Classification
(knn)

Agenda

- KNN concepts
- Preprocessing
  - Normalization
  - Dummy Coding
- Prediction
- Evaluation
- Implementation

Classification

- Predicting categorical values
- Methods
  - Nearest Neighbor
  - Naive Bayes
  - Decision Trees
  - SVM (dual use)
  - Neural Network (dual use)

KNN: K Nearest Neighbor

When to use

- relationships among the features and the target classes are numerous, complicated, or otherwise extremely difficult to understand

- items of similar class type tend to be fairly homogeneous

KNN Strengths and Weaknesses

Pros

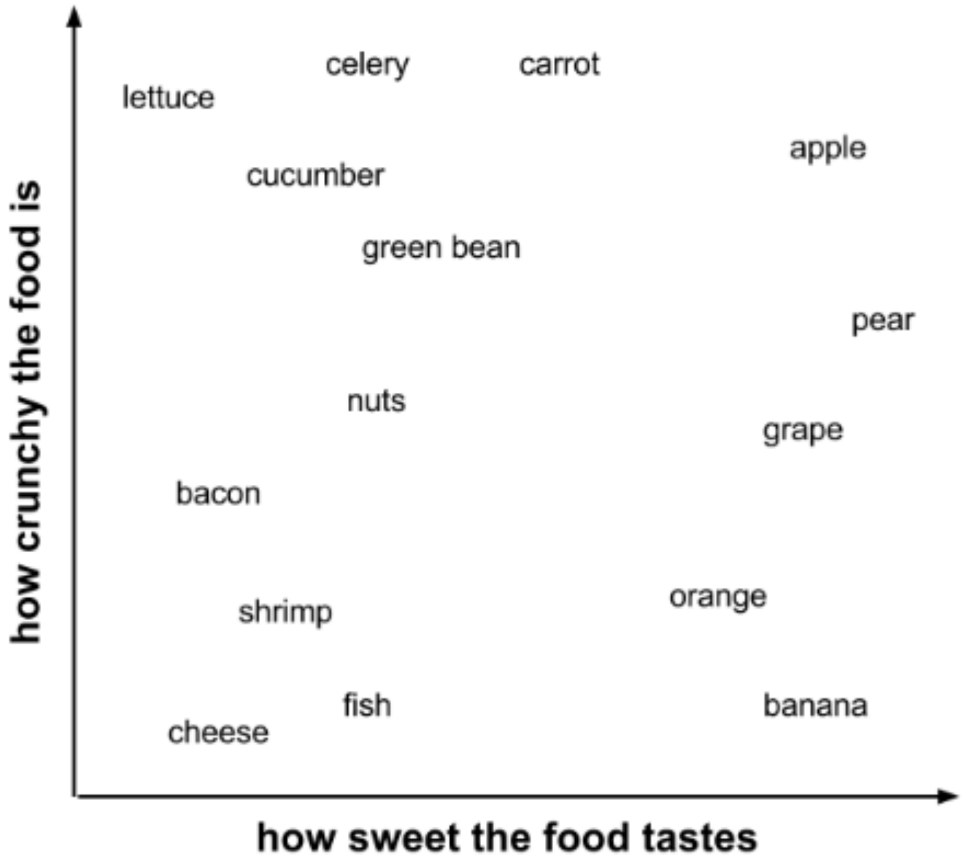
+ Simple and effective
+ No assumptions about the data distributions
+ Fast training

Cons

- There is no reusable / interpretable model
- Classification is slow
- Memory dependent
- Requires pre-processing of nominal features and missing data

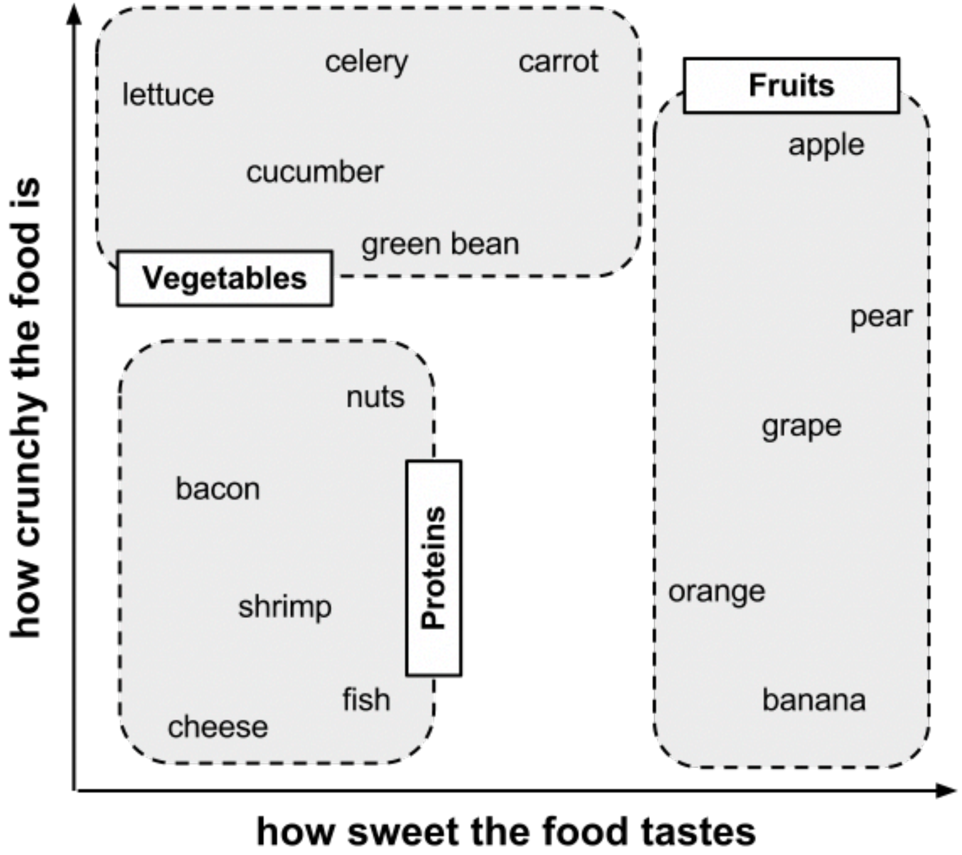
KNN illustration

ingredient	sweetness	crunchiness	food type
apple	10	9	fruit
bacon	1	4	protein
banana	10	1	fruit
carrot	7	10	vegetable
celery	3	10	vegetable
cheese	1	1	protein

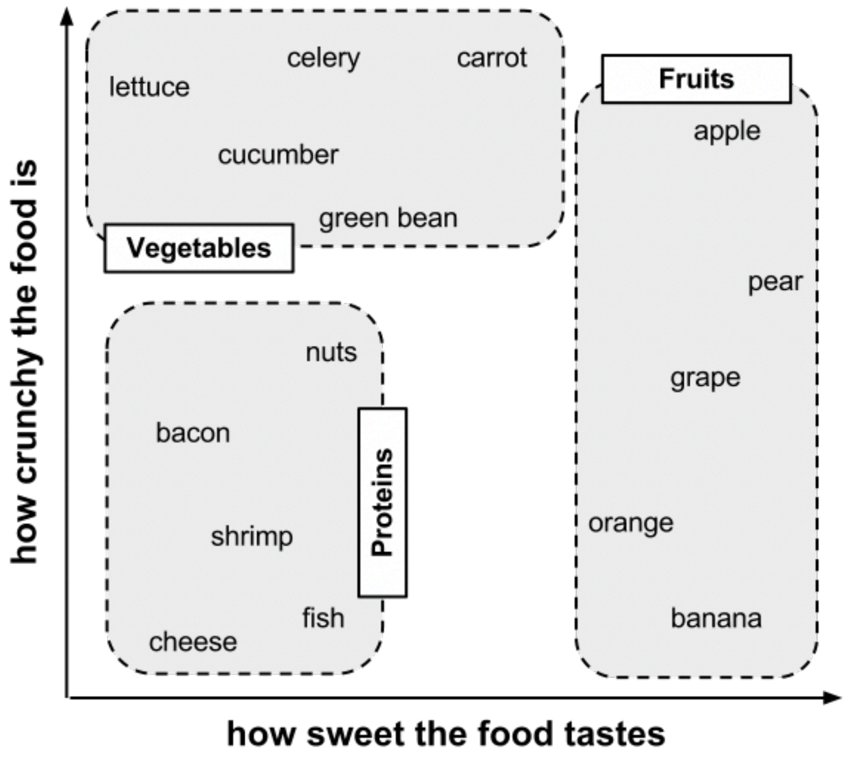


KNN illustration

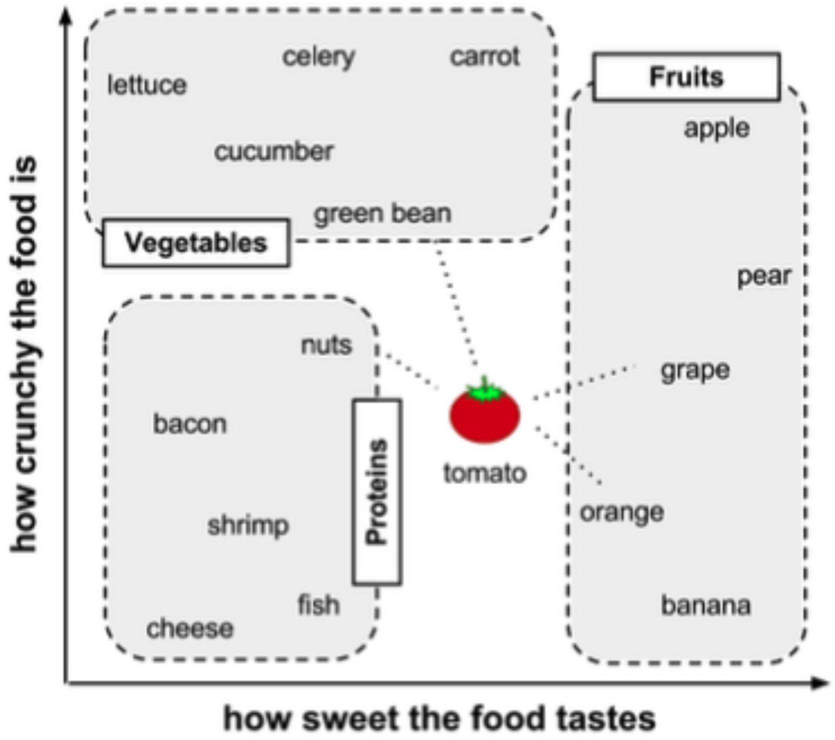
ingredient	sweetness	crunchiness	food type
apple	10	9	fruit
bacon	1	4	protein
banana	10	1	fruit
carrot	7	10	vegetable
celery	3	10	vegetable
cheese	1	1	protein



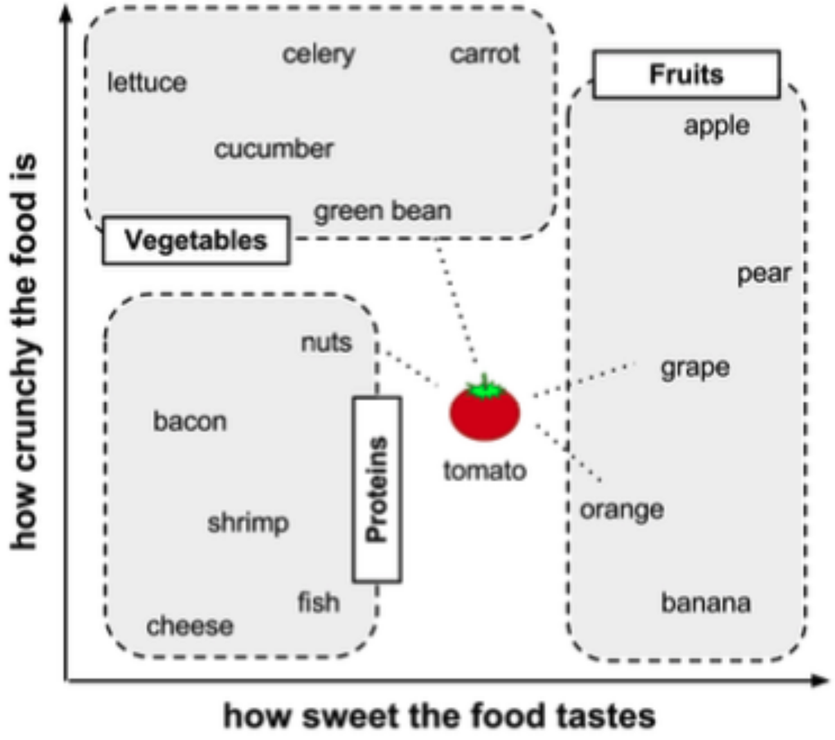
KNN illustration



KNN illustration



KNN illustration



Which category does tomato belong to ?

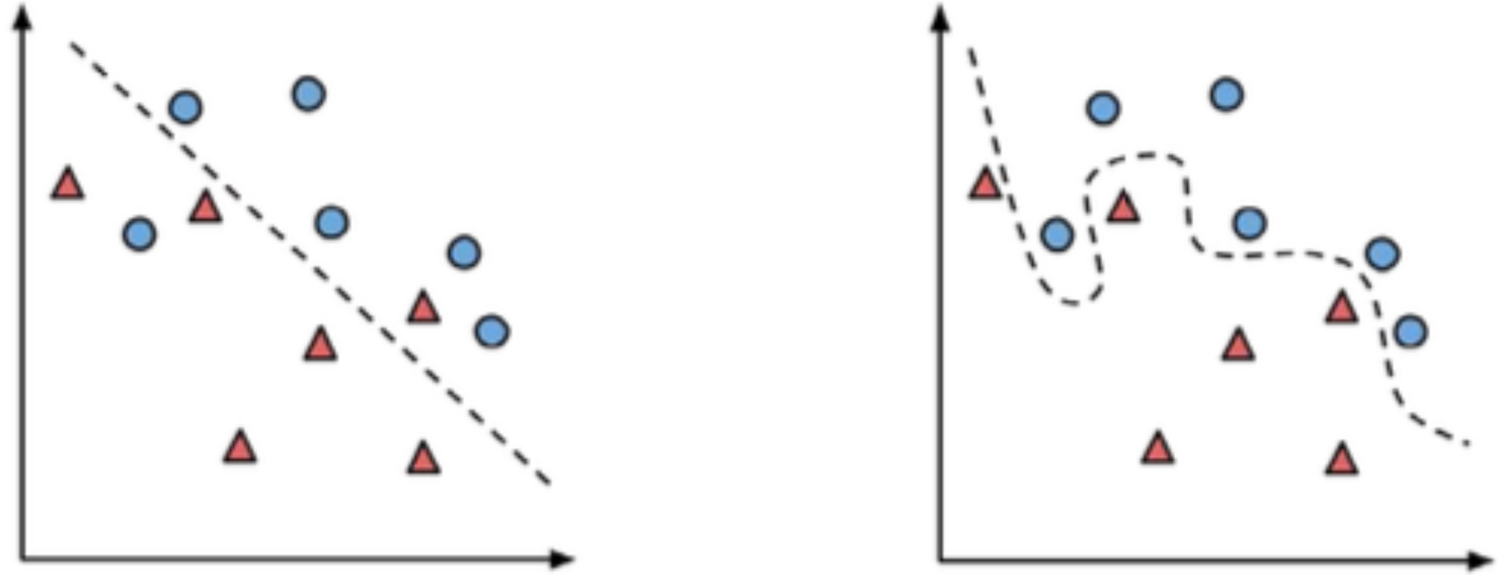
Tomato: sweetness=6 ; crunchiness=4

$$\mathrm{dist}(p,q)=\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\ldots+(p_n-q_n)^2}$$

$$\mathrm{dist}(tomato, green\ bean)=\sqrt{(6-3)^2+(4-7)^2}=4.2$$

ingredient	sweetness	crunchiness	food type	distance to the tomato
grape	8	5	fruit	*sqrt((6 - 8)^2 + (4 - 5)^2) = 2.2*
green bean	3	7	vegetable	*sqrt((6 - 3)^2 + (4 - 7)^2) = 4.2*
nuts	3	6	protein	*sqrt((6 - 3)^2 + (4 - 6)^2) = 3.6*
orange	7	3	fruit	*sqrt((6 - 7)^2 + (4 - 3)^2) = 1.4*

What is the best value of k



Larger k   Smaller k

The balance between overfitting and under-fitting the training data is a problem known as the **bias-variance tradeoff**

Preprocessing

1 - Normalization

2 - Nominal features - dummy coding

Normalization

Min-max normalization

- We need a way of "shrinking" or rescaling the various features such that each one contributes relatively equally to the distance formula

$$X_{new} = \frac{X - min(X)}{max(X) - min(X)}$$

- Values fall in a range between 0 and 1

Step functions to

```{r}

employee_recipe <- recipe(left_company ~ ., data = employee_training) %>%
                   step_range(all_numeric(), -all_outcomes()) %>%     # min_max_normalizat
                   step_dummy(all_nominal(), -all_outcomes())%>%
                   step_nzv(all_predictors())
```

- step_range(), applies min-max normalization
    - step_normalize(), standardize ($z_i = (x_i - \mu_x)/\sigma_x$)
- step_dummy(), creates dummy variables

step_range()

salary <dbl>	weekly_hours <dbl>	yrs_at_company <int>
118680.74	56	6
85576.44	42	10
46235.79	56	0
117226.84	50	8

salary <dbl>	weekly_hours <dbl>	yrs_at_company <dbl>
0.487322740	0.61538462	0.150
0.305716491	0.07692308	0.250
0.089898378	0.61538462	0.000
0.479346806	0.38461538	0.200

$$\boxed{X_{new} = \frac{X - min(X)}{max(X) - min(X)}}$$

Normalization

```
normalize<-function(x){(x-min(x))/(max(x)-min(x))}

wbc_data%>%mutate(n_radius_mean=normalize(radius_mean))%>%
          select(ends_with('radius_mean'))%>%
          arrange(radius_mean)%>%head()
```

```
##   radius_mean n_radius_mean
##         <dbl>         <dbl>
## 1        6.98        0
## 2        7.69        0.0336
## 3        7.73        0.0354
## 4        7.76        0.0369
## 5        8.20        0.0575
## 6        8.22        0.0586
```

Normalization

z-score standardization

- Rescales each of a feature's values in terms of how many standard deviations they fall above or below the mean value.
- The resulting value is called a z-score. The z-scores fall in an unbounded range of negative and positive numbers
- Used when outliers are important indication of a class (like tumor growth in cancer detection)

$$X_{new} = \frac{X - \mu}{\sigma} = \frac{X - mean(x)}{StdDev(x)}$$

step_normalize()

salary <dbl>	weekly_hours <dbl>	yrs_at_company <int>
118680.74	56	6
85576.44	42	10
46235.79	56	0
117226.84	50	8

salary <dbl>	weekly_hours <dbl>	yrs_at_company <dbl>
0.6327092094	1.27048268	-0.17722292
-0.2378880424	-1.66514407	0.48039485
-1.2724926113	1.27048268	-1.16364957
0.5944735977	0.01235693	0.15158596

$$\boxed{Z = \frac{X - \mu}{\sigma}}$$

z: standard score

μ: variable mean

σ: variable standard deviation

Z-score normalization

`scale()` performs z-score standardization

```
wbc_data%>%
mutate(n_radius_mean=normalize(radius_mean) ,s_radius_mean=scale(radius_mean))%>%
select(ends_with('radius_mean'))%>%arrange(radius_mean)%>%head()
```

```
##    radius_mean     n_radius_mean     s_radius_mean
##          <dbl>         <dbl>             <dbl>
## 1         6.98      0                    -2.03
## 2         7.69      0.0336               -1.83
## 3         7.73      0.0354               -1.82
## 4         7.76      0.0369               -1.81
## 5         8.20      0.0575               -1.68
## 6         8.22      0.0586               -1.68
```

Transform all Numeric variables

We can apply normalization function to every numeric variable.

```
normalize<-function(x){(x-min(x))/(max(x)-min(x))}

wbc_data= wbc_data%>%column_to_rownames("id")  ## Id is not a variable
wbc_n=wbc_data%>%mutate_if(is.numeric, normalize)
```

Preprocessing

1 - Normalization

2 - Nominal features - dummy coding

Dummy coding

- A value of 1 indicates one category, and 0 indicates the other. For instance, dummy coding for a gender variable could be constructed as:

$$\text{male} = \begin{cases} 1 & \text{if } x = \text{male} \\ 0 & \text{otherwise} \end{cases}$$

- An n-category nominal feature can be dummy coded by creating binary indicator variables for (n - 1) levels of the feature. For example, dummy coding for a three-category temperature variable (for example, hot, medium, or cold) could be set up as (3 - 1) = 2 features, as shown:

$$\text{hot} = \begin{cases} 1 & \text{if } x = \text{hot} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{medium} = \begin{cases} 1 & \text{if } x = \text{medium} \\ 0 & \text{otherwise} \end{cases}$$

Step dummy

left_company <fctr>	department <fctr>
Yes	Sales
No	Sales
Yes	Product Development
No	IT and Analytics

left_company <fctr>	department_Sales <dbl>	department_Research <dbl>	department_Product.Development <dbl>
Yes	1	0	0
No	1	0	0
Yes	0	0	1
No	0	0	0

Dummy coding (without tidy models)

```
##   MS_Zoning
## 1 A_agr
## 2 C_all
## 3 Floating_Village_Residential
...
```

```
f=formula('~MS_Zoning-1')
model.matrix(f, data=ames_data)
```

```
##      MS_ZoningA_agr MS_ZoningC_all MS_ZoningFloating_Village_Residential
## 1                 0              0                                     0
## 2                 0              0                                     0
## 3                 0              0                                     0
## 4                 0              0                                     0
## 5                 0              0                                     0
## 6                 0              0                                     0
## 7                 0              0                                     0
## 8                 0              0                                     0
## 9                 0              0                                     0
## 10                0              0                                     0
```

Converting all nominal features

```
categorical=ames_data%>%select_if(is.character)


models=map(.x=names(categorical), ~formula(paste(" ~ ",.x," -1", sep=“”)))

rs=map(models, ~model.matrix(.x, data=ames_data ))
all_dummies=Reduce(cbind, rs)%>%as_tibble()
```

Alternatively we can use **caret** library

```
require(caret)

dmy <- dummyVars(" ~ .", data = ames_data)
all_dummies2=data.frame(predict(dmy, newdata = ames_data))
```

Agenda

- KNN concepts
- Preprocessing
    - Normalization
    - Dummy Coding
- Prediction
- Evaluation
- Implementation

Prediction

- Prediction and model building is together
- Lazy algorithm   (instance-based learning)
  - No abstraction
  - Stores the data verbatim

Recipe and Model

```{r}

employee_recipe <- recipe(left_company ~ ., data = employee_training) %>%
                   step_range(all_numeric(), -all_outcomes()) %>%     # min_
                   #step_normalize(all_numeric(), -all_outcomes()) %>%
                   step_dummy(all_nominal(), -all_outcomes())%>%
                   step_nzv(all_predictors())
```

```{r}
knn_model<-nearest_neighbor(neighbors=7)%>%   ## default number of neighbors is 5
            set_engine("kknn")%>%
            set_mode("classification")
```

Workflow and fit

```
```{r}
knn_workflow<-workflow()%>%
            add_recipe(employee_recipe)%>%
            add_model(knn_model)
```


Fit Model
```{r}
model_fit<-knn_workflow%>%fit(data=employee_training)
```
```

Predict and Evaluate

~~~~
```{r}
test_result=predict(model_fit, new_data = employee_test)%>%bind_cols(employee_test%>%select(left_company))

conf_mat(test_result, truth = left_company, estimate = .pred_class)
```
~~~~

~~~~
```{r}
test_result=predict(model_fit, new_data = employee_test)%>%bind_cols(employee_test%>%select(left_company))

conf_mat(test_result, truth = left_company, estimate = .pred_class)
```
~~~~

~~~~
          Truth
Prediction Yes  No
       Yes  45  11
       No   14 297
~~~~

Selecting k

- Selecting optimal k requires running the model with different k values and see which one performed the best
- Odd numbers are preferred (even number of k's may result in ties)

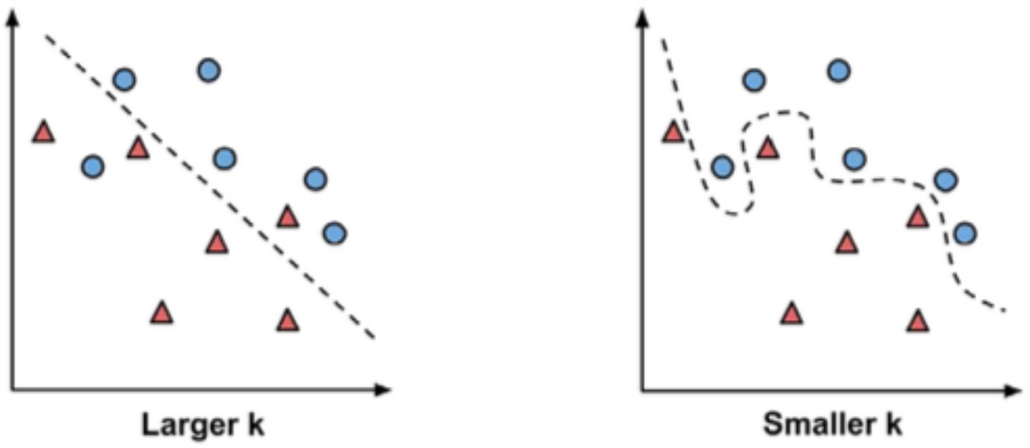
```
predictions=map(.x=seq(1,22,2), ~knn(train = train_wbc[,-1], test =
test_wbc[,-1], cl= train_wbc$diagnosis, k=.x ,prob = TRUE))

rs=map(.x=predictions, .f=~confusionMatrix(table(test_wbc$diagnosis, .x) ,p
ositive = 'M')$overall)

rs_df=Reduce(rbind, rs)%>%as.tibble()%>%
             mutate(run=seq(1,22,2))%>%select(run, everything())
```

Selecting the k - Parameter tuning

- What is the number of neighbors we should consider to get the best results



The balance between overfitting and under-fitting the training data is a problem known as the **bias-variance tradeoff**

tune() for parameter

````
```{r}
employee_folds <- vfold_cv(employee_training, v = 5)
```
````

- setup with v-fold cross validation

````
```{r}
knn_model2 <- nearest_neighbor(neighbors = tune()) %>%
            set_engine('kknn') %>%
            set_mode('classification')
```
````

````
```{r}
knn_workflow2 <- workflow() %>%
          add_model(knn_model2) %>%
          add_recipe(employee_recipe)
```
````

- enter tune() function instead of a predefined k value
- proceed the same way

Define the search grid for the best k

```{r}
k_grid <- tibble(neighbors = c(1:10, 20, 30, 50, 75, 100, 125, 150))
k_grid
```

neighbors <dbl>
1
2
3
4
5
6
7
8
9
10

1-10 of 17 rows

- create a table (tibble) of values that will be tried

search the grid

```{r}
set.seed(314)

knn_tuning <- knn_workflow2 %>%
              tune_grid(resamples = employee_folds,
                        grid = k_grid)
```

- tune_grid() uses a table (tibble) of parameters and plug it in where previously filled with =tune()
- previously we set neighbors= tune() and the neighbors column in k_grid will be used for the neighbors parameter

Finding the best parameter value

```
knn_tuning%>%show_best('accuracy')
```

neighbors <dbl>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>
20	accuracy	binary	0.9337927	5	0.007721810
30	accuracy	binary	0.9283464	5	0.008595037
10	accuracy	binary	0.9283422	5	0.010149714
50	accuracy	binary	0.9283340	5	0.012345119
9	accuracy	binary	0.9265282	5	0.009738308

- best model based on the 'accuracy' metric

Finalize the workflow with best value

```
## Select Best
```{r}
best_k<- knn_tuning%>% select_best(metric='accuracy')
```

finalize
```{r}
final_knn_wf<-knn_workflow2%>%finalize_workflow(best_k)
```
```

Fit the model

```
## Train and Eval with Last fit
```{r}
last_fit_knn<- final_knn_wf%>%last_fit(split=employee_split)
```

```{r}
last_fit_knn %>% collect_metrics()
```

```{r}
test_result<-last_fit_knn%>%collect_predictions()
```

```{r}
conf_mat(test_result, truth = left_company, estimate=.pred_class)
```

          Truth
Prediction Yes No
       Yes 45 11
       No   14 297
```

last_fit_knn() is combines the two steps below
1 - model fit with training data
2 - predict with testing data