CS 4375 Assignment 2 Solution

April 1, 2022

1 Nearest Neighbors

1.1 As Regression

It is possible to use k-nearest neighbors as regression, but regression solutions using k-nearest neighbors tend to be clunky and suboptimal. Simply, instances in the training dataset are labeled with values from $\mathbb{R}$ instead of a finite set of classes. Regression can easily be done with 1-nearest neighbors, where the label of the closest point in the training dataset is selected and returned. This scheme can possibly be extended to k-nearest neighbors for $k > 1$, where the average, median, or some other combination of the k nearest instances' labels is returned.

Even with these schemes, k-nearest neighbors is typically never used for regression. What happens when an instance far away from all points in the training dataset is evaluated during inference? Can we really attribute the labels of the nearest point(s) as the predicted value for this instance? In a majority of regression problems, points that are very different (far) from one another also have very different predicted/ground-truth values. As k-nearest neighbors cannot handle this issue, it is almost never used for regression. Another notable issue is that $\mathbb{R}$ has infinitely many values while k-nearest neighbors can only predict finitely many values. Hence, solutions such as linear regression, linear regression with polynomial features, and others tend to be much better choices.

1.2 Irrelevant Attributes

As discussed in class, k-nearest neighbors does not handle irrelevant attributes well. Consider a 3-nearest neighbors classifier that classifies one-dimensional points as depicted in Figure 1. Instances in the training set are filled, and the test instance is dashed.

Figure 1: An ideal 3-nearest neighbors classifier. The dashed point has a ground-truth red label. Hence, the dashed point is correctly classified as a red point.

We see that the 3-nearest neighbors classifier will assign a red classification to the test instance. What happens if we introduce an irrelevant feature to our training instances? Figure 2 shows a suboptimal 3-nearest neighbors classifier on the same instances that now have a y-component (feature).

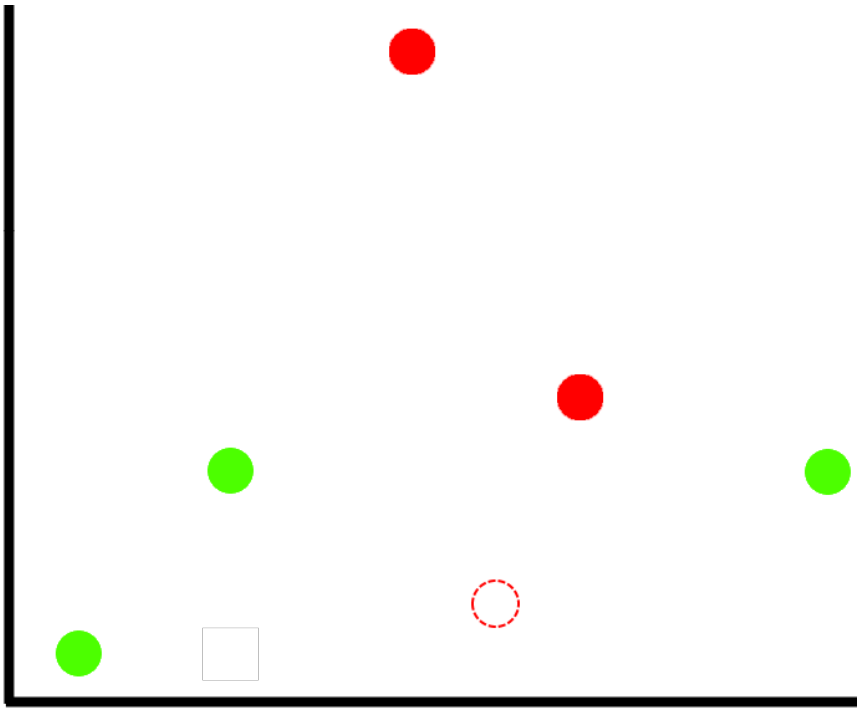

Figure 2: A suboptimal 3-nearest neighbors classifier. The dashed point has a ground-truth red label. However, the 3-nearest neighbors classifier assigns a green label, which does not match the ground-truth red label.

By introducing a y-component, the 3-nearest neighbors classifier now predicts that the test instance is green even though it has a red ground-truth label. This is because the two green training instances become closer than the second red training instance at the top of Figure 2. Hence, the irrelevant feature causes the 3-nearest neighbors classifier to make the wrong prediction. In general, we should remove all irrelevant features from the data before applying k-nearest neighbors.

1.3 Implementation

The code solution gives an implementation of k-nearest neighbors using NumPy.

2 Decision Trees

2.1 Information Gain and Conditional Gain

Recall that information gain is computed as the following:

$$I(X;Y) = H(Y) - H(Y|X)$$

where $H(Y)$ is the entropy of the random label variable Y and $H(Y|X)$ is the conditional entropy of Y given random feature variable X. Since entropy measures uncertainty in a random variable, our

objective should be to make decisions on the random feature variables $(X_1, X_2, ...)$ that provide the largest difference in uncertainty on the random label variable (Y). This is exactly what is captured by $I(X;Y)$. To given an example why maximizing $I(X;Y)$ is beneficial, let us examine the simple example in Figure 3.

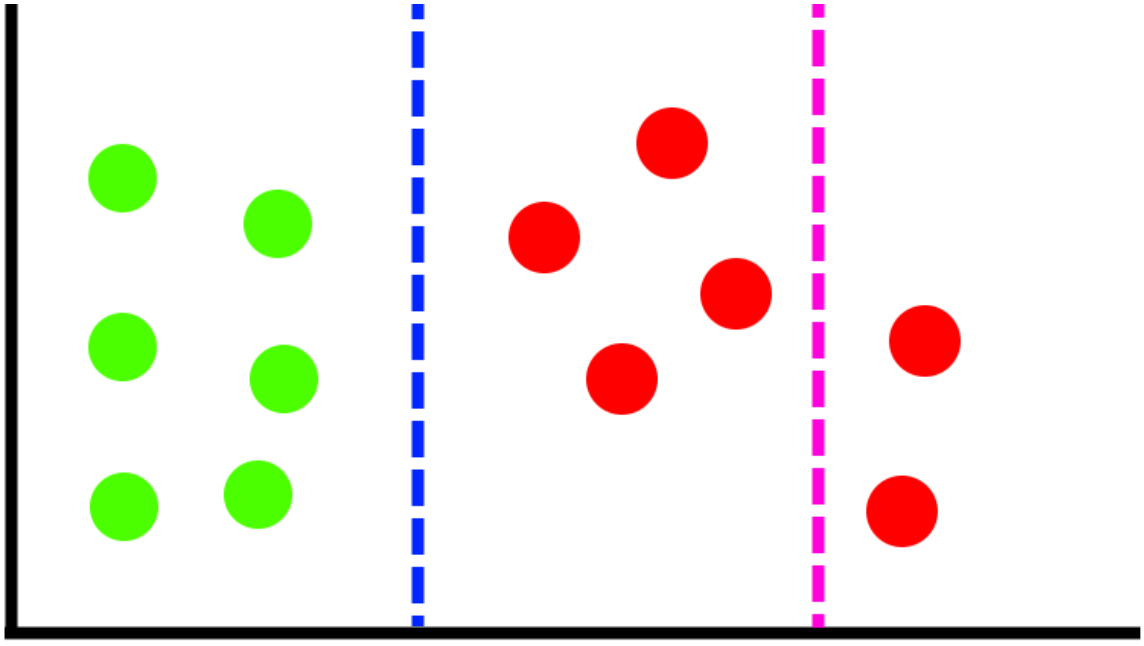

Figure 3: Two possible decisions to make in a decision tree. The blue line maximizes the information gain while the pink line does not. Clearly, the blue line's decision gives a definite answer; the pink line's decision still requires more deliberation if the classifier chooses the left side.

We see that $H(Y)$ is maximal since there is a 50/50 split between green points and red points (see lecture slides). To reduce this uncertainty, we make one of two decisions given by the blue and pink lines of Figure 3. The one that has maximum information gain is the same one that minimizes the conditional entropy. Recall that the conditional entropy is minimized when observing a particular value of Y becomes certain after observing one of $X_1, X_2, ...$ Hence, we should pick the blue line. The resulting tree decides the label in only one decision using this strategy. Had we picked the pink line, which does not result in maximum information gain, the decision tree would still need to make more decisions for all the points to the left of the pink line.

As we stated earlier, minimizing the conditional entropy $H(Y|X)$ is equivalent to maximizing the information gain. We can compare objectives:

$$\text{argmax}_X H(Y) - H(Y|X)$$

Since $H(Y)$ is a constant, the result is unchanged if we remove it:

$$\text{argmax}_X - H(Y|X) = \text{argmin}_X H(Y|X)$$

So, minimizing the conditional entropy is the same as maximizing the information gain.

2.2 Drawing a Decision Boundary

We saw that making splits that make the label prediction more certain maximizes information gain. Hence, we can apply the same strategy here to draw a decision boundary. This implies that we should draw lines that are right next to the green marks; more specifically, there should be no line that is tangential to a circle within the red ring of points. Assuming that the max depth is not bounded, a possible decision tree is given in Figure 5, which provides a decision boundary between red and green marks as shown in Figure 4. The decision tree was constructed with the following in mind:

1. The most red marks that can be initially split from the green marks by a horizontal or vertical line is to the left of the green square (roughly 12 red marks). This will minimize the conditional entropy the most for the previously listed reason, so it is the first decision (red line). As all points to the left of the line are red marks, we do not need to build a subtree for these points.

2. The most red marks to the right of the previous line that can be split from the green marks is to the right of the green square (roughly 10 red marks). For the same reason as before, we create another vertical line (given in orange). All points to the right of this new line are red marks, so we do not need to build a subtree for these points.

3. We can now either draw a horizontal line separating the top remaining red marks or the bottom remaining red marks. I choose the top portion. All points above this new line (given in blue) are red marks, so we do not need to build a subtree for these points.

4. We draw the final separating line (given in green) at the bottom. All points are now officially separated, ending the decision tree.

Based on Figure 5, we can clearly see how the max depth parameter might affect the structure. If we did have a max depth that was smaller than the depth of the tree given in Figure 5, then the right subtree at the max depth would be truncated. The subtree would then be replaced by a majority vote of the examples left in that space. Here's what different values of the max depth parameter would do:

1. If we had a max depth of 0 (not counting the classifications as nodes), we would keep only the root decision. However, either answer to the decision would predict red since the majority vote that replaces the right subtree is red.

2. If we had a max depth of 1, we would keep the top two decisions. Unlike before, the second decision will have a majority vote of green that replaces the right subtree.

3. If we had a max depth of 2, we would keep the top three decisions. Again, the third decision will have a majority vote of green that replaces the right subtree.

4. If we had a mad depth of 3 or greater, the decision tree remains unchanged. This is because the original tree perfectly classifies the space.

2.3 Max Depth in Decision Trees

We know that deeper decision trees allow us to classify our training dataset more accurately, resulting in lower bias. However, if our trees are too deep, then it becomes very easy for them to overfit the training data, resulting in poor test performance. Hence, how should practitioners handle the max depth hyperparameter? Let d be the max depth parameter. We study two questions here:

1. If $d = d_1$ and $d = d_2$ both produce decision trees with the same validation accuracy and $d_1 > d_2$, which decision tree should be chosen? Since the validation accuracy is the same, it gives the impression that either tree is suitable. However, we know that deeper trees have a higher likelihood to overfit than shallower trees. Since both trees are giving the same performance, we should choose the shallower tree as it is more likely to generalize better than the deeper tree (Occam's razor argument!). Accordingly, statement A) is correct while statement B) is incorrect.

2. Will a larger max depth parameter make it more likely to overfit? As previously stated, allowing for more decisions makes it easier to fit the decision tree more accurately on the training data. Consequently, this makes it easier to overfit the training data (bias-variance tradeoff). Hence, statement C) is correct while statement D) is incorrect.

3 Maximum a Posteriori Estimation

As we saw in class, maximum likelihood estimation can result in parameter values that are not truly believable given our *prior* knowledge about what value the parameter should take. The clear example was when we observed five heads after flipping a coin five times – the MLE estimate was $p_{heads} = 1$, which we know is not likely to occur when flipping a coin. Hence, we can incorporate our prior knowledge into the MLE objective via Bayes Rule:

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

where $p(\theta)$ is a prior distribution on the parameter θ (encoding our prior beliefs), $p(D|\theta)$ represents the data likelihood given parameter θ, and $p(\theta|D)$ represents the posterior distribution of θ having observed the data D ($p(D)$ can be viewed as a normalizing constant). Hence, while MLE solves

$$\text{argmax}_\theta p(D|\theta),$$

we instead solve the objective

$$\text{argmax}_\theta p(\theta|D) = \text{argmax}_\theta p(D|\theta)p(\theta),$$

which gives the MAP estimation. We now identify $p(D|\theta)$ and $p(\theta)$ from the problem statement, which are as follows:

$$p(D|\theta) = p(D|\mu, \sigma) = \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}$$
$$p(\mu) = \frac{1}{\beta\sqrt{2\pi}} e^{\frac{-(\mu-\nu)^2}{2\beta^2}}.$$

The data likelihood $p(D|\theta)$ is calculated using the fact that the data is independently, identically distributed according to a Gaussian distribution with mean μ and variance σ^2. The prior $p(\mu)$ is calculated using the fact that the mean (μ) is distributed according to a Gaussian distribution with mean ν and variance β^2. Because we do not specify a prior for σ, we assume that it has a uniform prior (like in maximum likelihood estimation) and therefore does not affect the optimization objective. The objective, then, is as follows:

$$\text{argmax}_\theta p(D|\theta)p(\theta) = \text{argmax}_{\mu,\sigma} \frac{1}{\beta\sqrt{2\pi}} e^{\frac{-(\mu-\nu)^2}{2\beta^2}} \times \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}.$$

We can make this easier to solve if we instead compute the log probability, which is an equivalent objective:

$$\begin{aligned}
&\text{argmax}_{\mu,\sigma} \log\left(\frac{1}{\beta\sqrt{2\pi}} e^{\frac{-(\mu-\nu)^2}{2\beta^2}} \times \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}\right) \\
&= \\
&\text{argmax}_{\mu,\sigma} \log\left(\frac{1}{\beta\sqrt{2\pi}} e^{\frac{-(\mu-\nu)^2}{2\beta^2}}\right) + \sum_{i=1}^{N} \log\left(\frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}\right) \\
&= \\
&\text{argmax}_{\mu,\sigma} \log\left(\frac{1}{\beta\sqrt{2\pi}}\right) + \log\left(e^{\frac{-(\mu-\nu)^2}{2\beta^2}}\right) + N\log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) + \sum_{i=1}^{N} \log\left(e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}\right) \\
&= \\
&\text{argmax}_{\mu,\sigma} \log\left(\frac{1}{\beta\sqrt{2\pi}}\right) + \frac{-(\mu-\nu)^2}{2\beta^2} + N\log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) + \sum_{i=1}^{N} \frac{-(x_i-\mu)^2}{2\sigma^2}.
\end{aligned}$$

This objective is concave, so we can exactly find μ and σ by solving for a stationary point. Since we are only interested in solving for μ, we will only calculate the partial derivative with respect to μ, set it to 0, and solve:

$$\begin{aligned}
\frac{\partial}{\partial\mu}\left[\log\left(\frac{1}{\beta\sqrt{2\pi}}\right) + \frac{-(\mu-\nu)^2}{2\beta^2} + N\log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) + \sum_{i=1}^{N} \frac{-(x_i-\mu)^2}{2\sigma^2}\right] &= \frac{\nu-\mu}{\beta^2} + \sum_{i=1}^{N} \frac{x_i-\mu}{\sigma^2} \\
&= \frac{\nu-\mu}{\beta^2} - \frac{N\mu}{\sigma^2} + \sum_{i=1}^{N} \frac{x_i}{\sigma^2}.
\end{aligned}$$

Solving when it is 0 gives

$$\frac{\nu-\mu}{\beta^2}-\frac{N\mu}{\sigma^2}+\sum_{i=1}^{N}\frac{x_i}{\sigma^2}=0$$

$$\Longrightarrow$$

$$\frac{\nu-\mu}{\beta^2}-\frac{N\mu}{\sigma^2}=-\sum_{i=1}^{N}\frac{x_i}{\sigma^2}$$

$$\Longrightarrow$$

$$\sigma^2\nu-\sigma^2\mu-\beta^2N\mu=-\beta^2\sum_{i=1}^{N}x_i$$

$$\Longrightarrow$$

$$-(\beta^2N+\sigma^2)\mu=-\sigma^2\nu-\beta^2\sum_{i=1}^{N}x_i$$

$$\Longrightarrow$$

$$\mu=\frac{\sigma^2\nu+\beta^2\sum_{i=1}^{N}x_i}{\beta^2N+\sigma^2},$$

which is the MAP estimator of the mean. Hence,

$$\mu_{MAP}=\frac{\sigma^2\nu+\beta^2\sum_{i=1}^{N}x_i}{\beta^2N+\sigma^2}.$$

Now, we are interested in what happens to μ_{MAP} as $N\to\infty$; that is, we are interested in what happens to μ_{MAP} as we see more and more data. As N gets larger and larger, we see that the numerator is dominated by the summation and that the denominator is dominated by β^2N. Hence, when N is large,

$$\mu_{MAP}\approx\frac{\beta^2\sum_{i=1}^{N}x_i}{\beta^2N}=\frac{\sum_{i=1}^{N}x_i}{N}=\mu_{MLE},$$

which matches the same observation we made about the coin-flip scenario in the lecture notes: the effect of the prior diminishes as we see more and more data, drawing the MAP estimation closer to the MLE estimation.

4 Decision Tree Implementation

Refer to the code solution for a possible implementation of decision trees.

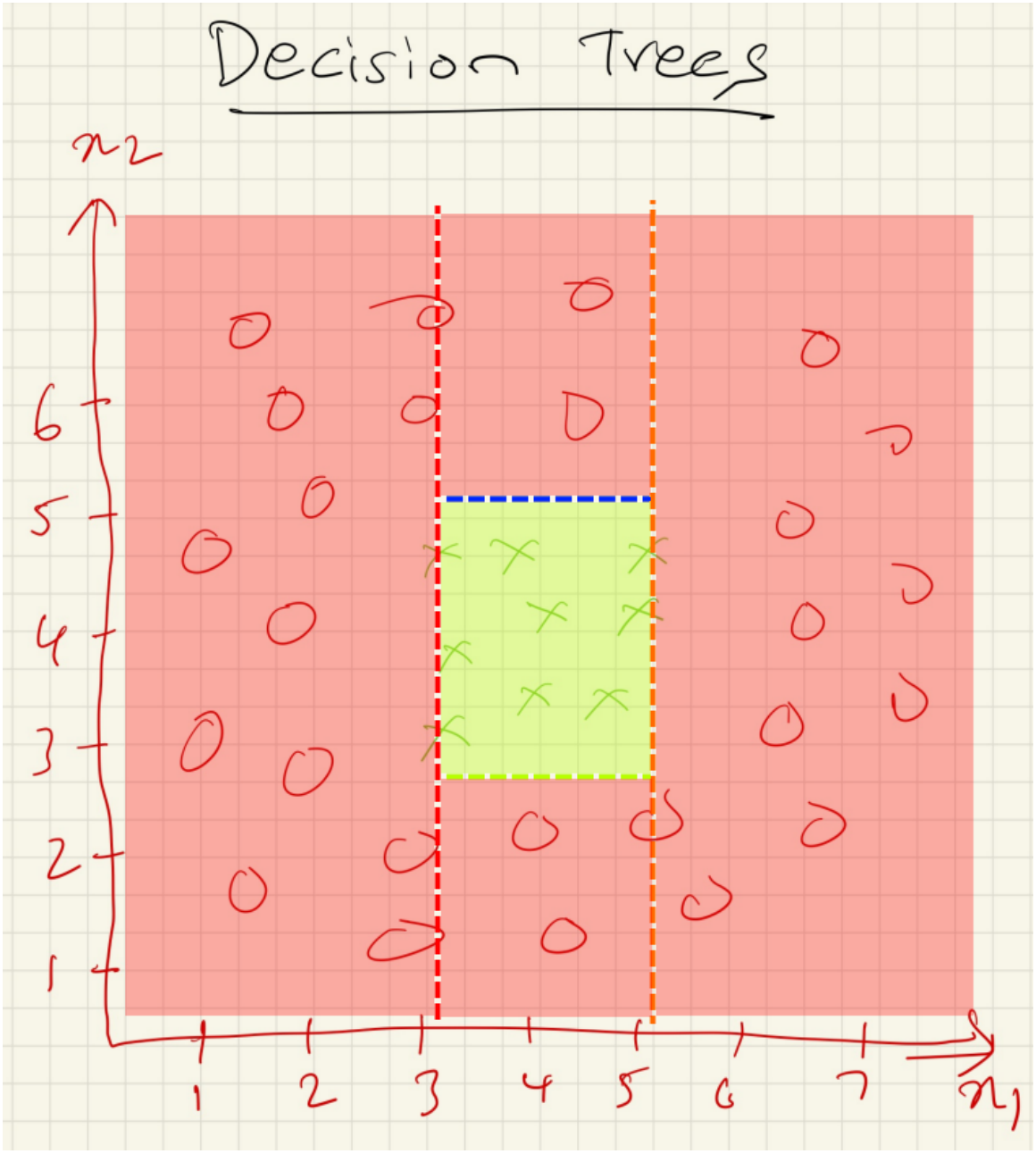

Figure 4: The decision boundary of the generated decision tree. The red line is the first decision made; the orange line is the second decision made; the blue line is the third decision made; and the green line is the fourth decision made. Shaded regions give the classification made by this tree.

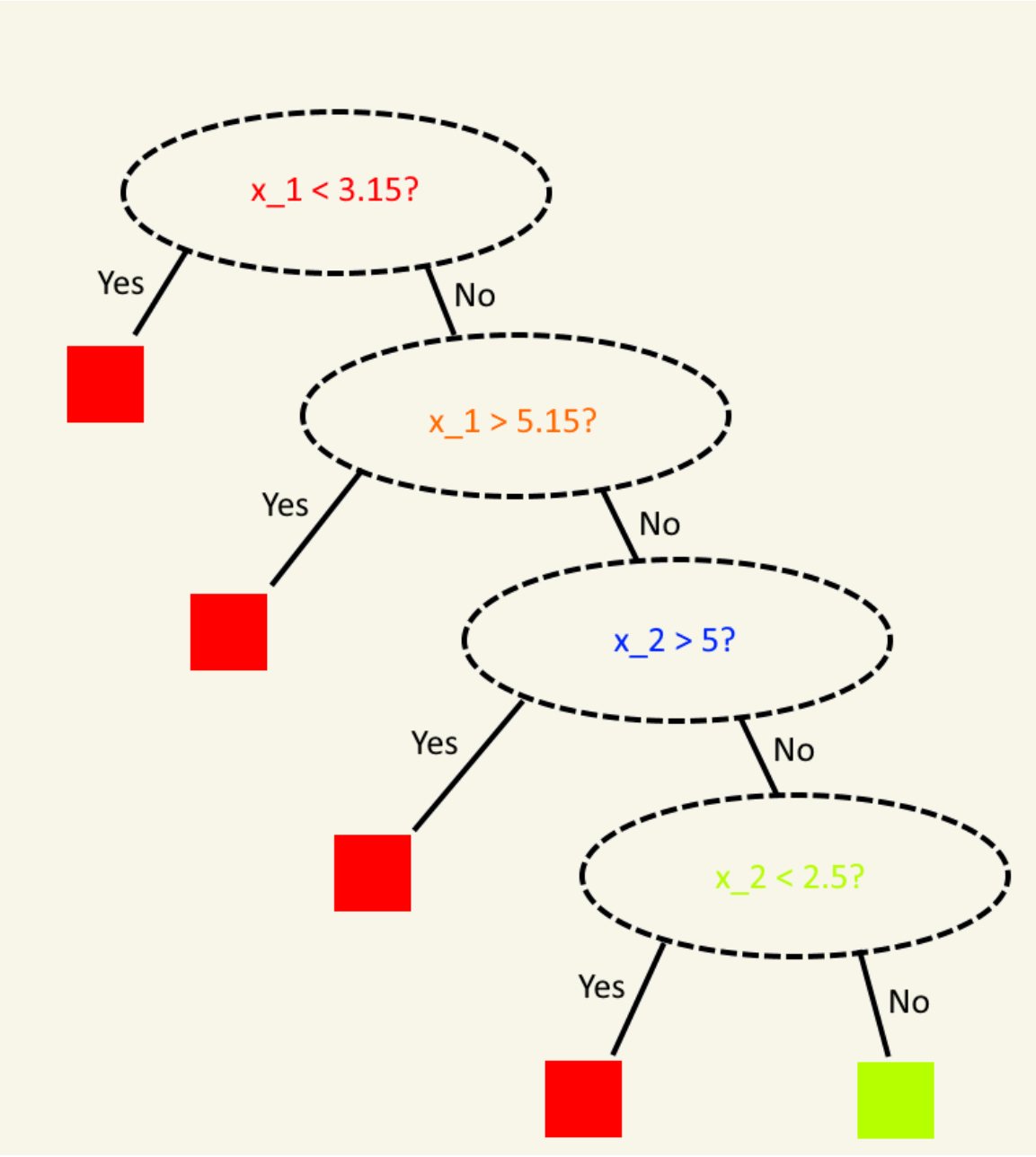

Figure 5: The resulting decision tree generated by maximizing the information gain. The decisions are color-coded after the decision boundary figure.