Some keras models for text classification

Simon Roth

Packages

```
pacman::p_load(tidyverse, keras, caret, e1071)
```

```
vis_table <- function(pred, real){
  tibble(preds = pred, real = real) %>%
    dplyr::count(preds, real) %>%
    dplyr::group_by(real) %>%
    dplyr::mutate(n_real = sum(n)) %>%
    ungroup() %>%
    dplyr::mutate(perc_real = round(n/n_real * 100, 1)) %>%
    dplyr::mutate(label = paste0(n, "\n", perc_real, "%")) %>%
    mutate(preds = factor(preds, levels = sort(unique(preds), decreasing = T))) %>%
    mutate(real = factor(real, levels = sort(unique(real), decreasing = F))) %>%
    #mutate(real = factor(real)) %>%
    ggplot(aes(real, preds, fill = n)) +
    ggplot2::geom_tile(alpha = 0.8) +
    #viridis::scale_fill_viridis(direction = -1) +
    scale_fill_gradient(low = "white", high = "black")+
    scale_x_discrete(position = "top") +
    ggthemes::theme_few() +
    theme(legend.position = "none", axis.text.x = element_text(angle = 30, hjust = -.1)) +
    coord_fixed() +
    labs(x = "Real value y", y = "Predicted value y hat") +
    ggplot2::geom_text(aes(label = label))
}
```

Data

- no missing data

```
tweets <- read_csv("../data/raw_tweets.csv") %>%
  arrange(sample(1:n(), size = n())) %>%
  glimpse
```

```
## Parsed with column specification:
## cols(
##   id = col_double(),
##   name = col_character(),
##   created = col_datetime(format = ""),
##   text = col_character()
## )
```

```
## Observations: 9,000
## Variables: 4
## $ id      <dbl> 1.083904e+18, 1.030147e+18, 1.025825e+18, 1.002008e+18...
## $ name    <chr> "fchollet", "khloekardashian", "fchollet", "KimKardash...
## $ created <dttm> 2019-01-12 01:48:36, 2018-08-16 17:39:15, 2018-08-04 ...
## $ text    <chr> "This is what I had to say about it in 2010:", "These ...
```

```
tweets %>%
  count(name, sort = T)
```

```
## # A tibble: 6 x 2
##   name                n
##   <chr>           <int>
## 1 fchollet         1500
## 2 hadleywickham    1500
## 3 khloekardashian  1500
## 4 KimKardashian    1500
## 5 kourtneykardash  1500
## 6 wesmckinn        1500
```

```
celeb_mat <- tweets$name %>%
  dummies::dummy()

celeb_names <- celeb_mat %>%
  colnames %>%
  str_remove("name|/Users/syro/MEGA/projects/celebrity-faceoff/code/keras_cnn.Rmd")

celeb_target <- celeb_mat %>%
  as_tibble %>%
  set_names(celeb_names) %>%
  glimpse
```

```
## Observations: 9,000
## Variables: 6
## $ fchollet        <int> 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0...
## $ hadleywickham   <int> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1...
## $ khloekardashian <int> 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0...
## $ KimKardashian   <int> 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0...
## $ kourtneykardash <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ wesmckinn       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0...
```

Train/ Split

```
set.seed(2019)
#split_id <- sample(c(T, F), size = nrow(tweets), replace = T, prob = c(.8, .2))
train_id <- read_csv("../data/train_tweets.csv") %>% pull(id)
```

```
## Parsed with column specification:
## cols(
```

```
##   id = col_double(),
##   name = col_character(),
##   created = col_datetime(format = ""),
##   text = col_character()
## )
```

```
test_id <- read_csv("../data/test_tweets.csv") %>% pull(id)
```

```
## Parsed with column specification:
## cols(
##   id = col_double(),
##   name = col_character(),
##   created = col_datetime(format = ""),
##   text = col_character()
## )
```

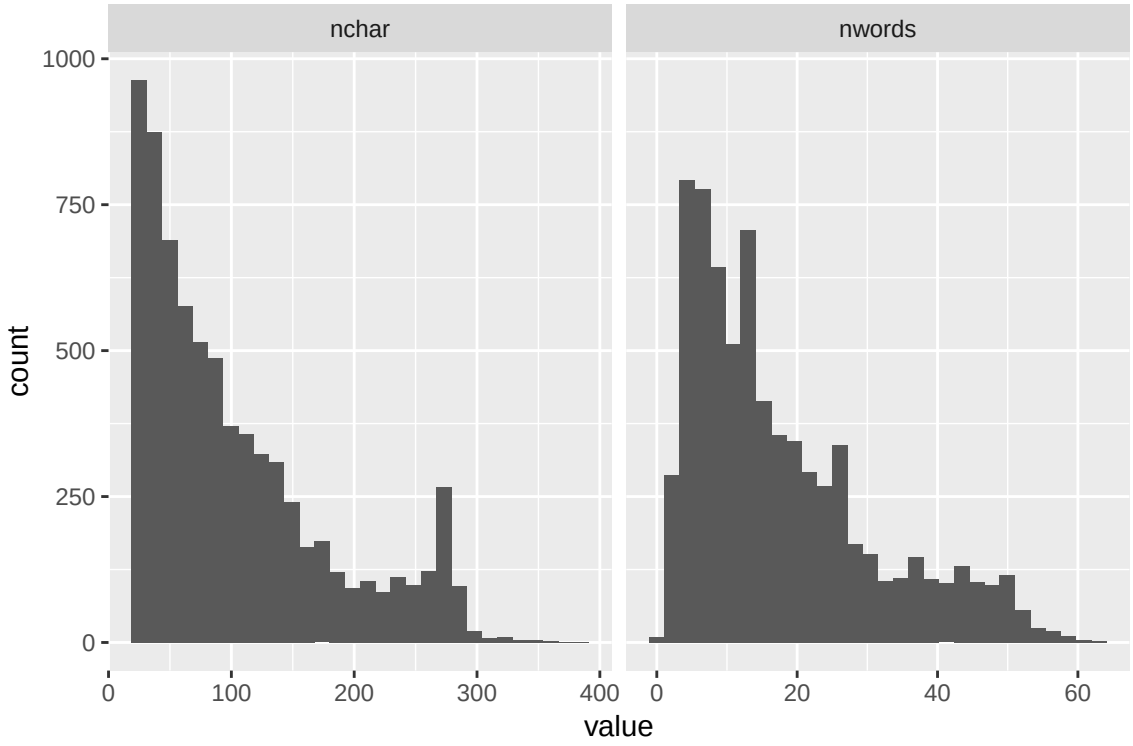
```
train <- tweets %>% filter(id %in% train_id)
test <- tweets %>% filter(id %in% test_id)

y_train <- celeb_target[tweets$id %in% train_id, ] %>% as.matrix
y_test <- celeb_target[tweets$id %in% test_id, ] %>% as.matrix
```

Text Preprocessing

- char_maxlen [200, 300]
- word_maxlen [25, 50]

```
train %>%
  mutate(nchar = str_length(text), nwords = str_count(text, "\\w+")) %>%
  select(nchar, nwords) %>%
  gather(var, value) %>%
  ggplot(aes(value)) +
  geom_histogram() +
  facet_wrap(~var, scales = "free_x")
```

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



- The vocab is pretty small max_features [2500, 3500]

```
train %>%
  tidytext::unnest_tokens(word, text, token = "words") %>%
  count(word, sort = T) %>%
  filter(n > 2)
```

```
## # A tibble: 4,718 x 2
##    word      n
##    <chr> <int>
##  1 the    4076
##  2 to     3222
##  3 a      2442
##  4 i      2389
##  5 of     2168
##  6 you    2099
##  7 and    2036
##  8 is     1791
##  9 in     1723
## 10 for    1410
## # ... with 4,708 more rows
```

Word Tokenization

- vectorize the text samples into a 2D integer tensor

```
library(keras)
maxlen <- 60
embedding_dim <- 128
batch_size <- 32
epochs <- 2
max_features <- 13488

tokenizer <- text_tokenizer(num_words = max_features, lower = F, split = " ", char_level = F)
fit_text_tokenizer(tokenizer, train$text)

x_train <- tokenizer %>%
  texts_to_sequences(train$text) %>%
  pad_sequences(maxlen = maxlen)

x_test <- tokenizer %>%
  texts_to_sequences(test$text) %>%
  pad_sequences(maxlen = maxlen)

dim(x_train)
```

```
## [1] 7200   60
```

Models

Baseline

- train a 1D convnet with global maxpooling

```
#inputs <- layer_input(shape = list(maxlen), dtype='int32')
baseline <- keras_model_sequential() %>%
  # We specify the maximum input length to our Embedding layer
  # so we can later flatten the embedded inputs
  layer_embedding(input_dim = max_features, output_dim = 50,
                  input_length = maxlen) %>%
  # We flatten the 3D tensor of embeddings
  # into a 2D tensor of shape `(samples, maxlen * output_dim)`
  layer_flatten() %>%
  # We add the classifier on top
  layer_dense(units = 6, activation = "softmax")

baseline %>% compile(
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("acc")
)

hist_baseline <- baseline %>%
  keras::fit(
    x_train, y_train,
    batch_size = batch_size,
    epochs = 2,
```

```
    suffle = T,
    #class_weight = weight_set,
    validation_split = .1
  )

score <- baseline %>%
  evaluate(
    x_test, y_test,
    batch_size = batch_size,
    verbose = 1
  )

cat('Test score:', score[[1]], '\n')
```

```
## Test score: 1.015242
```

```
cat('Test accuracy', score[[2]], '\n')
```

```
## Test accuracy 0.6472222
```

GRU + CNN

- Code
- Paper

```
inp <- layer_input(shape = list(maxlen), dtype = "int32", name = "input")
emm <- inp %>%
  layer_embedding(
    input_dim = max_features,
    output_dim = 128,
    input_length = maxlen
  )
  #layer_spatial_dropout_1d(rate = .1)

model_1 <- emm %>%
  bidirectional(layer_gru(units = 60, return_sequences = T, recurrent_dropout = 0.1)) %>%
  layer_conv_1d(30, 3, padding = "valid", activation = "relu", strides = 1)

model_2 <- emm %>%
  bidirectional(layer_gru(units = 30, return_sequences = T, recurrent_dropout = 0.1)) %>%
  layer_conv_1d(20, 2, padding = "valid", activation = "relu", strides = 1)

max_pool1 <- model_1 %>% layer_global_max_pooling_1d()
ave_pool1 <- model_1 %>% layer_global_average_pooling_1d()
max_pool2 <- model_2 %>% layer_global_max_pooling_1d()
ave_pool2 <- model_2 %>% layer_global_average_pooling_1d()

outp <- layer_concatenate(list(ave_pool1, max_pool1, ave_pool2, max_pool2)) %>%
        layer_dense(units = ncol(celeb_mat), activation = "softmax")

gru_cnn_model <- keras_model(inp, outp) %>%
```

```
  compile(
    optimizer = "adam",
    loss = "categorical_crossentropy",
    metrics = c("acc")
  )

summary(gru_cnn_model)
```

```
## ____________________________________________________________________________
## Layer (type)            Output Shape     Param #  Connected to
## ============================================================================
## input (InputLayer)      (None, 60)       0
## ____________________________________________________________________________
## embedding_2 (Embedding) (None, 60, 128)  1726464  input[0][0]
## ____________________________________________________________________________
## bidirectional_1 (Bidire (None, 60, 120)  68040    embedding_2[0][0]
## ____________________________________________________________________________
## bidirectional_2 (Bidire (None, 60, 60)   28620    embedding_2[0][0]
## ____________________________________________________________________________
## conv1d_1 (Conv1D)       (None, 58, 30)   10830    bidirectional_1[0][0]
## ____________________________________________________________________________
## conv1d_2 (Conv1D)       (None, 59, 20)   2420     bidirectional_2[0][0]
## ____________________________________________________________________________
## global_average_pooling1 (None, 30)       0        conv1d_1[0][0]
## ____________________________________________________________________________
## global_max_pooling1d_1  (None, 30)       0        conv1d_1[0][0]
## ____________________________________________________________________________
## global_average_pooling1 (None, 20)       0        conv1d_2[0][0]
## ____________________________________________________________________________
## global_max_pooling1d_2  (None, 20)       0        conv1d_2[0][0]
## ____________________________________________________________________________
## concatenate_1 (Concaten (None, 100)      0        global_average_pooling1d_
##                                                   global_max_pooling1d_1[0]
##                                                   global_average_pooling1d_
##                                                   global_max_pooling1d_2[0]
## ____________________________________________________________________________
## dense_2 (Dense)         (None, 6)        606      concatenate_1[0][0]
## ============================================================================
## Total params: 1,836,980
## Trainable params: 1,836,980
## Non-trainable params: 0
## ____________________________________________________________________________
```

```
hist_gru_cnn <- gru_cnn_model %>%
  keras::fit(
    x_train, y_train,
    batch_size = batch_size,
    epochs = 2,
    suffle = T,
    #class_weight = weight_set,
    validation_split = .1
  )
```

```
score <- gru_cnn_model %>%
  evaluate(
    x_test, y_test,
    batch_size = batch_size,
    verbose = 1
  )

cat('Test score:', score[[1]], '\n')
```

```
## Test score: 0.6793197
```

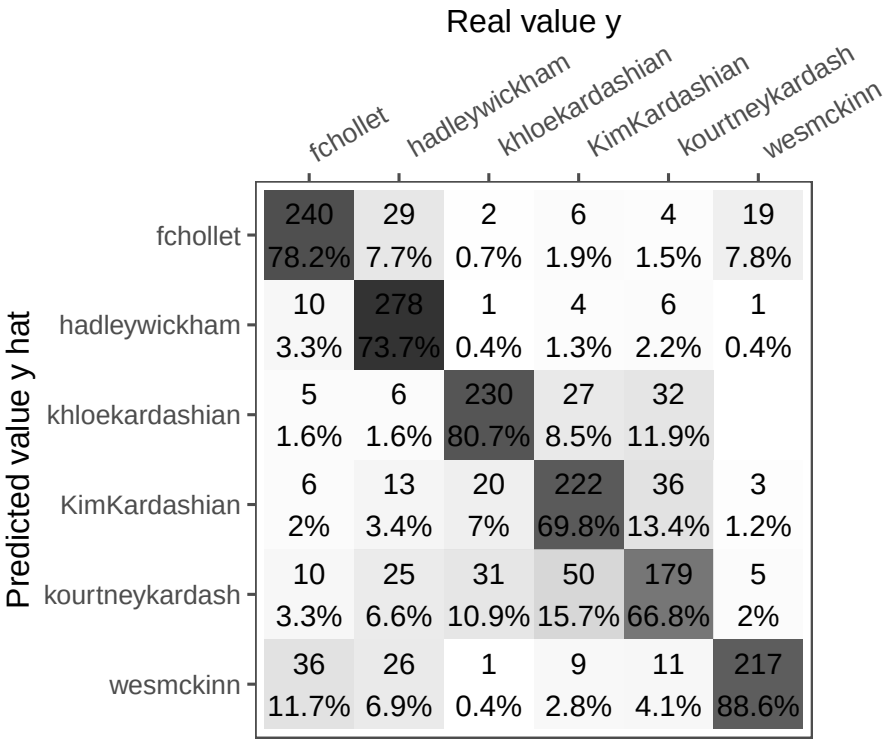
```
cat('Test accuracy', score[[2]], '\n')
```

```
## Test accuracy 0.7588889
```

```
pred_gru_cnn <- predict(gru_cnn_model, x = x_test)

pred_names <- pred_gru_cnn %>%
  as.data.frame() %>%
  set_names(celeb_names) %>%
  split(1:nrow(.)) %>%
  map_chr(~names(which.max(.x)))

real_names <- test$name
mean(real_names == pred_names)
```

```
## [1] 0.7588889
```

```
vis_table(real_names, pred_names)
```



Multi Channel CNN (mchannel_cnn)

- Code

```
# keras::k_clear_session()
# embed_size <- 64
# filter_sizes <- c(1, 2, 3, 4)
# num_filters <- 70

keras_mchannel_cnn <- function(
  max_features = 10000,
  embed_size = 128,
  maxlen = 50,
  filter_sizes = c(1, 2, 3, 4),
  num_filters = 50,
  embedding_matrix = NULL
){

  inputs <- keras::layer_input(shape = list(maxlen))

  if(!is.null(embedding_matrix)){
    embedding <- inputs %>%
      layer_embedding(
        input_dim = max_features,
        output_dim = embed_size,
        weights = list(embedding_matrix),
        input_length = maxlen,
        trainable = F
```

```
    )%>%
    #layer_spatial_dropout_1d(0.2) %>%
    layer_reshape(list(maxlen, embed_size, 1))
} else {
embedding<- inputs %>%
  layer_embedding(
    input_dim = max_features,
    output_dim = embed_size,
    input_length = maxlen
  ) %>%
  #layer_spatial_dropout_1d(0.2) %>%
  layer_reshape(list(maxlen, embed_size, 1))
}

block1 <- embedding %>%
  layer_conv_2d(
    num_filters,
    kernel_size = list(filter_sizes[1], embed_size),
    kernel_initializer = 'normal',
    activation='elu'
  ) %>%
  layer_max_pooling_2d(pool_size=list(maxlen - filter_sizes[1] + 1, 1))

block2 <- embedding %>%
  layer_conv_2d(
    num_filters,
    kernel_size = list(filter_sizes[2], embed_size),
    kernel_initializer = 'normal',
    activation='elu'
  ) %>%
  layer_max_pooling_2d(pool_size=list(maxlen - filter_sizes[2] + 1, 1))

block3 <- embedding %>%
  layer_conv_2d(
    num_filters,
    kernel_size = list(filter_sizes[3], embed_size),
    kernel_initializer = 'normal',
    activation='elu'
  ) %>%
  layer_max_pooling_2d(pool_size=list(maxlen - filter_sizes[3] + 1, 1))

block4 <- embedding %>%
  layer_conv_2d(
    num_filters,
    kernel_size = list(filter_sizes[4], embed_size),
    kernel_initializer = 'normal',
    activation='elu'
  ) %>%
  layer_max_pooling_2d(pool_size=list(maxlen - filter_sizes[4] + 1, 1))

z <- layer_concatenate(list(block1, block2, block3, block4), axis = 1) %>%
  layer_flatten()
```

```
  output <- z %>%
    layer_dense(ncol(celeb_mat), activation="softmax")

  mchannel_cnn <- keras::keras_model(inputs, output)

  return(mchannel_cnn)
}

mchannel_cnn <- keras_mchannel_cnn(max_features = max_features, embed_size = 128, maxlen = maxlen)

mchannel_cnn %>%
  compile(
    loss = "categorical_crossentropy",
    optimizer = "adam",
    metrics = "accuracy"
  )

summary(mchannel_cnn)
```

```
## __________________________________________________________________________
## Layer (type)            Output Shape     Param #  Connected to
## ==========================================================================
## input_1 (InputLayer)    (None, 60)       0
## __________________________________________________________________________
## embedding_3 (Embedding) (None, 60, 128)  1726464  input_1[0][0]
## __________________________________________________________________________
## reshape_1 (Reshape)     (None, 60, 128,  0        embedding_3[0][0]
## __________________________________________________________________________
## conv2d_1 (Conv2D)       (None, 60, 1, 50 6450     reshape_1[0][0]
## __________________________________________________________________________
## conv2d_2 (Conv2D)       (None, 59, 1, 50 12850    reshape_1[0][0]
## __________________________________________________________________________
## conv2d_3 (Conv2D)       (None, 58, 1, 50 19250    reshape_1[0][0]
## __________________________________________________________________________
## conv2d_4 (Conv2D)       (None, 57, 1, 50 25650    reshape_1[0][0]
## __________________________________________________________________________
## max_pooling2d_1 (MaxPoo (None, 1, 1, 50) 0        conv2d_1[0][0]
## __________________________________________________________________________
## max_pooling2d_2 (MaxPoo (None, 1, 1, 50) 0        conv2d_2[0][0]
## __________________________________________________________________________
## max_pooling2d_3 (MaxPoo (None, 1, 1, 50) 0        conv2d_3[0][0]
## __________________________________________________________________________
## max_pooling2d_4 (MaxPoo (None, 1, 1, 50) 0        conv2d_4[0][0]
## __________________________________________________________________________
## concatenate_2 (Concaten (None, 4, 1, 50) 0        max_pooling2d_1[0][0]
##                                                   max_pooling2d_2[0][0]
##                                                   max_pooling2d_3[0][0]
##                                                   max_pooling2d_4[0][0]
## __________________________________________________________________________
## flatten_2 (Flatten)     (None, 200)      0        concatenate_2[0][0]
## __________________________________________________________________________
## dense_3 (Dense)         (None, 6)        1206     flatten_2[0][0]
## ==========================================================================
```

```
## Total params: 1,791,870
## Trainable params: 1,791,870
## Non-trainable params: 0
## _________________________________________________________________________
```

```
mchannel_cnn_hist <- mchannel_cnn %>%
  keras::fit(
    x_train, y_train,
    batch_size = batch_size,
    shuffle = T,
    epochs = 3,
    validation_split = .1
  )
```

```
score <- mchannel_cnn %>%
  evaluate(
    x_test, y_test,
    batch_size = batch_size,
    verbose = 1
  )

cat('Test score:', score[[1]], '\n')
```

```
## Test score: 0.6165214
```

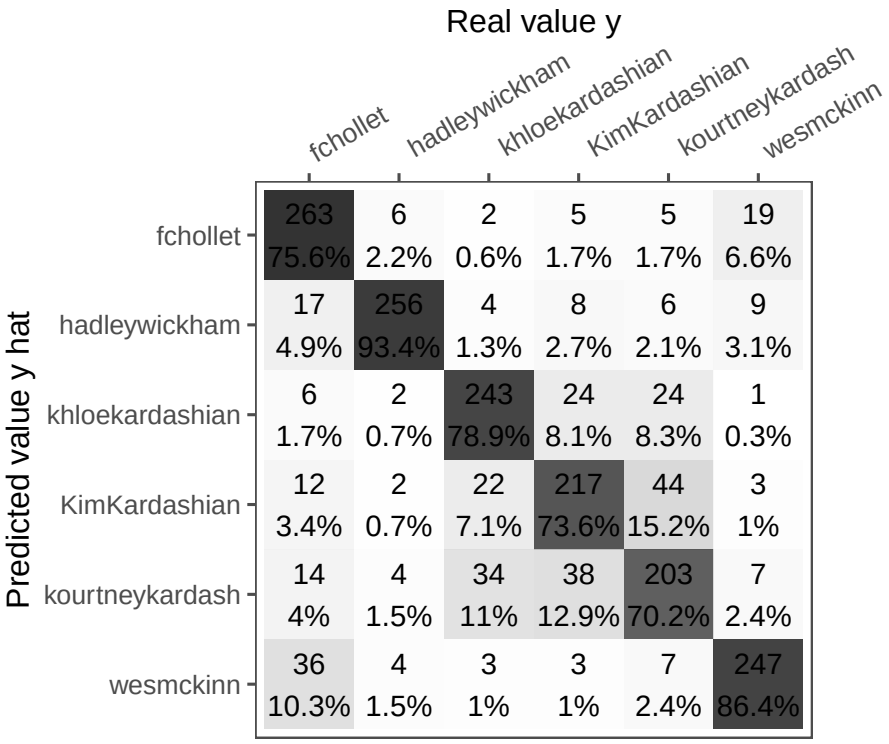
```
cat('Test accuracy', score[[2]], '\n')
```

```
## Test accuracy 0.7938889
```

```
pred_mchannel_cnn <- predict(mchannel_cnn, x = x_test)

pred_names <- pred_mchannel_cnn %>%
  as.data.frame() %>%
  set_names(celeb_names) %>%
  split(1:nrow(.)) %>%
  map_chr(~names(which.max(.x)))

real_names <- test$name
mean(real_names == pred_names)
```

```
## [1] 0.7938889
```

```
vis_table(real_names, pred_names)
```


Real value y
fchollet
hadleywickham
khloekardashian
KimKardashian
kourtneykardash
wesmckinn
Predicted value y hat
fchollet
hadleywickham
khloekardashian
KimKardashian
kourtneykardash
wesmckinn
263 75.6%
6 2.2%
2 0.6%
5 1.7%
5 1.7%
19 6.6%
17 4.9%
256 93.4%
4 1.3%
8 2.7%
6 2.1%
9 3.1%
6 1.7%
2 0.7%
243 78.9%
24 8.1%
24 8.3%
1 0.3%
12 3.4%
2 0.7%
22 7.1%
217 73.6%
44 15.2%
3 1%
14 4%
4 1.5%
34 11%
38 12.9%
203 70.2%
7 2.4%
36 10.3%
4 1.5%
3 1%
3 1%
7 2.4%
247 86.4%