KNN Classification

Classification

Our response variable is a category for each observation, not a number.

Example: can we predict party affiliation as a function of age?

We have data including the following variables for each of 944 participants in the 1996 American National Election Study:

- party affiliation ("Dem", "Ind", or "Rep"). This is our response variable, y_i
- age (range 19 to 91). This is our explanatory variable or feature, x_i

```
nes96 %>%
  select(age, party) %>%
  head()
```

```
##   age party
## 1  36   Rep
## 2  20   Dem
## 3  24   Dem
## 4  28   Dem
## 5  68   Dem
## 6  21   Dem
```

```
nrow(nes96)
```

```
## [1] 944
```

Here are the counts of how many participants are in each party:

```
nes96 %>%
  count(party)
```

```
## # A tibble: 3 x 2
##   party     n
##   <chr> <int>
## 1 Dem     488
## 2 Ind      37
## 3 Rep     419
```

Train/test split

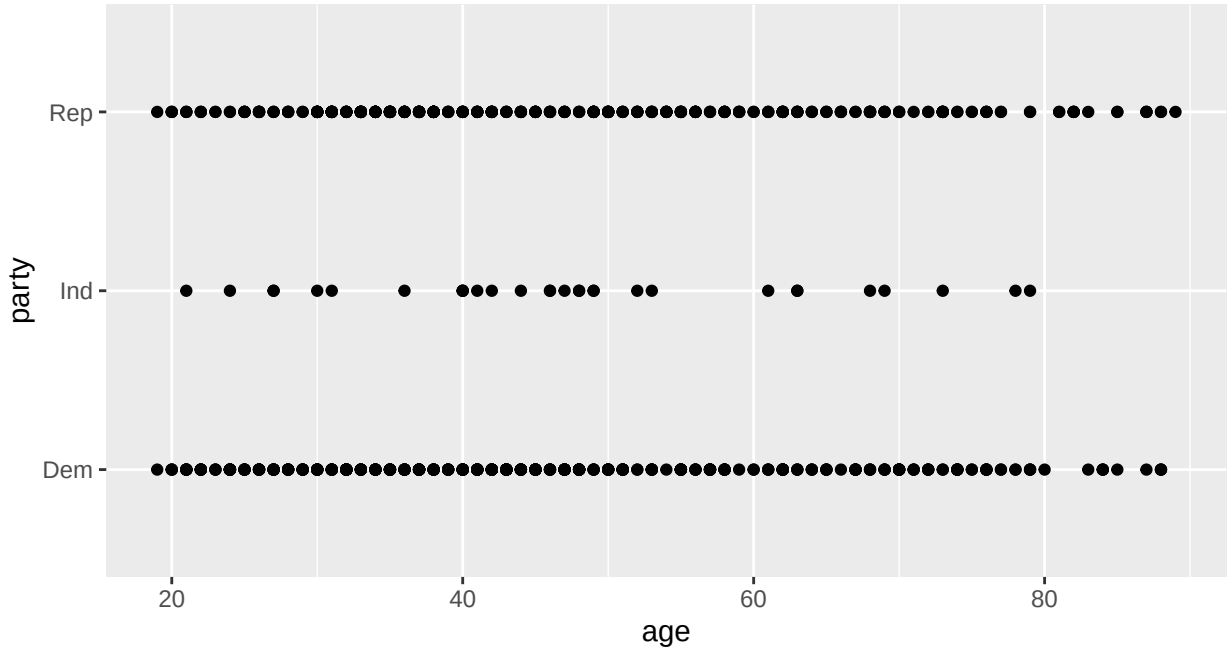
As with regression, we will evaluate model performance based on a test set.

```
set.seed(88412)
train_inds <- caret::createDataPartition(nes96$party, p = 0.8)
train_nes96 <- nes96 %>% slice(train_inds[[1]])
test_nes96 <- nes96 %>% slice(-train_inds[[1]])
```

Some plots

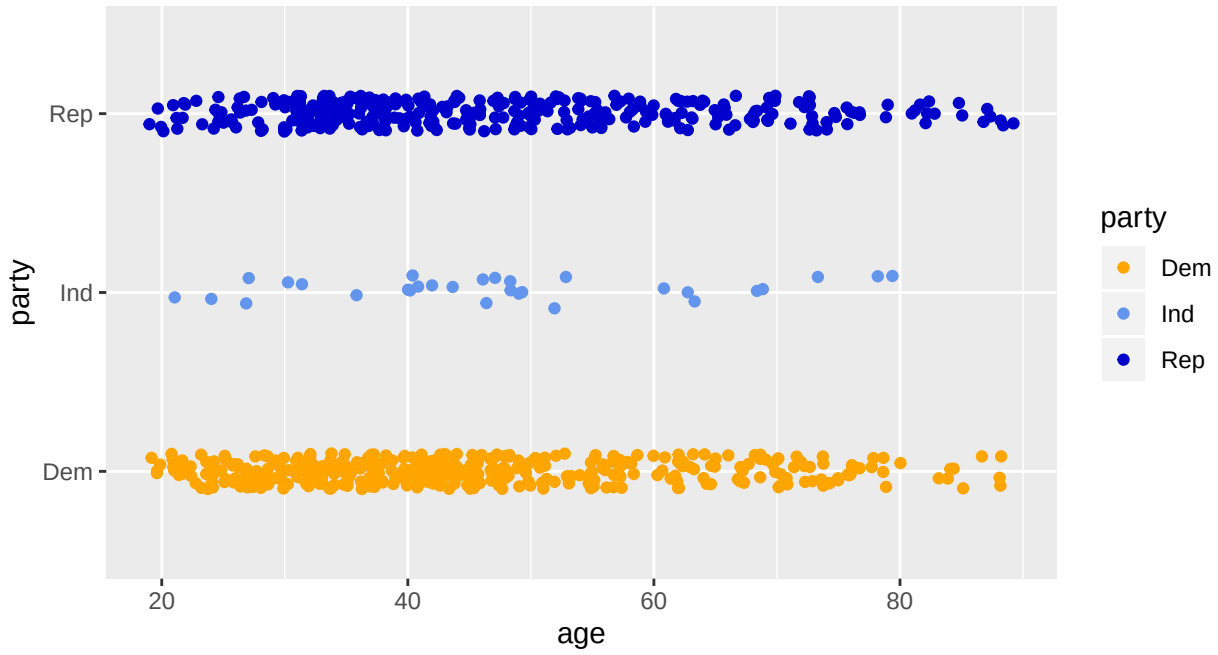
A scatter plot isn't that useful:

```
ggplot(data = train_nes96, mapping = aes(x = age, y = party)) +
  geom_point()
```



We can jitter the points, but still not that helpful:

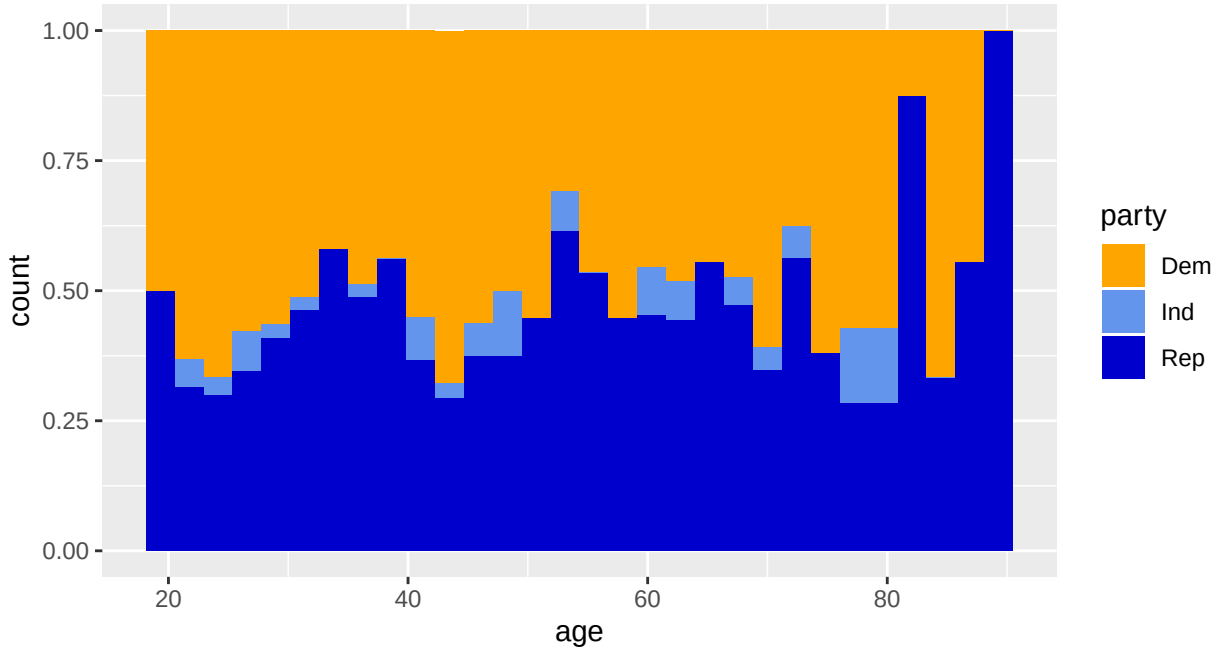
```
ggplot(data = train_nes96, mapping = aes(x = age, y = party, color = party)) +
  geom_point(position = position_jitter(height = 0.1)) +
  scale_color_manual(values = c("orange", "cornflowerblue", "mediumblue"))
```



How about a histogram? `postion = "fill"` says that within each bin, we want the bars to add up to 100%.

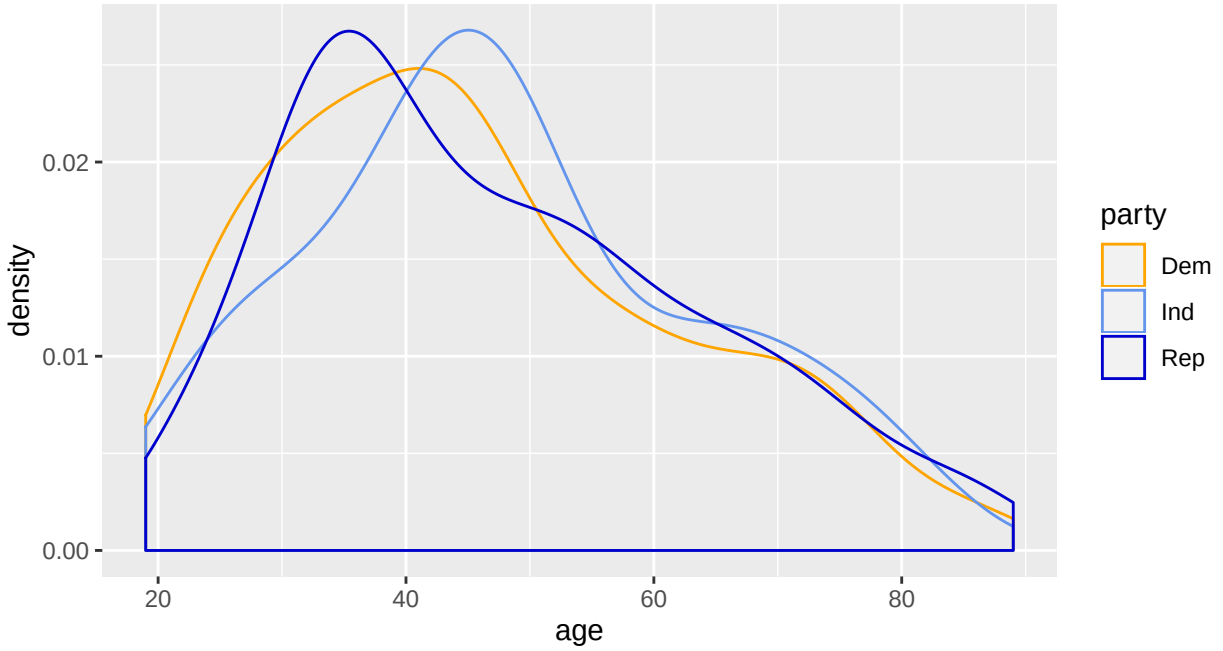
```
ggplot(data = train_nes96, mapping = aes(x = age, fill = party)) +
  geom_histogram(position = "fill") +
  scale_fill_manual(values = c("orange", "cornflowerblue", "mediumblue"))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



All of these put our response on the vertical axis, which is the easiest way to think about the model. We could also do something like a density plot of the explanatory variable colored by the response:

```
ggplot(data = train_nes96, mapping = aes(x = age, color = party)) +
  geom_density() +
  scale_color_manual(values = c("orange", "cornflowerblue", "mediumblue"))
```



R Code for K Nearest Neighbors for Classification

```
# "train" the KNN model
# this code is exactly the same as the code to do KNN regression!
knn_fit <- train(
  form = party ~ age,
  data = train_nes96,
  method = "knn",
  preProcess = "scale",
  trControl = trainControl(method = "none"),
  tuneGrid = data.frame(k = 100)
)

# to get estimated class membership probabilities, specify type = "prob" in the predict function
f_hats <- predict(knn_fit, newdata = test_nes96, type = "prob")
head(f_hats)
```

```
##         Dem         Ind       Rep
## 1 0.6310680 0.038834951 0.3300971
## 2 0.5045045 0.045045045 0.4504505
## 3 0.4690265 0.035398230 0.4955752
## 4 0.4433962 0.009433962 0.5471698
## 5 0.6310680 0.038834951 0.3300971
## 6 0.5488722 0.037593985 0.4135338
```

```
# to get the most likely class, leave out type or specify type = "raw" (the default)
# if the estimated class probability is the same for two classes, ties are broken at random
y_hats <- predict(knn_fit, newdata = test_nes96, type = "raw")
head(y_hats)
```

```
## [1] Dem Dem Rep Rep Dem Dem
## Levels: Dem Ind Rep
```

```
# classification error rate: what proportion of predicted parties are not equal to the observed party?
mean(y_hats != test_nes96$party)
```

```
## [1] 0.513369
```

```
# how does this compare to just predicting the most common class in the training set?
train_nes96 %>% count(party)
```

```
## # A tibble: 3 x 2
##   party     n
##   <chr> <int>
## 1 Dem     391
## 2 Ind      30
## 3 Rep     336
```

```
mean("Dem" != test_nes96$party)
```

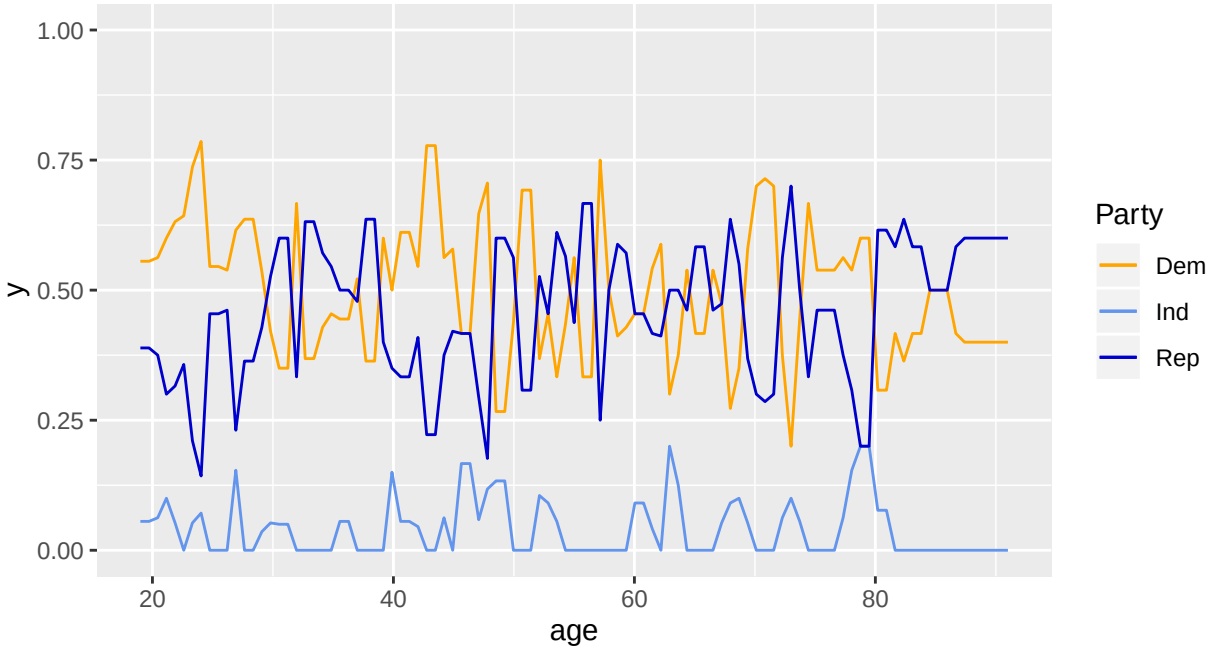
```
## [1] 0.4812834
```

Our model does slightly better than just guessing the most common party in the training set.

Here's a way to plot class membership probabilities as functions of age. It's admittedly a little awkward.

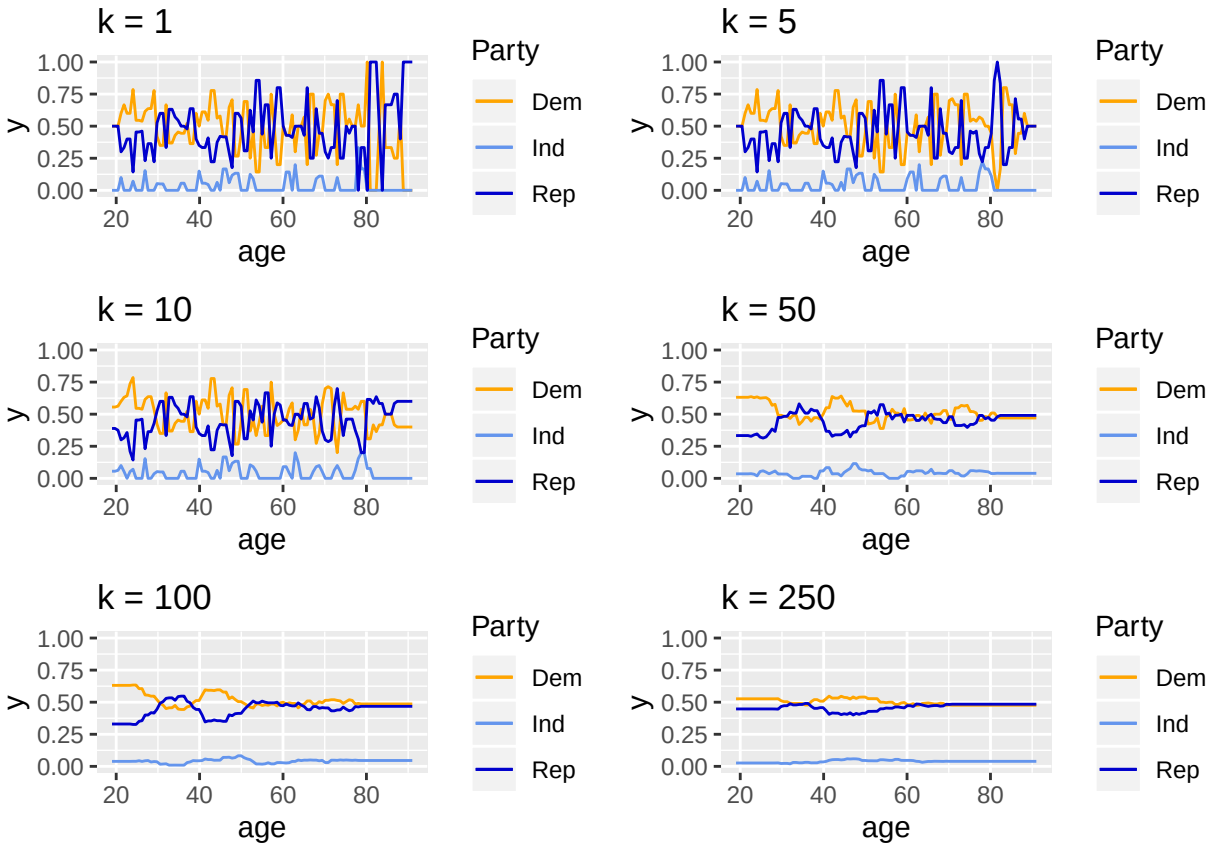
```
predict_knn_probs <- function(x, party) {
  f_hats <- predict(knn_fit, newdata = data.frame(age = x), type = "prob")
  f_hats[[party]]
}

ggplot(data = nes96, mapping = aes(x = age)) +
  stat_function(fun = predict_knn_probs,
    args = list(party = "Dem"),
    mapping = aes(color = "Dem")) +
  stat_function(fun = predict_knn_probs,
    args = list(party = "Ind"),
    mapping = aes(color = "Ind")) +
  stat_function(fun = predict_knn_probs,
    args = list(party = "Rep"),
    mapping = aes(color = "Rep")) +
  scale_color_manual("Party", values = c("orange", "cornflowerblue", "mediumblue")) +
  ylim(0, 1)
```



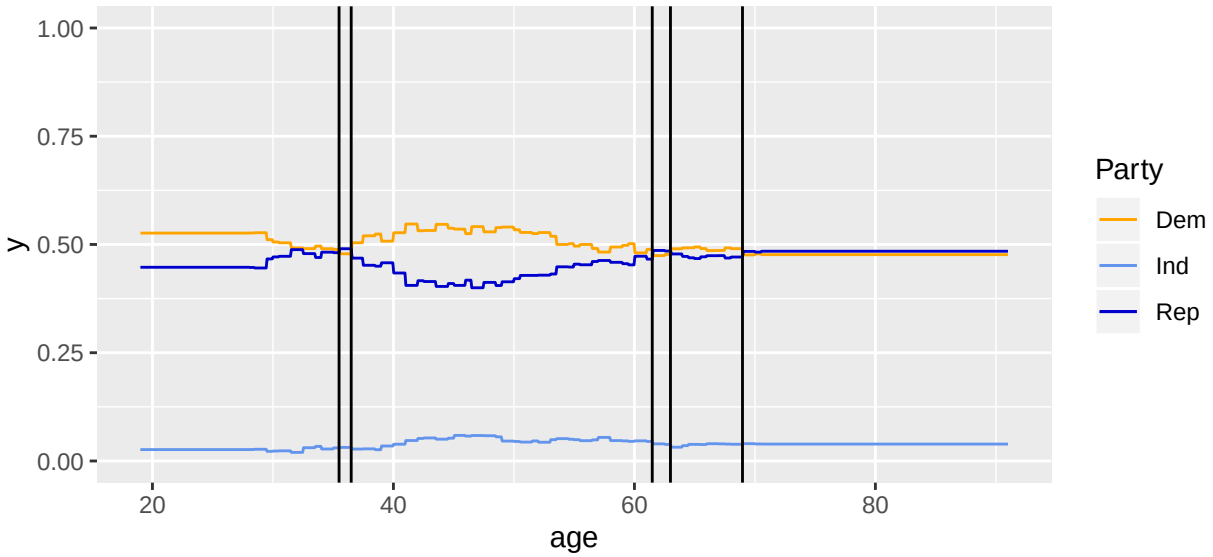
Flexibility is determined by k

Here are plots of the estimated class probability functions for several values of k (code suppressed):



Decision Boundaries

We won't explicitly calculate this for KNN, but it's nice to have in mind the concept of a *decision boundary*: the point at which the predicted value (class with highest estimated probability) changes. I've indicated the decision boundaries on the plot below for $k = 250$:



Note that there are generally fewer decision boundaries as k increases.

KNN with 2 features

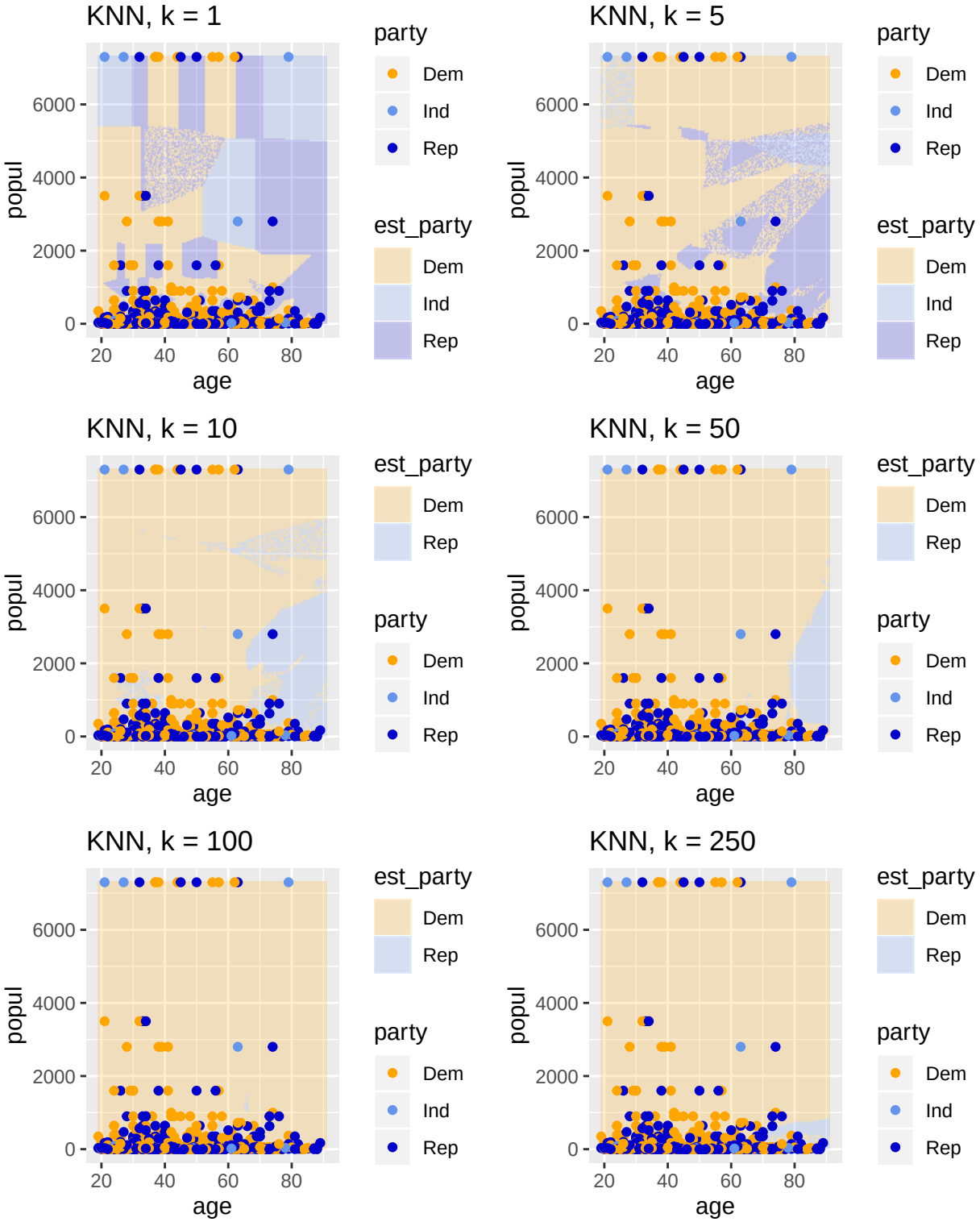
Suppose we use two variables to predict party affiliation:

- `age` (range 19 to 91). This is our first explanatory variable or feature, x_{i1}
- `popul` (range 0 to 7300) population of respondent's location in 1000s of people. This is our second feature, x_{i2}

With 2 inputs, the estimated class probability functions would have to be visualized in 3 dimensions (age, popul, and estimated class probability).

Instead, it's easier to display the decision boundaries in the two-dimensional feature space of values of (age, popul).

The plots below show these for a range of values of k:



Here's how you could make one of these plots:

```
# "train" the KNN model
knn_fit <- train(
  form = party ~ age + popul,
  data = train_nes96,
  method = "knn",
  preProcess = "scale",
  trControl = trainControl(method = "none"),
  tuneGrid = data.frame(k = 5)
)

# a grid of values for age and popul at which to get the estimated class.
# it's not a test data set in the sense that we don't have observations of party to go with these points,
# but we will treat it as a "test set" in the sense that we will obtain predictions at these points
test_grid <- expand.grid(
  age = seq(from = 19, to = 91, length = 201),
  popul = seq(from = 19, to = 7300, length = 201)
)
head(test_grid)
```

```
##     age popul
## 1 19.00    19
## 2 19.36    19
## 3 19.72    19
## 4 20.08    19
## 5 20.44    19
## 6 20.80    19
```

```
# use predict to find the estimated most likely class at each point in our grid
y_hats <- predict(knn_fit, newdata = test_grid, type = "raw")

# add the estimated types into the test_grid data frame
background_knn <- test_grid %>%
  mutate(
    est_party = y_hats
  )

# make the plot.  geom_raster does the shading in the background, alpha = 0.2 makes it transparent
ggplot() +
  geom_raster(data = background_knn,
    mapping = aes(x = age, y = popul, fill = est_party), alpha = 0.2) +
  geom_point(data = train_nes96, mapping = aes(x = age, y = popul, color = party)) +
  scale_color_manual("party", values = c("orange", "cornflowerblue", "mediumblue")) +
  scale_fill_manual(values = c("orange", "cornflowerblue", "mediumblue")) +
  ggtitle("KNN, k = 5")
```