive accuracy of each of the factors x in the mushroom data set, with PSI sorted from largest to smallest.*

Factor (x)	PSI	Predictive Power
⋮	⋮	⋮
`cap.shape`	0.0603	weak
`cap.surface`	0.0388	weak
⋮	⋮	⋮

(d) *[20 total points for this sub-problem]* Run the third code block and study the output. I've written a function called `univariate.exploration` that automates the process of repeating the first and second code blocks; run this function with each of the other 20 categorical predictors (save `odor` for last, for reasons that will become clear); in each case, pay particular attention to the table created by `tab.sum`, the diagnostic plot and the PSI value. Summarize Your findings by completing Table 2; I suggest that You use the phrases *extremely strong*, *strong*, *moderate*, *weak*, and *almost none* to describe the predictive power of each x variable *[15 points]*. If You were going to base the app on only one or two predictors, which ones look like the best candidates? Explain briefly *[5 points]*.

(e) *[15 total points for this sub-problem]* In the output from code block 3, the PSI for `cap.surface` came out 0.03877928, which we could round to 0.03878. That number appears somewhere else in the regression output; where? Read pages 748–749 in DeGroot and Schervish (2012), available on the course web page; based on Your reading of these pages, briefly explain what the number in the regression output is trying to measure *[5 points]*. Does it make sense that the PSI and this number are closely related? (This relation only holds for regressions with a dichotomous outcome; if y is continuous, the PSI doesn't make sense.) *[5 points]* Check several other sets of output from the `univariate.exploration` function with different predictors; is

the relation between the PSI and the number in the regression output always the same? *[5 points]*

Run the fourth code block, in which a linear regression model is fit with all available predictors (let's call this the *full model* $\mathcal{F}$), and study the output. You can see that `R` has a convenient way (`poisonous` $\sim$ `.`) to specify all of the predictors without having to name all of them. You can further see that prediction of the poisonous status of all $n = 8{,}124$ mushrooms using the full model is perfect: all of the truly poisonous mushrooms have estimated $P(y_i = 1 \,|\, \mathcal{L}\, \boldsymbol{x}_i\, \mathcal{F}\, \mathcal{B}) = 1$, and all of the truly edible mushrooms have estimated $P(y_i = 1 \,|\, \boldsymbol{x}_i\, \mathcal{L}\, \mathcal{F}\, \mathcal{B}) = 0$ (here $\boldsymbol{x}_i$ is the vector of predictor variables for mushroom i).

However, this evaluation of the predictive quality of $\mathcal{F}$ may overstate its accuracy, because we used the same (entire) data set both to fit $\mathcal{F}$ and then to see how good $\mathcal{F}$ is. As mentioned in class, *cross-validation (CV)* is a good way to check on the extent of any over-fitting: You partition the data set at random into non-overlapping subsets, fit the model on one subset, and evaluate the quality of the fit on another. A well-established CV method is called *s–fold cross-validation*: randomly partition the entire data set into s non-overlapping exhaustive subsets $\{S_1, \ldots, S_s\}$, and loop as j (say) goes from 1 to s: set aside subset S_j, fit the model M_j on the union of all of the other subsets, and evaluate the quality of the fit on S_j, by using M_j to predict all of the y values in S_j; when the loop is finished, average the resulting s quality estimates to get an overall evaluation of the model's predictive accuracy that avoids over-fitting.

(f) *[10 total points for this sub-problem]* Run the fifth code block, which implements s–fold CV with $s = 10$ (this choice has been shown to be reliable), and study the output, which is summarized in a graph and a number: the graph plots the cross-validated predictions against the true values of the outcome variable `poisonous`, and the number is the CV estimate of what's called the *root-mean-squared error (RMSE)* $\hat{\sigma}$ of the regression predictions, namely $\hat{\sigma} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$ (here $\hat{y}_i$ is the predicted value of y_i; what the code actually does is (a) compute the $s = 10$ separate estimates $\hat{\sigma}_j$ of the $RMSE$ arising from the cross-validation process and then (b) combine the $\hat{\sigma}_j$ values optimally with $\hat{\sigma} = \sqrt{\frac{1}{n}\sum_{j=1}^{s}\hat{\sigma}_j^2}$). Does the graph arising from the CV process support the idea that the predictions from the full model $\mathcal{F}$ are perfect, even when cross-validated? Explain briefly *[5 points]*. Does the cross-validated $RMSE$ value also support this idea? Explain briefly *[5 points]*.

Now that we've achieved the rare feat of perfect prediction, let's think about the app we're designing: do we really want to make users supply multiple-choice answers to 21 different questions about the mushroom they're thinking of eating? The next (and nearly final) task in this problem is to see if a subset of the full set of 21 predictors can do as well, or nearly as well, in predictive accuracy as the full model $\mathcal{F}$.

(g) *[10 total points for this sub-problem]* Run the sixth code block, which implements a method called *step-wise variable selection*, using the *Bayesian Information Criterion (BIC)* we talked about in class; recall that lower BIC values correspond to better models. The output of this code block is sufficiently voluminous that I put it into another `.txt` file, also available on the course web page:

`stat-206-mushroom-analysis-variable-selection-with-bic.txt`

Study the output from this code block. This implementation of the `R` function `step` starts with the *null model* consisting of just an intercept term, and then sequentially chooses the

Table 3: *Cross-tabulation of truth against what the app says for decision rules based on* `odor` *with two* $\hat{p}$ *cutoffs, 0.05 (left) and 0.01 (right).*

0.05 Cutoff

		Truth		
		Poisonous	Edible	Total
App Says	Poisonous			
	Edible			
	Total			

0.01 Cutoff

		Truth		
		Poisonous	Edible	Total
App Says	Poisonous			
	Edible			
	Total			

best variable not yet in the model and adds it. For the mushroom data, the algorithm goes through 11 iterations of this method, until it discovers that the model with 10 sequentially-best predictors yields perfect predictions, at which point it stops with an excellent and snarky warning message. By thinking about what the output is saying about the best subset of x variables, and in what order, answer the following three questions, as k goes from 1 to 3:

> If the app were going to be based on only k variable(s), which {one is}/{ones are} best?

Explain briefly *[10 points]*.

(h) *[25 total points for this sub-problem]* Finally, suppose that we tentatively decide to base the app only on the mushroom's `odor`. We would then still have to specify a decision rule to implement in the app, as a function of the $\hat{p}$ value it produces for a new mushroom. It's easy to show (You're not asked to show this) that the optimal decision rule is of the form

> If $\hat{p} \geq c$, declare the mushroom poisonous; otherwise declare it edible

for some $0 \leq c \leq 1$. Run the seventh code block, which summarizes the quality of two `odor`–based decision rules, one with $c = 0.05$ and the other with $c = 0.01$. Fill out Table 3 above by **carefully** re-arranging the output of the final two `table` commands in the code block *[10 points]*. Letting (as usual with classification rules) {App says poisonous} be a positive (+) finding and {App says edible} be a negative (–) result, use Your filled-out Table 3 to estimate the false-positive and false-negative rates for each of the 0.05– and 0.01–cutoff rules (You may wish to refer back to the *ELISA* case study in class and/or Take-Home Test 1) *[10 points]*. Considering the real-world implications of a false-positive error, and repeating for false-negative mistakes, is the 0.05–cutoff rule acceptable as a basis for our app? What about the 0.01–cutoff rule? Explain briefly in both cases *[5 points]*.

(i) *Extra credit [up to 20 extra points]:* Repeat part (h) (modifying code block 7 appropriately) with the app based on the best *two* predictor variables (instead of just `odor`). Is there now, with the two best predictors instead of one, an optimal cutoff that You regard as an acceptable trade-off between false-positive and false-negative mistakes, if You were going to sell the resulting app to wild-mushroom hunters? Explain briefly *[up to 10 extra-credit points]*; *[up to 10 more extra-credit points]* for repeating (h) with the best *three* predictors.

(B) *[125 total points for this problem]* Email has been around since 29 Oct 1969, when a researcher named Ray Tomlinson sent the first email message over a network called `ARPANET`. It appears that the first email *spam* message — `Google dictionary` defines spam as "irrelevant or inappropriate messages sent on the Internet to a large number of recipients" — was sent on 1 May 1978 (it was an advertisement for a presentation by the now-defunct `Digital Equipment Corporation` (DEC), sent by a `DEC` marketer to several hundred `ARPANET` users). It was reliably estimated in 2019 that

about 53% of all email messages worldwide as of Oct 2018 were spam. What can You, as an email user, do to minimize the amount of spam You receive in Your inbox?

One idea would be to construct a type of algorithm called a *spam filter*: a program that inputs an email message and outputs a tentative classification of that message as spam or not-spam (which is sometimes whimsically referred to as *ham*). This problem is about an attempt by researchers at Hewlett-Packard Labs (HPL) in 1999 to construct a spam filter that was *personalized* to a single individual named George Forman, the HPL postmaster at the time. George collected a representative sample of 1,813 messages that were certified as spam by him and by other HPL researchers, and he also obtained a representative sample of 2,788 email messages from HPL employees that were certified to be non-spam, yielding a data set with $n = 4{,}601$ messages.

Our data sets this quarter have all had the same structure: a matrix with one row for each *subject* of study (e.g., people, strands of copper wire, ...) and one column for each *variable* measured on the subjects. The HPL spam case study poses an interesting data-science problem: how do You summarize Your information when the 4,601 subjects You're studying are *documents*?

The HPL researchers did something reasonable (but probably far from optimal in terms of information-extraction):

- they identified 48 words that they thought might be good discriminators (about half possible spam and half perhaps ham) and computed the relative frequency of occurrence of each word in each email message (on a 0–100 percentage-point scale);
- they kept track of the relative frequency with which each of 6 special characters — ; ([! $ # — occurred in each message (again on a 0–100 percentage point scale); and
- they summarized information about the use of capital letters in 3 ways: the average length in each email message of consecutive runs of capital letters, the longest run of consecutive capital letters in the message, and the total number of consecutive capital letters in each message.

The file `spam-data-context.txt` on the course web page contains additional contextual information; please make a directory that will hold all of the files in this case study, and download and read the context file before continuing.

(a) *[10 total points for this sub-problem]* The following statement is made in the context file: *False positives (marking good mail as spam) are very undesirable.* Do You agree with this statement? Explain briefly. Do You agree with the following natural companion statement? *In this problem, false positives are worse than false negatives.* Explain briefly.

(b) *[10 total points for this sub-problem]* If You wanted to go beyond what the HPL researchers did in extracting spam-relevant information from their email messages, in what other ways would You data-mine the texts in these messages to obtain additional spam-predictive signal? Please suggest at least two additional methods, explaining briefly.

The HPL data set is on the course web page in the file `SPAM.txt` . I've written `R` code that helps You analyzes these data, in two files that are also on the course page: `spam-partial-data-analysis-R.txt` and `spam-analysis-functions-R.txt` , and there's a third file — `spam-data-analysis-variable-summary.txt` — that contains some of the output You'll produce. Please download all of these files into Your directory for this case study, before going on with the rest of the problem.

Our first task is to understand the scales on which the predictor and outcome variables currently live, by making graphical and numerical summaries of them, both one by one and by examining the bivariate relationships between each predictor and the outcome.

(c) *[15 total points for this sub-problem]* Run the first code block in the file `spam-partial-data-analysis-R.txt` and examine the output. With reference to the results from the `summary( raw.data.modeling )` command, what's the minimum value taken on by each of the 57 predictors? What's the relationship between the median and the mean for each predictor? What do these two aspects of the data set imply about the histogram shapes of the predictors? Explain briefly.

(d) *[10 total points for this sub-problem]* To obtain the histograms mentioned in part (c), run the second code block in three steps: run plotting code block 1 and examine the results; repeat with plotting code block 2; and repeat with plotting code block 3, this time making a PDF file of the resulting plot and including it in Your solutions. Did the histograms come out as You expected them to? Explain briefly.

The usual descriptive tools for examining the way quantitative predictors x and quantitative outcomes y relate to each other are scatterplots (graphical) and correlations (numerical). These methods were created for Bivariate Normal situations, which we emphatically do not have here, but they're useful anyway as long as we don't interpret their results inappropriately. For example, correlations are *attenuated* (diminished in absolute value) when x or y is binary; what would ordinarily be a mediocre correlation of (say) +0.36 with Bivariate Normal data is actually quite strong when y is dichotomous.

(e) *[10 total points for this sub-problem]* Run the third code block and examine the full output, which is truncated in the `.txt` file containing the code. Bearing in mind that y is coded 1 for spam and 0 for non-spam, and that the emails were chosen so that the result is a personal spam filter for George Forman, do the magnitudes and $+/-$ signs of any of the following correlations of y with predictors x surprise You? $x =$ { 'ch..4', 'hp', 'credit', 'george', 'ch..5', 'crl.long' } Explain briefly.

I bring up the issue of distributional shape of the predictors because research has shown that predictive signal is maximized (if the outcome is dichotomous) when the histogram of a quantitative predictor x is close to Normal. If this is not true of x on the original scale on which the data values were collected, You can often increase predictive power by *transforming* x to a new scale of measurement, using one of two approaches:

(I) You can look for an invertible function f such that the distribution of $f(x)$ is closer to Normality, or

(II) You can create a set of indicator variables from x by partitioning it at its quantiles, and model x as a *factor* instead of as a continuous predictor. If the sample size n is large, a reasonable choice is to cut x into 10 groups at its *deciles* (the 0th, 10th, 20th, ... and 100th percentiles).

All of the models we'll fit in this problem are instances of *logistic regression*, which has the general form (for $i = 1, \ldots n$)

$$\begin{aligned} (Y_i \,|\, p_i \, \mathcal{B}) &\overset{\text{I}}{\sim} \text{Bernoulli}(p_i) \\ \text{logit}(p_i) \triangleq \log\left(\frac{p_i}{1-p_i}\right) &= \sum_{j=1}^{k} \beta_j \, x_{ij} = (X \, \boldsymbol{\beta})_j \,; \end{aligned} \tag{6}$$

here $\overset{\text{I}}{\sim}$ means *are independently distributed as*, X is an $(n \times k)$ matrix whose first column is all 1s to accommodate the intercept term and all of whose other columns are vectors x_j containing the

values of the predictor variables, and $\boldsymbol{\beta}$ is a $(k \times 1)$ vector of (unknown) regression coefficients β_j. This model is a special case of the class of methods known as *generalized linear models* (GLMs); here the *link function* that connects the p_i to X is $\text{logit}(p_i)$.

`R` has a built-in function called `glm` that performs maximum-likelihood fitting of model (6) and other GLMs; with $\boldsymbol{y} = (y_1, \ldots, y_n)$ as the observed vector of binary outcomes and `x.1` and `x.2` as vectors containing the values of two of the predictor variables, the command `glm( y ~ x.1, family = binomial( link = 'logit' ) )` fits the logistic regression model in which the only predictor is `x.1`, and `glm( y ~ x.1 + x.2, family = binomial( link = 'logit' ) )` fits the corresponding model in which `x.1` and `x.2` are both included as predictors.

As with all statistical models, we need a method for judging when one logistic regression model is better than another. The `glm` function has a frequentist method called *AIC* built into it that goes hand-in-hand with maximum-likelihood fitting of a GLM; AIC is similar to *BIC* (which we discussed in class) in that they both trade off goodness-of-fit of a model against its complexity, but AIC and BIC have different complexity penalties. For any parametric model $\mathcal{M}$ containing a $(k \times 1)$ vector $\boldsymbol{\theta}$ of unknown quantities (parameters) and no random effects, it turns out that the complexity of $\mathcal{M}$ is meaningfully defined simply to be k, and goodness of fit can be measured by calculating the maximum value the log-likelihood function $\ell\ell(\boldsymbol{\theta} \,|\, \boldsymbol{y}\, \mathcal{M}\, \mathcal{B})$ defined by $\mathcal{M}$ takes on. With this in mind, AIC is computed as follows:

$$AIC(\mathcal{M} \,|\, \boldsymbol{y}\, \mathcal{B}) = -2\, \ell\ell(\hat{\boldsymbol{\theta}}_{MLE} \,|\, \boldsymbol{y}\, \mathcal{M}\, \mathcal{B}) + 2\, k\,. \tag{7}$$

Because we want the maximum log-likelihood value of a model to be big as an indication of its goodness of fit, the multiplication of $\ell\ell(\hat{\boldsymbol{\theta}}_{MLE} \,|\, \boldsymbol{y}\, \mathcal{M}\, \mathcal{B})$ by -2 means that models with *small* values of AIC are preferred (where by the phrase "AIC_1 is *smaller than* AIC_2" I mean that AIC_1 is *to the left of* AIC_2 *on the number line*, not that AIC_1 is *closer to 0* than AIC_2).

(f) *[10 total points for this sub-problem]* Run the fourth code block, stopping and starting as recommended; study the logic behind the commands and examine the output. This chunk of code implements an instance of method (I) mentioned above, with a transformation of the form $x \to \log(x + C)$ on the predictor variable `make` plus an indicator variable for `( make = 0 )`. Two logistic regressions are run in this code block: `spam` predicted by `make`, and `spam` predicted by `log( make + C )` and `make.is.0`. Identify the AIC values for these two models. Has this transformation succeeded in extracting additional predictive signal from `make`? Explain briefly.

Code block 5 implements an instance of method (II) mentioned above (creating categorical indicators from a continuous predictor x and using them instead of x itself), and also creates a diagnostic plot that helps to see how a predictor might optimally enter the modeling. If a predictor x has a non-linear relationship with $\text{logit}(p)$, the simplest form of non-linearity is quadratic, so people often create a new variable `x.squared <- x * x` and logistically regress y on both `x` and `x.squared` to explore this possibility. The function I've written for You does six things with a predictor `x`: (1) it creates the categorical variable `x.cut` from `x` by partitioning `x` at a user-specified number of quantiles, and runs the function `tab.sum` (from problem 2(A)) to examine the resulting predictive signal; (2) it regresses `y` on `x` and harvests the AIC value; (3) it repeats (2) with the predictors `x` and `x.squared` and grabs the resulting AIC; (4) it repeats (2) with the predictor `x.cut` and takes note of the AIC value that results; (5) it prints a summary table of the three AIC values to help You see which modeling approach is best; and (6) it creates an interesting plot.

The red dots in the plot are the observed `x` and `y` values, so that You can see where most of the data is concentrated as You scan from left to right; it computes the predicted `p.hat` values based

on `x`, transforms them to the logit scale, and plots a `lowess` smooth of the transformed `p.hat` values against `x`, together with a user-specified number of bootstrap replicate `lowess` curves, to give You an idea both of the the relationship between `x` and logit(`p.hat`) and the uncertainty in that relationship. (I've truncated the plot at $\hat{p} = 0.999$ and 0.001, to keep it from going so wide vertically that the information in the plot for $0.001 < \hat{p} < 0.999$ is harder to absorb.)

(g) *[40 total points for this sub-problem]* Run the fifth code block, stopping after each call to the function `logistic.regression.univariate.exploration` to examine the output before going on to the next call.

(i) Examine the `tab.sum` results for the predictor `make` with 10 requested categories. Is there predictive signal in `x.cut` with this choice of x? Explain briefly *[5 points]*.

(ii) Looking at the AIC summary table for `make` with `n.cutpoints` = 10, which of the three approaches to bringing this x into the modeling is most successful? Explain briefly *[5 points]*.

(iii) Does the diagnostic plot for the predictor `make` exhibit curvature in the relationship between `make` and logit(p)? Is this confirmed in the logistic regression results when y is modeled as a function of `make` and `make.squared`? (*Hint:* Look at the table of estimated coefficients for the quadratic regression model.) Explain briefly *[10 points]*.

(iv) Is there noticeably more predictive signal in `make` when `n.cutpoints` = 20 is specified than with `n.cutpoints` = 10? If we try to go on with `n.cutpoints` = 30 or 40 or ..., what do You expect will eventually happen? Explain briefly *[10 points]*.

(v) In part (e) of this question You probably noticed that the word `your` had the strongest univariate predictive signal. Is this fact reflected in the diagnostic plot for this predictor? Explain briefly *[5 points]*.

(vi) Is there more predictive signal in `x.cut` based on `make` with `n.cutpoints` = 10 (method (II)) than the corresponding signal in {`log( make + C )` + `make.is.0`} (method (I))? Explain briefly *[5 points]*.

(h) *[10 total points for this sub-problem]* Run the sixth code block; study the logic behind the commands and examine the full output of the `glm( )` function call, which is truncated in the `R` code file but available in full in the file `spam-data-analysis-glm-results.txt` on the course web page.

(i) As explained in the `R` code file, run the function

`logistic.regression.effect.of.adding.a.new.variable`

three times, with `beta.x` = 0, then −3, then +3. Would you say that the predicted probabilities of ($y = 1$) (the `p.hat` values in the function call) change dramatically as `beta.x` goes from −3 to +3? Explain briefly *[5 points]*.

(ii) Find the five predictor variables with the largest absolute z values in the `glm( )` output, and compare that list (in order) with the five predictor variables in code block 3 having the highest correlations with the outcome variable `spam`. Should it worry us that these lists are not the same? Explain briefly *[5 points]*.

(i) *[10 total points for this sub-problem]* Run the seventh code block, stopping and starting as recommended; study the logic behind the commands and examine the output.

(i) What value of the predictive separation index (PSI) does the logistic regression model with all 57 predictors achieve, when the model fit to the Modeling subset of data is evaluated on the same subset? Where would you regard these predictions on the scale {terrible, not great, okay, good, terrific}? Explain briefly *[5 points]*.

(ii) How much did the PSI change when the model (fit to the Modeling subset) was required to predict the email messages in the Validation data set? Does this mean that the logistic regression model is guilty of a {large, moderate, small} amount of over-fitting? Explain briefly *[5 points]*.

(j) Run the eighth code block, stopping and starting as recommended; study the logic behind the commands and examine the output. The point of this code block is to fit a Bayesian logistic regression model, to the Modeling subset, that shrinks unstable maximum-likelihood regression coefficients toward 0 to stabilize the prediction process.

(i) The predictors with the largest standardized regression coefficients in absolute value are `george` and `cs`. Has the Bayesian model succeeded in stabilizing the ML estimation process, at least to some extent, for those two predictors? What percentage of the $k = 57$ coefficients have been shrunken toward 0 in the Bayesian fitting process? Explain briefly *[10 points]*.

(ii) Are the Bayesian predictions dramatically more accurate than the ML results? Is the Bayesian model guilty of more over-fitting than the ML model, or less, or about the same? Explain briefly *[10 points]*.

(iii) Examine the tables, at the end of code block 8, that include uncertainty bands for the `p.hat` values from the Bayesian fitting. Would you say that, even with 3065 email messages in the Modeling subset, there is substantial uncertainty in the prediction process? Explain briefly *[5 points]*.

(k) Run the ninth code block; study the logic behind the commands and examine the output. This chunk of code fits (via ML) a logistic regression model in which we include not only all $k = 57$ predictors but also all *two-way interactions* between those variables; to create an interaction predictor between (say) x_1 and x_2, we just form the product $(x_1 \cdot x_2)$; this brings in *non-linear* predictive information, which can greatly improve the predictive process.

(i) Compare the typical magnitudes of the logistic regression coefficients in the all-two-way-interactions model with the corresponding values in the earlier regression with only the 57 *main effects* (as they're called); do you agree with the statement, "It looks like ML logistic regression has gone crazy"? Explain briefly *[5 points]*.

(ii) And yet, how do the predictions from the model with all two-way interactions compare in quality (e.g., PSI) with those from the earlier model that included only the main effects? Explain briefly *[5 points]*.

There's lots more to do with this data set (e.g., using a Bayesian/penalized-ML method called the *lasso* to fit the all-interactions model), but let's stop here for now.