XGBoost: The Art and Science of Communicating Machine Learning Algorithms

Amy Szadziewska, Peak

6th February 2018

Overview

- Why decision trees are interpretable but not good at predicting
- Why XGBoost is good at predicting but not interpretable
- How the xgboostExplainer R package can be used to get the best of both worlds

Dataset

TMDB 5000 Movie Dataset from Kaggle
https://www.kaggle.com/tmdb/tmdb-movie-metadata

The data includes the following information:

- Vote average
- Vote count
- Budget
- Revenue
- Genres
- Production companies
- Original language

The code I've used to process this data will be at the end of the presentation.

Decision trees

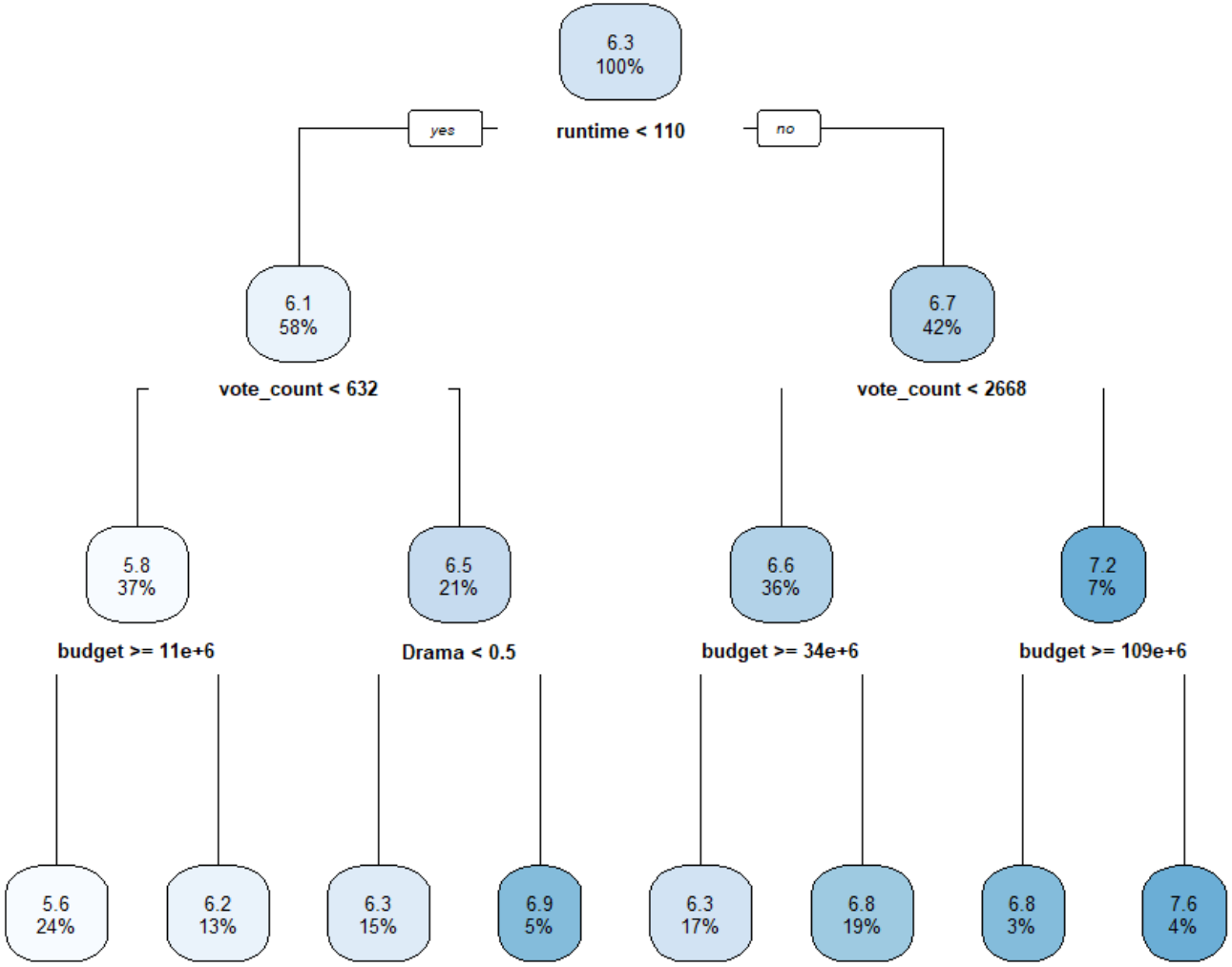
This decision tree predicts the IMDb score of a film.

It's easy to visualise, interpret & explain, which makes it easy to understand the reasons each film is predicted a certain score.

For example, we can predict the IMDb score for Avatar and understand how much each feature contributed to it's score:

- Runtime: 162 (6.7 – 6.3 = +0.4 score)
- Vote count: 11,800 (7.2 – 6.7 = +0.5 score)
- Budget: £237m (6.8 – 7.2 = -0.4 score)

Our prediction is 6.8 (vs. actual 7.2).



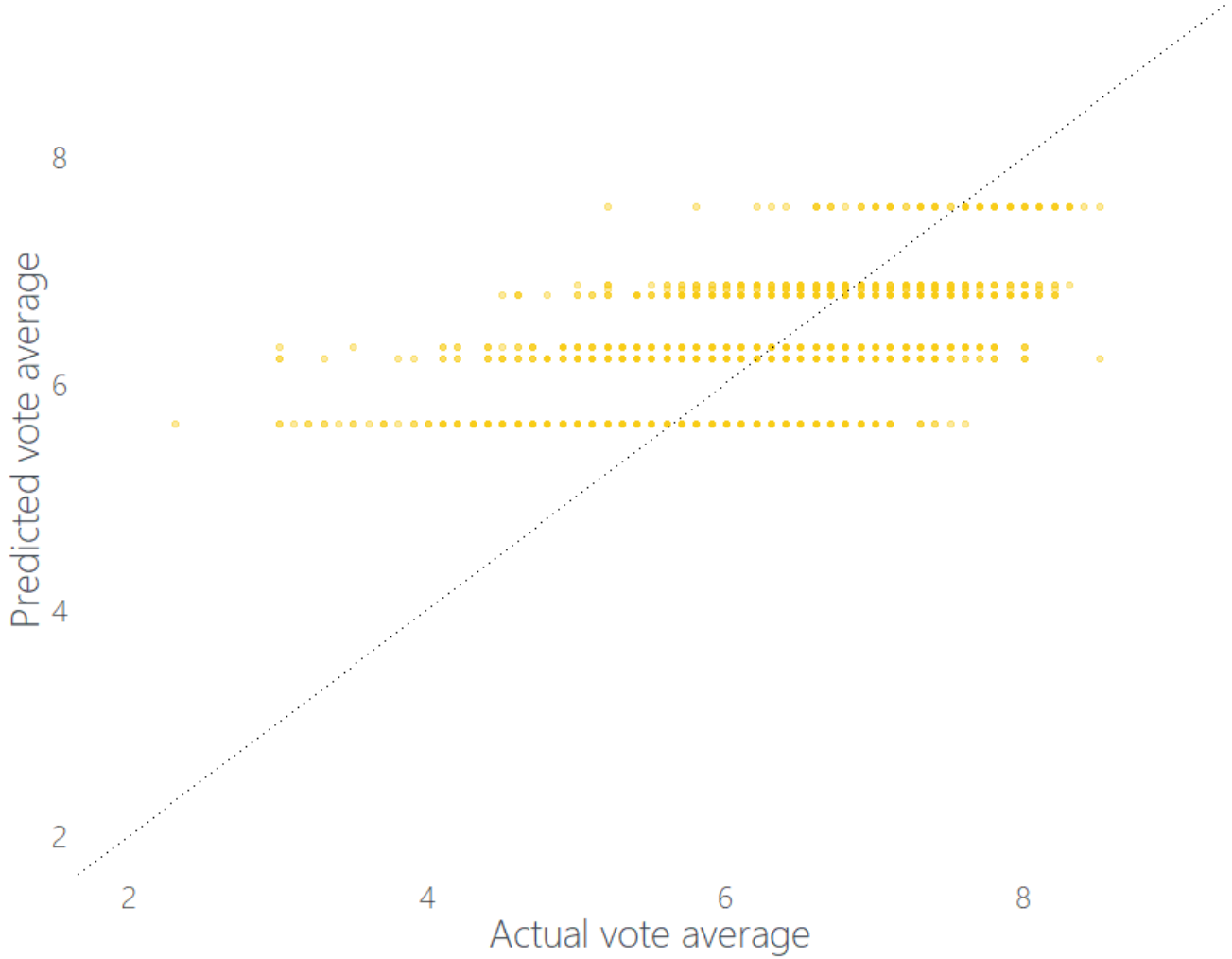
Decision trees

However, the interpretability of decision trees is also their downfall as they lack predictive power.

The plot on the right shows the Actual vs. Predicted IMDb scores for the films in the data.

For example, the films that were predicted to have the lowest scores (average 5.6) actually have values that vary from 2.3 to 7.6.

How can we improve these predictions?



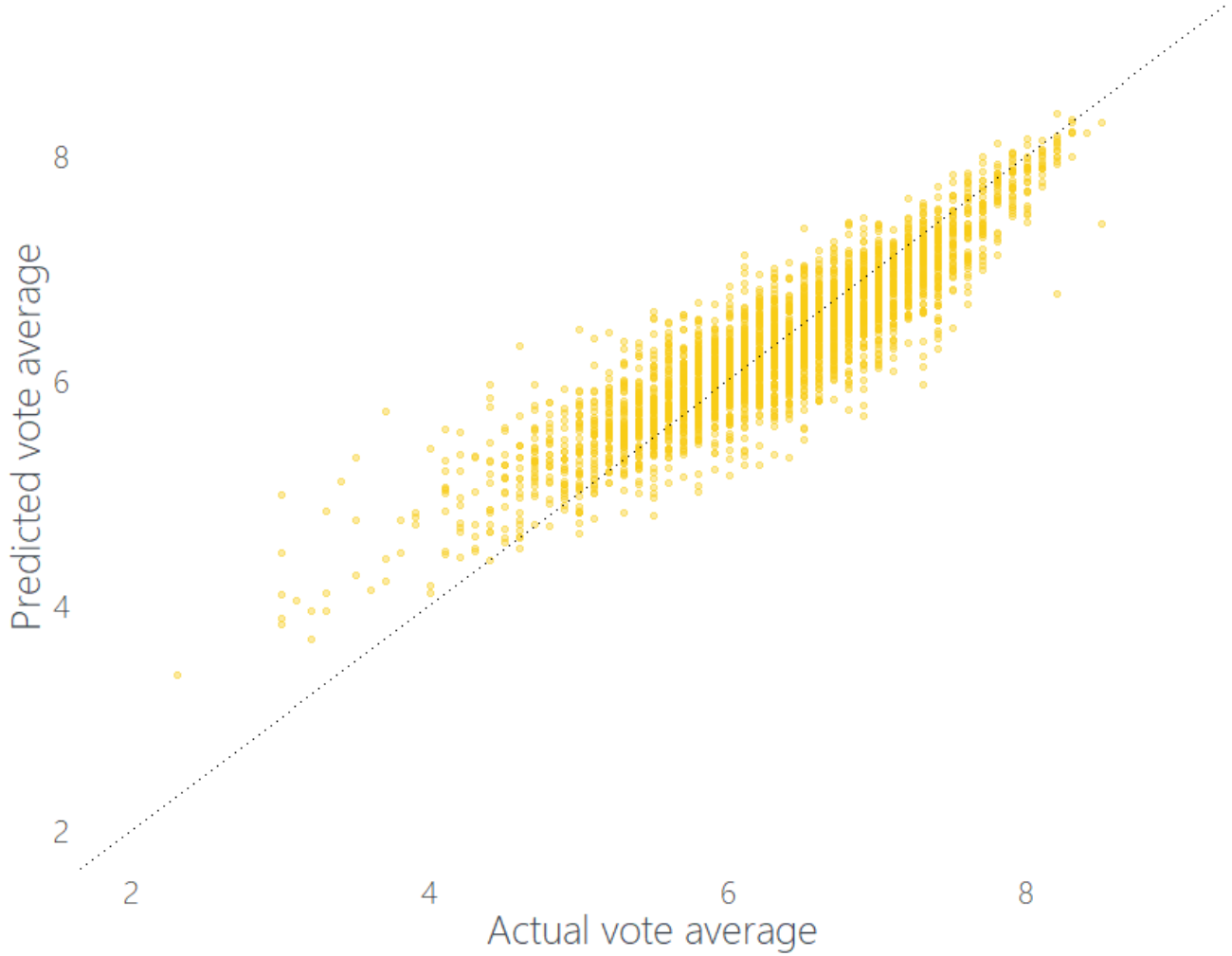
XGBoost

Enter Extreme Gradient Boosting A.K.A XGBoost

- Focused on computational speed and model performance
- Ensemble method i.e. builds multiple models and combines their results to increase performance
- Creates new models that predict the residuals of existing models which are added together to make the final prediction
- It uses a gradient descent algorithm to minimise the loss when creating new models

http://xgboost.readthedocs.io/en/latest/model.html

The plot on the right shows the Actual vs. Predicted IMDb scores using XGBoost which shows a huge improvement vs. the decision tree.



XGBoost

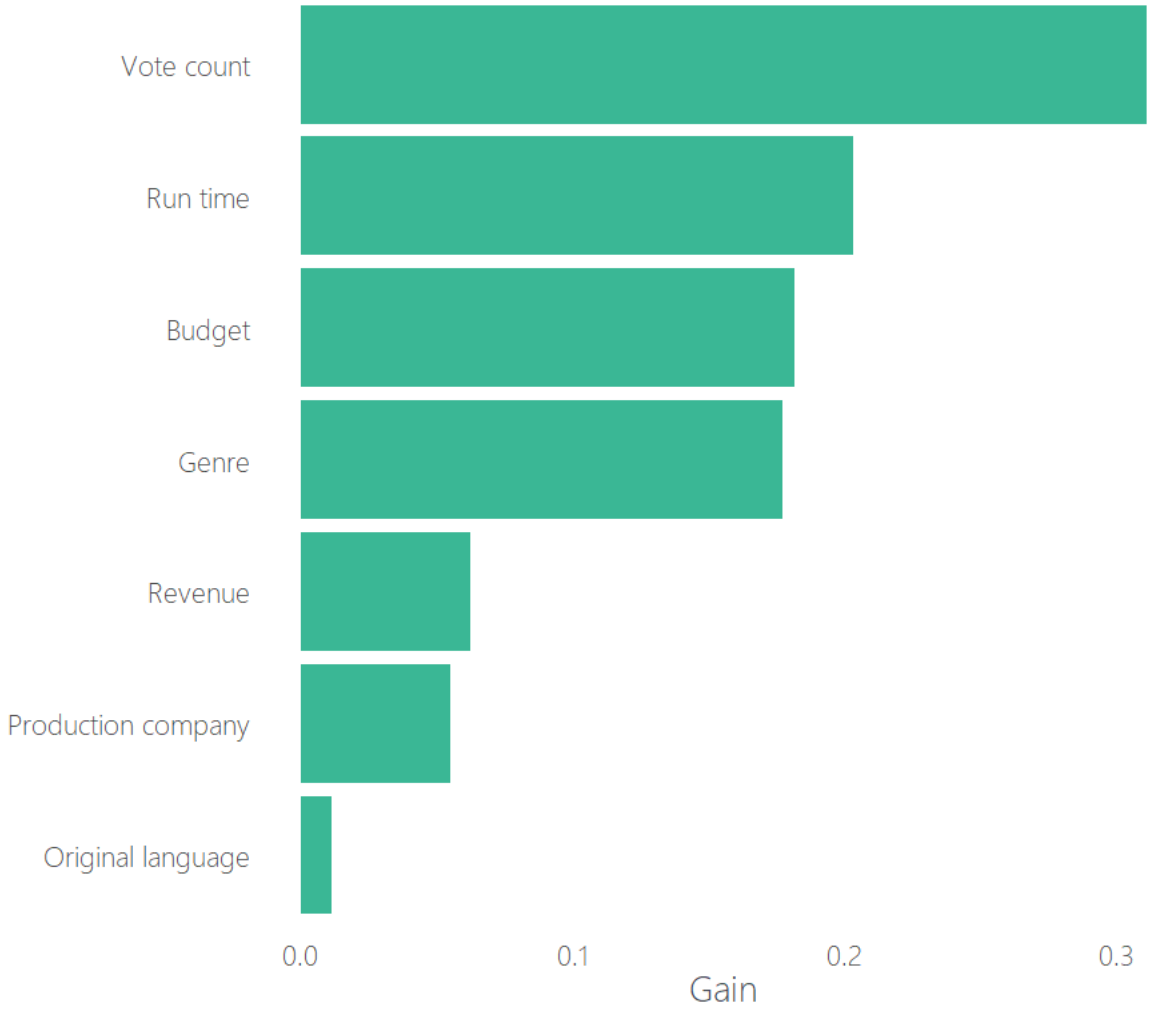
The problem with the xgboost package is that it doesn't allow you to understand the influence of each individual feature for a particular film.

For example, the predicted IMDb score for Avatar is 7.19 (vs. actual 7.2) – but why?

The closest we can get is the xgb.importance function which shows the 'importance' of each feature at an overall level.

The problems with this are:

- How do you explain the x-axis, 'Gain'?!
- There's no indication of relationship with the outcome (and sometimes it's not as simple as "as *x* increases, *y* increases")
- The importance is at an overall level, not for each individual prediction



The XGBoost Explainer

How can we make XGBoost as transparent as a single decision tree?

- The xgboostExplainer R package

How does xgboostExplainer work?

The way it works is by adding up the contributions of each feature for every tree in the ensemble, in exactly the same way we can for a decision tree.

The xgboost package contains a function xgb.model.dt.tree that breaks down the calculations that the algorithm is using to generate predictions which feeds into the xgboostExplainer package to express each prediction as the sum of feature impacts.

```
library(xgboost); library(Matrix); library(devtools);

## Prepare data and parameters for xgboost model
fmla <- formula(paste("vote_average ~ budget + original_language + revenue +
runtime + vote_count", paste(genres, collapse = " + "),
paste(top_production_companies, collapse = " + "), sep = " + "))
mm <- sparse.model.matrix(fmla, tmdb)
param <- list(booster = 'gbtree', objective = 'reg:linear')
outcome <- tmdb$vote_average

## Cross-validation and model
cv <- xgb.cv(data = mm, label = outcome,
             params = param, nfold = 5, nround = 3000,
             early_stopping_rounds = 10, nthread = 2)

bst <- xgboost(data = mm, label = outcome, params = param,
               nrounds = cv$best_iteration,
               base_score = mean(outcome))

## Install xgboostExplainer package
install_github("AppliedDataSciencePartners/xgboostExplainer")
library(xgboostExplainer)

## Build explainer
explainer <- buildExplainer(bst, xgb.DMatrix(mm, label = outcome), type =
"regression", base_score = mean(outcome))

## Get breakdowns for each of the predictions
pred.breakdown <- explainPredictions(bst, explainer, xgb.DMatrix(mm, label =
outcome))

## Waterfall chart for one prediction
showWaterfall(bst, explainer, xgb.DMatrix(mm, label = outcome), mm, idx = 1,
type = "regression", threshold = 0.03)
```

The XGBoost Explainer

buildExplainer: This function outputs an xgboostExplainer (a data table that stores the feature impact breakdown for each leaf of each tree in an xgboost model). It is required as input into the explainPredictions and showWaterfall functions.

explainPredictions: This function outputs the feature impact breakdown of a set of predictions made using an xgboost model.

showWaterfall: This function prints the feature impact breakdown for a single data row, and plots an accompanying waterfall chart.

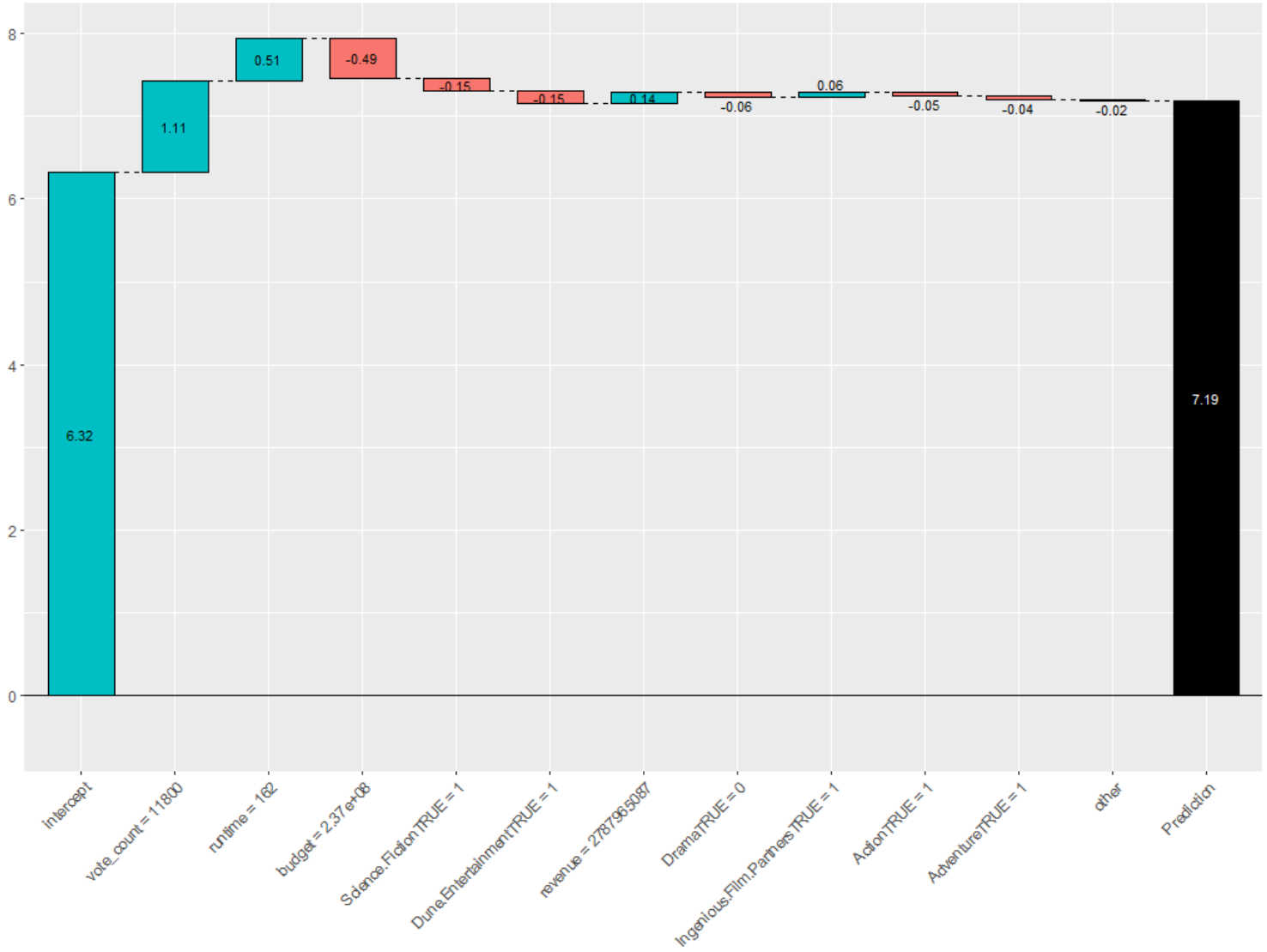
The XGBoost Explainer

We can now answer our question from earlier…

The predicted IMDb score for Avatar is 7.19 (vs. actual 7.2) – but why?

The prediction of 7.19 is broken down into the impact of each individual feature, where the intercept is the average score across all 5000 films (note that the base score can be changed).

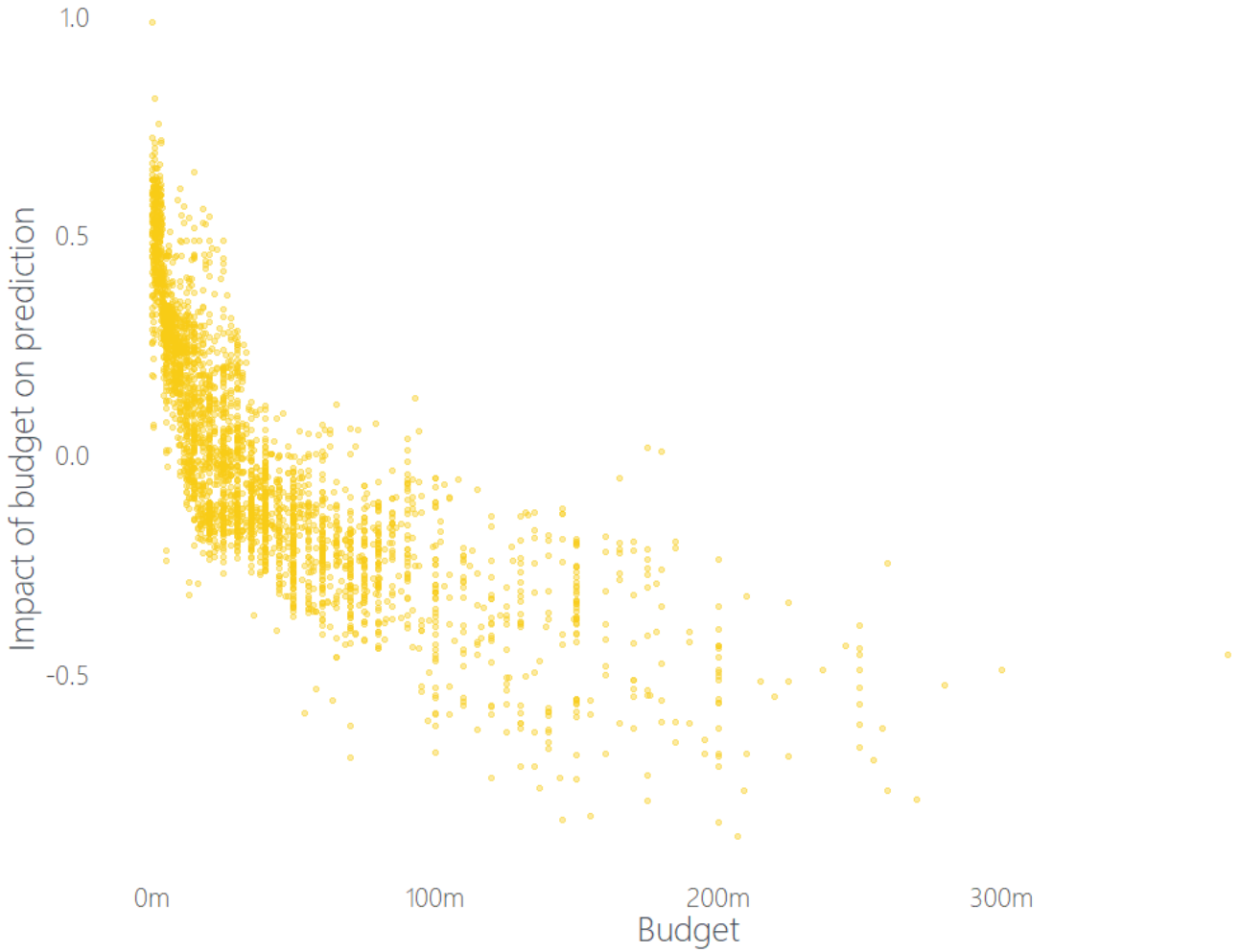
For example, the vote count contributes 1.11 to Avatar's score whereas it's high budget of £237m decreases the score by 0.49.



The XGBoost Explainer

It is also a lot easier to understand the relationship between a factor and the outcome, such as plot on the right which shows how budget impacts our IMDb score predictions.

Previously we'd have to plot the variable against actuals to try and spot relationships, which makes it more difficult to identify non-linear relationships and quantify the impact on the outcome.



The XGBoost Explainer

Let's revisit the problems with xgb.importance and see how xgboostExplainer overcomes them…

- How do you explain the x-axis, 'Gain'?!
  - Being able to breakdown our predictions allows us to explain how feature X causes an increase of Y for a particular prediction
- There's no indication of relationship with the outcome (and sometimes it's not as simple as "as *x* increases, *y* increases")
  - Plot a feature vs. its impact on the outcome
- The importance is at an overall level, not for each individual prediction
  - Use feature vs. impact and waterfall charts

Another example of how to use xgboostExplainer (on a classification problem as opposed to regression):

https://medium.com/applied-data-science/new-r-package-the-xgboost-explainer-51dd7d1aa211

Thank you for listening!
Any questions?

Full code – Prepare data

```
# Load libraries
library(dplyr); library(jsonlite); library(purrr); library(rpart); library(rpart.plot);
library(Matrix); library(xgboost); library(ggplot2); library(devtools); library(extrafont);
install_github("AppliedDataSciencePartners/xgboostExplainer")

library(xgboostExplainer)

# Load data
tmdb <- read.csv('~/Work/tmdb_5000_movies.csv', stringsAsFactors = FALSE)

# Prepare data ------------------------------------------------------------------
## Transform columns from JSON format
tmdb <- tmdb %>%
  mutate(genres = unlist(lapply(genres, FUN = function(x) paste(fromJSON(x)$name, collapse = ", "))),
         keywords = unlist(lapply(keywords, FUN = function(x) paste(fromJSON(x)$name, collapse = ", "))),
         production_companies = unlist(lapply(production_companies, FUN = function(x) paste(fromJSON(x)$name, collapse = ", "))),
         production_countries = unlist(lapply(production_countries, FUN = function(x) paste(fromJSON(x)$name, collapse = ", "))),
         spoken_languages = unlist(lapply(spoken_languages, FUN = function(x) paste(fromJSON(x)$name, collapse = ", "))))

## Filter out anything with no budget info etc. or
tmdb <- tmdb %>%
  filter(budget > 0, revenue > 0, vote_average > 0, vote_count > 0) %>%
  filter(genres != "", production_countries != "", production_companies != "", spoken_languages != "")

## Make columns for each genre
genres <- unique(unlist(strsplit(paste0(tmdb$genres, collapse = ", "), split = ", ")))
genres <- gsub(" ", ".", genres)

tmdb <- tmdb %>%
  bind_cols(tmdb$genres %>%
  map_dfr(function(x) as.data.frame(t(sapply(genres, FUN = function(y) grepl(y, x))))))

## Make columns for each production company
production_companies <- unique(unlist(strsplit(paste0(tmdb$production_companies, collapse = ", "), split = ", ")))
summary_production_companies <- summary(as.factor(unlist(strsplit(paste0(tmdb$production_companies, collapse = ", "), split = ", "))), maxsum = 4000)
top_production_companies <- names(summary_production_companies)[summary_production_companies > 10]
top_production_companies[grepl("^[0-9]", top_production_companies)] <- paste0("x", top_production_companies[grepl("^[0-9]", top_production_companies)])
top_production_companies <- gsub("[[:punct:]]", "", top_production_companies)
top_production_companies <- gsub(" ", ".", top_production_companies)

tmdb <- tmdb %>%
  bind_cols(tmdb$production_companies %>%
              map_dfr(function(x) as.data.frame(t(sapply(top_production_companies, FUN = function(y) grepl(y, x))))))

## Make columns for each production country
production_countries <- unique(unlist(strsplit(paste0(tmdb$production_countries, collapse = ", "), split = ", ")))

tmdb <- tmdb %>%
  bind_cols(tmdb$production_countries %>%
              map_dfr(function(x) as.data.frame(t(sapply(production_countries, FUN = function(y) grepl(y, x))))))

## Make columns for spoken languages
spoken_languages <- unique(unlist(strsplit(paste0(tmdb$spoken_languages, collapse = ", "), split = ", ")))

tmdb <- tmdb %>%
  bind_cols(tmdb$spoken_languages %>%
              map_dfr(function(x) as.data.frame(t(sapply(spoken_languages, FUN = function(y) grepl(y, x))))))

## Remove any spaces from column names
names(tmdb) <- gsub(" ", ".", names(tmdb))
```

Full code – Decision tree

```
# Make models -----------------------------------------------------------------
## Decision tree
fmla <- formula(paste("vote_average ~ budget + original_language + revenue + runtime + vote_count",
                      paste(genres, collapse = " + "),
                      paste(top_production_companies, collapse = " + "),
                      sep = " + "))

tree <- rpart(fmla, tmdb, method = "anova", control = rpart.control(maxdepth = 3))

rpart.plot(tree, cex = 0.8)

tmdb$pred_tree <- predict(tree, tmdb)

ggplot(data = tmdb, aes(x = vote_average, y = pred_tree)) +
  geom_point(alpha = 0.4, colour = "#F7CB15") +
  geom_abline(slope = 1, intercept = 0, linetype = 'dotted') +
  labs(x = 'Actual vote average', y = 'Predicted vote average') +
  coord_cartesian(xlim = c(2,9), ylim = c(2,9)) +
  theme(panel.background = element_blank(),
        panel.grid = element_blank(),
        panel.border = element_blank(),
        plot.background = element_blank(),
        legend.position = "none",
        axis.ticks.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text.x = element_text(size = 16),
        axis.text.y = element_text(size = 16),
        axis.title.x = element_text(size = 20),
        axis.title.y = element_text(size = 20),
        text = element_text(family = "Segoe UI Light", color = "#434A54"))
```

Full code – XGBoost

```
## XGBoost
mm <- sparse.model.matrix(fmla, tmdb)
param <- list(booster = 'gbtree', objective = 'reg:linear')
outcome <- tmdb$vote_average

## Cross-validation and model
cv <- xgb.cv(data = mm, label = outcome,
             params = param, nfold = 5, nround = 3000, early_stopping_rounds = 10, nthread = 2)

bst <- xgboost(data = mm, label = outcome, params = param, nrounds = cv$best_iteration, base_score = mean(outcome))

## Fit of model
tmdb$pred <- predict(bst, mm)

ggplot(data = tmdb, aes(x = vote_average, y = pred)) +
  geom_point(alpha = 0.4, colour = "#F7CB15") +
  geom_abline(slope = 1, intercept = 0, linetype = 'dotted') +
  coord_cartesian(xlim = c(2, 9), ylim = c(2, 9)) +
  labs(x = 'Actual vote average', y = 'Predicted vote average') +
  theme(panel.background = element_blank(),
        panel.grid = element_blank(),
        panel.border = element_blank(),
        plot.background = element_blank(),
        legend.position = "none",
        axis.ticks.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text.x = element_text(size = 16),
        axis.text.y = element_text(size = 16),
        axis.title.x = element_text(size = 20),
        axis.title.y = element_text(size = 20),
        text = element_text(family = "Segoe UI Light", color = "#434A54"))

importance <- xgb.importance(feature_names = colnames(mm), model = bst)
importance <- importance %>%
  mutate(Group = case_when(gsub("TRUE", "", Feature) %in% genres ~ 'Genre',
                           gsub("TRUE", "", Feature) %in% top_production_companies ~ 'Production company',
                           grepl("original_language", Feature) ~ 'original_language',
                           TRUE ~ Feature))

ggplot(data = importance %>% group_by(Group) %>% summarise(Gain = sum(Gain)), aes(x = reorder(Group, Gain), y = Gain)) +
  geom_bar(stat = 'identity', fill = "#3AB795") +
  coord_flip() +
  scale_x_discrete(labels = c('vote_count' = 'Vote count',
                              'runtime' = 'Run time',
                              'budget' = 'Budget',
                              'Genre' = 'Genre',
                              'revenue' = 'Revenue',
                              'Production company' = 'Production company',
                              'original_language' = 'Original language')) +
  labs(x = '', y = 'Gain') +
  theme(panel.background = element_blank(),
        panel.grid = element_blank(),
        panel.border = element_blank(),
        plot.background = element_blank(),
        legend.position = "none",
        axis.ticks.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text.x = element_text(size = 16),
        axis.text.y = element_text(size = 16),
        axis.title.x = element_text(size = 20),
        axis.title.y = element_text(size = 20),
        text = element_text(family = "Segoe UI Light", color = "#434A54"))
```

Full code – XGBoost Explainer

```
## XGBoost explainer
explainer <- buildExplainer(bst, xgb.DMatrix(mm, label = outcome), type = "regression", base_score = mean(outcome))
pred.breakdown <- explainPredictions(bst, explainer, xgb.DMatrix(mm, label = outcome))

# Avatar
showWaterfall(bst, explainer, xgb.DMatrix(mm, label = outcome), mm,
              idx = 1, type = "regression", threshold = 0.03)

# Plot of how scores vary by budget
ggplot(data = data.frame(Budget = mm[,'budget'], Pred = pred.breakdown[,budget]),
       aes(x = Budget, y = Pred)) +
  geom_point(alpha = 0.4, colour = "#F7CB15") +
  scale_x_continuous(labels = function(x) paste0(x / 1000000, 'm')) +
  labs(x = 'Budget', y = 'Impact of budget on prediction') +
  theme(panel.background = element_blank(),
        panel.grid = element_blank(),
        panel.border = element_blank(),
        plot.background = element_blank(),
        legend.position = "none",
        axis.ticks.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text.x = element_text(size = 16),
        axis.text.y = element_text(size = 16),
        axis.title.x = element_text(size = 20),
        axis.title.y = element_text(size = 20),
        text = element_text(family = "Segoe UI Light", color = "#434A54"))
```