Classification

Lecture 7

Connor Dowd

April 20th, 2021

Today's Class

1. Review
   - LASSO
   - Deviance vs MSE
   - Cross-Validation
   - R help docs – An explainer
2. KNN
3. Binomial Classificiation: Probabilities to Predictions
4. Misclassification: Sensitivity, Specificity, ROC Curve
5. Multinomial Regression

Quick Review

LASSO

The lasso penalty is the l_1 norm of our parameters: $pen(\beta) = \sum |\beta_j|$, which penalizes larger coefficients and non-zero coefficients. So our estimates are:

$$\hat{\beta}_\lambda = \underset{\beta}{\text{argmin}} \left(Dev(\beta) + \lambda \sum_{j=1}^{p} |\beta_j| \right)$$

For a given model of the error terms, $Dev(\beta)$ is held fixed, so we can make comparisons within the logistic or linear regressions, but not between them easily.

When we switch logistic $\rightarrow$ linear, we change Dev. We can still make comparisons betwen these models but we need to ensure we are comparing the same thing.

LASSO – MSE vs Deviance

For the logistic LASSO, our deviance is approximately:

$$Dev_{logit}(\beta) \propto \sum_{i=1}^{n}[log(1 + exp(x_i'\beta)) - y_i x_i'\beta]$$

While for our linear LASSO, our deviance is approximately:

$$Dev_{linear}(\beta) \approx \frac{1}{\sigma^2}\sum_{i=1}^{n}(y - x'\beta)^2 \propto \widehat{MSE}$$

This difference derives from the different *likelihoods* each model uses to model their different ideas about the error distribution.

But there is only one true error distribution.

Model Choice

When deciding which model to use, we used the out-of-sample deviance to select a value of λ for each LASSO type. But we still need to choose between these two very different models.

That choice is still driven by out of sample prediction errors. We need to compare the prediction errors using the same yardstick.

So we could compare all the prediction errors using the binomial deviance, or using the MSE, or using some other loss function.

Model Choice

Critically (and we will see this again), cross-validation gives us a good understanding of our out-of-sample error.

We can answer questions like "When I build a model in this way, what do its out of sample errors look like?"

Which means we can also answer questions like "Which model building process has better out-of-sample predictions?"

All of this is because of the very trustworthy error distributions we get from Cross-validation.

R help documents.

I've realized I should give you a quick overview of how the R help documents are structured.

There are three main questions I turn to the help documents with.

1. Where in the output is x?
2. What do I put in the input to do y?
3. How do I do z?

By far, the help documents are most useful for (1) and (2).

```
?cv.glmnet
```

Basic Structure of Help Docs

1. **Description**: Basic explanation of the function in question.

Basic Structure of Help Docs

1. **Description**: Basic explanation of the function in question.
2. **Usage**: This shows the function (and close relatives), as well as most of the arguments to the function, and their defaults.

Basic Structure of Help Docs

1. **Description**: Basic explanation of the function in question.
2. **Usage**: This shows the function (and close relatives), as well as most of the arguments to the function, and their defaults.
3. **Arguments**: This lists out every argument you can enter, describing what it must be, and what it controls.

Basic Structure of Help Docs

1. **Description**: Basic explanation of the function in question.
2. **Usage**: This shows the function (and close relatives), as well as most of the arguments to the function, and their defaults.
3. **Arguments**: This lists out every argument you can enter, describing what it must be, and what it controls.
4. **Details**: In-depth description of the function, often including further detail about inputs, sometimes math.

Basic Structure of Help Docs

1. **Description**: Basic explanation of the function in question.
2. **Usage**: This shows the function (and close relatives), as well as most of the arguments to the function, and their defaults.
3. **Arguments**: This lists out every argument you can enter, describing what it must be, and what it controls.
4. **Details**: In-depth description of the function, often including further detail about inputs, sometimes math.
5. **Value**: In depth description of the output of the function.

Basic Structure of Help Docs

1. **Description**: Basic explanation of the function in question.
2. **Usage**: This shows the function (and close relatives), as well as most of the arguments to the function, and their defaults.
3. **Arguments**: This lists out every argument you can enter, describing what it must be, and what it controls.
4. **Details**: In-depth description of the function, often including further detail about inputs, sometimes math.
5. **Value**: In depth description of the output of the function.
6. **Authors/Refs**: People and places with even more details

Basic Structure of Help Docs

1. **Description**: Basic explanation of the function in question.
2. **Usage**: This shows the function (and close relatives), as well as most of the arguments to the function, and their defaults.
3. **Arguments**: This lists out every argument you can enter, describing what it must be, and what it controls.
4. **Details**: In-depth description of the function, often including further detail about inputs, sometimes math.
5. **Value**: In depth description of the output of the function.
6. **Authors/Refs**: People and places with even more details
7. **See Also**: other closely related functions. Many model tools will mention "`plot`, `predict`, and `coef`" methods for `cv.glmnet`. That implies functions like `coef.cv.glmnet` and `plot.cv.glmnet` you can look up.

Basic Structure of Help Docs

1. **Description**: Basic explanation of the function in question.
2. **Usage**: This shows the function (and close relatives), as well as most of the arguments to the function, and their defaults.
3. **Arguments**: This lists out every argument you can enter, describing what it must be, and what it controls.
4. **Details**: In-depth description of the function, often including further detail about inputs, sometimes math.
5. **Value**: In depth description of the output of the function.
6. **Authors/Refs**: People and places with even more details
7. **See Also**: other closely related functions. Many model tools will mention "`plot`, `predict`, and `coef`" methods for `cv.glmnet`. That implies functions like `coef.cv.glmnet` and `plot.cv.glmnet` you can look up.
8. **Examples**: example code running the function on basic data.

Classification

Basic Setting

Just as in our basic prediction problems, we have data with n observations $(\mathbf{x}_i, y_i)$ of something.

But now y_i is qualitative rather than quantitative. Membership in some category $\{1, ..., M\}$

The basic problem then is the following: Given new observation covariates $\mathbf{x}_i^{new}$, what is the class label y_i^{new}?

The quality of any classifier can be determined by its misclassification risk:

$$P[\hat{y}_i^{new} \neq y_i^{new}]$$

Motivation

How does this differ from basic logistic question of predicting $P[y_i^{new} = 1]$?

1. We may have many categories, not just two.
2. We may have different discrete actions we take depending on classification.
   - In some domains, as $P[y_i = 1]$ changes, our actions change smoothly.
     - e.g. as P[stock x goes up] we buy more of it.
   - In other domains, as $P[y_i = 1]$ changes, our actions change suddenly.
     - e.g. as P[get into college i|I apply] goes up, we switch from "don't apply" to "apply", there is mostly no "apply a little bit more"

Motivation

This leads to situations where we want to categorize, as it affects our decisions.

We face decisions not on a spectrum, so the most useful interpretation of our predictions may not be on that spectrum.

$\Longrightarrow$ classification.

Optimal Classifier

Presuming you have no preference between different types of misclassification (**LOSS FUNCTION CLAIM**), there is an optimal classifier, known as the Bayes Classifier.

$$\hat{y}_i^{new} = \underset{j \in \{1,\ldots,M\}}{\text{argmax}} \; P[y_i^{new} = j | \mathbf{x}_i^{new}]$$

Find the prediction which is *most* likely (not necessarily all that likely – e.g. $P[\hat{y} = y] < 0.5$ is common). This will minimize the misclassification risk.

Bayes Classifier

Unfortunately, we don't know $P[y = j|\mathbf{x}_i^{new}]$. So the Bayes classifier is in some ways an unattainable standard.

But we can estimate it!

Estimating P

There are many tools for estimating $P[y|x]$ given the training data.

- We can use parametric tools.
  - Assume $P[y|x]$ is some function of unknown parameters β, estimate those, and make predictions.
    - Sound familiar? Logistic Regression does this.
- We can use non-parametric tools.
  - Estimate $P[y|x]$ directly without any parameters.
    - K Nearest Neighbors (KNN)

KNN

KNN Basics

Basic Idea: Estimate $P[y|x]$ locally using the labels of similar observations in the training data.

KNN: What is the most common class near x^{new}?

1. Take the K nearest neighbors $x_{i,1}, ..., x_{i,K}$ of x^{new} in the training data
   - Nearness is (usually) Euclidean distance: $\sqrt{\sum_{j=1}^{p}(x_j^{new} - x_{i,k,j})^2}$
2. Estimate $P[y = j|x] = \sum_{i=1}^{K} 1(y_i = j)$
3. Select the class j with the highest probability.

KNN – Details

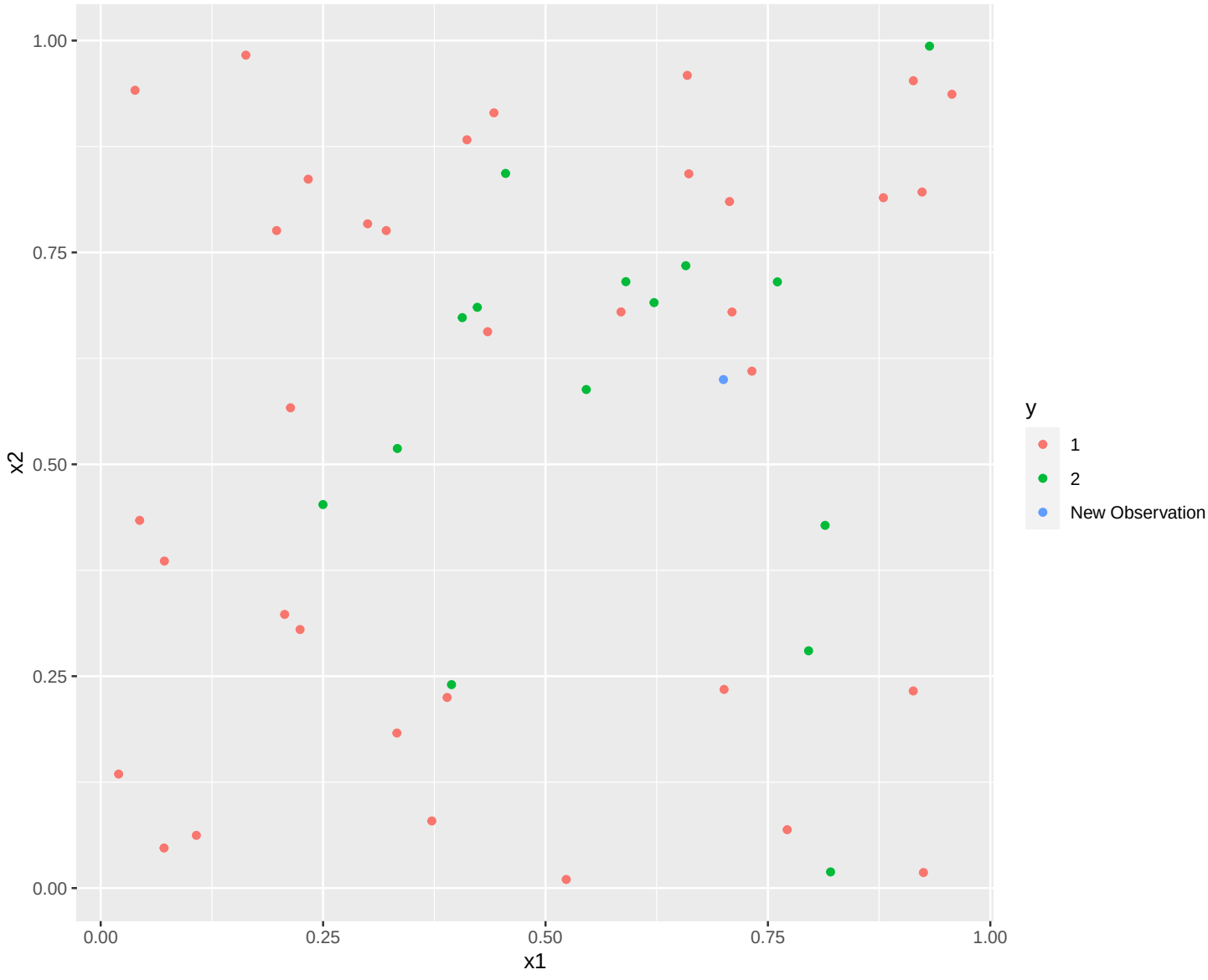
This (again) is sensitive to the scale of each covariate x. So we will rescale them all by standard deviations.

This will be sensitive to K, and we need to pick it.

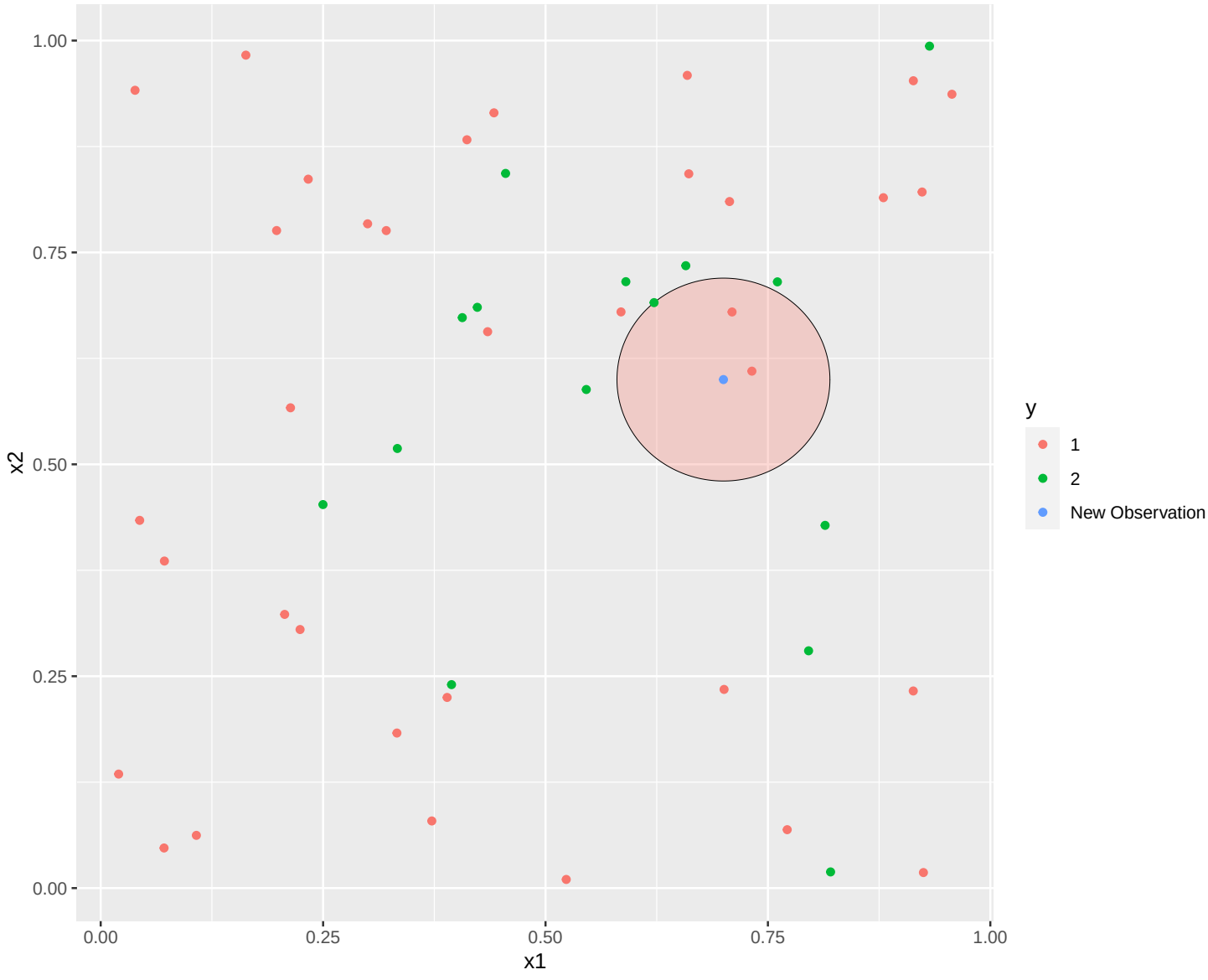
- $K = n$ – we just take the mean across the entire training data.
- $K = 1$ – Whatever observation happens to be closest will be our prediction.

Cross-validation here will help.

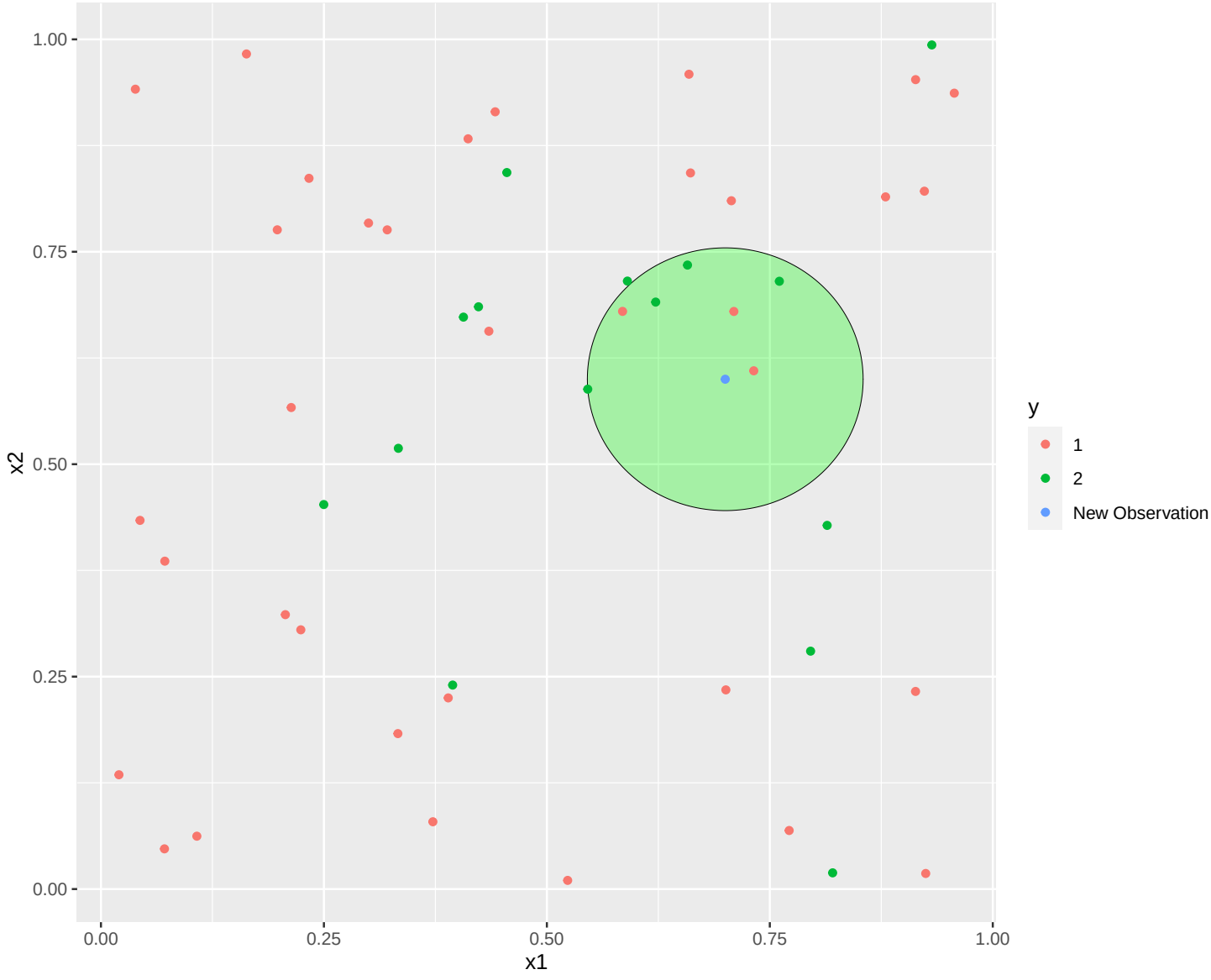
KNN Example Data



KNN Example $K = 3$

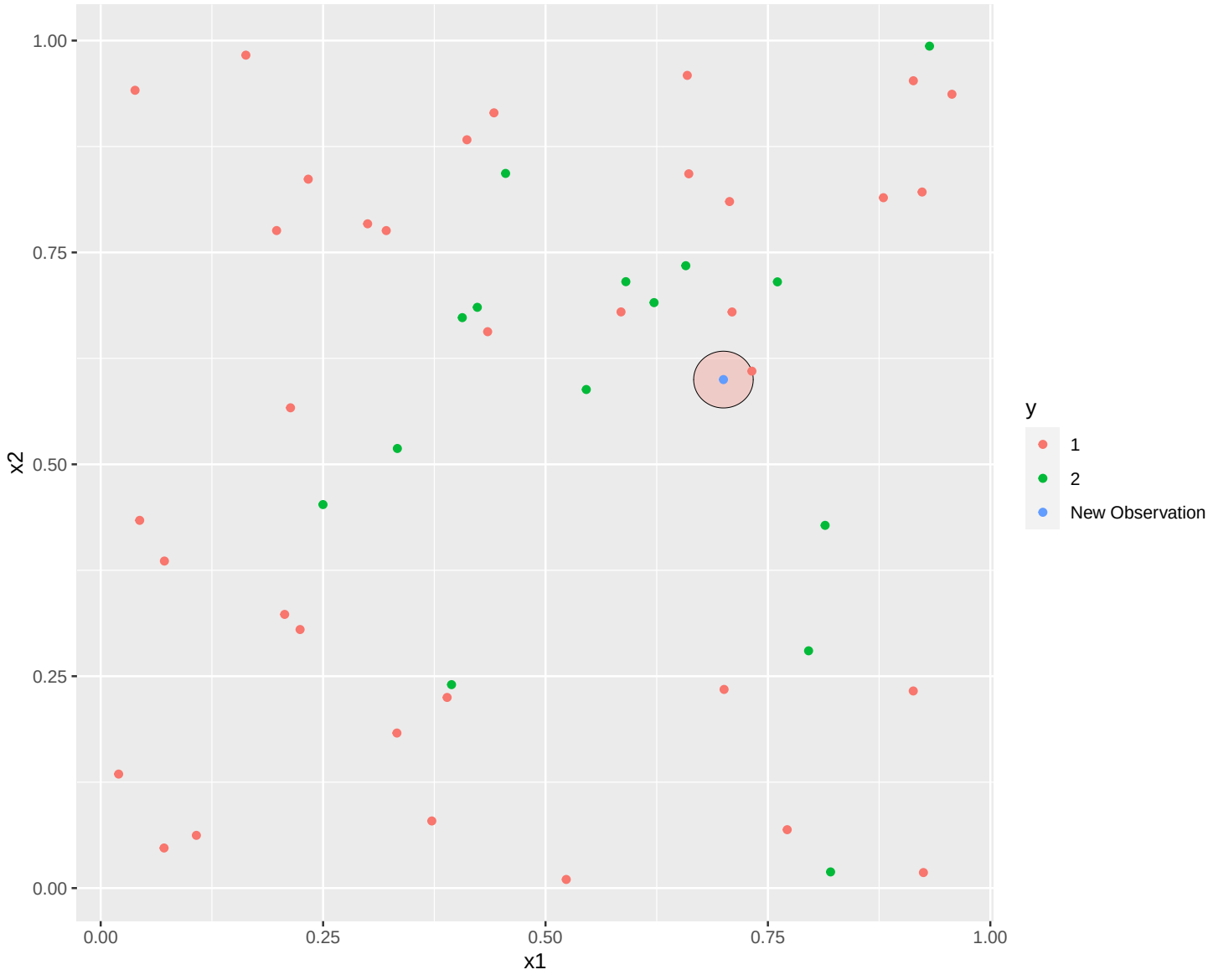


KNN Example $K = 7$

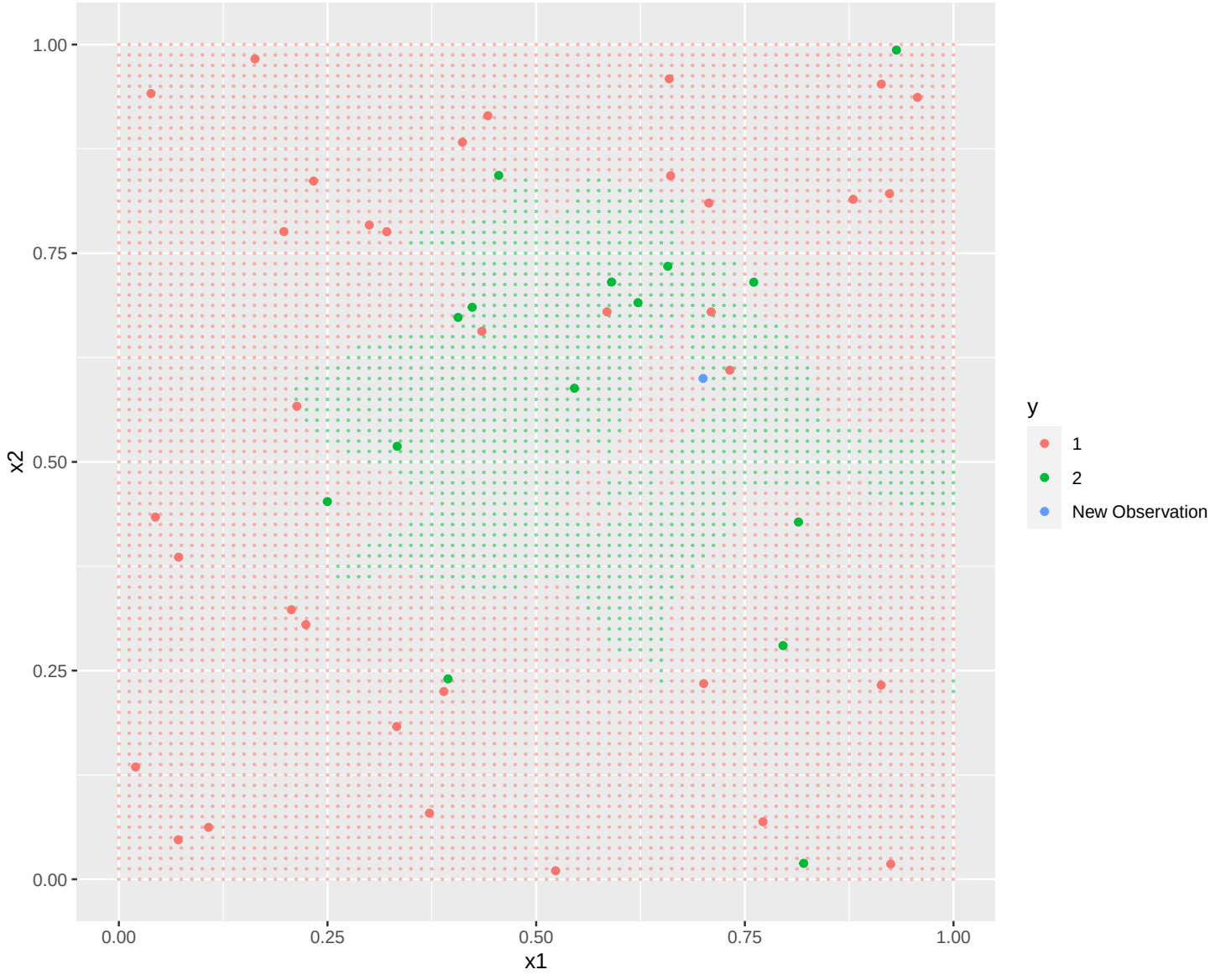


The relative 'vote counts' are a very crude estimate of probability.

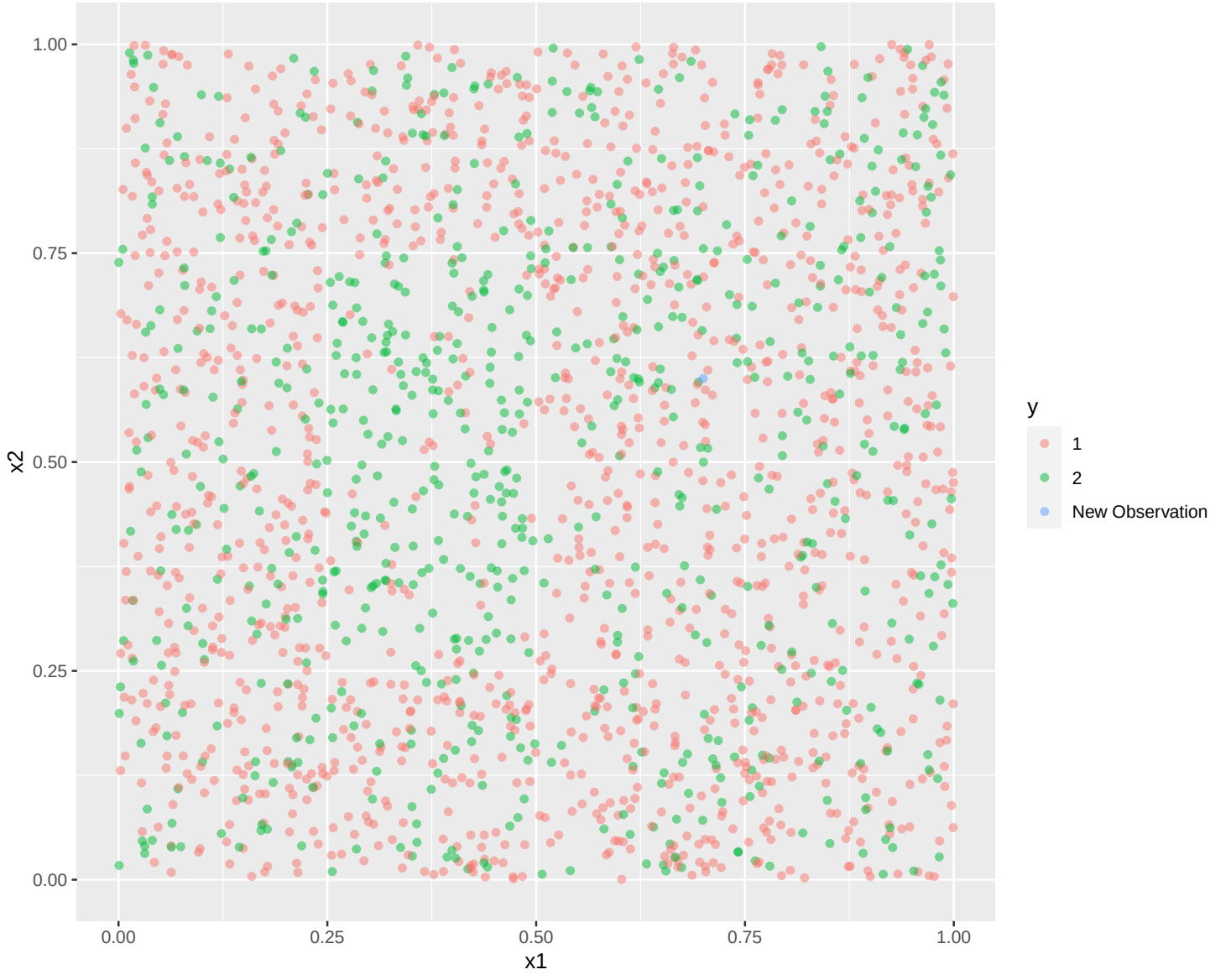
KNN Example $K = 1$



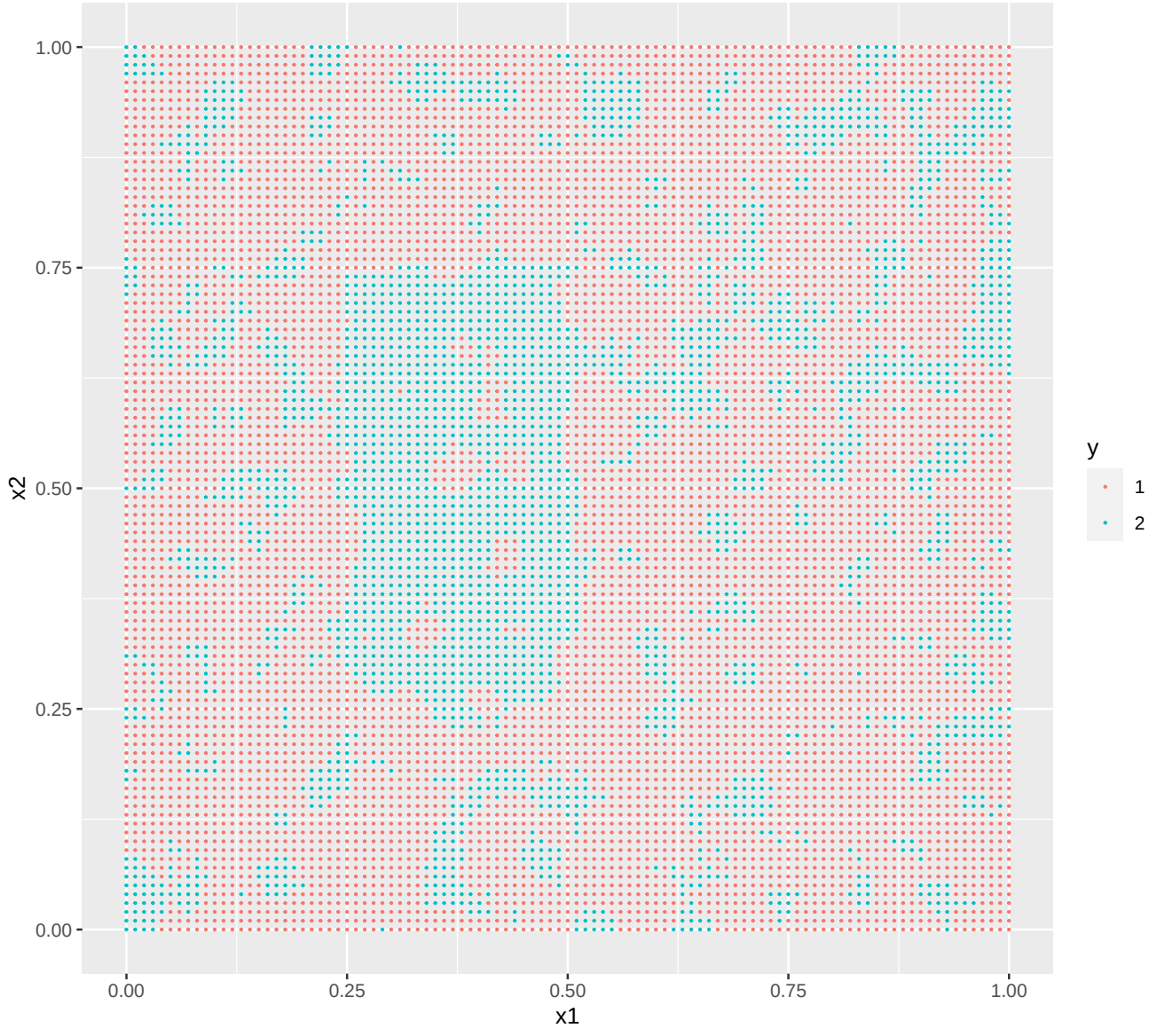
KNN Example


1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
1
2
New Observation

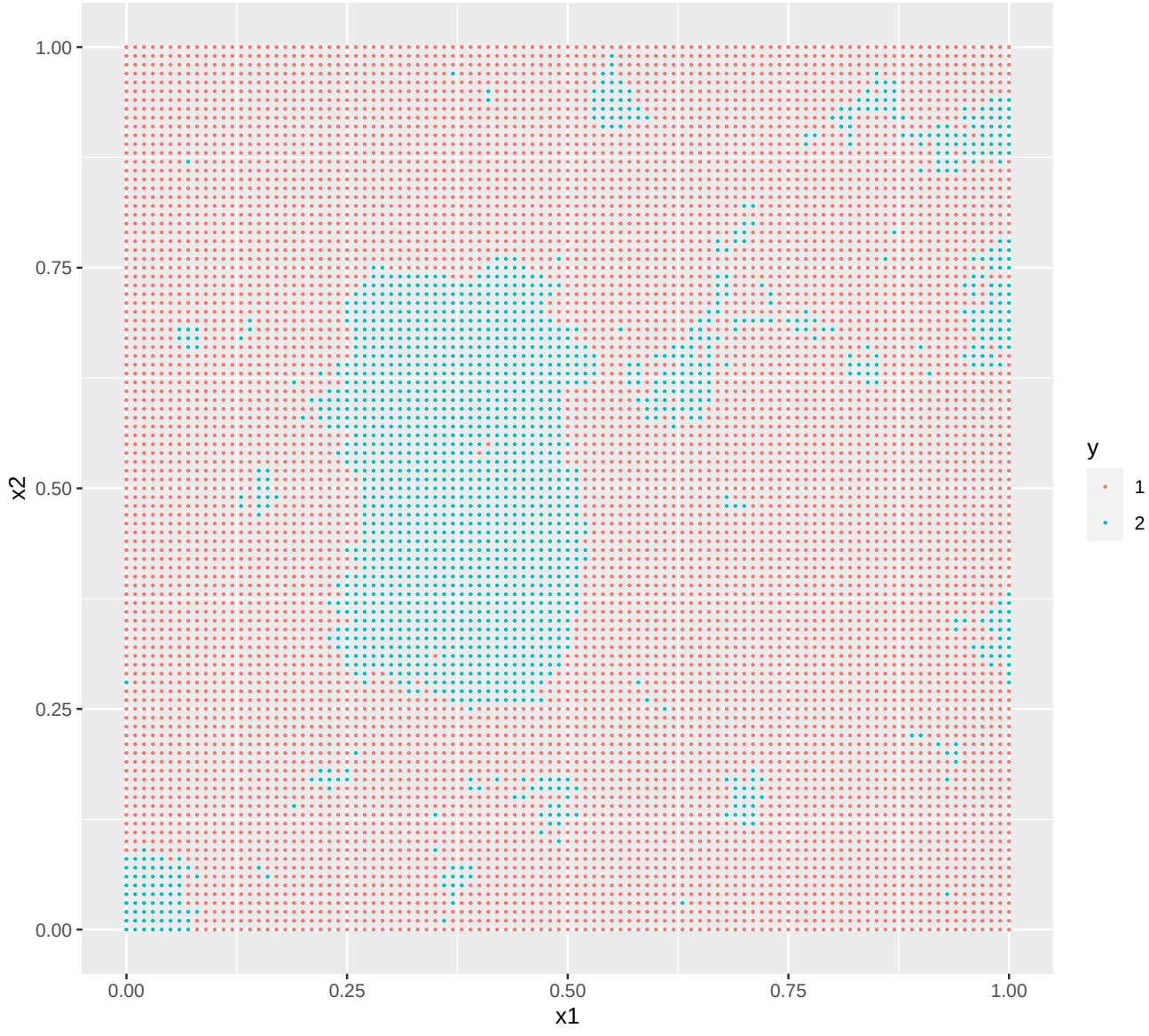
KNN Example: More Data



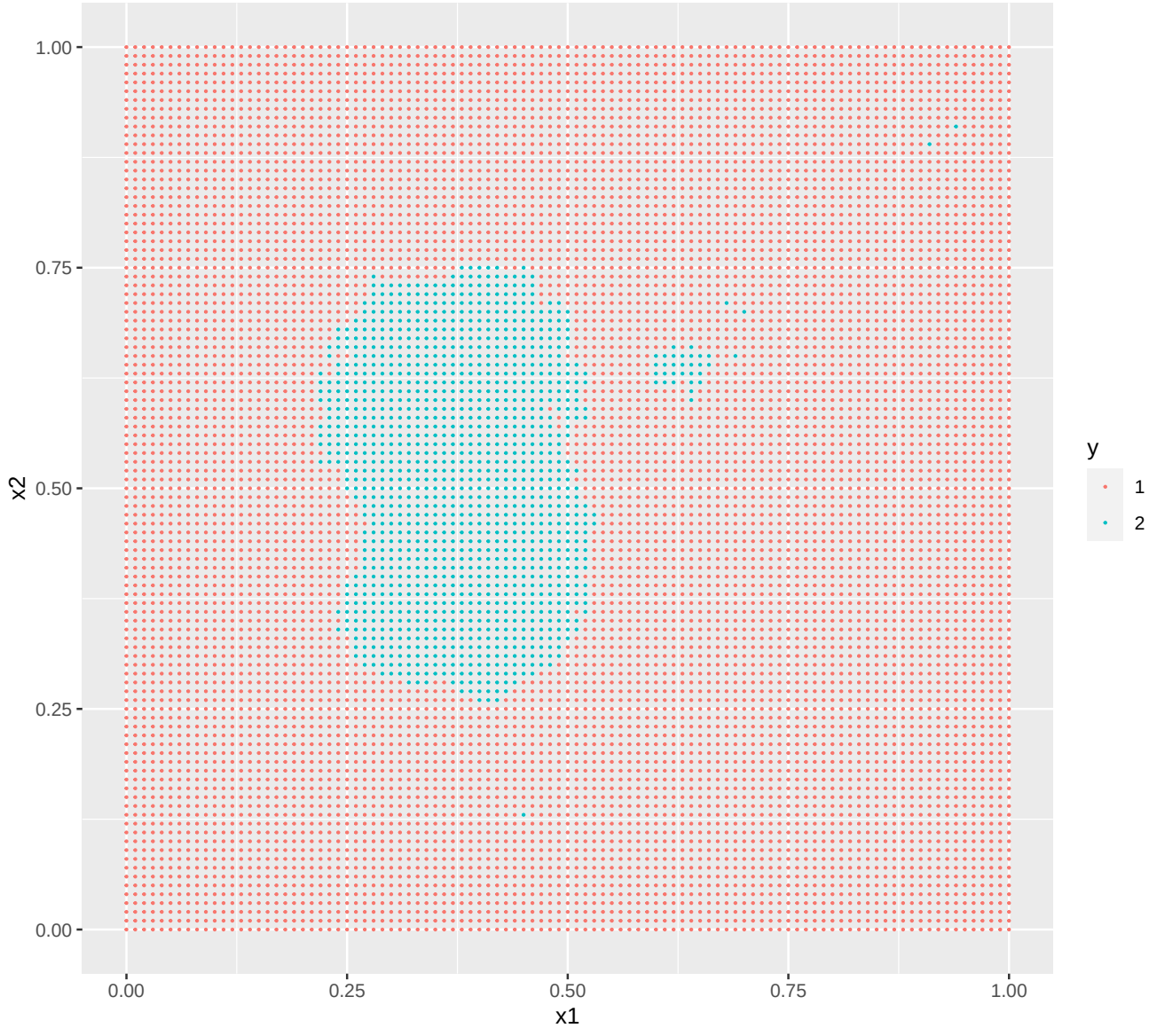
KNN Example $K = 3$



KNN Example $K = 11$

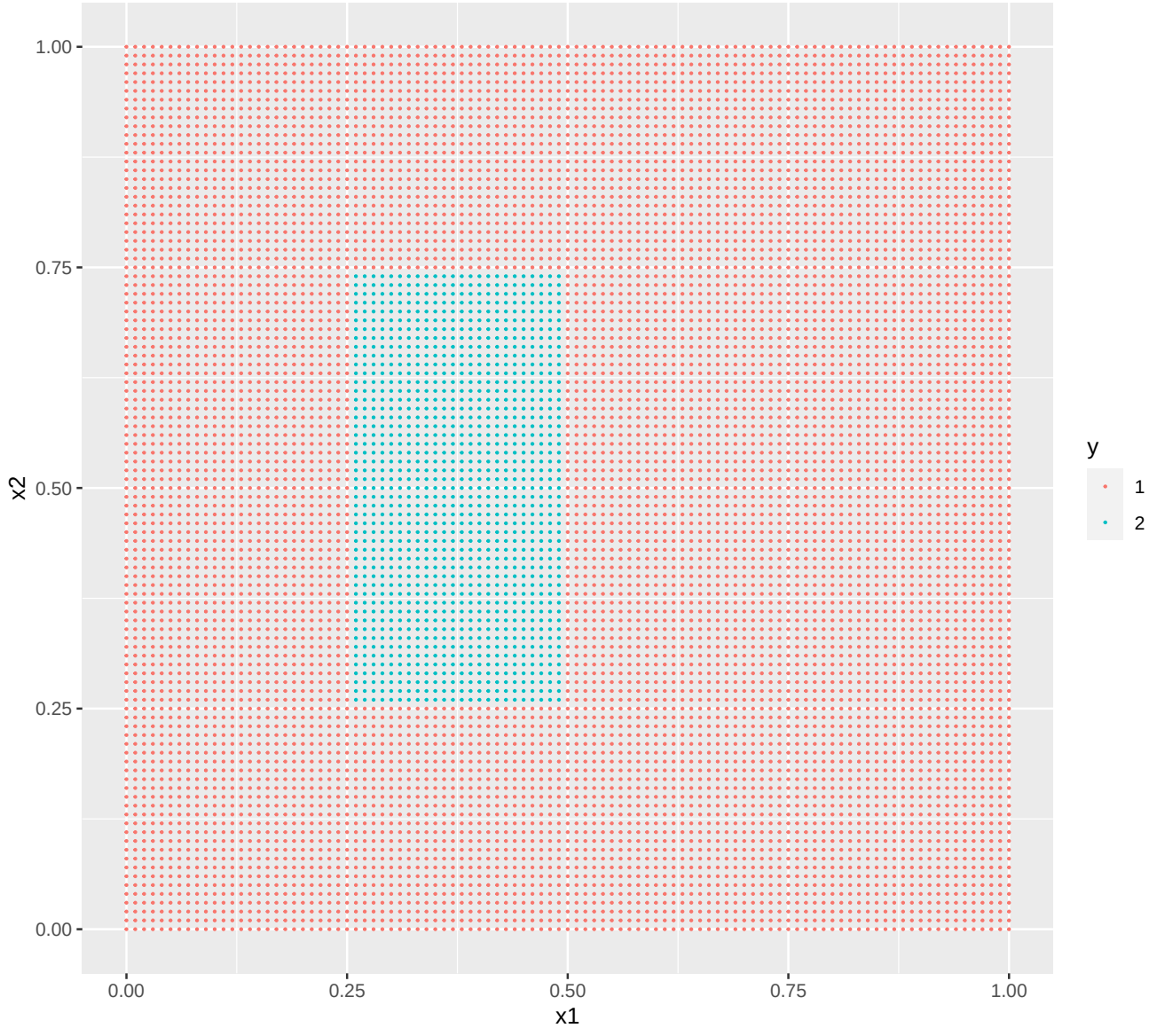


KNN Example $K = 31$



KNN – Example Base Truth

The optimal prediction scheme



KNN

Higher K leads to higher *in-sample* training error. (Proportion incorrect).

Lower K leads to higher flexibility $\implies$ overfitting and poor OOS misclassification.

KNN

- Pros:
  - Straightforward
  - Easily adjust to multiple categories.
  - Outperform other classifiers in settings where local details matter
- Cons:
  - Computing neighbors (just finding them) can be costly with big n or p
  - No variable selection built in
  - Choosing k is tricky – Cross-validation can work – but it is unstable.
    - Remember, LASSO benefitted from *stability* of estimates
  - Classification is very sensitive to k
  - Only other output is rough local probabilities.
    - Without good probabilities, assessing uncertainty is hard.

Binary Classification

Many problems can be reduced to binary classification (as above).

KNNs are a useful non-parametric classification tool.

Logits are a useful parametric classification tool.
- Remember Spam?

- Logits yield parametric decision boundaries. Easy to interpret.
- Logits are global methods. Use *all* the training data to inform predictions.
  - The probability estimates are more useful and more stable.
- Logits can do variable selection.

Credit Classification Example

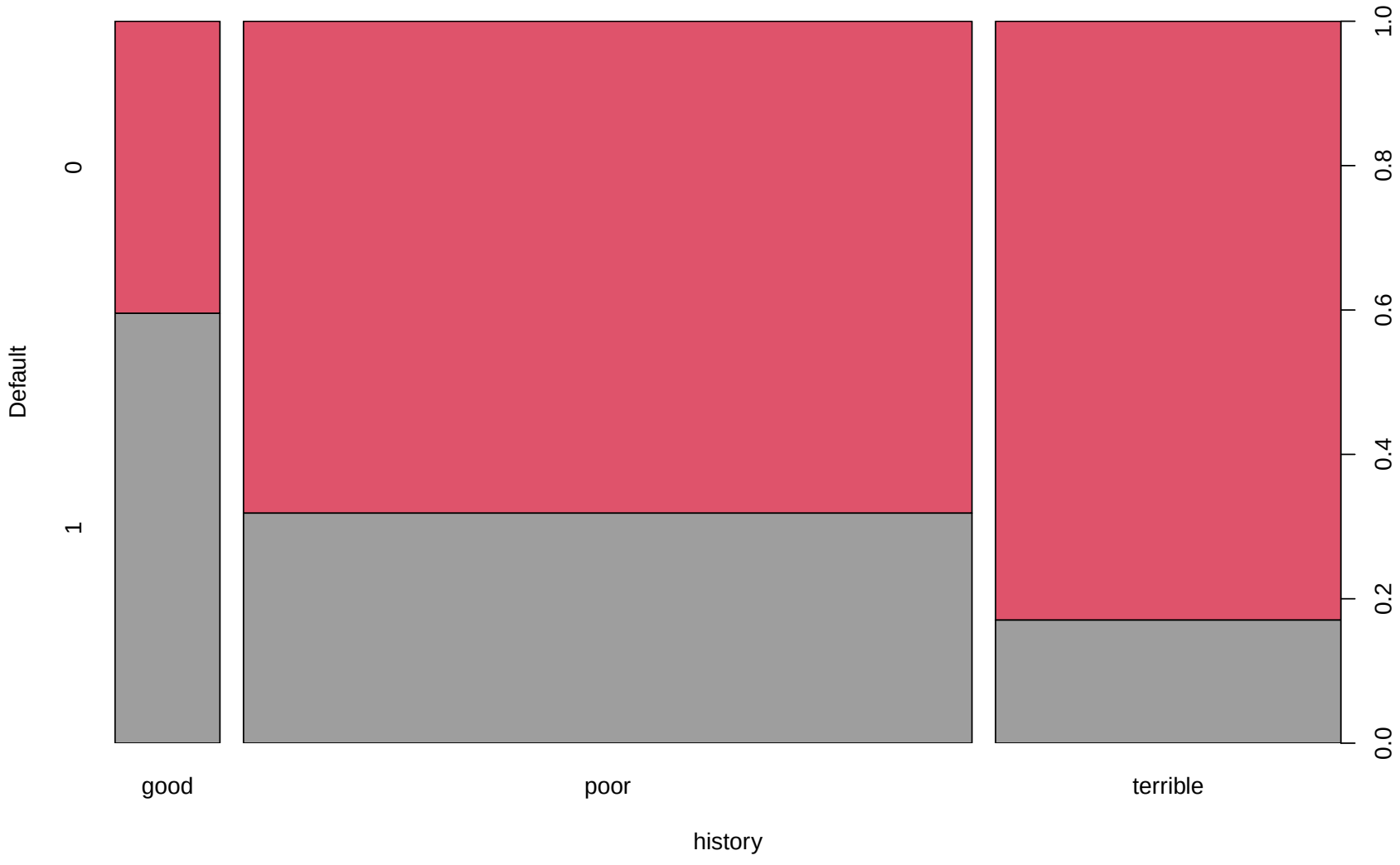
German loan/default data. "Predict performance of new loans" – want to predict default probability.

Going to try to use borrower and loan characteristics.

Messy Data.

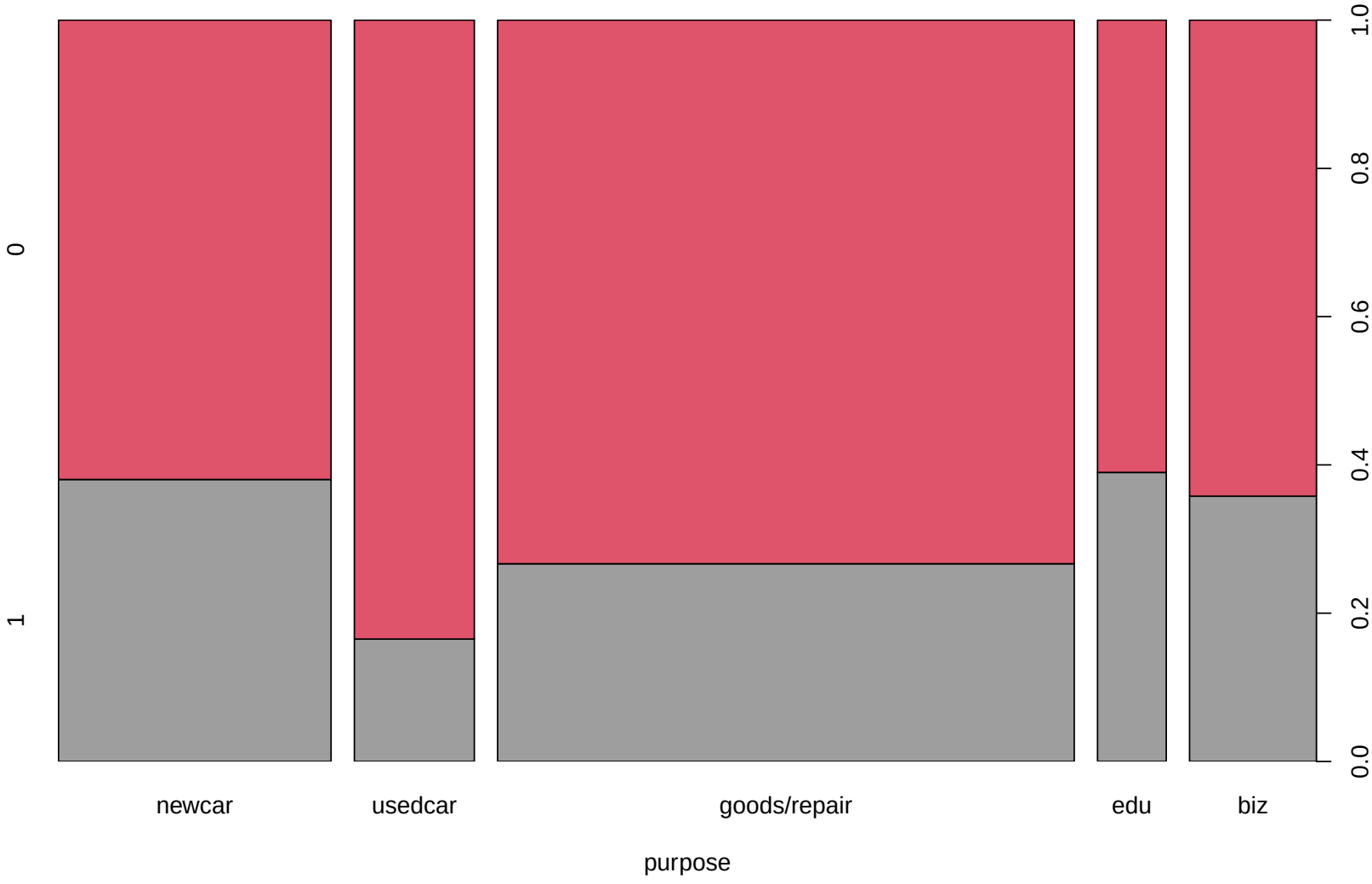
Beware

This is not ‘randomly sampled’ data.



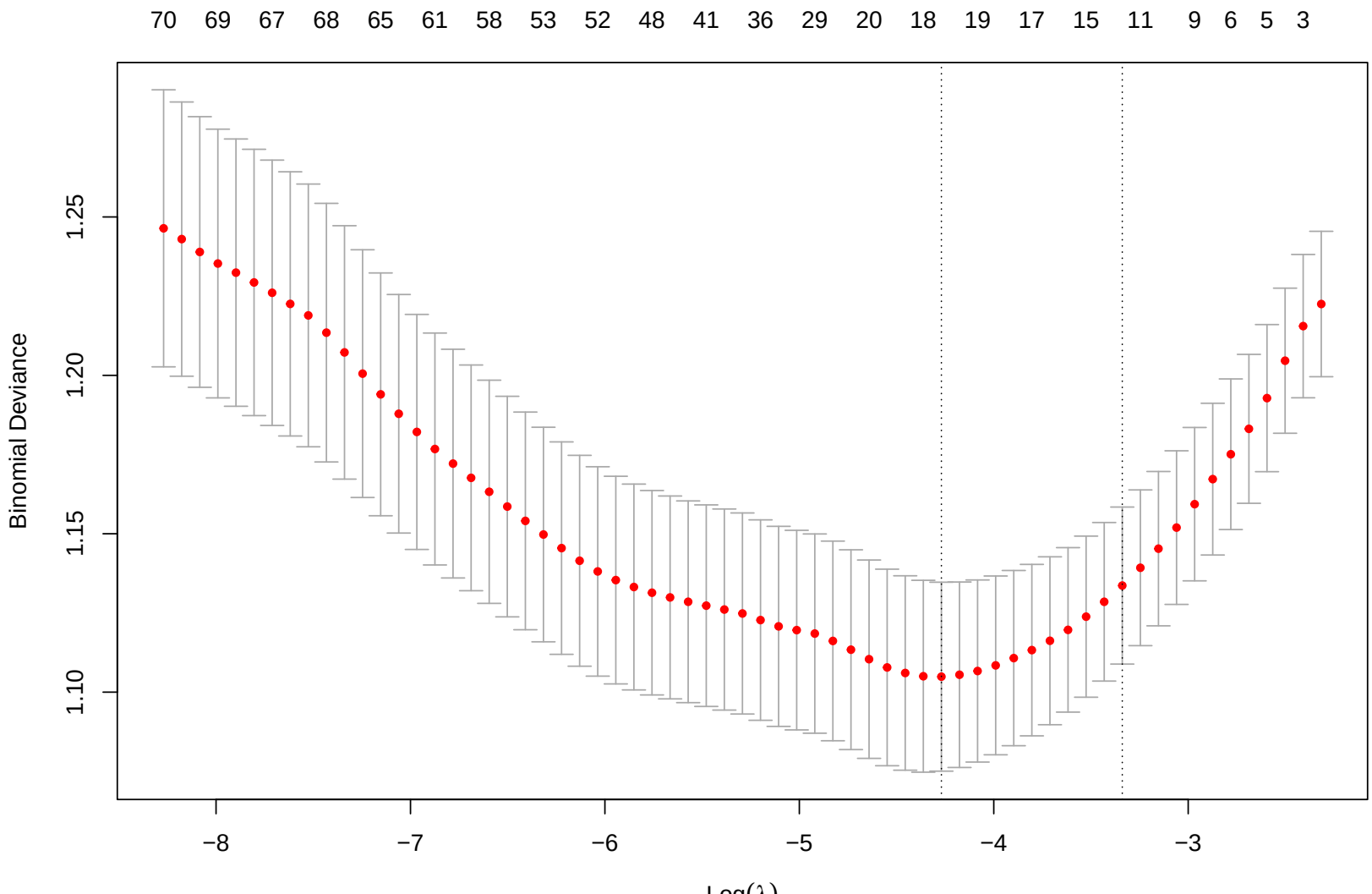
Beware

Consider your data sources carefully.



German Credit LASSO

```
credscore = cv.glmnet(credx, default, family="binomial")
plot(credscore)
```



German Credit LASSO

```
sum(coef(credscore, s="lambda.1se")!=0) # 1se
```

```
## [1] 13
```

```
sum(coef(credscore, s="lambda.min")!=0) # min
```

```
## [1] 21
```

Decision Making

There are two ways to be wrong in this binary problem.

- False Positive predict $\hat{y} = 1$ when $y = 0$.
  - Classify as defaulters when they are not
- False Negatives predict $\hat{y} = 0$ when $y = 1$.
  - Classify as non-defaulters when they are.

Both mistakes are bad, but sometimes one is worse than the other. Logistic regression gives us an estimate of $P[y = 1|x]$. And the Bayes decision rule classifies purely based on probabilities. When $P[y = 1|x] > 0.5$, classify as 1.

But instead of minimizing misclassification risk, we want to minimize our **loss**.

Using probabilities for Decisions

To make optimal decisions you need to account for probabilities and costs.

If for each loan, you make \$0.25 when repaid, and lose \$1 when not, then we only *expect* to make 'profits' when $P[y = 1] < 0.2$.

So we may want our classifier to use a different threshold.

False Positive Rate

Much like in lecture 1, we we can think about our *rate* of false positives.

- False Positive Rate = #Misclassified as 1 / #classified as 1
- False Negative Rate = #Misclassified as 0/ #Classified as 0

Sensitivity and Specificity

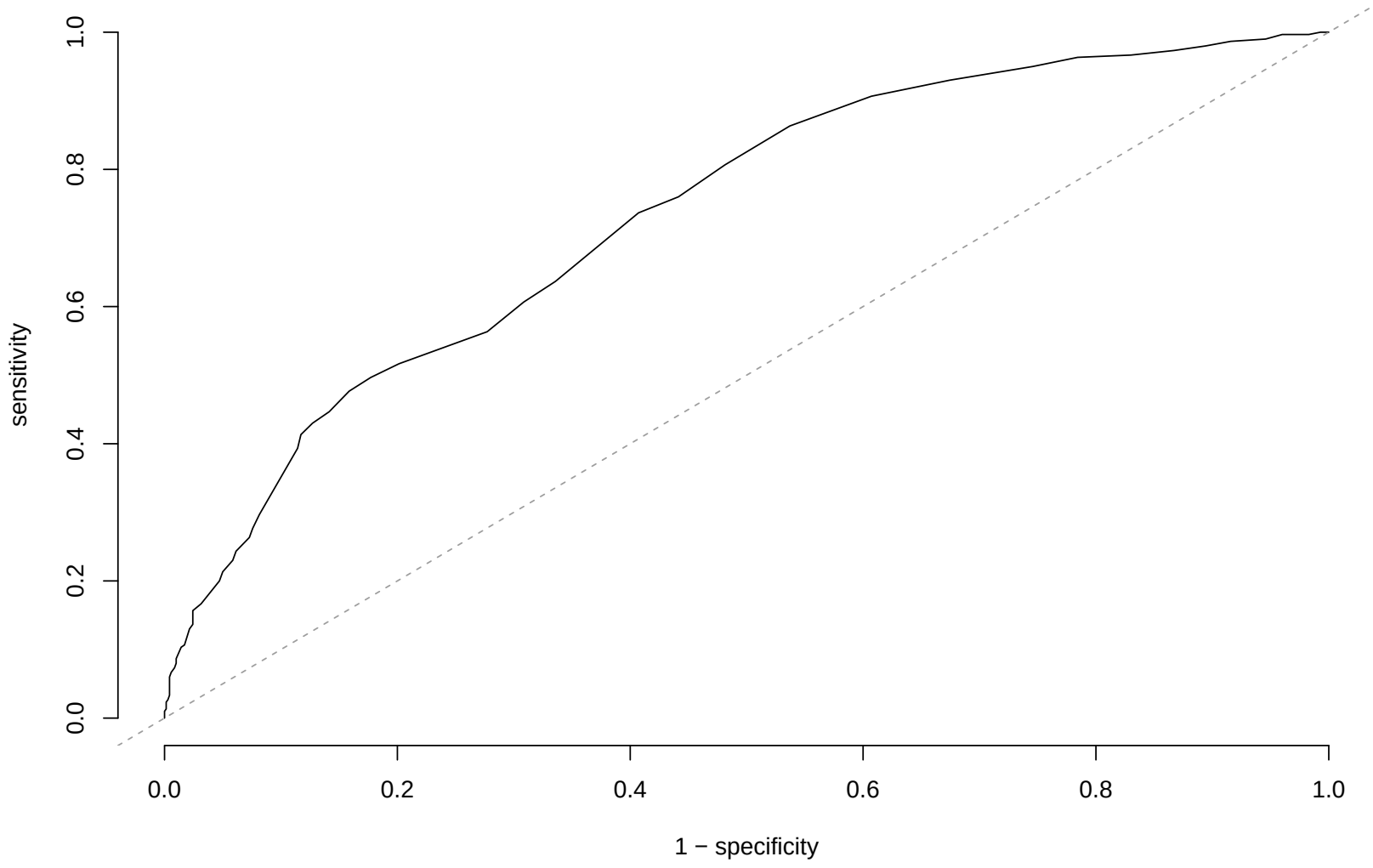
But we may also want to think about sensitivity and specificity.

- Sensitivity: proportion of true $y = 1$ classified as such.
- Specificity: proportion of true $y = 0$ classified as such.

A rule is sensitive if it mostly gets the 1s right. A rule is specific if it mostly gets the 0s right.

ROC Curve

We can plot the ROC curve for different choices of threshold.



Wrap up

Things to do

HW 3 is due tomorrow night.

See you Thursday

Rehash

- Classification is a different type of problem
- KNN is a useful tool for using local data to make predictions.
- Logits can also be used for classifications
- Prediction Errors in this setting are closely related to eachother

Bye!