KNN in R

Nate Wells

Math 243: Stat Learning

November 1st, 2021

Outline

In today's class, we will...

- Implement KNN in R
- Compare KNN and Logistic Regression

Section 1

KNN in R

The Unsinkable Example

The `Titanic` data set contains information on passengers of the *Titanic*

The Unsinkable Example

The `Titanic` data set contains information on passengers of the *Titanic*

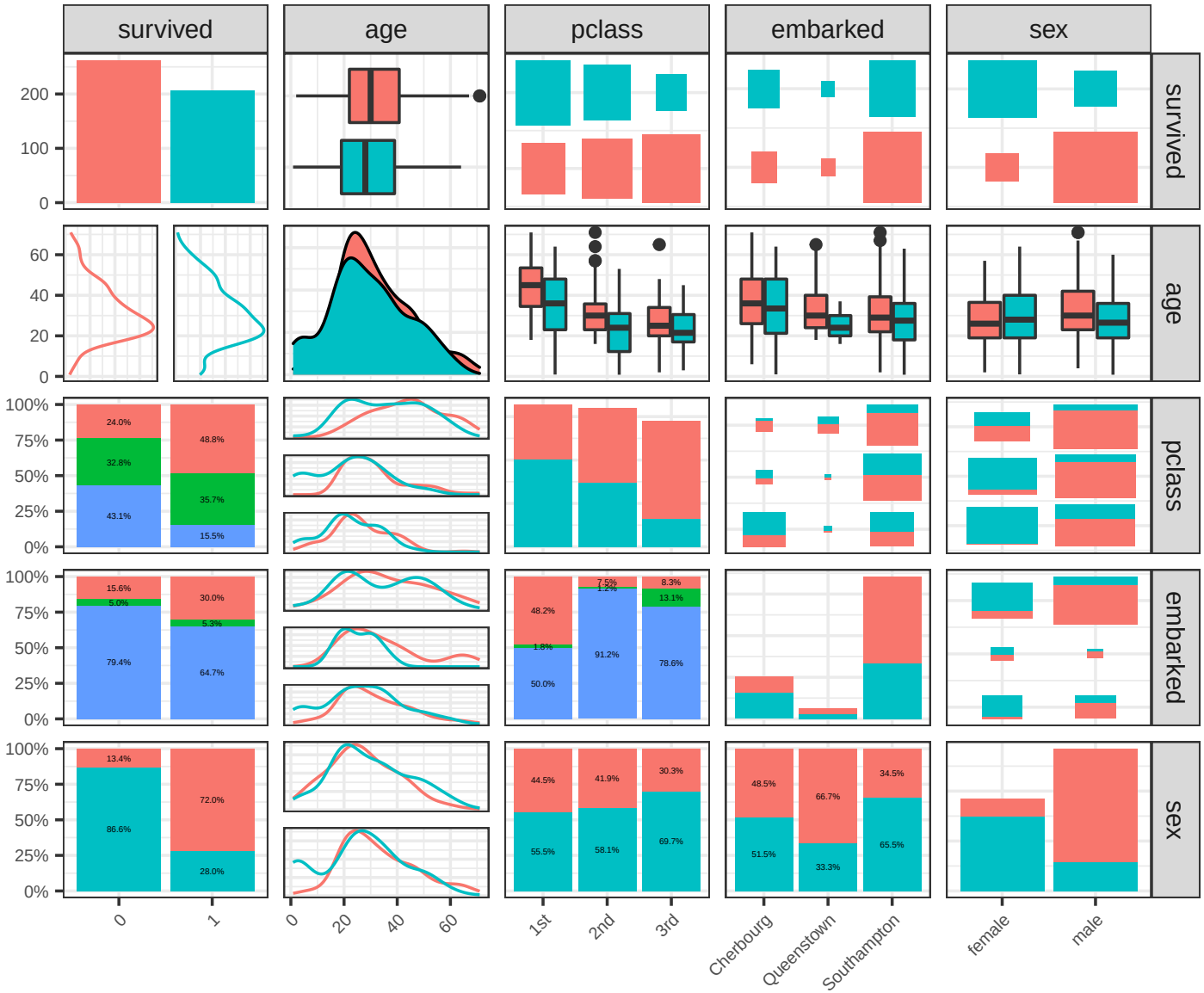
- Goal: Build a model with high predictive accuracy using several predictors

```
Titanic<-read_csv("data/titanic.csv") %>%
  select(survived, age, pclass, embarked, sex) %>%
  drop_na() %>% mutate(survived = as.factor(survived))

Titanic %>% count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   627
```

```
library(rsample)
set.seed(111)
Titanic_split <- initial_split(Titanic, prop = .8, stata = survived)
Titanic_train <- training(Titanic_split)
Titanic_test <- testing(Titanic_split)
```

Data Analysis



Data Analysis

- What trends do you notice among variables?

KNN

- Recall: The KNN model estimates the conditional probability $P(Y = 1 \mid X)$ as

$$P(Y = 1 \mid X = x_0) \approx \frac{1}{K} \sum_{i \in N_0} I(y_i = 1)$$

KNN

- Recall: The KNN model estimates the conditional probability $P(Y = 1 \mid X)$ as

$$P(Y = 1 \mid X = x_0) \approx \frac{1}{K} \sum_{i \in N_0} I(y_i = 1)$$

- In R, we have two options for KNN:
  - Use `knn` from `class` (textbook's choice)
  - Use the `kknn` function in the `kknn` package (preferable)

KNN

- Recall: The KNN model estimates the conditional probability $P(Y = 1 \mid X)$ as

$$P(Y = 1 \mid X = x_0) \approx \frac{1}{K} \sum_{i \in N_0} I(y_i = 1)$$

- In R, we have two options for KNN:
  - Use `knn` from `class` (textbook's choice)
  - Use the `kknn` function in the `kknn` package (preferable)
- Both `kknn` and `knn` fit a model **and** makes predictions all in one command.

KNN

- Recall: The KNN model estimates the conditional probability $P(Y = 1 \mid X)$ as

$$P(Y = 1 \mid X = x_0) \approx \frac{1}{K} \sum_{i \in N_0} I(y_i = 1)$$

- In R, we have two options for KNN:
  - Use `knn` from `class` (textbook's choice)
  - Use the `kknn` function in the `kknn` package (preferable)
- Both `kknn` and `knn` fit a model **and** makes predictions all in one command.
  - Compare `tolm` and `glm`, which first fit a model and then make predictions using `predict`

KNN

- Recall: The KNN model estimates the conditional probability $P(Y = 1 \mid X)$ as

$$P(Y = 1 \mid X = x_0) \approx \frac{1}{K} \sum_{i \in N_0} I(y_i = 1)$$

- In R, we have two options for KNN:
  - Use `knn` from `class` (textbook's choice)
  - Use the `kknn` function in the `kknn` package (preferable)
- Both `kknn` and `knn` fit a model **and** makes predictions all in one command.
  - Compare `tolm` and `glm`, which first fit a model and then make predictions using `predict`
- The `knn` function requires a model matrix (use `model.matrix`); while `kknn` instead uses a formula (`y ~ .`)

KNN

- Recall: The KNN model estimates the conditional probability $P(Y = 1 \mid X)$ as

$$P(Y = 1 \mid X = x_0) \approx \frac{1}{K} \sum_{i \in N_0} I(y_i = 1)$$

- In R, we have two options for KNN:
  - Use `knn` from `class` (textbook's choice)
  - Use the `kknn` function in the `kknn` package (preferable)
- Both `kknn` and `knn` fit a model **and** makes predictions all in one command.
  - Compare `tolm` and `glm`, which first fit a model and then make predictions using `predict`
- The `knn` function requires a model matrix (use `model.matrix`); while `kknn` instead uses a formula (`y ~ .`)
- KNN works best if predictors are standardized first (why?)
  - `knn` requires us to standardize using `scale`
  - `kknn` standardizes for us

KNN

- Recall: The KNN model estimates the conditional probability $P(Y = 1 \mid X)$ as

$$P(Y = 1 \mid X = x_0) \approx \frac{1}{K} \sum_{i \in N_0} I(y_i = 1)$$

- In R, we have two options for KNN:
  - Use `knn` from `class` (textbook's choice)
  - Use the `kknn` function in the `kknn` package (preferable)
- Both `kknn` and `knn` fit a model **and** makes predictions all in one command.
  - Compare `tolm` and `glm`, which first fit a model and then make predictions using `predict`
- The `knn` function requires a model matrix (use `model.matrix`); while `kknn` instead uses a formula (`y ~ .`)
- KNN works best if predictors are standardized first (why?)
  - `knn` requires us to standardize using `scale`
  - `kknn` standardizes for us
- `kknn` allows us to (optionally) weight observations **by** distance

KNN in R

- Using `kknn` with $k = 5$

```
library(kknn)
Titanic_fit5 <- kknn(survived ~., train = Titanic_train, test = Titanic_test,
                     k = 5, kernel = "rectangular")
```

- Setting `kernel = "rectangular"` corresponds to classic KNN

KNN in R

- Using `kknn` with $k = 5$

```
library(kknn)
Titanic_fit5 <- kknn(survived ~., train = Titanic_train, test = Titanic_test,
                     k = 5, kernel = "rectangular")
```

- Setting `kernel = "rectangular"` corresponds to classic KNN
- The output of `kknn` is a list with components:
  - `fitted.values`, a vector of predicted classes
  - `prob`, a matrix of predicted class probabilities
  - `CL`, a matrix of classes of the k nearest neighbors
  - `D`, a matrix of distances of the k nearest neighbors

KNN in R

- Using `kknn` with $k = 5$

```
library(kknn)
Titanic_fit5 <- kknn(survived ~., train = Titanic_train, test = Titanic_test,
                     k = 5, kernel = "rectangular")
```

- Setting `kernel = "rectangular"` corresponds to classic KNN
- The output of `kknn` is a list with components:
  - `fitted.values`, a vector of predicted classes
  - `prob`, a matrix of predicted class probabilities
  - `CL`, a matrix of classes of the k nearest neighbors
  - `D`, a matrix of distances of the k nearest neighbors

```
Titanic_fit5$fitted.values %>% head()
```

```
## [1] 1 1 1 1 0 1
## Levels: 0 1
```

```
Titanic_fit5$prob %>% head()
```

```
##         0   1
## [1,] 0.0 1.0
## [2,] 0.0 1.0
## [3,] 0.2 0.8
## [4,] 0.0 1.0
## [5,] 1.0 0.0
## [6,] 0.0 1.0
```

Model Performance

- Create dataframe of results:

```
titanic_results_knn <- data.frame(obs = Titanic_test$survived,
                                  preds = Titanic_fit5$fitted.values,
                                  probs = Titanic_fit5$prob[,2])
```

Model Performance

- Create dataframe of results:

```
titanic_results_knn <- data.frame(obs = Titanic_test$survived,
                                  preds = Titanic_fit5$fitted.values,
                                  probs = Titanic_fit5$prob[,2])
```

- Metrics via yardstick package

```
library(yardstick)
conf_mat(titanic_results_knn, truth = obs, estimate = preds)
```

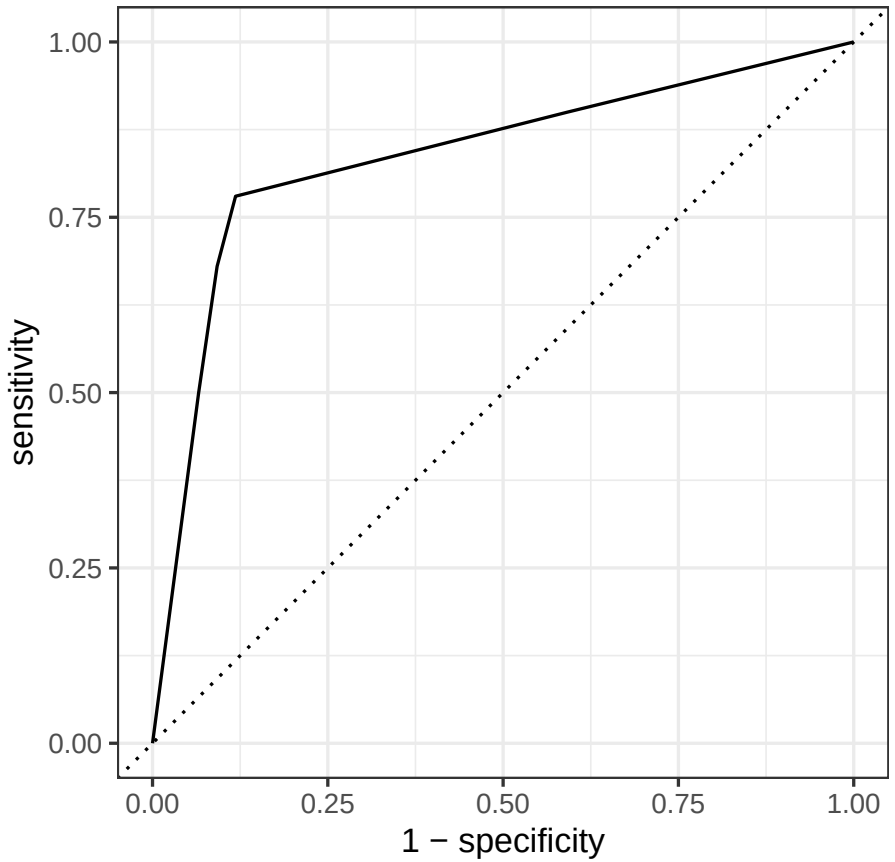
```
##           Truth
## Prediction  0  1
##          0 67 11
##          1  9 39
```

```
accuracy(titanic_results_knn, truth = obs, estimate = preds)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.841
```

ROC Curve

```
roc_curve(titanic_results_knn, truth = obs, estimate = probs,  event_level = "second") %>%
  autoplot()
```



- Why is ROC plot so linear?

Tuning KNN parameters

- How would our classifier behave for different values of k?

Tuning KNN parameters

- How would our classifier behave for different values of k?
- To find out, we use the `train.kknn` function, which performs LOOCV
  - k-fold CV is not recommended for KNN, due to computation time concerns

Tuning KNN parameters

- How would our classifier behave for different values of k?
- To find out, we use the `train.kknn` function, which performs LOOCV
  - k-fold CV is not recommended for KNN, due to computation time concerns

```
titanic_tune <- train.kknn(survived ~., data = Titanic_train,
                           kmax = 40, kernel = "rectangular")
```

Tuning KNN parameters

- How would our classifier behave for different values of k?
- To find out, we use the `train.kknn` function, which performs LOOCV
  - k-fold CV is not recommended for KNN, due to computation time concerns

```
titanic_tune <- train.kknn(survived ~., data = Titanic_train,
                           kmax = 40, kernel = "rectangular")
```

- Instead of `kmax`, can supply a vector of k values with `ks=`
- `train.kknn` creates a list with several components:
  - `MISCLASS`, a vector of misclassifation error for each k
  - `fitted.values`, a list of vectors of predictions for all k
  - `best.parameters`, the best parameter value for k

Tuning KNN parameters

- How would our classifier behave for different values of k?
- To find out, we use the `train.kknn` function, which performs LOOCV
  - k-fold CV is not recommended for KNN, due to computation time concerns

```
titanic_tune <- train.kknn(survived ~., data = Titanic_train,
                           kmax = 40, kernel = "rectangular")
```

- Instead of `kmax`, can supply a vector of k values with `ks=`
- `train.kknn` creates a list with several components:
  - `MISCLASS`, a vector of misclassifation error for each k
  - `fitted.values`, a list of vectors of predictions for all k
  - `best.parameters`, the best parameter value for k

```
titanic_tune$best.parameters
```

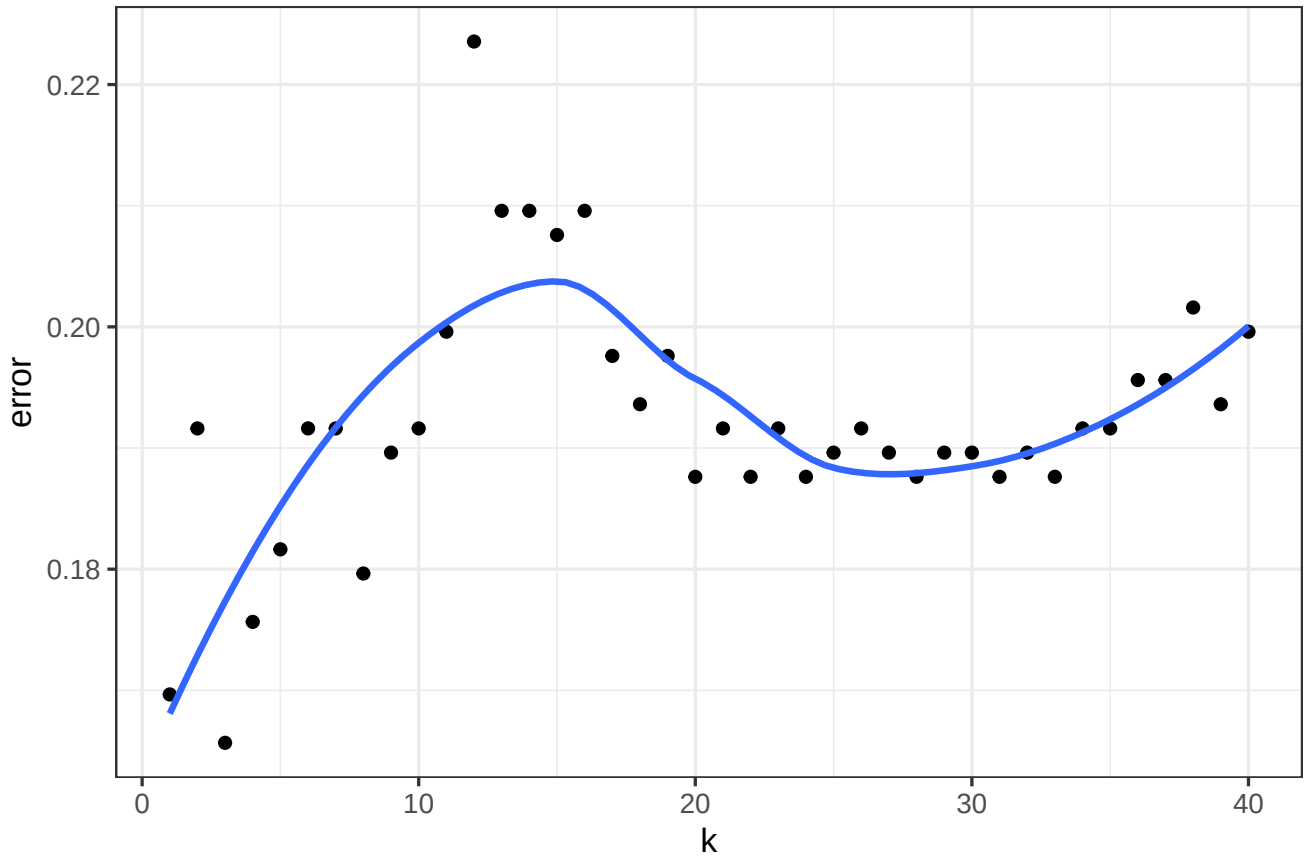
```
## $kernel
## [1] "rectangular"
##
## $k
## [1] 3
```

```
titanic_tune$MISCLASS %>% head()
```

```
##   rectangular
## 1   0.1696607
## 2   0.1916168
## 3   0.1656687
## 4   0.1756487
## 5   0.1816367
## 6   0.1916168
```

Bias-Variance Trade-off

```
as.data.frame(titanic_tune$MISCLASS) %>%
  rownames_to_column(var = "k") %>%
  mutate(error = rectangular, k = as.numeric(k)) %>%
ggplot(aes(x = k, y = error))+geom_point()+geom_smooth(se = F)+theme_bw()
```



Performance on Test Set

- Let's predict with $k = 1, 3, 5, 8, 25$.
  - First, we create a data frame with `obs`, `preds`, and `probs` for all models.

Performance on Test Set

- Let's predict with $k = 1, 3, 5, 8, 25$.
  - First, we create a data frame with obs, preds, and probs for all models.

```
titanic_results<- data.frame()
for (k in c(1,3,5,8,25)){
  Titanic_fit <- kknn(survived ~., train = Titanic_train, test = Titanic_test,
                      k = k, kernel = "rectangular")
  titanic_results<-rbind(titanic_results,
        data.frame(model = rep(k, times = length(Titanic_test$survived)),
                   obs = Titanic_test$survived,
                   preds = Titanic_fit$fitted.values,
                   probs = Titanic_fit$prob[,2]))
}
```

Performance on Test Set

- Let's predict with $k = 1, 3, 5, 8, 25$.
  - First, we create a data frame with obs, preds, and probs for all models.

```
titanic_results<- data.frame()
for (k in c(1,3,5,8,25)){
  Titanic_fit <- kknn(survived ~., train = Titanic_train, test = Titanic_test,
                      k = k, kernel = "rectangular")
  titanic_results<-rbind(titanic_results,
        data.frame(model = rep(k, times = length(Titanic_test$survived)),
                   obs = Titanic_test$survived,
                   preds = Titanic_fit$fitted.values,
                   probs = Titanic_fit$prob[,2]))
}
```

```
titanic_results %>% group_by(model) %>% accuracy(truth = obs, estimate = preds)
```

```
## # A tibble: 5 x 4
##   model .metric  .estimator .estimate
##   <dbl> <chr>    <chr>          <dbl>
## 1     1 accuracy binary         0.675
## 2     3 accuracy binary         0.770
## 3     5 accuracy binary         0.841
## 4     8 accuracy binary         0.817
## 5    25 accuracy binary         0.778
```

Comparison with Logistic Regression

```
Titanic_lr <- glm(survived ~., data = Titanic_train, family = "binomial")

lr_probs <- predict(Titanic_lr, Titanic_test, type = "response")
lr_preds <- ifelse(lr_probs>=.5, "1", "0")

titanic_results<- rbind(titanic_results,
        data.frame(model = rep("logistic regression", times = length(Titanic_test$survived)),
                   obs = Titanic_test$survived,
                   preds = lr_preds,
                   probs = lr_probs))
```

Comparison with Logistic Regression

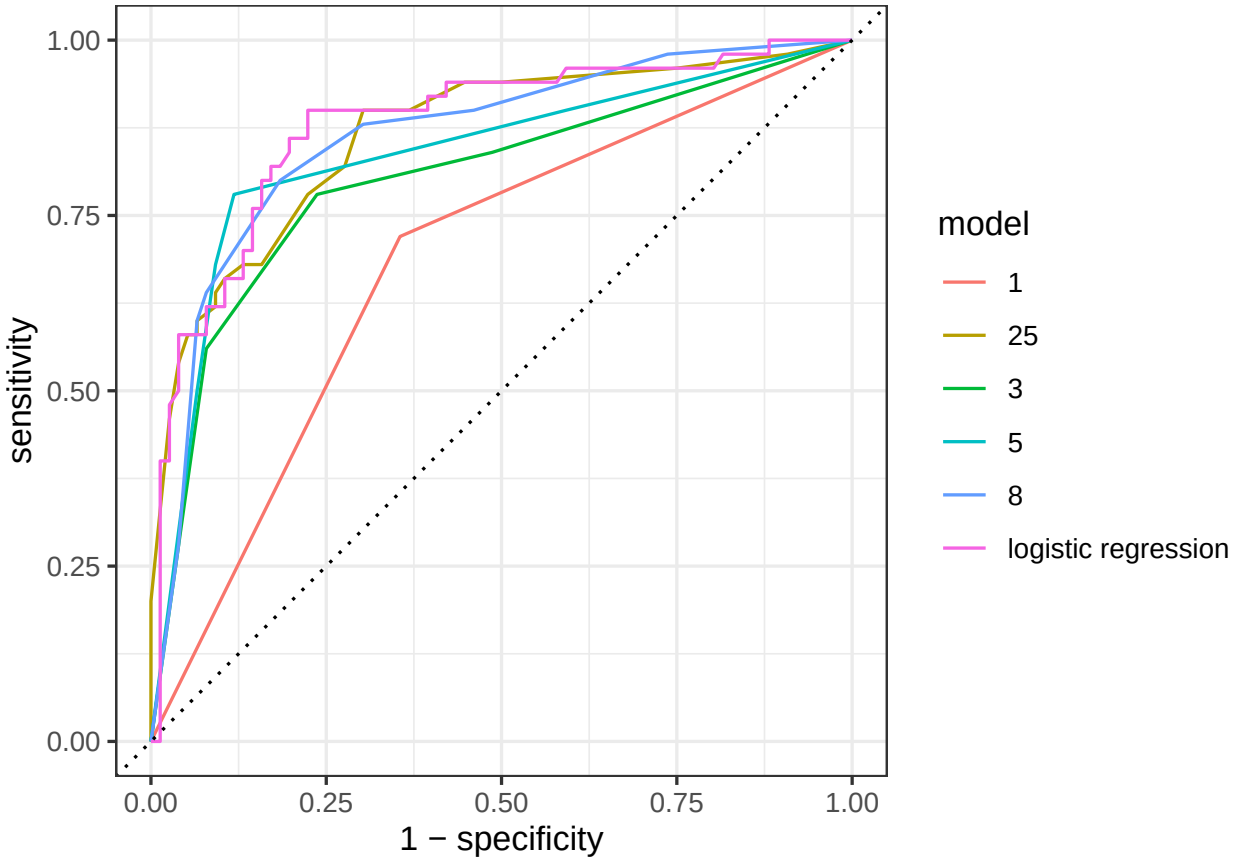
```
Titanic_lr <- glm(survived ~., data = Titanic_train, family = "binomial")

lr_probs <- predict(Titanic_lr, Titanic_test, type = "response")
lr_preds <- ifelse(lr_probs>=.5, "1", "0")

titanic_results<- rbind(titanic_results,
        data.frame(model = rep("logistic regression", times = length(Titanic_test$survived)),
                   obs = Titanic_test$survived,
                   preds = lr_preds,
                   probs = lr_probs))
```

```
titanic_results %>% group_by(model) %>% accuracy(truth = obs, estimate = preds) %>% arrange(desc
```

```
## # A tibble: 6 x 4
##   model               .metric  .estimator .estimate
##   <chr>               <chr>    <chr>          <dbl>
## 1 5                   accuracy binary         0.841
## 2 8                   accuracy binary         0.817
## 3 logistic regression accuracy binary         0.810
## 4 25                  accuracy binary         0.778
## 5 3                   accuracy binary         0.770
## 6 1                   accuracy binary         0.675
```

ROC Curves

```
titanic_results %>% group_by(model) %>%
  roc_curve(obs, probs,event_level = "second") %>%
  autoplot()
```



ROC Curves

```
titanic_results %>% group_by(model) %>%
  roc_auc(obs, probs, event_level = "second") %>% arrange(desc(.estimate))
```

```
## # A tibble: 6 x 4
##   model               .metric .estimator .estimate
##   <chr>               <chr>   <chr>          <dbl>
## 1 logistic regression roc_auc binary         0.879
## 2 25                  roc_auc binary         0.866
## 3 8                   roc_auc binary         0.861
## 4 5                   roc_auc binary         0.837
## 5 3                   roc_auc binary         0.802
## 6 1                   roc_auc binary         0.682
```