Problem 1. In this problem we explore the bias-variance trade-off for \( k \)-nearest-neighbors (\( k \)-NN).

Recall how \( k \)-NN works: we are given a training dataset \( X, Y \), and whenever we want to make a prediction on a new datapoint \( \bar{X} \), we average the \( Y \) values of its nearest \( k \) neighbors in \( X \).

Formally, given a vector \( \bar{X} \), let \( \mathcal{N}_k(\bar{X}) \) be the indices of the \( k \) nearest covariate vectors (in terms of Euclidean distance) to \( \bar{X} \); i.e., these are the indices of the \( k \) observations for which \( \|X_i - \bar{X}\| \) is smallest. Then the \( k \)-NN prediction at \( \bar{X} \) is:

\[
\hat{f}(\bar{X}) = \frac{1}{k} \sum_{\ell \in \mathcal{N}_k(\bar{X})} Y_\ell.
\]

For the rest of this problem, we assume the the population model is \( Y = f(\bar{X}) + \varepsilon \), where \( \mathbb{E}[\varepsilon | \bar{X}] = 0 \), and \( \text{Var}(\varepsilon | \bar{X}) = \sigma^2 \), and that errors are independent across observations.

(a) Explain why given a covariate vector \( \bar{X} \):

\[
\mathbb{E}_Y \left[ \sum_{\ell \in \mathcal{N}_k(\bar{X})} Y_\ell | X, \bar{X} \right] = \sum_{\ell \in \mathcal{N}_k(\bar{X})} f(X_\ell).
\]

Use this fact to conclude that the bias of \( k \)-NN is equal to:

\[
f(\bar{X}) - \frac{1}{k} \sum_{\ell \in \mathcal{N}_k(\bar{X})} f(X_\ell).
\]

(b) On the other hand, explain why the variance of \( k \)-NN is \( \sigma^2 / k \). (\textit{Hint}: What is the variance of a sum of independent random variables?)

(c) Suppose you fit a linear regression on the data, and you send your model \( \hat{\beta} \) to a friend by email so that he can also make predictions if presented with new data. Then, you fit the \( k \)-NN model, meaning that you find the \( k \) you think is best, and then you send your model again to your friend so that he can compare both models. What’s an obvious disadvantage of \( k \)-NN? (Suppose you have as much data as Facebook, so rows are users and columns represent a few covariates of each account).

\textit{Note:} While the idea of \( k \)-NN is quite natural, it certainly does not work in high-dimensional settings (\( p \geq n \)) as the amount of data you need to truly fill the space grows exponentially with \( p \).

\(^1\text{For this problem you can assume there are no ties.}\)
This is a well-known phenomenon called the curse of dimensionality. You can read more about it here: [https://en.wikipedia.org/wiki/Curse_of_dimensionality](https://en.wikipedia.org/wiki/Curse_of_dimensionality)

**Problem 2.** (Naive Bayes) In this problem, we study a particularly simple approach to classification: the naive Bayes classifier. This is one of the most widely used classification techniques, and despite its simplicity is known to work surprisingly well in a wide range of settings.

We build the naive Bayes’ classifier for a spam detection problem. The data we work with is the dataset [spam.csv](http://web.stanford.edu/class/msande226/spam.csv). This dataset has 4601 rows; each row represents information about a single e-mail message. There are 49 columns. The last column indicates whether the email was a spam or not (1 means spam, 0 means not spam). The remaining 48 columns are binary values. Every column represents a word, and the value in a particular row indicates whether that word was present in the email or not (1 means the word was present, 0 means that it was not present).

As we discussed in class, if you knew the population model, the optimal classifier for 0-1 loss would be the Bayes classifier: given a covariate vector $\vec{X}$, if the message is more likely to be spam than not, then predict 1; otherwise predict 0.

Of course, in practice we don’t know the population model. In this problem, we take an approach to approximating the Bayes’ classifier that assumes that the different covariates are independent.

In the population model, given a covariate vector $\vec{X}$, we can expand the conditional probability of $Y$ given $\vec{X}$ as follows:

$$
\mathbb{P}(Y = y | \vec{X}) = \frac{\mathbb{P}(Y = y, \vec{X})}{\mathbb{P}(\vec{X})} = \frac{\mathbb{P}(\vec{X} | Y = y) \mathbb{P}(Y = y)}{\mathbb{P}(\vec{X})}.
$$

If this quantity is larger for $y = 0$ than $y = 1$, then the Bayes’ classifier returns 0; otherwise it returns 1. Note that the denominator does not depend on $\vec{X}$. With this in mind, the naive Bayes classifier works as follows:

- Approximate $\mathbb{P}(Y = 0)$ and $\mathbb{P}(Y = 1)$ by the empirical frequency of 0’s and 1’s in the data; i.e., for $y = 0, 1$ let:

$$
\hat{Q}_y = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{Y_i = y\}.
$$

- Approximate $\mathbb{P}(\vec{X} | Y = y)$ by assuming that the covariates are independent given $y$; i.e., for each covariate $j$, $x = 0, 1$, and $y = 0, 1$, define:

$$
\hat{q}_j(x|y) = \frac{\sum_{i=1}^{n} \mathbb{I}\{X_{ij} = x, Y_i = y\}}{\sum_{i=1}^{n} \mathbb{I}\{Y_i = y\}}.
$$

---

This dataset is derived from a spam dataset in the UCI ML repository; see [https://archive.ics.uci.edu/ml/datasets/Spambase](https://archive.ics.uci.edu/ml/datasets/Spambase).
In words, this is the fraction of observations where $Y_i = y$, for which also $X_{ij} = x$. We then approximate $\Pr(\vec{X} | Y = y)$ by the following product of empirical frequencies:

$$\prod_{j=1}^{p} \hat{q}_j(X_j | y).$$

- Putting it together, the naive Bayes classifier works as follows. Given a new covariate $\vec{X}$, compute:

$$\hat{Q}_0 \prod_{j=1}^{p} \hat{q}_j(X_j | 0), \quad (*)$$

and

$$\hat{Q}_1 \prod_{j=1}^{p} \hat{q}_j(X_j | 1). \quad (***)$$

If $(*) > (***)$, then the classifier returns 0; otherwise it returns 1. (Ties can be broken at random.)

(a) Read in the data. Set the seed to be 123 using:

```r
set.seed(123)
```

Then split the data into training (75%) and test (25%) sets. If your original data frame was `df`, you can do this as follows:

```r
train.ind = sample(1:nrow(df), 0.75*nrow(df))
df.train = df[train.ind,]
df.test = df[-train.ind,]
```

You don’t need to report the results of this step.

(b) Now explore your training data. In particular, compute the sample correlation between between the occurrence of each word, and whether or not the message is spam.

Which 5 words are most positively correlated with a message being spam? Which 5 words are most negatively correlated with a message being spam?

*Hint:* The `cor` function computes sample correlations.

(c) In the next several steps we will build a naive Bayes classifier from the training data. First, compute $\hat{Q}_1$ from the training data. (Note that $\hat{Q}_1$ is the fraction of emails that are spam, and that $\hat{Q}_0 = 1 - \hat{Q}_1$ — so we only need to compute $\hat{Q}_1$.) Store the resulting value as `Q1`, and report it.
(d) Next, for each covariate (word) \( j \), let \( q_j(1|y) \) be defined as above. As before, we have \( q_j(0|y) = 1 - q_j(1|y) \), so we only need to compute \( q_j(1|y) \) for \( y = 0, 1 \).

To do this, start by creating two empty vectors \( q_0 = \text{rep}(\text{NA}, 48) \) and \( q_1 = \text{rep}(\text{NA}, 48) \), and then fill them in so that the \( j \)'th entry of \( q_0 \) is \( q_j(1|0) \), and the \( j \)'th entry of \( q_1 \) is \( q_j(1|1) \). Report the resulting vectors.

(e) Now run the following R code:

```r
spamNB = function(x, thresh = 0) {
  P1 = log(Q1) + sum( x*log(q1) + (1-x)*log(1 - q1) )
  P0 = log(Q0) + sum( x*log(q0) + (1-x)*log(1 - q0) )
  if (P1 - P0 > thresh)
    return(1)
  else
    return(0)
}
```

This function creates a classifier using the values you’ve computed. In addition, the function has a parameter \( \text{thresh} \); we’ll return to this below. For now just note that by default \( \text{thresh} = 0 \).

The function takes as input a binary vector \( x \), and returns either 0 or 1. Assuming \( Q1, q0, q1 \) were all computed as in the previous part, explain why this function implements the naive Bayes classifier as defined above. Can you think of why we take \( \log \)'s of the frequencies in the code above?

(f) You will now apply the preceding classifier to your test set. In particular:

- Apply the \( \text{spamNB} \) function to the covariate vector in each of the rows of your test set.
- Compute the 0-1 loss on each observation: you incur a loss of 1 if the observation is misclassified, and 0 otherwise.
- Average the 0-1 loss across observations of the test set.

Report your resulting mean 0-1 loss on the test set.

(g) Finally, we will \textit{tune} the classifier that we developed in the preceding part, by varying the threshold used.

Here is some code that you will need to complete for this part:

```r
thresholds = seq(-50, 50, 1)
TP = rep(NA, length(thresholds))
TN = rep(NA, length(thresholds))
FP = rep(NA, length(thresholds))
FN = rep(NA, length(thresholds))
```
for (t in 1:length(thresholds)) {
    # START HERE
    # Compute the predictions made by spamNB() on
    # the test set using thresh = thresholds[t]
    # Record no. of true positives in TP[t],
    # no. of false positives in FP[t], etc.
    # END HERE
}

# Compute TPR, FPR, 0-1 loss
TPR = TP/(TP + FN)
FPR = FP/(FP + TN)
zero.one.loss = (FP+FN)/n.test

results.df = data.frame(thresholds, TPR, FPR, zero.one.loss)

# Plot an ROC curve
library(ggplot2)
ggplot(data = results.df, aes(x = FPR, y = TPR)) + geom_point()

(Here n.test is the number of observations in your test set.)

This code does the following: you loop over thresholds from −50 to 50, and apply spamNB
to make predictions on your test set.

You need to complete the for loop (between the START and END comments), so that (i)
predictions are made on the test set using the given threshold; and (ii) using the resulting
predictions, you compute the number of true positive, false positives, etc. The remaining
code uses what you’ve done to plot an ROC curve.

Explain what value of the threshold you would choose and why. How does the threshold you
picked perform in terms of 0-1 loss? If you wanted to minimize 0-1 loss, which threshold
would you pick?
Problem 3. This problem is meant to serve as practice to get you ready for your mini-project. We will use tools in R to explore data, to use cross validation, and perform stepwise regression. The techniques should be helpful for exploring your own dataset for the project. For this problem, we will rely on functions from the packages GGally, cvTools and MASS.

Note that some extra hints are provided at the bottom of the question, and feel free to be brief in your write-up of the answers.

We consider a wine quality data set that you can download here: [https://archive.ics.uci.edu/ml/datasets/Wine+Quality](https://archive.ics.uci.edu/ml/datasets/Wine+Quality) This page also gives some basic information about the attributes.

We will build a predictive model for the the quality score based on chemical properties of the wine.

(a) Download the data for both red and white wines, and combine the datasets into a single data frame. Don’t forget to add a column that specifies the color of the wine, which should be treated as a factor. You don’t need to report any results for this step.

(b) Split the data into a training and test set. The training set should have 5000 observations. To be able to replicate your results, please use `set.seed(1)`. You can use the `sample` function to obtain a random set of indices. You don’t need to report any results for this step.

(c) Use the `ggpairs` (part of GGally) function to visualize your data (if you have trouble with ggplot, you can use the built-in `pairs`). Make sure you use only your training set. Comment briefly on notable correlations. Also, identify covariates that you want to transform (find at least one).

*Note:* Using a subset of your training data may speed up the plotting. If you do this, make sure you take a random subset.

(d) Before you transform your data, fit a linear model using all covariates to predict `quality`; this will serve as a baseline model. Use cross-validation to get an estimate for the prediction error (see the hints). Report your findings.

(e) Apply transformations that you deem appropriate, and fit a new linear model. Report your findings, including estimates of prediction error. Did the transformations improve your predictions?

(f) Fit a linear model that includes all two-way interactions. Note that this is convenient to do using the formula `quality ~ . + .:.`. Give an estimate of prediction error.

(g) Now you will use the `stepAIC` function to carry out stepwise regression. Perform backward selection using both AIC and BIC to find optimal models. What are the CV estimates of error for the models you find? How many terms do the models contain?

(h) Pick your favorite model. This can be one of the models above, but you are also free to propose any other model. Find the mean square error on both the train and the test set. How do the train and test error compare to the CV errors you have found earlier in the problem set?
Hints:

- To turn a column named \( \text{col} \) of a dataframe \( \text{df} \) into a factor, you can use \( \text{df}$\text{col} = \text{factor(df}$\text{col}) \).
  
  In addition, to construct a new data frame by combining rows of two separate data frames \( \text{df1} \) and \( \text{df2} \), you can use \( \text{rbind(df1,df2)} \).

- To sample a set of indices for your train-test split, you can use

  ```r
  in.train = sample(nrow(df), size = M)
  train = df[in.train, ]
  test = df[-in.train, ]
  ```

  where \( M \) is the number of rows you want in the training set.

- To fit a linear model \( Y \sim X \) (where \( X, Y \) are columns of a data frame \( \text{df} \)) and perform cross validation with 10 folds, you can do:

  ```r
  model = lm(Y ~ X, data=df)
  model.cv = cvFit(model, data=df, y=df$Y, K=10)
  ```

  *Note*: By default \( \text{cvFit} \) reports the RMSE (root mean square error). You will need to square this to get mean squared error.

- Similarly, to perform stepwise regression, use \( \text{stepAIC} \) as follows:

  ```r
  full.model = lm(Y ~ X + Z + X:Z, data=df)
  selected.model = stepAIC(full.model, direction="backward", k=2)
  ```

  Here, \( k=2 \) indicates that we want to use AIC. We can use BIC by setting \( k=\log(nrow(df)) \). (In the function call, this just varies the coefficient on the model complexity term in the AIC/BIC expressions from lecture.) To run forward stepwise regression, you would have to also include a scope, with both lower bound and upper bound for the model.