Homework 4

PSTAT 131/231, Fall 2020

Due on Friday Decmber 11, 2020 at 11:59 pm

In this homework, the problems will be more computationally intensive than previous problems in this class. You should expect the code for some problems to take a couple of minutes to complete. Re-knitting your file can take a long time so you should consider using `cache=TRUE` option in R chunks that involve code which takes a while to run. Another option would be to work some of the more computationally demanding problems in separate Rmd. Please load the following packages for this homework:

```
library(tidyverse)
library(tree)
library(randomForest)
library(gbm)
library(ROCR)
library(e1071)
library(imager)
```

1. Fundamentals of the bootstrap

In the first part of this problem we will explore the fact that approximately 1/3 of the observations in a bootstrap sample are *out-of-bag*.

a) Given a sample of size n, what is the probability that any observation j is *not* in in a bootstrap sample? Express your answer as a function of n.

b) Compute the above probability for $n = 1000$.

c) Verify that your calculation is reasonable by resampling the numbers 1 to 1000 with replace and printing the number of missing observations. Hint: use the `unique` and `length` functions to identify how many unique observations are in the sample.

Here we'll use the bootstrap to compute uncertainty about a parameter of interest.

d) By November 19, 2015, Stephen Curry, an NBA basketball player regarded as one of the best players currently in the game, had made 62 out of 126 three point shot attempts (49.2%). His three point field goal percentage of 0.492, if he maintains it, will be one of the best all time for a single season. Use bootstrap resampling on a sequence of 62 1's (makes) and 64 0's (misses). For each bootstrap sample compute and save the sample mean (e.g. bootstrap FG% for the player). Use 1000 bootstrap samples to plot a histogram of those values. Compute the 95% bootstrap confidence interval for Stephen Curry's "true" end-of-season FG% using the `quantile` function in R. Print the endpoints of this interval. However, this estimate, and the associated uncertainty, exclude information about his career performance as well as the typical shooting skill for other players in the league. For reference, prior to this year, Stephen Curry had made about 43% of all three point shots in his career. Despite the fact that the bootstrap histogram shows that it is about equally likely that Curry's true skill is greater or elss than 0.492, why do you expect that his end-of-season field goal percentage will in fact be lower than his percentage on 11/19? *Hint:* look up the phenomenon known as "regression to the mean".

2. Eigenfaces

In this problem we will use PCA to explore variation in images of faces. Load the data saved in `faces_array.RData` with the `load` function. This will load a 100 x 100 x 1000 *array* of data. An array is a generalization of a matrix to more than 2 dimensions. In this example, the first two dimensions index the pixels in a 100 x 100 black and white image of a face. The last dimension is the index for one of 1000 face images. The faces used in this example are from 1000 images scraped from the internet. See https://cyberextruder.com/face-matching-data-set-download/ for more info.

```
load("faces_array.RData")
```

Although it is natural to think about an stack of 1000 matrices representing each of the face images, to run PCA we need to input a single matrix. To do this, we'll convert each 100 x 100 matrix to a single vector of length 100*100 = 10000. When you call `as.numeric` on a matrix, it stacks each of the columns in the matrix into one large vector. Thus, we can think of our data as 1000 observations of a 10000 variables (one variable per pixel). Run the following code to get a matrix of face observations.

```
face_mat <- sapply(1:1000, function(i) as.numeric(faces_array[, , i]))  %>% t
```

When we want to visualization an image, we need to take the 10000 dimensional vector and reconstruct it as a matrix. The code `plot_face` takes a single 10000 dimensional vector (e.g. a column of `face_mat`), converts it back to a matrix, and plots the resulting image. You can test this functionality by printing a random face from the dataset: `plot_face(face_mat[sample(1000, 1), ])`.

```
plot_face <- function(image_vector) {
  plot(as.cimg(t(matrix(image_vector, ncol=100))), axes=FALSE, asp=1)
}
```

a) Find the "average" face in this dataset by averaging all of the columns in `face_mat`. Plot the average face by calling `plot_face` on the average.

b) Run PCA on `face_mat` setting `center=TRUE` and `scale=FALSE`. In class we mentioned that in general it is best if `scale=TRUE` because it puts all variables on the same scale and we don't have to worry about the units of the variables (remember, the scale of the variables affects our results). In general, this is good practice, especially when the predictor variables are of mixed types. Here, each variable represents a single pixel intensity (in black & white) and so all variables already have the same units and same scale (minimum of 0 and maximum of 255). In this case, setting `scale=FALSE` actually seems to give slightly better results. Plot the PVE and cumulative PVE from the PCA. How many PCs do you need to explain at least 50% of the total variation in the face images?

c) Plot the first 16 principle component directions as faces using the `plot_face` function (these are the columns of the `rotation` matrix). Early researchers termed these "eigenfaces" since they are eigenvectors of the matrix of faces. The code below will adjust the margins of you plot and specifies a layout for the 16 images. `par(mfrow=c(4,4))` specifies a grid of 4 x 4 images. Each time you call `plot_face` it will plot the next face in one of the new grid cells. All you need to do is call `plot_face` 16 times (please use a `for` loop). Note that these images describe "directions" of maximum variability in the face images. You should interpret light and dark regions in the eigenfaces as regions of high *contrast*, e.g. your interpretation should not change if you inverted black and white in the images.

d) In this part, we will examine faces that have the highest and lowest values for specific PCs. Plot the faces with the 5 largest values on PC1 and the 5 smallest values for PC1. Based on the example faces, and the first eigenface from the previous part and the 10 example images, what aspect of variability in the face images is captured by the first component.

e) Repeat part d) but now display example faces with the largest and smallest values on principal component 5. Again, discuss what aspect of variability in the face images is best captured by this principal component. Based on your results, which principal component, (1 or 5) would be more useful as a feature in a face recognition model (e.g. a model which predicts the identity of the individual in an image)

f) (**231 only**) In this example we will demonstrate how we can use PCA for image compression. Pick any face image from the dataset. We can "compress" the image by expressing the image as a linear combination of a smaller number of eigenfaces. In class we mentioned that PCA is closely related to the singular value

decomposition (SVD) of a matrix. Using the SVD, on the centered original image vector, $\mathbf{x}$ (the 1000×10000 matrix), we can express $\mathbf{x} = UDV^T = \mathbf{z}V^T$, where here V is the matrix of eigenvectors (the eigenfaces from the part c, e.g. `rotation` returned by `prcomp`) and D is the standard deviation of the principal components (`sdev` returned by `prcomp`). Together, $\mathbf{z} = UD$ are the coordinates of the princpal components in the rotated space (`prcomp$x`). We can come up with a "low rank" approximation to $\mathbf{x}$ by throwing away the lower variance components. Specifically, we can approximate the original vector by using only the first k columns of V and the first k columns of z: $\tilde{x} = \mathbf{z}^{1:k}V^{1:k^T}$. Compute the compressed representations of the face of your choosing, $\tilde{x}$ for values of $k \in \{10, 50, 100, 300\}$. Note that you should add the "average face", μ from part a) back onto $\tilde{x}$ before plotting, since the x has been centered. Plot the 4 compressed face images, $\tilde{x} + \mu$ as well as the original image in a single row by specifying `par(mfrow=c(1, 5))` prior to each call of `plot_face`.

3. Logistic regression with polynomial features

a) In class, we have used polynomial linear regression several times as an example for model complexity and the bias variance tradeoff. We can also introduce polynomial logistic regression models to derive more sophisticated classification functions by introducing additional features. Use `read_csv` to load `nonlinear.csv` and plot the data. Plot each point colored according to its class, `Y`.

b) Fit a logistic regression model of `Y` on `X1` and `X2`. The decision boundary can be visualized by making predictions of class labels over finely sampled grid points that cover your region (sample space) of interest. The following code will create grid points over the sample space as below:

```
# grid of points over sample space
gr <- expand.grid(X1=seq(-5, 5, by=0.1),  # sample points in X1
                  X2=seq(-5, 5, by=0.1))  # sample points in X2
```

For each point in `gr`, predict a class label using the logistic regression model. You should classify based on the probability being greater or less than 1/2. Visualize your predictions at each point on the grid using the `geom_raster` function. This function colors in rectangles on the defined grid and is a good way to visualize your decision boundary. Set the `fill` aesthetic to your predicted label and outside of the `aes` use `alpha=0.5` to set the transparency of your predictions. Plot the observed data, colored by label, over the predictions using `geom_point`.

c) Fit a model involving 2nd degree polynomial of `X1` and `X2` with interaction terms. You should use the `poly()` function. Inspect result of the fit using `summary()`. Plot the resulting decision boundary.

d) Using the same procedure, fit a logistic regression model with 5-th degree polynomials without any interaction terms. Inspect result of the fit using `summary()`. Plot the resulting decision boundary and discuss the result. Explain the reason for any strange behvaior.

e) Qualitatively, compare the relative magnitudes of coefficients of in the two polynomial models and the linear model. What do you notice? Your answer should mention bias, variance and/or overfitting.

f) (**231 required, 131 extra credit**) Create 3 bootstrap replicates of the original dataset. Fit the linear model and the 5th order polynomial to each of the bootstrap replicates. Plot class predictions on the grid of values for each of both linear and 5th order fits, from each of the bootstrap samples. There should be six plots total. Discuss what you see in the context of your answer to the previous question.

4. Predicting insurance policy purchases

This question involves the use of the "Caravan" data set, which contains 5822 real customer records. Each record consists of 86 variables, containing sociodemographic data (variables 1-43) and product ownership (variables 44-86), grouped by zip code. In this problem we will focus on predicted the variable "Purchase" which indicates whether the customer purchased a caravan insurance policy. For more information see http://www.liacs.nl/~putten/library/cc2000/data.html.

a) When you load the "ISLR" library, the variable `Caravan` is automatically loaded into your environment. Split `Carvan` into a training set consisting of the first 1000 observations and a test set consisting of the remaining observations.

b) Fit a boosting model to the training set with `Purchase` as the response and the other variables as predictors. Use the `gbm` to fit a 1,000 tree boosted model and set the shrinkage value of 0.01. Which predictors appear to be the most important (Hint: use the `summary` function)?

c) Now fit a random forest model to the same training set from the previous problem. Set `importance=TRUE` but use the default parameter values for all other inputs to the `randomForest` function. Print the random forest object returned by the random forest function. What is the out-of-bag estimate of error? How many variables were subsampled at each split in the trees? How many trees were used to fit the data? Look at the variable importance. Is the order of important variables similar for both boosting and random forest models?

d) Use both models to predict the response on the test data. Predict that a person will make a purchase if the estimated probability of purchase is greater than 20 %. Print the confusion matrix for both the boosting and random forest models. In the random forest model, what fraction of the people predicted to make a purchase do in fact make one? Note: use the `predict` function with `type="prob"` for random forests and `type="resonpse"` for the boosting algorithm.

5. An SVMs prediction of drug use

In this problem we return to an analysis of the drug use dataset. Load the drug use data using `read_csv`:

```
drug_use <- read_csv('drug.csv',
                     col_names = c('ID','Age','Gender','Education','Country','Ethnicity',
                                   'Nscore','Escore','Oscore','Ascore','Cscore','Impulsive',
                                  'SS','Alcohol','Amphet','Amyl','Benzos','Caff','Cannabis',
                                  'Choc','Coke','Crack','Ecstasy','Heroin','Ketamine','Legalh','LSD',
                                  'Meth', 'Mushrooms', 'Nicotine', 'Semer','VSA'))
```

a) Split the data into training and test data. Use a random sample of 1500 observations for the training data and the rest as test data. Use a support vector machine to predict `recent_cannabis_use` using only the subset of predictors between `Age` and `SS` variables as on homework 3. Unlike homework 3, do not bother mutating the features into factors. Use a "radial" kernel and a cost of 1. Generate and print the confusion matrix of the predictions against the test data.

b) Use the `tune` function to perform cross validation over the set of cost parameters: `cost=c(0.001, 0.01, 0.1, 1,10,100)`. What is the optimal cost and corresponding cross validated training error for this model? Print the confusion matrix for the best model. The best model can be found in the `best.model` variable returned by `tune`.

c) (**231 only**) One drawback with the support vector machine is that the method does not explicitly output a probability or likelihood for the class labels. One way to estimate class probabilities is with the bootstrap. Create 200 bootstrap replicates of the training data, fit the svm with the best cost parameter learned in the previous part, and predict the class labels on the *test* data for every bootstrapped fit. Calculate the class probability for each observation as the fraction of the bootstrap predictions assigned to each class (out of the 200). Use these probabilities and the true test set labels to plot an ROC curve for this SVM model. Note: the code may take a couple of minutes to complete.