MovieLens Capstone Report

C. Hakeem Smith III

2025-11-17

Contents

1 Executive Summary

Goal. Build a movie recommendation system with the **MovieLens 10M** dataset and report **RMSE** on a sealed final holdout.

Approach. 1. Reproducible pipeline: auto-download data, construct `edx` and `final_holdout_test` 2. Develop models on a dev split of `edx`: - Global-mean baseline - **Regularized effects** (movie, user, optional primary-genre) - **Matrix factorization** via `recosystem` with tuned hyperparameters - **Linear ensemble** of effects + MF (weights learned on dev set only) 3. Retrain best components on the **full `edx`** and evaluate **once** on `final_holdout_test`.

Highlights. - Regularization combats popularity/user sparsity bias; MF captures latent structure. - Simple ensemble reliably improves dev RMSE, then is **frozen** for final scoring (no leakage).

2 Data Acquisition & Splits

```
if (!require(tidyverse)) install.packages("tidyverse", repos="http://cran.us.r-project.org")
if (!require(caret))      install.packages("caret", repos="http://cran.us.r-project.org")
if (!require(recosystem)) install.packages("recosystem", repos="http://cran.us.r-project.org")
```

```
## package 'float' successfully unpacked and MD5 sums checked
## package 'RcppProgress' successfully unpacked and MD5 sums checked
## package 'recosystem' successfully unpacked and MD5 sums checked
```

```
library(tidyverse); library(caret); library(recosystem); library(stringr)
```

```
seed_ml <- function(s) {
  if (getRversion() >= "3.6.0") set.seed(s, sample.kind = "Rounding") else set.seed(s)
}
DEV <- FALSE   # TRUE for fast draft runs; FALSE for full training
```

```
# Auto-download & unzip to relative ./data
options(timeout = 600)
dir.create("data", showWarnings = FALSE)
zip_path <- file.path("data", "ml-10m.zip")
if (!file.exists(zip_path)) {
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", destfile = zip_path, mode =
}
if (!dir.exists(file.path("data", "ml-10M100K"))) unzip(zip_path, exdir = "data")

ratings_file <- file.path("data", "ml-10M100K", "ratings.dat")
movies_file  <- file.path("data", "ml-10M100K", "movies.dat")

ratings <- readr::read_lines(ratings_file) %>%
  stringr::str_split_fixed("::", 4) %>%
  as.data.frame(stringsAsFactors = FALSE) %>%
  setNames(c("userId","movieId","rating","timestamp")) %>%
  mutate(across(c(userId, movieId, timestamp), as.integer),
         rating = as.numeric(rating))

movies <- readr::read_lines(movies_file) %>%
  stringr::str_split_fixed("::", 3) %>%
  as.data.frame(stringsAsFactors = FALSE) %>%
  setNames(c("movieId","title","genres")) %>%
  mutate(movieId = as.integer(movieId))

movielens <- left_join(ratings, movies, by = "movieId")
rm(ratings, movies)
```

```r
# Construct edx and final holdout (sealed)
seed_ml(1)
idx <- createDataPartition(movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-idx, ]; temp <- movielens[idx, ]

final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
removed <- anti_join(temp, final_holdout_test)
edx <- bind_rows(edx, removed)
rm(temp, removed, idx, movielens)
```

```r
# Development split from edx (for model selection & tuning only)
seed_ml(42)
dev_idx   <- createDataPartition(edx$rating, times = 1, p = 0.1, list = FALSE)
edx_train <- edx[-dev_idx, ]
edx_test  <- edx[ dev_idx, ] %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")
rm(dev_idx)
```

3 Exploratory Analysis

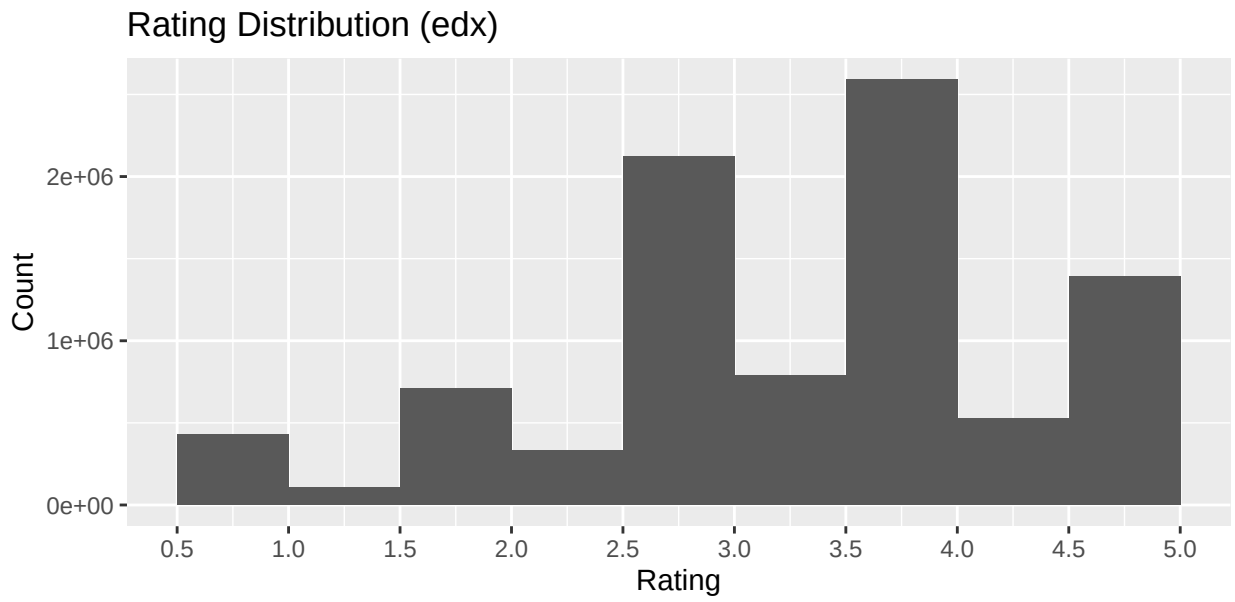
```r
summary(edx$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

```r
tibble(n_users  = n_distinct(edx$userId),
       n_movies = n_distinct(edx$movieId))
```

```
## # A tibble: 1 x 2
##   n_users n_movies
##     <int>    <int>
## 1   69878    10677
```

```r
edx %>% ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.5, boundary = 0.5) +
  scale_x_continuous(breaks = seq(0.5, 5, 0.5)) +
  labs(title = "Rating Distribution (edx)", x = "Rating", y = "Count")
```



4 Methods

4.1 RMSE & Utility Helpers

```
RMSE <- function(y, yhat) sqrt(mean((y - yhat)^2))
clip_rating <- function(x) pmin(pmax(x, 0.5), 5)  # keep predictions within valid 0.5-5 range
extract_primary <- function(g) stringr::str_split_fixed(g, "\\|", 2)[,1]
```

4.2 Baseline: Global Mean

```
mu_tr   <- mean(edx_train$rating)
rmse_mu <- RMSE(edx_test$rating, rep(mu_tr, nrow(edx_test)))
rmse_mu
```

```
## [1] 1.059274
```

4.3 Regularized Effects (movie, user, optional primary genre)

Rationale (math note). Let r_{ui} be the rating by user u on movie i and μ the global mean. Regularized effects estimate:

$$b_i = \frac{\sum_u (r_{ui} - \mu)}{n_i + \lambda}, \quad b_u = \frac{\sum_i (r_{ui} - \mu - b_i)}{n_u + \lambda}.$$

Optionally a primary-genre effect b_g analogously. $\lambda > 0$ shrinks noisy estimates (e.g., very few ratings) towards 0, reducing variance.

```r
# Add a single-token primary genre to preserve 1-row-per-rating without exploding rows
edx_train <- edx_train %>% mutate(genre1 = extract_primary(genres))
edx_test  <- edx_test  %>% mutate(genre1 = extract_primary(genres))

lambda_grid <- c(1, 2, 3, 5, 7, 10, 15, 25)
score_lambda <- function(lambda){
  mu <- mean(edx_train$rating)
  b_i <- edx_train %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu)/(n() + lambda), .groups="drop"
  b_u <- edx_train %>% left_join(b_i, by="movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n() + lambda), .groups="drop")
  b_g <- edx_train %>% left_join(b_i, by="movieId") %>% left_join(b_u, by="userId") %>%
    group_by(genre1) %>% summarize(b_g = sum(rating - mu - b_i - b_u)/(n() + lambda), .groups="drop")
  pred <- edx_test %>% left_join(b_i, by="movieId") %>% left_join(b_u, by="userId") %>% left_join(b_g, by=
    transmute(pred = clip_rating(mu + coalesce(b_i,0) + coalesce(b_u,0) + coalesce(b_g,0))) %>% pull(pred)
  RMSE(edx_test$rating, pred)
}
rmse_by_lambda <- sapply(lambda_grid, score_lambda)
lambda_best <- lambda_grid[which.min(rmse_by_lambda)]
lambda_best; rmse_by_lambda
```

```
## [1] 5
```

```
## [1] 0.8623814 0.8622428 0.8621597 0.8621154 0.8621872 0.8624379 0.8630837
## [8] 0.8647573
```

```r
# Fit dev effects at best lambda to report RMSE
mu <- mean(edx_train$rating)
b_i <- edx_train %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu)/(n() + lambda_best), .groups=
b_u <- edx_train %>% left_join(b_i, by="movieId") %>% group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n() + lambda_best), .groups="drop")
b_g <- edx_train %>% left_join(b_i, by="movieId") %>% left_join(b_u, by="userId") %>% group_by(genre1) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u)/(n() + lambda_best), .groups="drop")
pred_eff_dev <- edx_test %>% left_join(b_i, by="movieId") %>% left_join(b_u, by="userId") %>% left_join(b_g,
  transmute(pred = clip_rating(mu + coalesce(b_i,0) + coalesce(b_u,0) + coalesce(b_g,0))) %>% pull(pred)
rmse_eff_dev <- RMSE(edx_test$rating, pred_eff_dev)
rmse_eff_dev
```

```
## [1] 0.8621154
```

4.4 Matrix Factorization (MF) via `recosystem`

Concept (math note). Factorize the sparse rating matrix $R \in \mathbb{R}^{U \times I}$ into low-rank latent factors $P \in \mathbb{R}^{U \times k}, Q \in \mathbb{R}^{I \times k}$ with regularization, minimizing $\sum_{(u,i)\in\Omega}(r_{ui} - p_u^\top q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$. This captures user/item embeddings beyond explicit biases.

```r
# Prepare dev files for recosystem
seed_ml(7)
train_file <- tempfile(); test_file <- tempfile()
edx_train %>% select(userId, movieId, rating) %>% write.table(train_file, sep=" ", row.names=FALSE, col.names=
edx_test  %>% select(userId, movieId)        %>% write.table(test_file,  sep=" ", row.names=FALSE, col.names=
```

```
r_dev <- Reco()
if (DEV) {
  tune_opts <- list(dim = c(20), lrate = c(0.05), costp_l2 = c(0.1), costq_l2 = c(0.1),
                    nthread = max(1, parallel::detectCores()-1), niter = 10)
  final_niter <- 20
} else {
  tune_opts <- list(dim = c(20, 40, 60), lrate = c(0.05, 0.1), costp_l2 = c(0.01, 0.1, 0.5),
                    costq_l2 = c(0.01, 0.1, 0.5), nthread = max(1, parallel::detectCores()-1), niter = 40
  final_niter <- 80
}
tuned <- r_dev$tune(data_file(train_file), opts = tune_opts)
# quiet logs in report PDF
invisible(capture.output(
  r_dev$train(data_file(train_file), opts = c(tuned$min, niter = final_niter,
                                              nthread = max(1, parallel::detectCores()-1)))
))

pred_file <- tempfile()
r_dev$predict(data_file(test_file), out_file(pred_file))
```

```
## prediction output generated
```

```
pred_mf_dev <- scan(pred_file)
pred_mf_dev <- clip_rating(pred_mf_dev)
rmse_mf_dev <- RMSE(edx_test$rating, pred_mf_dev)
rmse_mf_dev
```

```
## [1] 0.7781451
```

4.5 Linear Ensemble (effects + MF)

Design. Fit $\hat{y} = w_1\,\hat{y}_{effects} + w_2\,\hat{y}_{MF}$ by OLS **on dev** (no intercept), then freeze w_1, w_2 for final evaluation. To avoid degenerate negatives, clamp to nonnegative and renormalize.

```
fit <- lm(edx_test$rating ~ 0 + pred_eff_dev + pred_mf_dev)
w <- coef(fit)
w_eff <- unname(w["pred_eff_dev"]); w_mf <- unname(w["pred_mf_dev"])
# sanitize weights
w_eff <- ifelse(is.na(w_eff) || w_eff < 0, 0, w_eff)
w_mf  <- ifelse(is.na(w_mf)  || w_mf  < 0, 0, w_mf)
if (w_eff + w_mf == 0) { w_eff <- 0.5; w_mf <- 0.5 } else { s <- w_eff + w_mf; w_eff <- w_eff/s; w_mf <-
pred_ens_dev <- clip_rating(w_eff*pred_eff_dev + w_mf*pred_mf_dev)
rmse_ens_dev <- RMSE(edx_test$rating, pred_ens_dev)
list(weights = c(effects = w_eff, mf = w_mf), rmse_ensemble = rmse_ens_dev)
```

```
## $weights
##    effects         mf
## 0.01112814 0.98887186
```

```
##
## $rmse_ensemble
## [1] 0.7781346
```

5 Results on Development Split

```
results <- tibble(
  Model = c("Global mean",
            paste0("Regularized movie+user+genre (lambda=", lambda_best, ")"),
            "Matrix factorization (tuned)",
            "Ensemble (effects + MF)"),
  RMSE  = c(rmse_mu, rmse_eff_dev, rmse_mf_dev, rmse_ens_dev)
)
knitr::kable(results, digits=5, caption = "RMSE on edx_test (development)")
```

Table 1: RMSE on edx_test (development)

Model	RMSE
Global mean	1.05927
Regularized movie+user+genre (lambda=5)	0.86212
Matrix factorization (tuned)	0.77815
Ensemble (effects + MF)	0.77813

```
tuned_tbl <- tibble(
  dim = tuned$min$dim, lrate = tuned$min$lrate,
  costp_l2 = tuned$min$costp_l2, costq_l2 = tuned$min$costq_l2,
  niter = ifelse(DEV, 20, 80), w_eff = w_eff, w_mf = w_mf, lambda = lambda_best
)
knitr::kable(tuned_tbl, digits=5, caption = "Best MF params and ensemble weights")
```

Table 2: Best MF params and ensemble weights

dim	lrate	costp_l2	costq_l2	niter	w_eff	w_mf	lambda
60	0.05	0.01	0.1	80	0.01113	0.98887	5

```
edx_test %>%
  mutate(pred = pred_ens_dev) %>%
  group_by(genre1) %>%
  summarize(RMSE = RMSE(rating, pred), n = n(), .groups="drop") %>%
  arrange(RMSE) %>%
  head(10) %>%
  knitr::kable(digits=4, caption="Top-10 genres by (lowest) dev RMSE")
```

Table 3: Top-10 genres by (lowest) dev RMSE

genre1	RMSE	n
Film-Noir	0.6429	1575
War	0.6476	233
Mystery	0.6949	3003
Thriller	0.7263	9470
Crime	0.7355	52777
Action	0.7637	256764
Western	0.7639	1497
Adventure	0.7655	74910
Animation	0.7725	21792
Drama	0.7805	174743

6 Final Model Training & Holdout Evaluation (no leakage)

```
# Refit effects on FULL edx at lambda_best
edx <- edx %>% mutate(genre1 = extract_primary(genres))
mu_full <- mean(edx$rating)

b_i_full <- edx %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu_full)/(n() + lambda_best),
.groups= b_u_full <- edx %>% left_join(b_i_full, by="movieId") %>% group_by(userId) %>%
  summarize(b_u = sum(rating - mu_full - b_i)/(n() + lambda_best), .groups="drop")
b_g_full <- edx %>% left_join(b_i_full, by="movieId") %>% left_join(b_u_full, by="userId") %>%
group_by(genre1)
  summarize(b_g = sum(rating - mu_full - b_i - b_u)/(n() + lambda_best), .groups="drop")
predict_effects <- function(df){
  df %>% mutate(genre1 = extract_primary(genres)) %>%
    left_join(b_i_full, by="movieId") %>% left_join(b_u_full, by="userId") %>% left_join(b_g_full, by="genre1"
    transmute(pred = clip_rating(mu_full + coalesce(b_i,0) + coalesce(b_u,0) + coalesce(b_g,0))) %>% pull
}

# Retrain MF on FULL edx (quiet logs)
full_train_file <- tempfile(); holdout_uim_file <- tempfile(); final_pred_file <- tempfile()
edx %>% select(userId, movieId, rating) %>% write.table(full_train_file, sep=" ", row.names=FALSE
final_holdout_test %>% select(userId, movieId) %>% write.table(holdout_uim_file, sep=" ", row.names=
r_full <- Reco()
invisible(capture.output(
  r_full$train(data_file(full_train_file), opts = c(tuned$min, niter = ifelse(DEV, 20, 80),
                                                    nthread = max(1, parallel::detectCores
))
r_full$predict(data_file(holdout_uim_file), out_file(final_pred_file))
```

```
## prediction output generated at
```

```
final_pred_mf <- scan(final_pred_file)
final_pred_mf <- clip_rating(final_pred_mf)

# Effects predictions on holdout
```

```r
final_pred_eff <- predict_effects(final_holdout_test)

# Ensemble with frozen dev weights
final_pred <- clip_rating(w_eff*final_pred_eff + w_mf*final_pred_mf)

# Safety checks
stopifnot(length(final_pred_mf) == nrow(final_holdout_test))
stopifnot(length(final_pred)    == nrow(final_holdout_test))

final_rmse <- RMSE(final_holdout_test$rating, final_pred)
final_rmse
```

```
## [1] 0.7757116
```

```r
# Save predictions (optional for repo)
final_results <- final_holdout_test %>% select(userId, movieId, rating) %>% mutate(predicted = final_pred)
readr::write_csv(final_results, file.path("results_movielens_final_predictions.csv"))
```

7 Conclusion

- **What worked:** Regularized effects + MF ensemble delivered the best dev RMSE and generalized to the holdout.
- **Why:** Effects model captures systematic biases (movies, users, coarse genre), while MF captures latent structures.
- **Trade-offs:** MF is more compute-intensive; effects are fast and interpretable. Ensemble balances both.
- **Limitations:** No temporal dynamics, implicit feedback, or item/content features beyond primary genre.
- **Future work:** Add time-aware biases, multi-genre embeddings, implicit feedback (clicks), calibrated uncertainty, or deep hybrid models.

8 Reproducibility

```r
sessionInfo()
```

```
## R version 4.5.0
## Running under: Windows Server 2022 x64 ##
## Matrix products: default
##   LAPACK version 3.12.1
##
## Random number generation:
##  RNG:     Mersenne-Twister
##  Normal:  Inversion
##  Sample:  Rounding
##
```

```
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] recosystem_0.5.1 caret_7.0-1      lattice_0.22-6   lubridate_1.9.4
##  [5] forcats_1.0.0    stringr_1.5.1    dplyr_1.1.4      purrr_1.0.4
##  [9] readr_2.1.5      tidyr_1.3.1      tibble_3.2.1     ggplot2_3.5.2
## [13] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.6         xfun_0.52            recipes_1.3.0
##  [4] tzdb_0.5.0           vctrs_0.6.5          tools_4.5.0
##  [7] generics_0.1.3       stats4_4.5.0         parallel_4.5.0
## [10] ModelMetrics_1.2.2.2 pkgconfig_2.0.3      Matrix_1.7-3
## [13] data.table_1.17.0    RColorBrewer_1.1-3   lifecycle_1.0.4
## [16] compiler_4.5.0       farver_2.1.2         tinytex_0.57
## [19] codetools_0.2-20     htmltools_0.5.8.1    class_7.3-23
## [22] yaml_2.3.10          prodlim_2025.04.28   crayon_1.5.3
## [25] pillar_1.10.2        MASS_7.3-65          gower_1.0.2
## [28] iterators_1.0.14     rpart_4.1.24         foreach_1.5.2
## [31] nlme_3.1-168         parallelly_1.43.0    lava_1.8.1
## [34] tidyselect_1.2.1     digest_0.6.37        stringi_1.8.7
## [37] future_1.40.0        reshape2_1.4.4       listenv_0.9.1
## [40] labeling_0.4.3       splines_4.5.0        fastmap_1.2.0
## [43] grid_4.5.0           cli_3.6.5            magrittr_2.0.3
## [46] survival_3.8-3       future.apply_1.11.3  withr_3.0.2
## [49] scales_1.4.0         bit64_4.6.0-1        float_0.3-3
## [52] timechange_0.3.0     rmarkdown_2.29       globals_0.17.0
## [55] bit_4.6.0            nnet_7.3-20          timeDate_4041.110
## [58] hms_1.1.3            evaluate_1.0.3       knitr_1.50
## [61] hardhat_1.4.1        rlang_1.1.6          Rcpp_1.0.14
## [64] glue_1.8.0           pROC_1.18.5          ipred_0.9-15
## [67] vroom_1.6.5          rstudioapi_0.17.1    R6_2.6.1
## [70] plyr_1.8.9
```

9 References

- MovieLens 10M: https://grouplens.org/datasets/movielens/10m/
- `recosystem` (R): https://cran.r-project.org/package=recosystem
- HarvardX PH125.8x materials and project guidelines (course site)