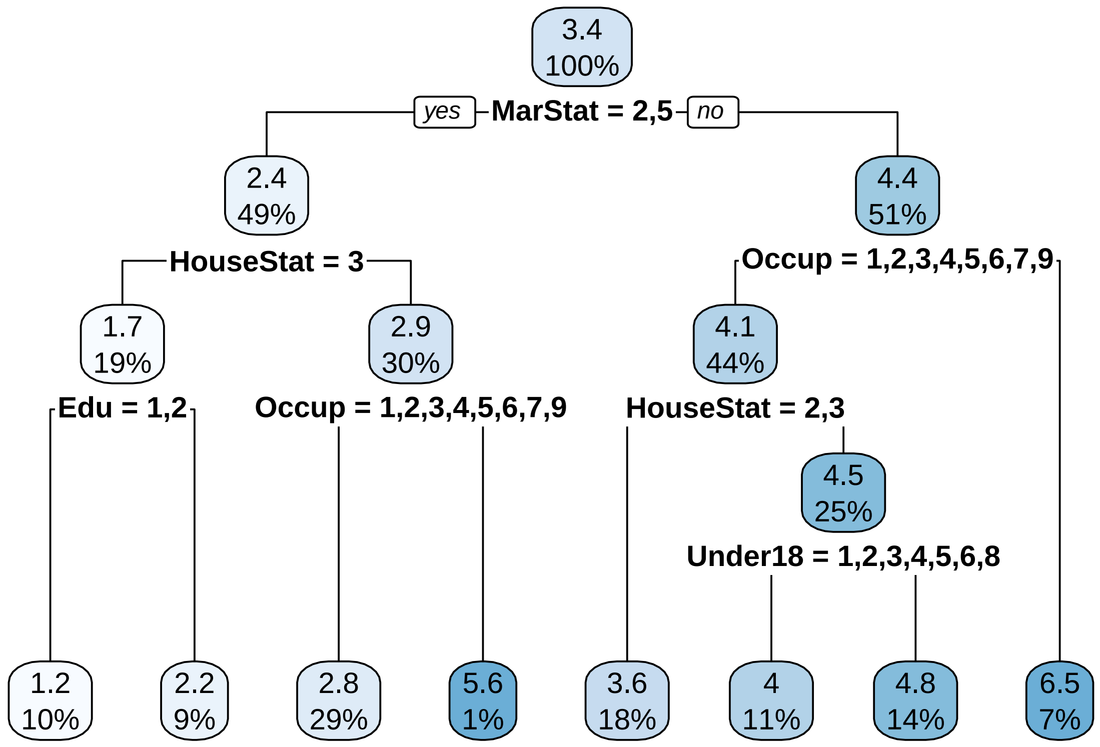
Stats 315 Homework 1

Problem 1:

```
age <- read.csv("age_stats315B.csv")

# Factors
# Occupation: 2
# Type of Home: 3
# Sex: 4
# Marital Status: 5
# Dual Incomes: 9
# Householder Status:12
# Ethnic classification: 13
# Language at home: 14
factor_columns <- c(2,3,4,5,9,12,13,14)
age[, factor_columns] <- apply(age[, factor_columns], 2, as.factor)

# Ordinal:
# Education: 6
# Anuual income: 7
# How long in SF:8
# Persons in household: 10
# Person in household under 18: 11
ordinal_columns <- c(6, 7, 8, 10, 11)
age[, ordinal_columns] <- apply(age[, ordinal_columns], 2, as.ordered)

# Train model
model <- rpart(age ~ ., data = age)

#age <- read.csv("age_stats315B.csv")
me <- c(2, 6, 1, 2, 5, 5, 2, 5, 1, 4, 0, 3, 7, 1)

rpart.plot(model)
```



Write a short report about the relation between the age and the other demographic predictors as obtained from the RPART output

The optimal tree found has a total of 7 splits, and obtains an error of 33% on the training data. Variables in order of their predictive power: Marital Status, Occupation, householder status, dual income status, income, education level, number of persons in the household under 18, Type of Home, and number of persons in the household.

The primary split divides the observations between "married, widowed, or separated" (right branch) and "never married" (left branch). The unmarried people are sorted by whether they live with their parents, whether they have finished only early education, and whether they are retired. Contrastingly, the married people are considered based on their retirement status, their housing status and the number of people in their household under 18 (presumably kids).

(a)

Were surrogate splits used in the construction of the optimal tree you obtained? What does a surrogate split mean? Give an example of a surrogate split from your optimal decision tree. Which variable is the split on? Which variable(s) is the surrogate split on?

Yes surrogate splits were used; `summary(model)` shows these surrogates (not shown). Surrogate splits are used to handle missing values. As a tree is constructed, surrogate splits are selected and stored alongside primary splits; these help decide whether an observation should fall into the right or left daughter node when the primary split variable is missing. Surrogate variables are selected at each node based on their agreement with the primary variable. In essence, a "good" surrogate split is one which would sort the data in a similar way to the primary split. For example, the primary split at Node 2 of our tree is housing status, but 92 observations at the node are missing that variable. Instead, these 92 are split based on `under 18`, which has 0.715 agreement with the primary split.

(b)

Using your optimal decision tree, predict your age.

```
# Steal datatypes
myage <- data.frame(rbind(age[1, ], me))
myage[, factor_columns] <- apply(myage[, factor_columns], 2, as.factor)
myage[, ordinal_columns] <- apply(myage[, ordinal_columns], 2, as.ordered)
myage <- myage[2, ] # just include me

myage.preds <- predict(model, myage[-1]) # exclude actual age

age_categories <- c("17 and under", "18 thru 24", "25 thru 34", "35 thru 44", "45 thru 54", "55 thru 64
print("Actual Values")
```

```
## [1] "Actual Values"
```

```
print(age_categories[round(myage$age)])
```

```
## [1] "18 thru 24"
```

```
print("Predicted values")
```

```
## [1] "Predicted values"
```

```
print(age_categories[round(myage.preds)])
```

```
## [1] "18 thru 24"
```

```
if(age_categories[round(myage$age)] == age_categories[round(myage.preds)]){
  print("That's correct!")
}
```

```
## [1] "That's correct!"
```

The model correctly predicted that Christine was 18 through 24!

Problem 2:

```
house <- read.csv("housetype_stats315B.csv")

train_ind <- sample(1:nrow(house), size = nrow(house) * .9)
test_ind <- seq(1,nrow(house))[!(seq(1,nrow(house)) %in% train_ind)]

factor_columns <- c("TypeHome", "sex", "MarStat", "Occup", "HouseStat", "Ethnic", "Lang")
ordinal_columns <- c("age", "Edu", "Income", "LiveBA", "DualInc", "Persons", "Under18")

house[,factor_columns] <- apply(house[,factor_columns], 2, as.factor)
house[,ordinal_columns] <- apply(house[,ordinal_columns], 2, as.ordered)

house_train <- house[train_ind, ]
house_test <- house[test_ind, ]

model <- rpart(TypeHome ~ ., data = house_train)

preds_train <- predict(model, house_train[,-1])
```

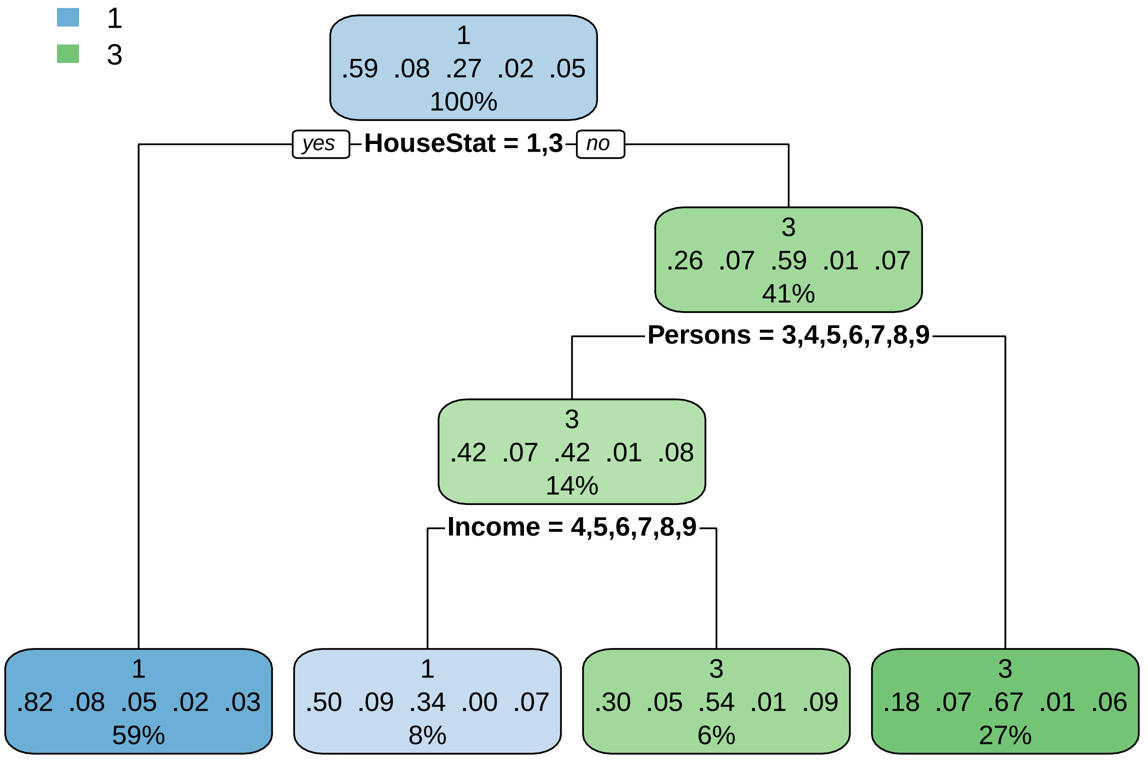
```r
preds_train <- apply(preds_train, 1, function(x) return(which(x == max(x))))
err_train <- sum(preds_train != house_train[,1]) / length(preds_train)

preds <- predict(model, house_test[,-1])
preds <- apply(preds, 1, function(x) return(which(x == max(x))))
err <- sum(preds != house_test[,1])/ length(preds)


print(paste("Training error", err_train))
```

```
## [1] "Training error 0.263222783873752"
```

```r
print(paste("Testing error", err))
```

```
## [1] "Testing error 0.254988913525499"
```

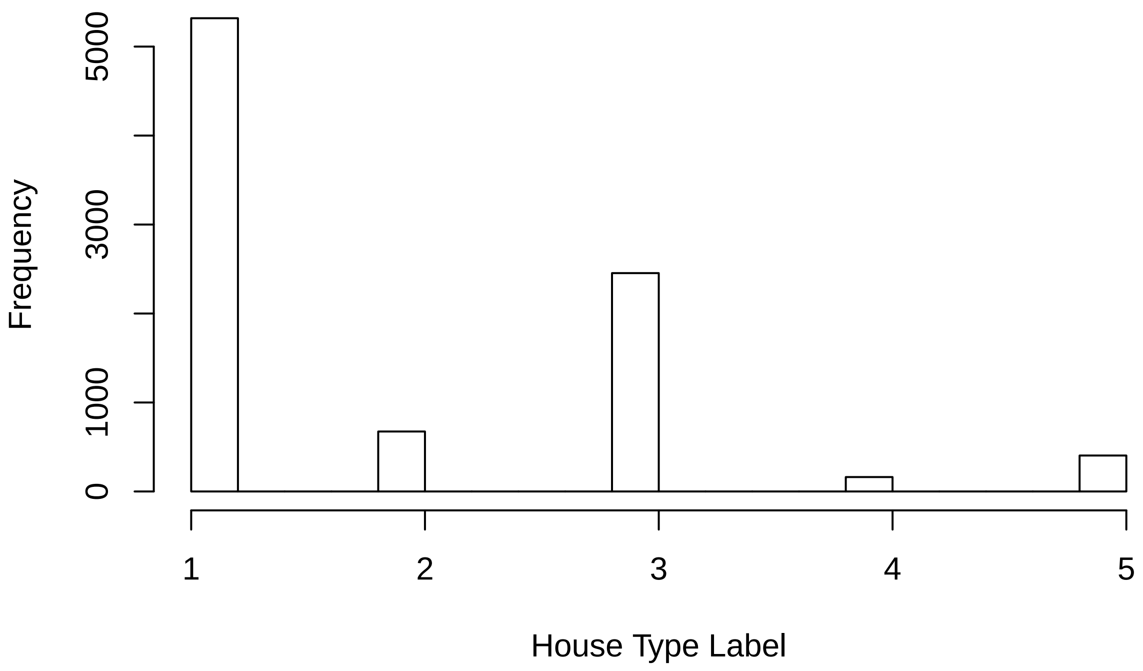
```r
rpart.plot(model)
```



Mis-classification Error Estimate: 25%

Interestingly, even though the outcome we are trying to predict has five possible values, the tree only ever predicts class 1 (house) or 3 (apartment). If we examine the trees leaf nodes, we can see that the majority class in any lead is never 2,4, or 5. This is probably because these classes are under-represented in the dataset as a whole (see below). As a result, the optimal tree minimizes empirical loss simply by training a classification scheme for types 1 and 3, and never predicting the minority classes.

```r
hist(as.numeric(house$TypeHome), main = NULL, xlab = "House Type Label")
```



Problem 3:

What are the two main reasons why a model that accurately describes the data used to build it, may not do a good job describing future data?

- 1. Data obtained in the future may be from a different distribution as that from the past (training data). For instance, if training data on stock prices was taken from 1950, it would make very bad predictions on 2010 stock data because industry emphases have changed. This phenomenon is known as concept drift.
- 2. The model may have overfit to the data used to build it, so while it can perform well on that specific data, it cannot generalize to different inputs in the same range.

Problem 4:

Why can't the prediction function be chosen from the class of all possible functions?

The optimal function chosen from the class of all possible functions would theoretically have 0 bias, which seems appealing. However, simply representing the class of all possible functions is a challenge, and choosing an optimum of a score criterion over such a massive space would be computationally impossible. Even if this problem were feasible, the variance in the optimal choice of function over this function class is likely to be astronomical.

Problem 5:

What is the definition of the target function for a given problem? Is it always an accurate function for prediction? Why/why not?

The target function for a given problem is that function, out of the class of all possible functions, that minimizes the expected prediction risk, $E_{y,\vec{x}}\left[L(y,\vec{x})\right]$. It is not always an accurate function for prediction, especially if there is no association between the inputs and the desired output quantity. For example, if we were trying to predict whether a patient will be diagnosed with cancer based on their favorite color, even the best possible predictive model for this problem formulation would still probably perform poorly.

Problem 6:

Is the empirical risk evaluated on the training data always the best surrogate for the actual (population) prediction risk? Why/why not? In what settings would it be expected to be good?

No. - 1. If the training data does not match the population distribution, empirical training risk will likely strongly underestimate prediction risk. For instance, a model for predicting cancer status built on data from 1970 which excels on the training set may perform poorly if we apply it to modernday patients. - 2. Even if the training and testing data are drawn from the same distribution, empirical risk on the training data is still an underestimate of population risk. This is because the training algorithm (more-or-less) specifically builds a model to minimize training empirical risk. Models which are fit too closely to chance fluctuations in the training data may have very low empirical risk (for example, a tree with the maximum number of splits), but very high population risk. This is why overfitting is such an important problem.

Problem 7:

Misclassification risk is defined as $E_{yx}L(y, c(x))$, the expectation over y and x of the loss incurred from predicting $c(x)$ when the answer was y. In our case, that loss is simply an indicator function that is 0 when the classifications are the same and 0 otherwise. $E_{yx}I(y \neq c(x)) = mean(I(y \neq c(x)))$ which is equivalent to the classification error rate!

The general Bayes decision rule for unequal costs is:

$$\hat{k} = min_{k \in 1...K} R(c_k|x)$$

where $R(c_k|x) = \sum_{l=1}^{K} L_{kl} P(y = c_l|x)$, and L_{kl} is the loss incurred from predicting class k when the true class is l. In our case, $L_{kl} = I(k \neq l)$.

In our case, since $L_{kl} = I(k \neq l)$, the risk of predicting c_k is

$$\sum_{l \neq k} P(y = c_l|x).$$

In essence, this is just the probability that we are incorrect when we predict c_k. This quantity is minimized when we choose the most probable label for Y given x. Thus, our Bayes rule is:

$$\hat{k} = max_{k \in 1...K} P(y = c_k|x).$$

Problem 8:

We use our Bayes decision rule as a classifier, using estimated probabilities for $P(c_l|x)$ and predicting the k that gives us the lowest risk. We get a low error rate on our predicted classes (think low test error). Does this imply accurate estimates of $P(c_l|x)$? No - 1. Imagine we knew we had unbalanced datasets. We could do a very bad job of predicting the probabilities of each class. However, we could skew the loss in such a way that the risk for the correct class was lower, even though the probability of the correct class was also low, because the penalty for misclassification of THAT class was high. - 2. During the task of classification, you lose information. Imagining a two class classification problem with 0-1 loss, you could construct a situation where your classifier obtains 0 classification error on a test set, but predicts every probability at ~49% off from it's actual value.

Problem 9:

Explain the bias-variance trade-off.

The term bias-variance trade-off encompasses the idea that selecting a function from a larger class of possible functions will likely result in a function that is closer to the target (less bias), however, because you chose from a larger pool, the function you get from using a slightly different training set can vary a lot (more variance). A lot of variation per training set means that your model will not be as generalizable to other datasets.

Problem 10:

Why not choose surrogate splits to best predict the outcome variable y rather than the primary split?

When you use surrogate splits, you lose information about the primary variable the algorithm wished to split on. You have only as much information as you have correlation between the surrogate and the primary variable.

Problem 11:

Value that minimizes the MSE is the mean:

$$\sum_{i=1}^{N}(y_i - F(x))^2 = \tag{1}$$

$$\sum_{i=1}^{N} y_i^2 - 2y_iF(x) + F(x)^2 \tag{2}$$

Taking the derivative with respect to $F(x)$ and set equal to 0. Then solve for F(x)

$$0 - 2\sum_{i=1}^{N} y_i + 2\sum_{i=1}^{N} F(x) = 0 \tag{3}$$

$$F(x) = \frac{\sum_{i=1}^{N} y_i}{n} \tag{4}$$

We see that the value of F(x) that minimizes the MSE is the mean of the y values in the set $i \in examples$

The value that minimizes the square risk criteria for a given region m, c_m, will naturally be the mean of the y values that fall within that region. This can be expressed as:

$$c_m = \frac{\sum_{i=1}^{N} y_i I(x_i \in R_m}{\sum_{i=1}^{N} I(x_i \in R_m)} \tag{5}$$

Problem 12:

Incomplete - writing down what I have, but this could be wrong/on the wrong track

The contribution of R_m to the squared error loss is

$$\sum_{i:x_i \in R_m} [y_i - \bar{y}]^2$$

So, naturally, the improvement in squared error loss is:

$$= \sum_{i:x_i \in R_m} [y_i - \bar{y}]^2 - \sum_{i:x_i \in R_{m^l}} [y_i - \bar{y}_l]^2 - \sum_{i:x_i \in R_{m^r}} [y_i - \bar{y}_r]^2$$

Expanding each of these three terms, we see that the sums of y_i terms cancel each other out:

$$\begin{aligned}
&= \sum_{i:x_i \in R_m} \left[y_i^2 - 2y_i\bar{y} + \bar{y}^2)\right] - \sum_{i:x_i \in R_{m^l}} \left[y_i^2 - 2y_i\bar{y}_l + \bar{y}_l^2)\right] - \sum_{i:x_i \in R_{m^r}} \left[y_i - 2y_i\bar{y}_r + \bar{y}_r^2\right] \\
&= \sum_{i:x_i \in R_m} \left[-2y_i\bar{y} + \bar{y}^2)\right] - \sum_{i:x_i \in R_{m^l}} \left[-2y_i\bar{y}_l + \bar{y}_l^2)\right] - \sum_{i:x_i \in R_{m^r}} \left[-2y_i\bar{y}_r + \bar{y}_r^2\right]
\end{aligned}$$

We now simplify each term independently:

$$\begin{aligned}
\sum_{i:x_i \in R_m} \left[-2y_i\bar{y} + \bar{y}^2)\right] &= -2\bar{y} \sum_{i:x_i \in R_m} y_i + n\bar{y}^2 \\
&= -2\bar{y}(n\bar{y}) + n\bar{y}^2 \\
&= -n\bar{y}^2
\end{aligned}$$

$$\begin{aligned}
\sum_{i:x_i \in R_l} \left[-2y_i\bar{y}_l + \bar{y}_l^2)\right] &= -2\bar{y}_l \sum_{i:x_i \in R_m} y_i + n\bar{y}_l^2 \\
&= -2\bar{y}_l(n\bar{y}_l) + n\bar{y}_l^2 \\
&= -n\bar{y}_l^2
\end{aligned}$$

$$\begin{aligned}
\sum_{i:x_i \in R_l} \left[-2y_i\bar{y}_r + \bar{y}_r^2)\right] &= -2\bar{y}_r \sum_{i:x_i \in R_m} y_i + n\bar{y}_r^2 \\
&= -2\bar{y}_r(n\bar{y}_r) + n\bar{y}_r^2 \\
&= -n\bar{y}_r^2
\end{aligned}$$

The improvement in the squared error loss can now be expressed as:

$$\begin{aligned} -n\bar{y}^2 - (-n\bar{y}_l^2) - (-n\bar{y}_r^2) \\ -n\bar{y}^2 + n\bar{y}_l^2 + n\bar{y}_r^2 \end{aligned}$$

We now express $\bar{y}$ in terms of y_l and y_r

$$\begin{aligned} \bar{y} &= \frac{1}{n}(n_l\bar{y}_l + n_r\bar{y}_r) \\ \bar{y}^2 &= \frac{1}{n^2}(n_l^2\bar{y}_l^2 + 2n_ln_r\bar{y}_l\bar{y}_r + n_r^2\bar{y}_r^2) \end{aligned}$$

and substitute into our expression for the improvement

$$\begin{aligned} &\frac{-1}{n}(n_l^2\bar{y}_l^2 + 2n_ln_r\bar{y}_l\bar{y}_r + n_r^2\bar{y}_r^2) + n\bar{y}_l^2 + n\bar{y}_r^2 \\ &\bar{y}_l^2(\frac{-n_l^2}{n} + n_l) + \bar{y}_r^2(\frac{-n_r^2}{n} + n_r) - 2n_ln_r\bar{y}_l\bar{y}_r \end{aligned}$$

Lastly, we simplify $(\frac{-n_l^2}{n} + n_l)$. The same logic can be applied to $(\frac{-n_r^2}{n} + n_r)$

$$\begin{aligned} (\frac{-n_l^2}{n} + n_l) &= n_l(1 - \frac{n_l}{n}) \\ &= n_l(\frac{n}{n} - \frac{n_l}{n}) \\ &= n_l(\frac{n_l + n_r}{n} - \frac{n_l}{n}) \\ &= n_l(\frac{n_r}{n}) \\ &= \frac{n_ln_r}{n} \end{aligned}$$

And Substitute:

$$\begin{aligned} &\bar{y}_l^2(\frac{n_ln_r}{n}) + \bar{y}_r^2(\frac{n_ln_r}{n}) - 2n_ln_r\bar{y}_l\bar{y}_r \\ &\frac{n_ln_r}{n}(\bar{y_l}^2 - \bar{y_r}^2) \end{aligned}$$

Problem 13:

When a new number, x_0 is added to a set, the new mean $\bar{x}_{new}$ is:

$$\bar{x}_{new} = \frac{n\bar{x}_{old} + x_0}{n+1}$$

Likewise removing an observation gives us:

$$\bar{x}_{new} = \frac{n\bar{x}_{old} - x_0}{n-1}$$

Thus, we can use the following update rule when y_i moves from the left to the right partition:

$$\bar{y}_l \leftarrow \frac{n_l\bar{y}_l - y_i}{n_l - 1}$$

$$\bar{y}_r \leftarrow \frac{n_r\bar{y}_r + y_i}{n_r + 1}$$

$$n_l \leftarrow n_l - 1$$

$$n_r \leftarrow n_r + 1$$

Finally, we just recalculate the change in prediction risk:

$$\frac{n_r n_l}{n}(\bar{y}_l - \bar{y}_r)^2$$

Problem 14:

Increasing the size of the function class is not always a good idea, even if allows us to achieve a smaller training error. There are two main reasons this might hurt:

1. It may be more difficult to optimize over a larger class of functions.
2. Although increasing the size of the function class often decreases bias, it tends to increase variance. In essence, even though the optimal f within the larger function class may be closer to the target, our actual $\hat{f}$ after training may vary wildly depending on chance patterns in the training data. This may cause our mse on future data to increase, rather than decrease.

Likewise, decreasing the size of the function class is also not always the best path. This may make the selection of $\hat{f}$ simpler and decrease the variance in the $\hat{f}$ we choose, but it may also introduce bias. Inessense, we may be able to choose the optimum f over a smaller function class more easily and consistently, but that optimum may be far from the target simply because the function class is small.

In reality, we need to make decisions about our function class in order to balance bias and variance (and maintain feasibility).

Problem 15:

Advantages: 1. By splitting in a way with higher resolution (more than just in a binary way), likely one will be able to achieve improved accuracy on the training data. 2. Enabling more splits per node at a variable right away may be easier/more effective than downstream splitting on different variables. For example if the dataset has one predictor variable completely correllated with the outcome and the predictor variable takes on values in $n > 2$ different intervals, then splitting on that variable in more than a binary way immediately achieves perfect accurcacy.

Disadvantages: 1. If the number of splits at a node is not prespecified, this may greatly increases the computational burden of splitting and recursive searching, and if the we require $k > 2$ splits per node, this may induce redundancy as maybe a binary split was sufficient for making a decision over a specific variable. 2. Allowing for a greater number of splits per node reduces the number of data points available at each child tree, so having more splits may lead to overfitting in the end.

Problem 16:

Advantages: The primary advantage in this case is that by having linear combination splits, we would almost certainly increase accuracy in our trees. Especially since a major disadvantage of regression trees is that we're restricted to piecewise continuous functions, enabling piecewise linear splitting might greatly increase accuracy especially in regions with a larger range or variance of output values.

Disadvantages: 1. Inevitably, this strategy would induce a greater computational burden in calculating optimal separating hyperplanes. 2. This strategy might lead to overfitting of the data if we don't regularize the procedure somehow. 3. Because the space of functions we're considering (piecewise linear, which is a superset of piecewise constant), the variance of our estimate will increase, so we're more vulnerable to biases in our dataset. 4. Perhaps the greatest cost is that our predictions become much less naturally interpretable. Instead of having a set of statements about where each variable in certain intervals or categories as a way to make predictions, we instead have sets of statements about how linear combinations of variables fall into one half-space or another, which is much less obviously interpretable than in the simple splits case.