Chocolate Chip Pancakes

Ajay Mandlekar
Anand Poozhikunnel
Rahul Bachal
What we Tried

- SVD
- SVD++
- TimeSVD++
- KNN
- KNN on Residuals of SVDs
- RBM
- Blending on Probe
- Blending on Quiz
General Notes on Models

- Models were trained using Stochastic Gradient Descent (SGD), with no shuffling
- For most part, stuck with 60 epochs for training
- Used a parameter scan to cycle through many different permutations of all the parameters
- Used Multithreading, by splitting each model into 8 different threads, to speed up time per epoch
Latent Factor Models

- **SVD**
  - Originally just used movie and user factors, then added bias terms which improved performance
  - Reduced time per epoch to under 10 seconds using multi-threading
  - Performance increased with number of factors used, however, with diminishing returns
  - Best performance of a single model was 5.2% using 2000 factors
Latent Factor Models

- SVD++
  - Began with a naïve implementation which took on the order of 10 hours per epoch
  - Moved to a smarter implementation which reduced time to on the order of 10 seconds per epoch
  - At first, the performance of this model was only 5.3% above water (on a good day)
  - Then, a few well-hidden bugs were exterminated, giving a performance boost up to about 6.3% above water
  - Additional parameter scans resulted in around 6.6% above water
Latent Factor Models

• Automatic Parameter Tuner 2 - APT2
  • Implemented from paper by Toscher and Jahrer
  • For a parameter $p$, try new values $p*e$ and $p/e$ for $e < 1$.
  • Increase $e$ by $e^{0.9}$ if new values result in higher RMSE

• Running parameter scan for 6 parameters with 2 passes through all parameters and 5 tries per parameter took $\sim 10$ hours
• New parameters improved score by $\sim 10$-20 bps
Latent Factor Models

- TimeSVD++
  - Implemented three variants of TimeSVD++.
  - Model A) Only utilized movie bias time effects and linear user bias time effects.
    - Movies were binned by date and each bin had its own parameter.
    - Deviation function over a user’s rating date from the user’s mean date.
    - Each user had an alpha parameter scaling the deviation.
    - Performance – 6.75% above water
  - Model T) Added constant bias terms for every date, for every user.
    - Performance – 7.2% above water
Latent Factor Models

• **TimeSVD++**
  • Model C) Added linear user factor time effects to user factors.
    • Performance – 7.05% above water
    • Worse than Model B, so did not pursue this.
  • Did not implement full TimeSVD++ (has constant bias terms for every user factor) due to memory issues.
K Nearest Neighbors (KNN)

• Used Pearson Correlation Coefficients as the similarity metric.
• Used fisher / inverse fisher transform to get lower bound on 95% confidence interval on Pearson correlation

• Weight for movie m2 as a neighbor of movie m1 is simply their correlation squared times the logarithm of the number of common viewers.

• Pure kNN model prediction - a weighted average over ratings of the K neighbors with highest weights.

• Performance – roughly 1% above water.
KNN

- The best use of KNN was to feed it the residuals of another model in order to generate modified qual predictions.

- Generally improved performance by about 0.3% for most models.
Restricted Boltzmann Machines (RBM)

- Used graphchi package to run RBM.
- Performance – 2.5% above water after some fine tuning of parameters.
- Wanted to apply kNN to residuals of RBM but couldn’t find out how to get residuals out of graphchi.
Blending

• First blended on probe.
  • Worked well, but wanted to train our models on probe.

• Moved to blending on quiz.
  • Allowed us to use probe in training, but added danger of overfitting to quiz set.

• We also tried clipping values below 1 and above 5 before and after blending models – worked well.

• Also added constant model which consisted of quiz avg ratings.
Blending

• RBMs blended well with latent factor models despite bad performance.

• kNN blended poorly 😞

• However, kNN on residuals of SVD, SVD++, timeSVD++ all contributed nicely to the blend.

• Ironically, blending in our incorrect SVD++ implementation helped our score.
The Mystery Machine

- Constant Average Model
- 3 SVD models
- 8 SVD++ models (some were incorrect)
- 3 timeSVD++ models
- 4 kNN on SVD++ models
- 5 kNN on timeSVD++ models
- 1 ALS model (alternating least squares)
- 6 RBM models
- 2 pure kNN models
- Total – 32 models
The Mystery Machine

- We clipped each model before we added it to the blend
- We used the RMSE of each model pre-clipping, however
- The model which was simply just a constant guess for all tuples improved the blend surprisingly well, despite being a bad standalone model
What We Learned

• Implementing algorithms from papers.
• Machine Learning on large data sets.
  • Optimizing code to speed up performance.
    • Memory used (available RAM, cache performance and locality)
    • Run time and smart ways to group computations.
    • Thinking conservatively about the limitations of hardware and what is absolutely needed for the algorithm.
• Lesson in debugging.
  • Things may appear to be working, but they might not actually be working.
    • SVD++ - Thought it was working (doing better than SVD), then talked to other groups and realized that ours was doing much worse than it should.
    • 2 hours of intense debugging later, found the problem.
Thank you

Professor Yaser Abu-Mostafa
The TA’s
   Especially Joseph Choi
Yehuda Koren and the Netflix Forums
Avi, Nand and Sarthak of 3 Shades of Brown