HW2

Zayd Abdalla, Areeya Aksornpan

3/15/2021

Problem 1: visualization

```
read_data <- function(df) {
  path <- paste("https://raw.githubusercontent.com/jgscott/ECO395M/master/dat
a/",
                    df, sep = "")
  df <- read_csv(path)
  return(df)
}

capmetro <- read_data("capmetro_UT.csv") %>%
  mutate(day_of_week = factor(day_of_week,
                              levels = c("Mon", "Tue", "Wed","Thu",
                                         "Fri","Sat","Sun")),
         month = factor(month, levels = c("Sep", "Oct","Nov")))

##
## -- Column specification ----------------------------------------------------
------
## cols(
##   timestamp = col_datetime(format = ""),
##   boarding = col_double(),
##   alighting = col_double(),
##   day_of_week = col_character(),
##   temperature = col_double(),
##   hour_of_day = col_double(),
##   month = col_character(),
##   weekend = col_character()
## )

Figure1 <-
  capmetro %>%
  group_by(hour_of_day, day_of_week, month) %>%
  mutate(avg_boarding = mean(boarding)) %>%
  ungroup() %>%
  ggplot() +
  geom_line(aes(x = hour_of_day, y = avg_boarding, color = month)) +
  scale_x_continuous(expand = c(0,0), limits = c(0, 24),
                     breaks = seq(10, 20, 5)) +
  scale_y_continuous(expand = c(0,0), limits = c(0, 200)) +
  scale_color_ft("Month") +
  facet_wrap(. ~ day_of_week, scales = "free") +
  labs(x = "Hour of day", y = "Average boarding",
```

```
      title = "Average bus ridership around UT",
      subtitle = "Tracked by Optical Scanner",
      caption = "Source: Capital Metro") +
  theme_ipsum(grid = "XY", axis = "xy")
```

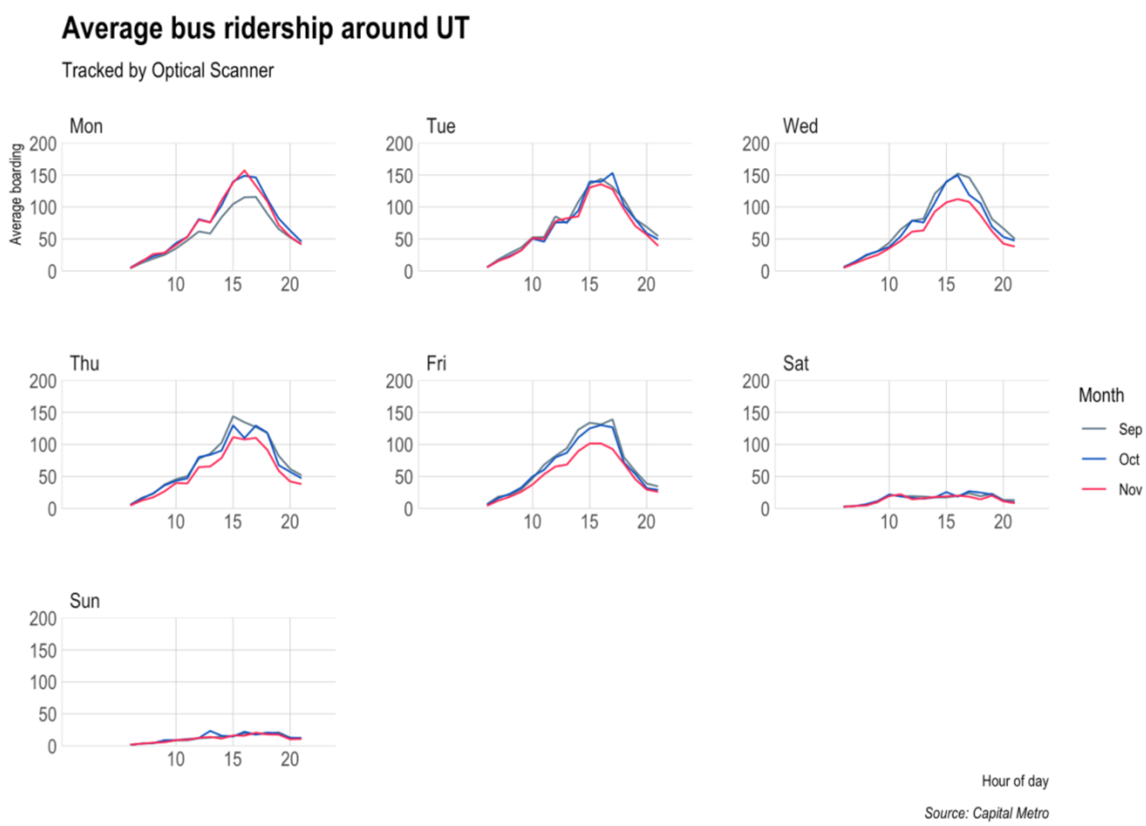


Figure1 illustrates the average CapMetro bus boardings-tracked by Optical Scanner-on weekdays in September, October, and November. The hour of peak boarding appears broadly similar across days, generally peaking around the 17th hour (5pm). This result is intuitive since most people finish school/work around that time. However, weekends tend to not peak in average bus boardings around certain hours as sharply, which supports my intuition that these trends are indicating work commutes. One guess for the decline in average boardings on Mondays in September is that the first Monday of September is Labor Day. Since Labor Day is a holiday, work commutes that day will decline relative to other Mondays, so the average bus boardings in September declines. One guess for the decline in average boardings on Weds/Thurs/Fri in November are because many schools and occupations go on break after the Tuesday before Thanksgiving, which gives people time off from work, so they are less likely to commute on those days.

```
Figure2 <-
  capmetro %>%
  group_by(timestamp, hour_of_day) %>%
  mutate(avg_boarding = mean(boarding)) %>%
```

```r
ggplot() +
geom_point(aes(x = temperature, y = avg_boarding, color = weekend)) +
scale_x_continuous(expand = c(0,0), limits = c(30, 100),
                   breaks = seq(40, 100, 20)) +
scale_y_continuous(expand = c(0,0), limits = c(0, 300)) +
scale_color_ft() +
facet_wrap(. ~ hour_of_day, scales = "free") +
labs(x = "Temperature", y = "Boarding",
     title = "Average bus ridership around UT by temperature",
     subtitle = "Faceted by hour of day",
     caption = "Source: Capital Metro") +
theme_ipsum(grid = "XY", axis = "xy") +
theme(legend.title = element_blank())
```

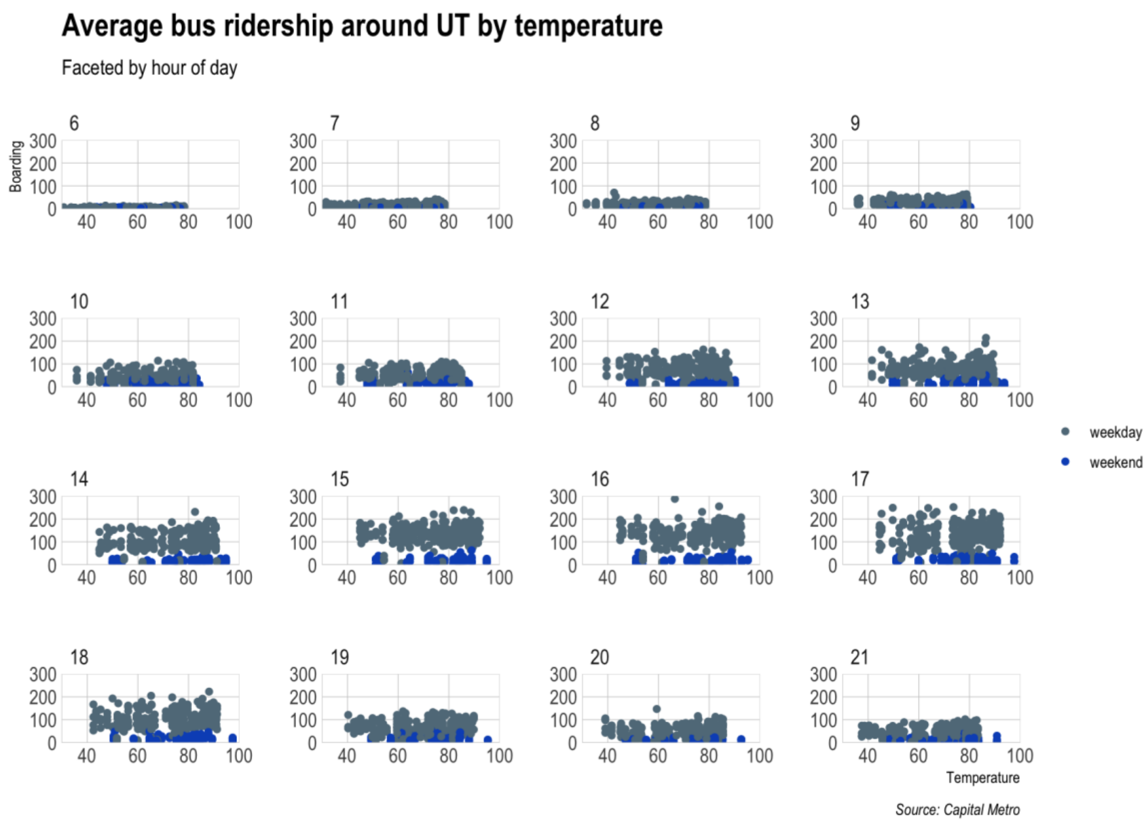


Figure2 shows average ridership, by temperature, which is faceted by hour of the day (6am to 10pm), and averaged by 15-minute increments. Gray indicates weekdays and blue indicates weekends. When we hold hour of day and weekend status constant, temperature does not appear to noticeably change the average ridership of UT students. The changes in bus demand seems to be more related to the time of day since the average boardings at each hour is pretty similar across temperatures.

Problem 2: Saratoga House Prices

```
saratoga <- mosaicData::SaratogaHouses

#create the train/test split.

set.seed(300)

saratoga_split <- initial_split(saratoga, strata = "price", prop = 0.75)
saratoga_train <- training(saratoga_split)
saratoga_test  <- testing(saratoga_split)

dim(saratoga_train)

## [1] 1298   16

dim(saratoga_split)

##   analysis assessment          n          p
##       1298        430       1728         16

#use cross-validation to split training set into k-folds.

# 3 fold cross validation
saratoga_fold <- vfold_cv(saratoga_train, v = 3, repeats = 1, strata = "price
")

# Linear and Knn models
lin_mod <-
    linear_reg() %>%
    set_mode("regression") %>%
    set_engine("lm")
lin_mod

## Linear Regression Model Specification (regression)
##
## Computational engine: lm

knn_mod <-
  nearest_neighbor(
    mode = "regression",
    neighbors = tune("K"),
  ) %>%
  set_engine("kknn")
knn_mod

## K-Nearest Neighbor Model Specification (regression)
##
## Main Arguments:
##   neighbors = tune("K")
```

```
##
## Computational engine: kknn
```

```
#Use tidymodels to feature engineer: rescaling and standardizing variables
saratoga_wf <-
  workflow() %>%
  add_formula(price ~ .) %>%
  # log price
  step_log(price) %>%
  # mean impute numeric variables
  step_meanimpute(all_numeric(), -all_outcomes()) %>%
  # rescale all numeric variables to lie between 0 and 1
  step_range(all_numeric(), min = 0, max = 1) %>%
  # one-hot
  step_dummy(fuel, centralAir, heating, newConstruction, waterfront, sewer) %
>%
  # remove predictor variables that are almost the same for every entry
  step_nzv(all_predictors())
saratoga_wf
```

```
## == Workflow ============================================================
======
## Preprocessor: Formula
## Model: None
##
## -- Preprocessor --------------------------------------------------------
------
## price ~ .
```

```
#Fitting LM model

set.seed(400)
lm_rs <-
  saratoga_wf %>%
  add_model(lin_mod) %>%
  fit_resamples(
    resamples = saratoga_fold,
    control = control_resamples(save_pred = TRUE)
  )



#Fitting KNN model



set.seed(400)
# feature engineering
knn_recipe <-
  recipe(price ~ ., data = saratoga_train) %>%
  # log price
  step_log(price) %>%
```

```
  # mean impute numeric variables
  step_meanimpute(all_numeric(), -all_outcomes()) %>%
  # rescale all numeric variables to lie between 0 and 1
  step_range(all_numeric(), min = 0, max = 1) %>%
  # one-hot
  step_dummy(fuel, centralAir, heating, newConstruction, waterfront, sewer) %
>%
  # remove predictor variables that are almost the same for every entry
  step_nzv(all_predictors())
# workflow
knn_wf <-
  workflow() %>%
  add_model(knn_mod) %>%
  add_recipe(knn_recipe)
# hyperparameter tuning
gridvals <- tibble(K = seq(1, 200))

knn_rs <-
  knn_wf %>%
  tune_grid(
    resamples = saratoga_fold,
    grid = gridvals,
    control = control_resamples(save_pred = TRUE))
knn_rs

## # Tuning results
## # 3-fold cross-validation using stratification
## # A tibble: 3 x 5
##   splits             id    .metrics           .notes           .predictions
##   <list>             <chr> <list>             <list>           <list>
## 1 <split [863/435]> Fold1 <tibble [400 x 5~ <tibble [0 x 1~ <tibble [87,00
0 x 5~
## 2 <split [866/432]> Fold2 <tibble [400 x 5~ <tibble [0 x 1~ <tibble [86,40
0 x 5~
## 3 <split [867/431]> Fold3 <tibble [400 x 5~ <tibble [0 x 1~ <tibble [86,20
0 x 5~

set.seed(400)

# Display only minimum RMSE
knn_min <- knn_rs %>%
  collect_metrics() %>%
  filter(.metric == "rmse") %>%
  filter(mean == min(mean))
knn_min

## # A tibble: 1 x 7
##       K .metric .estimator   mean     n std_err .config
##   <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1    28 rmse    standard   0.0667     3 0.00700 Preprocessor1_Model028
```

```r
## Evaluate Models

# Evaluate Linear Model
final_lm_wf <-
  saratoga_wf %>%
  add_model(lin_mod)

lm_fit <-
  final_lm_wf %>%
  last_fit(split = saratoga_split)
lm_fit %>% collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard   54998.    Preprocessor1_Model1
## 2 rsq     standard       0.642 Preprocessor1_Model1
```

```r
lm_results <-
  lm_fit %>%
  collect_predictions()
# view results
lm_results
```

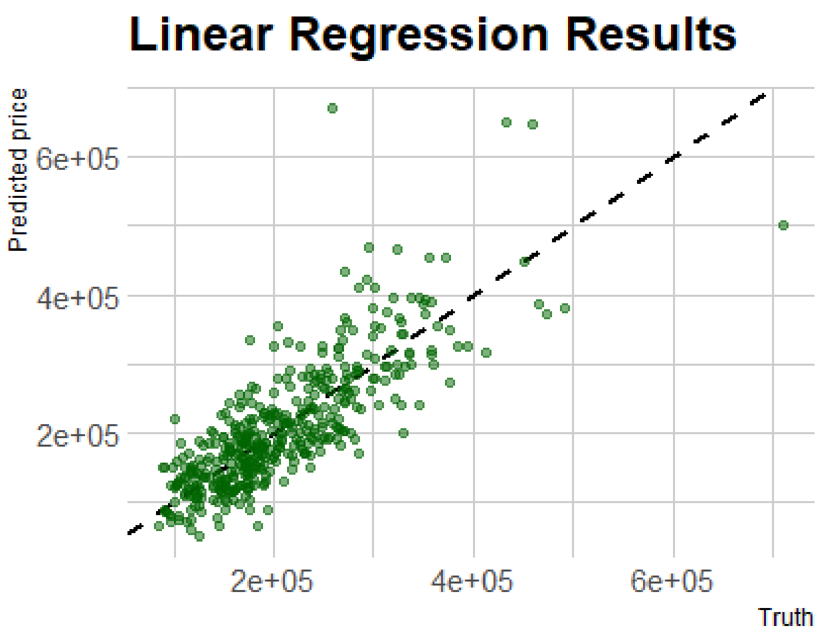
```
## # A tibble: 430 x 5
##    id                 .pred  .row  price .config
##    <chr>              <dbl> <int>  <int> <chr>
##  1 train/test split 188781.     8 170000 Preprocessor1_Model1
##  2 train/test split 176225.     9  90000 Preprocessor1_Model1
##  3 train/test split 226408.    11 325000 Preprocessor1_Model1
##  4 train/test split 277084.    26 248800 Preprocessor1_Model1
##  5 train/test split 135004.    27 135000 Preprocessor1_Model1
##  6 train/test split 178201.    30 140000 Preprocessor1_Model1
##  7 train/test split 225475.    32 187000 Preprocessor1_Model1
##  8 train/test split 232186.    36 169900 Preprocessor1_Model1
##  9 train/test split 218866.    37 209900 Preprocessor1_Model1
## 10 train/test split 229481.    38 169900 Preprocessor1_Model1
## # ... with 420 more rows
```

```r
lm_fit$.workflow[[1]] %>%
  tidy() %>%
  kable(digits = 4, "pipe")
```

term	estimate	std.error	statistic	p.value
(Intercept)	119752.7855	22181.0183	5.3989	0.0000
lotSize	7563.3591	2435.0849	3.1060	0.0019
age	-200.0885	68.0400	-2.9407	0.0033
landValue	0.9023	0.0560	16.1176	0.0000
livingArea	67.8700	5.3229	12.7505	0.0000

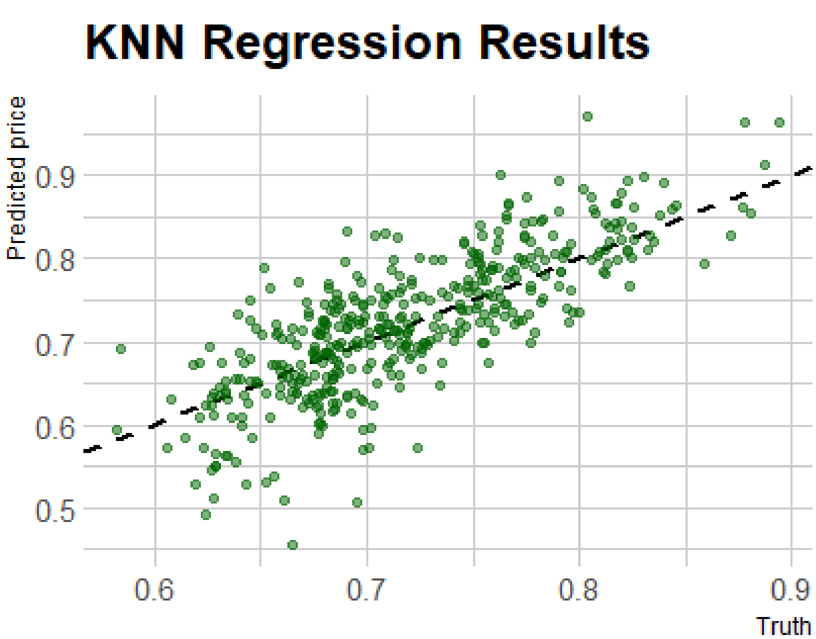
pctCollege	-138.4247	182.4636	-0.7586	0.4482
bedrooms	-8262.4653	3008.5782	-2.7463	0.0061
fireplaces	3083.3949	3487.1396	0.8842	0.3767
bathrooms	25595.5289	3998.7661	6.4009	0.0000
rooms	3512.9442	1134.4768	3.0965	0.0020
heatinghot water/steam	-10979.9277	4959.6157	-2.2139	0.0270
heatingelectric	1278.0498	14903.4144	0.0858	0.9317
fuelelectric	-12672.5264	14752.0410	-0.8590	0.3905
fueloil	-41.3979	5894.1796	-0.0070	0.9944
sewerpublic/commercial	1031.3498	4326.6126	0.2384	0.8116
sewernone	-12456.5061	20182.1907	-0.6172	0.5372
waterfrontNo	-136887.7740	17052.8276	-8.0273	0.0000
newConstructionNo	49508.0622	8594.8209	5.7602	0.0000
centralAirNo	-9271.3629	4143.6560	-2.2375	0.0254

```
# LM Graphically
lm_results %>%
  ggplot(aes(.pred, price)) +
  geom_abline(lty = 2, color = "black", size = 1) +
  geom_point(alpha = 0.5, color = "dark green") +
  labs(
    title = 'Linear Regression Results',
    x = "Truth",
    y = "Predicted price",
    color = NULL
  ) +
  theme_ipsum()
```




```
# Evaluate KNN Model

final_knn_wf <-
  knn_wf %>%
  finalize_workflow(knn_min)
knn_fit <-
  final_knn_wf %>%
  last_fit(split = saratoga_split)
knn_fit %>% collect_metrics()

## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard      0.0524 Preprocessor1_Model1
## 2 rsq     standard      0.616  Preprocessor1_Model1

# predictions
knn_results <-
  knn_fit %>%
  collect_predictions()
# view results
knn_results

## # A tibble: 430 x 5
##    id                .pred  .row price .config
##    <chr>             <dbl> <int> <dbl> <chr>
##  1 train/test split 0.688     8 0.699 Preprocessor1_Model1
```

```
##  2 train/test split 0.701     9 0.573 Preprocessor1_Model1
##  3 train/test split 0.704    11 0.828 Preprocessor1_Model1
##  4 train/test split 0.756    26 0.775 Preprocessor1_Model1
##  5 train/test split 0.645    27 0.653 Preprocessor1_Model1
##  6 train/test split 0.715    30 0.661 Preprocessor1_Model1
##  7 train/test split 0.731    32 0.718 Preprocessor1_Model1
##  8 train/test split 0.777    36 0.699 Preprocessor1_Model1
##  9 train/test split 0.744    37 0.741 Preprocessor1_Model1
## 10 train/test split 0.755    38 0.699 Preprocessor1_Model1
## # ... with 420 more rows

# KNN Graphically
knn_results %>%
  ggplot(aes(.pred, price)) +
  geom_abline(lty = 2, color = "black", size = 1) +
  geom_point(alpha = 0.5, color = "dark green") +
  labs(
    title = 'KNN Regression Results',
    x = "Truth",
    y = "Predicted price",
    color = NULL
  ) +
  theme_ipsum()
```

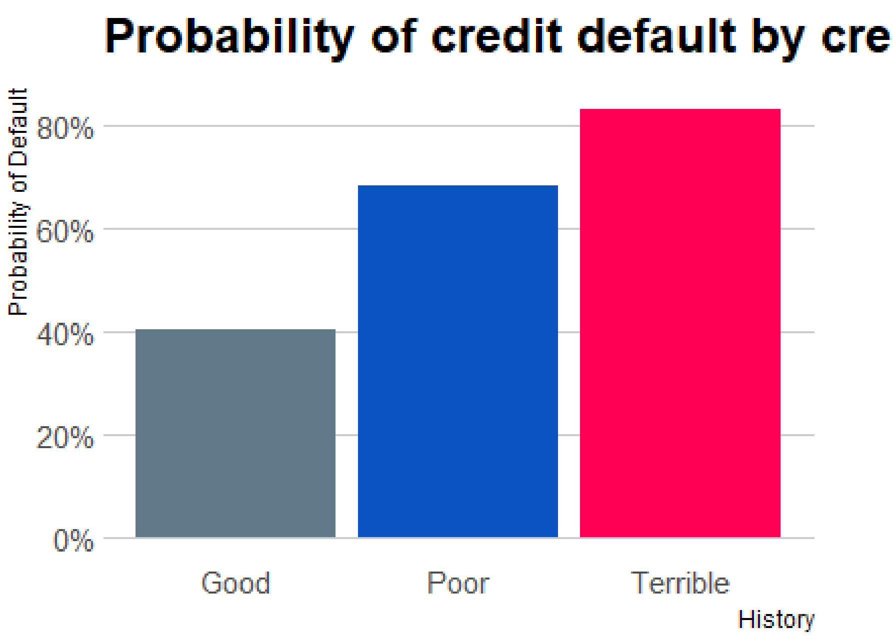


We built two models-a linear model and a KNN model-to predict the price of houses. The base model appeared to perform quite well, so we decided to tweak it by feature engineering to improve the accuracy. We standardized numeric variables to values between (0,1), applied a log transformation to the price variable and created dummy variables for all “character” encoded variables. Next, both our linear and KNN regression models were trained on 3 folds without repetition. We gave the KNN model a hyperparameter (neighbors) that was tuned using a tuning grid. Then, these models were fit on out-of-sample data and we found that our linear model clearly outperformed the medium model from class. However, our KNN model heavily outperformed even our improved linear model. This exercise illustrates the capability of KNN models to adapt to non-linearities of the data in order to achieve better fits in predicting pricing of houses.

Problem 3: Classification and retrospective sampling

```
german_credit <-
  read_data("german_credit.csv") %>%
  select(-1) %>%
  # Factoring outcomes
  mutate(Default = as.factor(Default))

## Warning: Missing column names filled in: 'X1' [1]

##
## -- Column specification ---------------------------------------------------
------
## cols(
##   .default = col_character(),
##   X1 = col_double(),
##   Default = col_double(),
##   duration = col_double(),
##   amount = col_double(),
##   installment = col_double(),
##   residence = col_double(),
##   age = col_double(),
##   cards = col_double(),
##   liable = col_double(),
##   rent = col_logical()
## )
## i Use `spec()` for the full column specifications.

# Build logistic regression model


german_credit %>%
  group_by(Default, history) %>%
  add_tally() %>%
  rename(num_default = n) %>%
  distinct(history, num_default) %>%
  ungroup() %>%
  group_by(history) %>%
  mutate(tot_default = sum(num_default),
         prob_default = (num_default / tot_default) * 100) %>%
  filter(Default == 0) %>%
```

```
  ggplot() +
  geom_col(aes(x = history, y = prob_default,
           fill = history)) +
  scale_y_continuous(labels = function(x) paste0(x, "%")) +
  scale_x_discrete(labels = c("Good", "Poor", "Terrible")) +
  scale_fill_ft() +
  labs(x = "History", y = "Probability of Default",
      title = "Probability of credit default by credit history") +
  theme_ipsum(grid = "Y") +
  theme(legend.title = element_blank(),
        legend.position = "None")
```




```
# Train test
set.seed(395)
german_split <- initial_split(german_credit, strata = "Default", prop = 0.75)
german_train <- training(german_split)
german_test  <- testing(german_split)
# 3 fold cross validation (for speed)
german_fold <- vfold_cv(german_train, v = 3, repeats = 1, strata = "Default")
german_fold
```

```
## #  3-fold cross-validation using stratification
## # A tibble: 3 x 2
##   splits            id
##   <list>            <chr>
## 1 <split [500/250]> Fold1
```

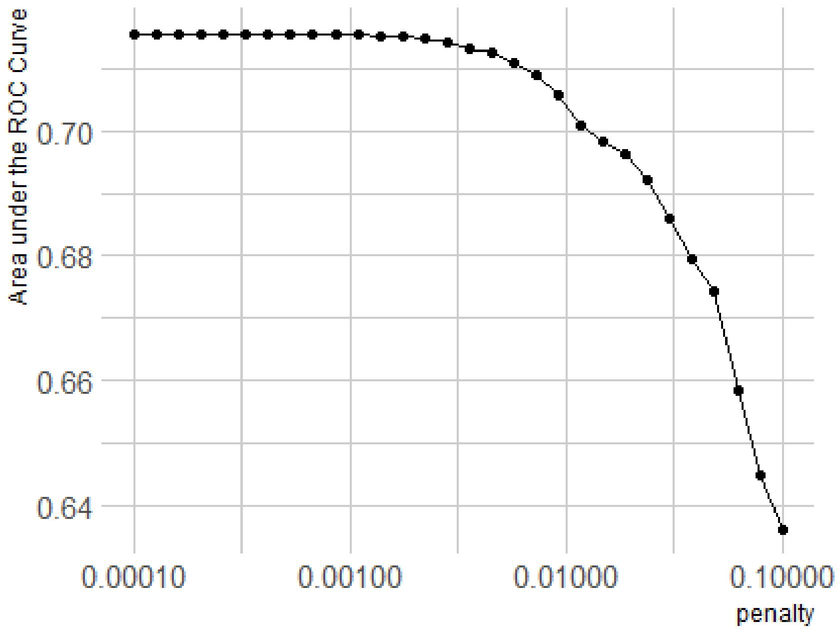
```
## 2 <split [500/250]> Fold2
## 3 <split [500/250]> Fold3
```

```r
# Model engine

log_mod <-
  logistic_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet") %>%
  set_mode("classification")
log_mod
```

```
## Logistic Regression Model Specification (classification)
##
## Main Arguments:
##   penalty = tune()
##   mixture = 1
##
## Computational engine: glmnet
```

```r
# recipe and workflow.

set.seed(350)
# varlist to keep
varlist <- c("Default", "duration", "amount", "installment", "age",
             "history", "purpose", "foreign")
# recipe
log_rec <-
  recipe(Default ~ ., data = german_train) %>%
  # remove vars not in varlist
  step_rm(setdiff(colnames(german_credit), varlist)) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())

# workflow
log_wf <-
  workflow() %>%
  add_model(log_mod) %>%
  add_recipe(log_rec)


# Tune grid


log_grid <- tibble(penalty = 10^seq(-4, -1, length.out = 30))

set.seed(350)
log_rs <-
  log_wf %>%
  tune_grid(german_fold,
```

```
            grid = log_grid,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(roc_auc))
log_rs

## # Tuning results
## # 3-fold cross-validation using stratification
## # A tibble: 3 x 5
##   splits            id    .metrics          .notes           .predictions
##   <list>            <chr> <list>            <list>           <list>
## 1 <split [500/250]> Fold1 <tibble [30 x 5]> <tibble [0 x 1]> <tibble [7,50
0 x 6~
## 2 <split [500/250]> Fold2 <tibble [30 x 5]> <tibble [0 x 1]> <tibble [7,50
0 x 6~
## 3 <split [500/250]> Fold3 <tibble [30 x 5]> <tibble [0 x 1]> <tibble [7,50
0 x 6~

log_rs %>%
  collect_metrics() %>%
  ggplot(aes(x = penalty, y = mean)) +
  geom_point() +
  geom_line() +
  ylab("Area under the ROC Curve") +
  scale_x_log10(labels = scales::label_number()) +
  theme_ipsum()
```




```r
top_models <-
  log_rs %>%
  show_best("roc_auc", n = 20) %>%
  arrange(penalty)
top_models %>% kbl(format = "pipe", booktabs = T)
```

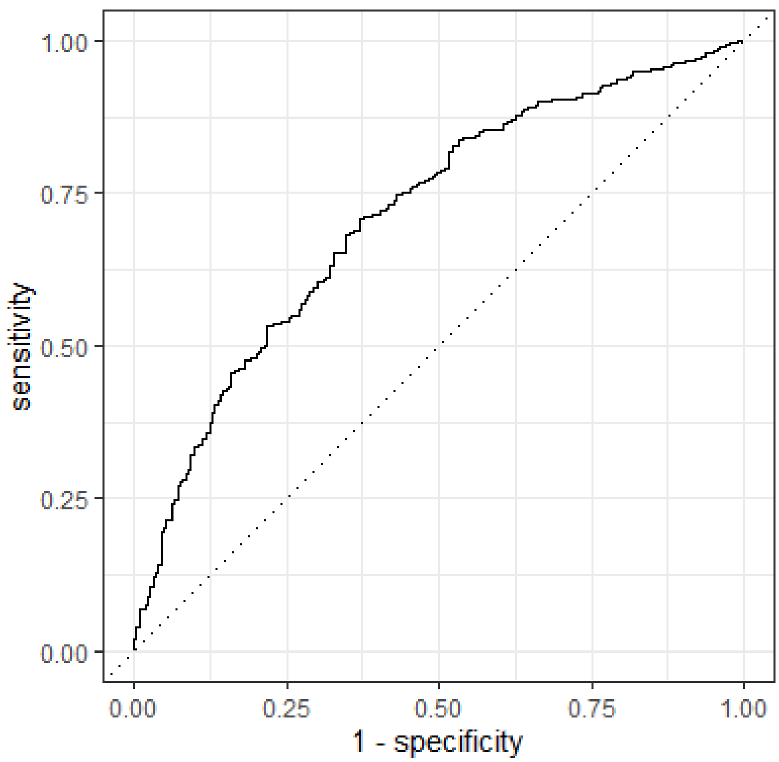
penalty	.metric	.estimator	mean	n	std_err	.config
0.0001000	roc_auc	binary	0.7156063	3	0.0286988	Preprocessor1_Model01
0.0001269	roc_auc	binary	0.7156063	3	0.0286988	Preprocessor1_Model02
0.0001610	roc_auc	binary	0.7156063	3	0.0286988	Preprocessor1_Model03
0.0002043	roc_auc	binary	0.7156063	3	0.0286988	Preprocessor1_Model04
0.0002593	roc_auc	binary	0.7156063	3	0.0286988	Preprocessor1_Model05
0.0003290	roc_auc	binary	0.7156063	3	0.0286988	Preprocessor1_Model06
0.0004175	roc_auc	binary	0.7156063	3	0.0286988	Preprocessor1_Model07
0.0005298	roc_auc	binary	0.7156571	3	0.0286530	Preprocessor1_Model08
0.0006723	roc_auc	binary	0.7155810	3	0.0286226	Preprocessor1_Model09
0.0008532	roc_auc	binary	0.7155302	3	0.0287753	Preprocessor1_Model10
0.0010826	roc_auc	binary	0.7155302	3	0.0287700	Preprocessor1_Model11
0.0013738	roc_auc	binary	0.7152762	3	0.0285927	Preprocessor1_Model12
0.0017433	roc_auc	binary	0.7152762	3	0.0286611	Preprocessor1_Model13
0.0022122	roc_auc	binary	0.7147429	3	0.0286857	Preprocessor1_Model14

0.0028072	roc_auc	binary	0.7142857	3	0.0286611	Preprocessor1_Model15
0.0035622	roc_auc	binary	0.7133206	3	0.0283669	Preprocessor1_Model16
0.0045204	roc_auc	binary	0.7124571	3	0.0281729	Preprocessor1_Model17
0.0057362	roc_auc	binary	0.7108317	3	0.0279406	Preprocessor1_Model18
0.0072790	roc_auc	binary	0.7089524	3	0.0271651	Preprocessor1_Model19
0.0092367	roc_auc	binary	0.7058794	3	0.0267176	Preprocessor1_Model20

```
log_rs %>%
  select_best()
```

```
## # A tibble: 1 x 2
##    penalty .config
##      <dbl> <chr>
## 1 0.000530 Preprocessor1_Model08
```

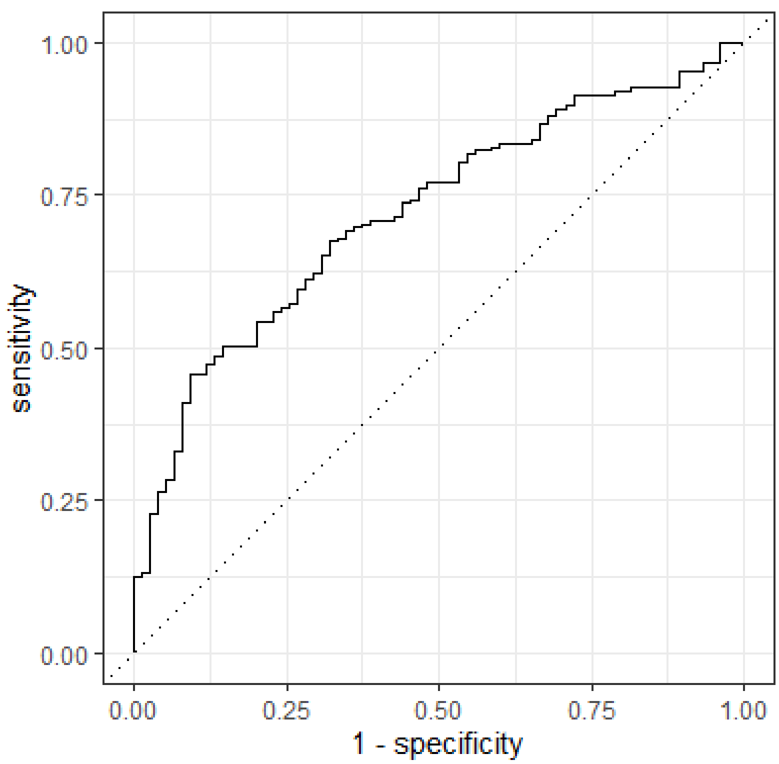
```
# Model 8 seems to be the best
# Graphically
log_best <-
  log_rs %>%
  collect_metrics() %>%
  arrange(penalty) %>%
  slice(8)
log_auc <-
  log_rs %>%
  collect_predictions(parameters = log_best) %>%
  roc_curve(Default, .pred_0) %>%
  mutate(model = "Logistic Regression")
autoplot(log_auc)
```




```
final_log_wf <-
  log_wf %>%
  finalize_workflow(log_best)
log_fit <-
  final_log_wf %>%
  last_fit(split = german_split)
log_fit %>% collect_metrics()

## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.704 Preprocessor1_Model1
## 2 roc_auc  binary         0.721 Preprocessor1_Model1

log_results <-
  log_fit %>%
  collect_predictions()

log_results

## # A tibble: 250 x 7
##    id               .pred_0 .pred_1  .row .pred_class Default .config
##    <chr>              <dbl>   <dbl> <int> <fct>       <fct>   <chr>
##  1 train/test split   0.805   0.195    14 0           1       Preprocessor
1_Mod~
##  2 train/test split   0.654   0.346    16 0           1       Preprocessor
1_Mod~
```

```
## 3 train/test split   0.842  0.158     22 0            0        Preprocessor
1_Mod~
## 4 train/test split   0.667  0.333     27 0            0        Preprocessor
1_Mod~
## 5 train/test split   0.661  0.339     32 0            0        Preprocessor
1_Mod~
## 6 train/test split   0.917  0.0827    34 0            0        Preprocessor
1_Mod~
## 7 train/test split   0.578  0.422     41 0            0        Preprocessor
1_Mod~
## 8 train/test split   0.749  0.251     42 0            0        Preprocessor
1_Mod~
## 9 train/test split   0.937  0.0627    48 0            0        Preprocessor
1_Mod~
## 10 train/test split  0.803  0.197     49 0            0        Preprocessor
1_Mod~
## # ... with 240 more rows
```

```
log_results %>%
  roc_curve(Default, .pred_0) %>%
  autoplot()
```




```
# Confusion matrix
cm <- log_results %>%
  conf_mat(Default, .pred_class)
cm
```

```
##           Truth
## Prediction   0   1
##          0 151  50
##          1  24  25

# Poor sampling
german_credit %>%
  group_by(history) %>%
  tally() %>%
  kbl(format = "pipe")
```

history	n
good	89
poor	618
terrible	293

Our model is accurate roughly 74.4 percent of the time, which is not ideal since our null model that assumes no one will default would be correct 70 percent of the time. We believe the data is not likely ideal for predicting due to the weight of the poorly sampled history variable. Specifically, observe the vast disparity in sampling above.

Problem 4: Children and Hotel Reservations

```
hotels_dev <-
  read_data("hotels_dev.csv") %>%
  mutate(children = as.factor(children))

##
## -- Column specification --------------------------------------------------
------
## cols(
##   .default = col_double(),
##   hotel = col_character(),
##   meal = col_character(),
##   market_segment = col_character(),
##   distribution_channel = col_character(),
##   reserved_room_type = col_character(),
##   assigned_room_type = col_character(),
##   deposit_type = col_character(),
##   customer_type = col_character(),
##   required_car_parking_spaces = col_character(),
##   arrival_date = col_date(format = "")
## )
## i Use `spec()` for the full column specifications.

  hotels_dev %>%
  count(children) %>%
  mutate(prop = round( n/sum(n), 3)) %>%
  mutate(children = if_else(children == 1, "children", "none")) %>%
  kbl("pipe")
```

children	n	prop
none	41365	0.919
children	3635	0.081

```
# Children only make up about 8% of the sample

hotel_splits <- initial_split(hotels_dev, strata = children)
hotel_train <- training(hotel_splits)
hotel_test <- testing(hotel_splits)

train_val_set <- validation_split(hotel_train, strata = children, prop = 0.8)

# Proportion of children in train/test
# train
hotel_train %>%
  count(children) %>%
  mutate(prop = round( n/sum(n), 3)) %>%
  mutate(children = if_else(children == 1, "children", "none")) %>%
  kbl("pipe")
```

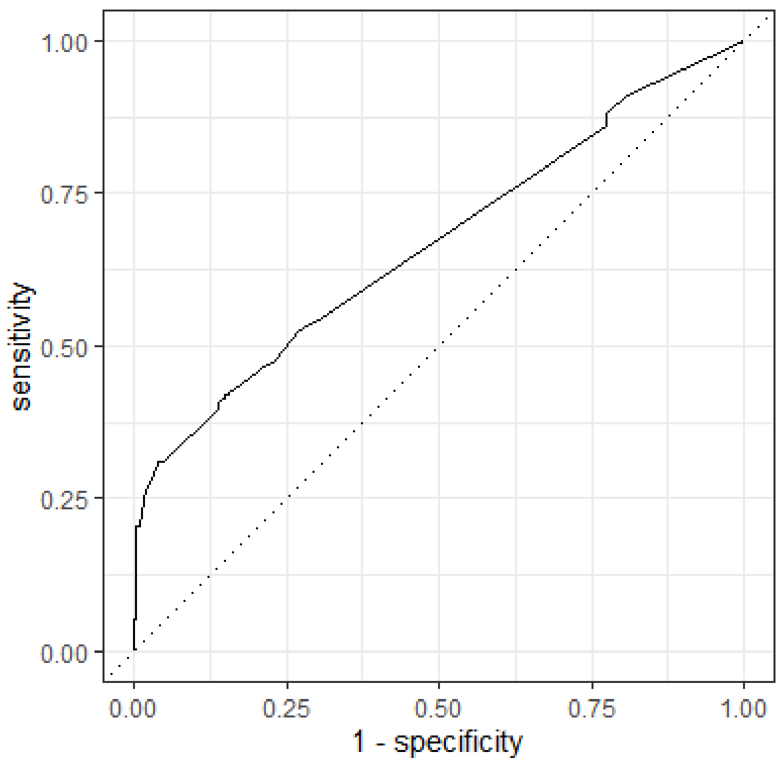
children	n	prop
none	31033	0.919
children	2717	0.081

```
# test
hotel_test %>%
  count(children) %>%
  mutate(prop = round( n/sum(n), 3)) %>%
  mutate(children = if_else(children == 1, "children", "none")) %>%
  kbl("pipe")
```

children	n	prop
none	10332	0.918
children	918	0.082

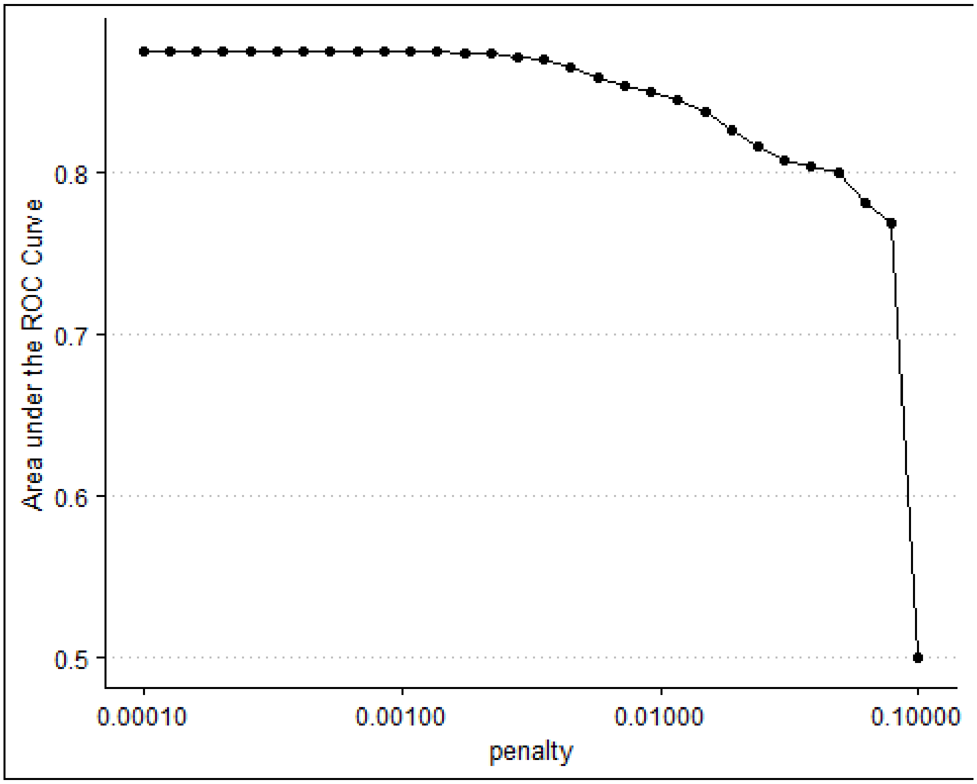
```
# Both the splits are similar in proportion for children and no children.

## Baseline models

# penalized logistic regression model

log_mod_base1 <-
  logistic_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet")

# Preprocess recipe
log_mod_base1_recipe <-
  recipe(children ~ market_segment + adults + customer_type + is_repeated_gue
st,
```

```r
         data = hotel_train) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())

# Preprocess
log_mod_base1_wrkflow <-
  workflow() %>%
  add_model(log_mod_base1) %>%
  add_recipe(log_mod_base1_recipe)


# Tune hyperparameter

lr_reg_grid <- tibble(penalty = 10^seq(-4, -1, length.out = 30))
log_base1_res <-
  log_mod_base1_wrkflow %>%
  tune_grid(train_val_set,
            grid = lr_reg_grid,
            control = control_grid(save_pred = T),
            metrics = metric_set(roc_auc))


# select the best model
log_base1_res %>%
  select_best()

## # A tibble: 1 x 2
##   penalty .config
##     <dbl> <chr>
## 1 0.00924 Preprocessor1_Model20

best_mod_base1 <-
  log_base1_res %>%
  collect_metrics() %>%
  slice(20)

# roc curve
log_base1_res %>%
  collect_predictions(parameters = best_mod_base1) %>%
  roc_curve(children, .pred_0) %>%
  mutate(model = "Logistic Regression") %>%
  autoplot()
```

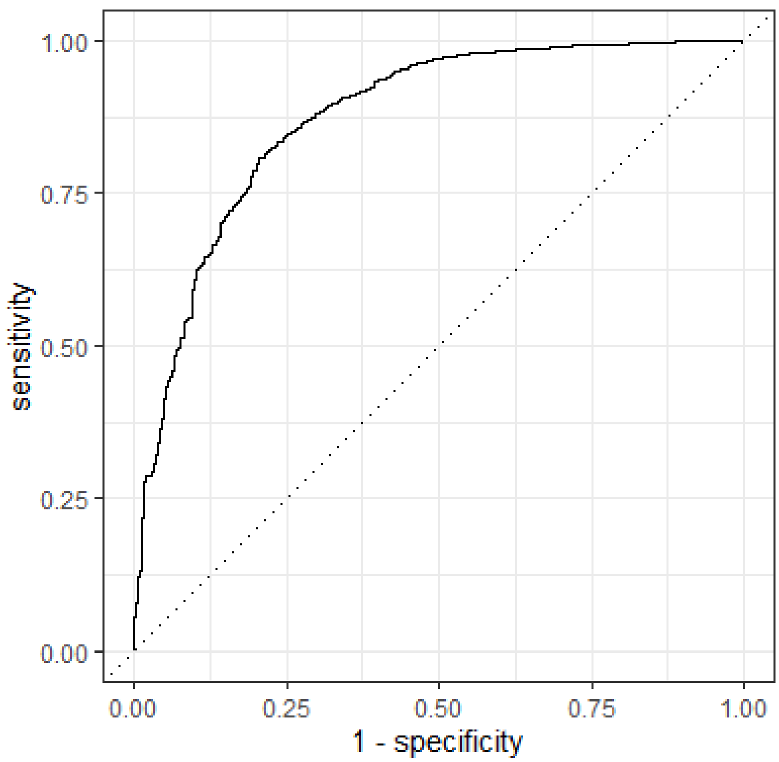



```
# This is awful
# Confusion matrix

param_final <-
  log_base1_res %>%
  select_best(metric = "roc_auc")
log_mod_base1_wrkflow <-
  log_mod_base1_wrkflow %>%
  finalize_workflow(param_final)
base1_fit <-
  log_mod_base1_wrkflow %>%
  last_fit(hotel_splits)
base1_pred <-
  base1_fit %>%
  collect_predictions()
base1_pred %>%
  conf_mat(truth = children, estimate = .pred_class)

##             Truth
## Prediction     0     1
##          0 10332   918
##          1     0     0

# Baseline 2

# Preprocess recipe
holidays <- c("AllSouls", "AshWednesday", "ChristmasEve", "Easter",
```

```
              "ChristmasDay", "GoodFriday", "NewYearsDay", "PalmSunday")
log_mod_base2_recipe <-
  recipe(children ~ .,
         data = hotel_train) %>%

  step_date(arrival_date) %>%
  step_holiday(arrival_date, holidays = holidays) %>%
  step_rm(arrival_date) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%

  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())
# Preprocess
log_mod_base2_wrkflow <-
  workflow() %>%
  add_model(log_mod_base1) %>%
  add_recipe(log_mod_base2_recipe)


# Tune Hyperparameter

log_base2_res <-
  log_mod_base2_wrkflow %>%
  tune_grid(train_val_set,
            grid = lr_reg_grid,
            control = control_grid(save_pred = T),
            metrics = metric_set(roc_auc))


log_base2_res %>%
  collect_metrics() %>%
  ggplot(aes(x = penalty, y = mean)) +
  geom_point() +
  geom_line() +
  ylab("Area under the ROC Curve") +
  scale_x_log10(labels = scales::label_number()) +
  theme_clean()
```




```
# select the best model
log_base2_res %>%
  select_best()

## # A tibble: 1 x 2
##    penalty .config
##      <dbl> <chr>
## 1 0.000530 Preprocessor1_Model08

best_mod_base2 <-
  log_base1_res %>%
  collect_metrics() %>%
  slice(8)

# roc curve
log_base2_res %>%
  collect_predictions(parameters = best_mod_base2) %>%
  roc_curve(children, .pred_0) %>%
  mutate(model = "Logistic Regression") %>%
  autoplot()
```



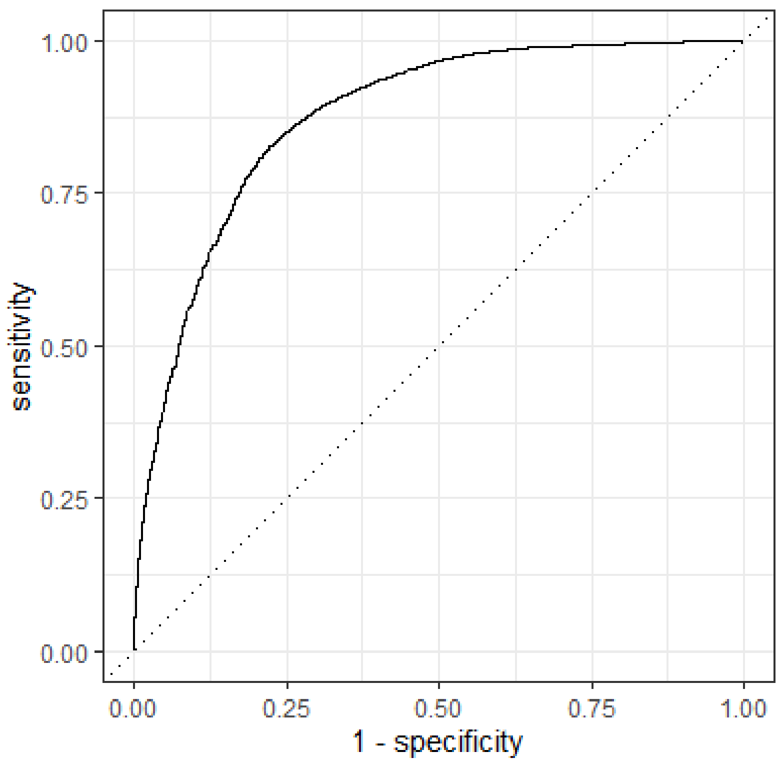

```
# Not great, but better than baseline1 at least

param_final <-
  log_base2_res %>%
  select_best(metric = "roc_auc")
log_mod_base2_wrkflow <-
  log_mod_base2_wrkflow %>%
  finalize_workflow(param_final)
base2_fit <-
  log_mod_base2_wrkflow %>%
  last_fit(hotel_splits)
base2_pred <-
  base2_fit %>%
  collect_predictions()
base2_pred %>%
  conf_mat(truth = children, estimate = .pred_class)

##             Truth
## Prediction     0     1
##            0 10218   620
##            1   114   298

# Better


## Best Linear Model
```

```r
set.seed(400)
# train/test

hotel_splits2 <- initial_split(hotels_dev, strata = children)
hotel_train2 <- training(hotel_splits2)
hotel_test2 <- testing(hotel_splits2)

# cross-val folds
hotel_cv <- vfold_cv(hotel_train2, v = 10, repeats = 1, strata = children)

# validation set
hotels_val <- read_data("hotels_val.csv") %>%
  mutate(children = as.factor(children))

##
## -- Column specification ----------------------------------------------------
------
## cols(
##   .default = col_double(),
##   hotel = col_character(),
##   meal = col_character(),
##   market_segment = col_character(),
##   distribution_channel = col_character(),
##   reserved_room_type = col_character(),
##   assigned_room_type = col_character(),
##   deposit_type = col_character(),
##   customer_type = col_character(),
##   required_car_parking_spaces = col_character(),
##   arrival_date = col_date(format = "")
## )
## i Use `spec()` for the full column specifications.

log_mod_rec <-
  recipe(children ~ .,
         data = hotel_train2) %>%

  step_date(arrival_date) %>%
  step_holiday(arrival_date, holidays = timeDate::listHolidays("US")) %>%
  step_rm(arrival_date) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())
log_mod <-
  logistic_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet")

# Preprocess
log_mod_wrkflow <-
  workflow() %>%
```

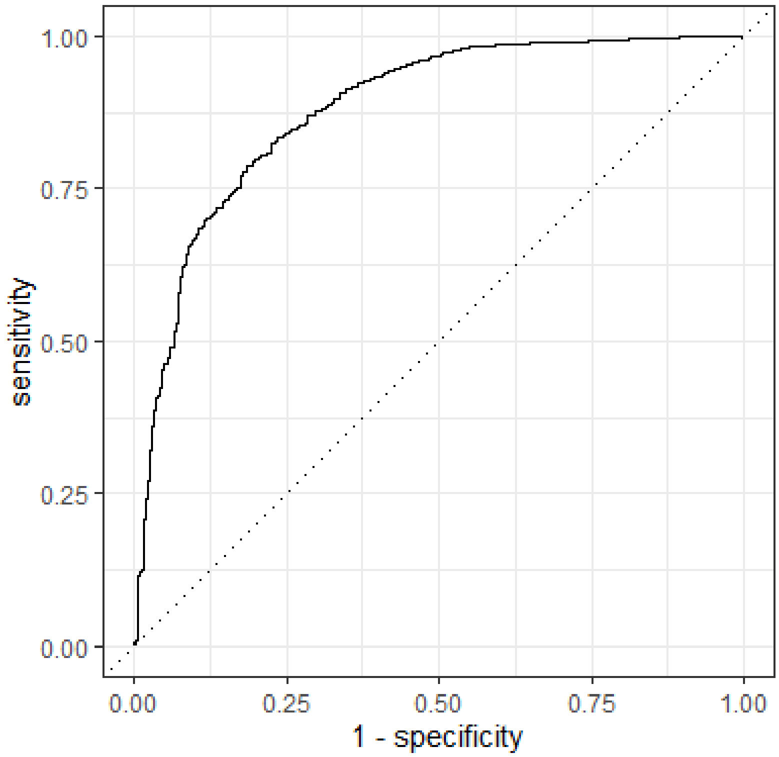
```r
  add_model(log_mod) %>%
  add_recipe(log_mod_rec)

# fit to  validation set
hotel_res <-
  log_mod_wrkflow %>%
  tune_grid(grid = lr_reg_grid,
            resamples = hotel_cv,
            control = control_grid(save_pred = T),
            metrics = metric_set(roc_auc))
top_models <-
  hotel_res %>%
  show_best("roc_auc", n = 15) %>%
  arrange(penalty)
hotel_best <-
  hotel_res %>%
  collect_metrics() %>%
  arrange(penalty) %>%
  slice(8)

hotel_best %>% kbl("pipe")
```

penalty	.metric	.estimator	mean	n	std_err	.config
0.0013738	roc_auc	binary	0.8749515	10	0.0035659	Preprocessor1_Model08

```r
# roc curve
hotel_res %>%
  collect_predictions(parameters = hotel_best) %>%
  roc_curve(children, .pred_0) %>%
  mutate(model = "Logistic Regression") %>%
  autoplot()
```




```
### Model Validation 1 using new dataset

trained_wf <-
  log_mod_wrkflow %>%
  finalize_workflow(hotel_best) %>%
  fit(hotels_dev)
hotel_preds <-
  trained_wf %>%
  predict(hotels_val) %>%
  bind_cols(hotels_val %>% select(children))
hotel_final_pred <-
  trained_wf %>%
  predict(hotels_val, type = "prob") %>%
  bind_cols(hotel_preds)
hotel_final_pred %>%
  conf_mat(truth = children, .pred_class)

##             Truth
## Prediction    0    1
##           0 4542  261
##           1   55  141

hotel_final_pred %>%
  roc_curve(children, .pred_0) %>%
  mutate(model = "Logistic Regression") %>%
  autoplot()
```




```
### Model Validation 2

# create v-folds
hotel_folds <- vfold_cv(hotels_val, v = 20)


hotel_fold_fit <-
  trained_wf %>%
  fit_resamples(
    resamples = hotel_folds,
    control = control_resamples(save_pred = TRUE)
  )


# predicted probabilities
pred_sums <- list()
for (i in 1:20) {
  pred_sums <-
    hotel_fold_fit$.predictions[[i]] %>%
    summarize(sum_pred = sum(as.numeric(.pred_class))) %>%
    pull(sum_pred) %>%
    append(pred_sums)
}

# actual probabilities
actual_sums <- list()
```

```r
for (i in 1:20) {
  actual_sums <-
    hotel_fold_fit$.predictions[[i]] %>%
    summarize(sum_actual = sum(as.numeric(children))) %>%
    pull(sum_actual) %>%
    append(actual_sums)
}
# colnames
names <- tibble("Folds" = c("Actual", "Predicted"))

probs <-
  as_tibble(actual_sums, .name_repair = "unique") %>%
  rbind(as_tibble(pred_sums, .name_repair = "unique"))

## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
## * ...

## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
## * ...

# Table
cbind(names, probs) %>%
  kable(col.names =
          append("Folds", make.unique(c("Folds", rep("v", 21)), sep = "")[3:2
2]),
        caption = "Sum of probabilities")
```

Sum of probabilities

Folds

v1

v2

v3

v4

v5

v6

v7

v8

v9

v10

v11

v12

v13

v14

v15

v16

v17

v18

v19

v20

Actual

268

273

273

266

265

267

275

268

271

271

273

269

272

263

273

276

269

275

266

268

Predicted

255

260

262

255

256

261

264

258

260

266

263

262

259

256

261

259

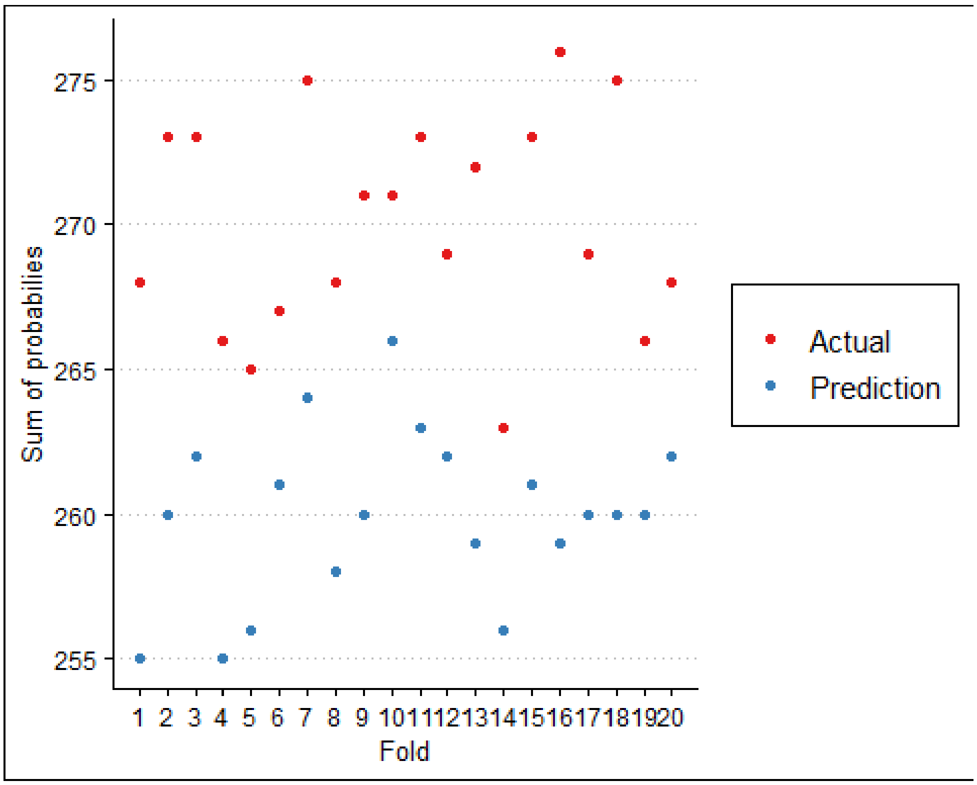
260

260

260

262

```
tibble(fold = seq(1, 20, 1),
       Actual = unlist(actual_sums),
       Prediction = unlist(pred_sums)) %>%
  pivot_longer(!fold, names_to = "names", values_to = "vals") %>%
  ggplot() +
  geom_point(aes(x = fold, y = vals, color = names))  +
  labs(x = "Fold", y = "Sum of probabilies") +
  scale_x_continuous(breaks = seq(1, 20, 1)) +
  scale_color_brewer(palette = "Set1") +
  theme_clean() +
  theme(legend.title = element_blank())
```




```
mean_err <- sum(probs[1,] - probs[2,]) / 20
glue::glue("The average mean error is {mean_err}")

## The average mean error is 10.1

The sum of probabilities is fairly similar across folds.
```