# Lessons Learned from the Netflix Contest

Arthur Dunbar

# Background

- From Wikipedia:
  - “The Netflix Prize was an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films, i.e. without the users or the films being identified except by numbers assigned for the contest.”
- Two teams dead tied for first place on the test set:
  - BellKor's Pragmatic Chaos
  - The Ensemble
- Both are covered here, but technically BellKor won because they submitted their algorithm 20 minutes earlier. This even though The Ensemble did better on the training set (barely).
- Lesson one learned! Be 21 minutes faster! ;-D
  - Don't feel too bad, that's the reason they won the competition, but the actual test set data was slightly better for them:
    - BellKor RMSE: 0.856704
    - The Ensemble RMSE: 0.856714

# Background

- Netflix dataset was 100,480,507 date-stamped movie rating performed by anonymous Netflix customers between December 31st 1999 and December 31st 2005. (Netflix has been around since 1997)
- This was 480,189 users and 17,770 movies

# Background

- There was a hold-out set of 4.2 million ratings for a test set consisting of the last 9 movies rated by each user that had rated at lest 18 movies.
  - This was split in to 3 sets, Probe, Quiz and Test
    - Probe was sent out with the training set and labeled
    - Quiz set was used for feedback. The results on this were published along the way and they are what showed upon the leaderboard
    - Test is what they actually had to do best on to win the competition. They were not informed of their performance on this so the system could not be gamed by making small tweaks and rerunning

# Background

- This hold-out set biased the results in favor of those algorithms that could better predict people with few previous ratings because each user contributes 9
- Also very sparse. Most users do not have an opinion logged on most movies
- To qualify you had to improve on Netflix's Cinematch by 10%+

# Note

- This is guessing a rating
  - [1,5] is the range of what it's guessing
  - This means, though it is machine learning, it is not a normal classification problem because 1 means something relative to 2 …. 5.
  - 1 is lower than 5 and not just a separate class

# Feature-Weighted Linear Stacking Sill Takacs Mackey and Lin

- Use an algorithm Feature Weighted Linear Stacking (FWLS) “that incorporates meta-features for improved accuracy while retaining the well-known virtues of linear regression regarding speed, stability, and interpretability.”
- FWLS combines model predictions linearly using coefficients that are themselves linear functions of meta-features

# Intro

- Stacking (aka blending) – two tiers of classifiers
  - Tier 1 learns original data
  - Tier 2 learns based on Tier 1 output
- Meta features
  - Features like the number of products rated by a user, or the number of days since release

# Feature Weighted Linear Stacking

- Let X represent the space of inputs, $g_1$ to $g_L$ that denote the learned prediction functions of L machine learning methods
  - $g_i : X \rightarrow R$ for all i
- Let $f_1$ to $f_M$ represent a collection M meta-feature functions to be used in blending
  - Each $f_i$ maps each data point x in X to its corresponding meta feature $f_i(x)$ in R

# Feature Weighted Linear Stacking

- Standard linear regression stacking seeks a blend prediction function b

$$b(x) = \sum_i w_i g_i(x), \forall x \in \mathcal{X}$$

- Were w is a constant in R

# Feature Weighted Linear Stacking

- Feature Weighted Linear Stacking however has $w_i$ as a function

$$w_i(x) = \sum_j v_{ij} f_j(x), \forall x \in \mathcal{X}$$

- For learned weights $v_{ij}$ in R, this changes b to

$$b(x) = \sum_{i,j} v_{ij} f_j(x) g_i(x), \forall x \in \mathcal{X}$$

- Making this optimization function for FWLS (where y(x) is the target prediction for a datapoint x in the subset of X~ used to train the stacking parameters):

$$\min_v \sum_{x \in \tilde{\mathcal{X}}} \sum_{i,j} (v_{ij} f_j(x) g_i(x) - y(x))^2.$$

# Feature Weighted Linear Stacking

- This gives a b which is linear in $v_{ij}$, and we can use a single linear regression to estimate those parameters.
- The input variables of the regression are the M*L products $f_j(x)g_i(x)$ aka the meta-feature function and model predictor evaluated at each x in X
  - Reminder $f_j(x)$ is the function that gives us x's jth meta-feature
  - $g_i(x)$ is the function that gives us x's ith learning algorithm

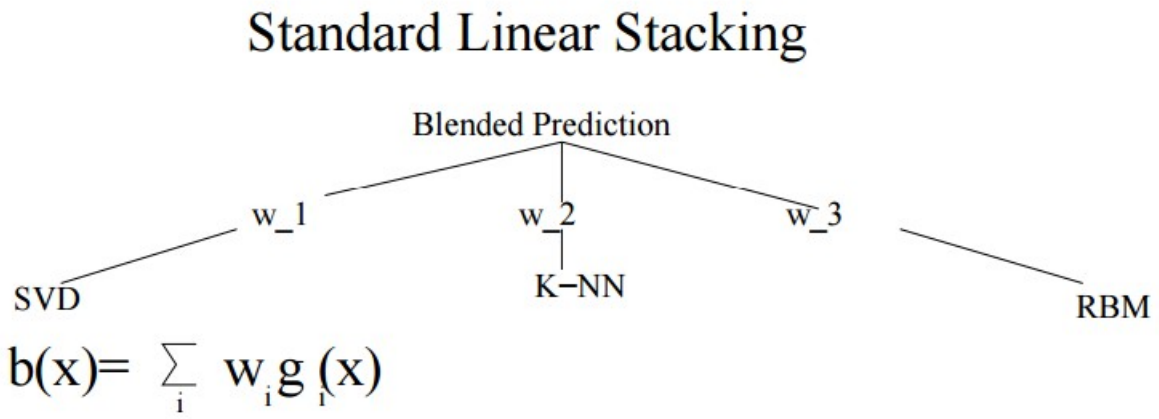


Figure 1: FWLS forms a linear combination of products of model outputs and meta-features

# Feature Weighted Linear Stacking

- The $g_i(x)$'s are represented as SVD, K-NN, RBM in the figure (these are 'collaborative filtering algorithms')
  - SVD is singular value decomposition
  - 'This simplest version of this approach represents each user and each movie as a vector of F real numbers. The predicted rating is given by the dot product of the user vector with the movie vector'
    - As far as I can tell SVD is used to make something more linear by removing dimensions from it
- FWLS can be interpreted as
  - 'a regression against all possible two-way products of models and meta-features.'
  - 'as a kind of bipartite quadratic regression against a set of independent variables consisting of the models and the meta-features'
  - 'a linear combination of models where the coefficients of the combination vary as a function of meta-features'
- All true, the first one is the easiest to program though

# Feature Weighted Linear Stacking

- Constant factors are involved
- To represent constant factors, the meta-feature I'm going to call “Always guess 1” is added.
- It acts like a bias factor

# FWLS Implementation

- N data points
- Matrix A is N x M*L dimensional with elements
  - $A_{n(i+L(j-1))} = f_j(x_n)g_i(x_n)$
  - $x_n$ is the input vector for the the nth data point in X
- Linear regression with Tikhonov regularization:
  - Solve this system: $(A^TA + \lambda I)v = A^ty$
  - $O(NM^2L^2 + M^3L^3)$ fortunately N>>>M & L

# Full list of meta features 1

Table 1: Meta-Features used for Netflix Prize model blending

| | |
|---|---|
| 1 | A constant 1 voting feature (this allows the original predictors to be regressed against in addition to their interaction with the voting features) |
| 2 | A binary variable indicating whether the user rated more than 3 movies on this particular date |
| 3 | The log of the number of times the movie has been rated |
| 4 | The log of the number of distinct dates on which a user has rated movies |
| 5 | A bayesian estimate of the mean rating of the movie after having subtracted out the user's bayesian-estimated mean |
| 6 | The log of the number of user ratings |
| 7 | The mean rating of the user, shrunk in a standard bayesian way towards the mean over all users of the simple averages of the users |
| 8 | The norm of the SVD factor vector of the user from a 10-factor SVD trained on the residuals of global effects |
| 9 | The norm of the SVD factor vector of the movie from a 10-factor SVD trained on the residuals of global effects |
| 10 | The log of the sum of the positive correlations of movies the user has already rated with the movie to be predicted |
| 11 | The standard deviation of the prediction of a 60-factor ordinal SVD |
| 12 | Log of the average number of user ratings for those users who rated the movie |
| 13 | The log of the standard deviation of the dates on which the movie was rated. Multiple ratings on the same date are represented multiple times in this calculation |
| 14 | The percentage of the correlation sum in feature 10 accounted for by the top 20 percent of the most correlated movies the user has rated. |

# Full list of meta features 2

| | |
|---|---|
| 15 | The standard deviation of the date-specific user means from a model which has separate user means (a.k.a biases) for each date |
| 16 | The standard deviation of the user ratings |
| 17 | The standard deviation of the movie ratings |
| 18 | The log of (rating date - first user rating date + 1) |
| 19 | The log of the number of user ratings on the date + 1 |
| 20 | The maximum correlation of the movie with any other movie, regardless of whether the other movies have been rated by the user or not |
| 21 | Feature 3 times Feature 6, i.e., the log of the number of user ratings times the log of the number of movie ratings |
| 22 | Among pairs of users who rated the movie, the average overlap in the sets of movies the two users rated, where overlap is defined as the percentage of movies in the smaller of the two sets which are also in the larger of the two sets. |
| 23 | The percentage of ratings of the movie which were the only rating of the day for the user |
| 24 | The (regularized) average number of movie ratings for the movies rated by the user. |
| 25 | First, a movie-movie matrix was created with entries containing the probability that the pair of movies was rated on the same day conditional on a user having rated both movies. Then, for each movie, the correlation between this probability vector over all movies and the vector of ratings correlations with all movies was computed |

# Meta-Features Explanations (some of them)

- There are 25 of these (previous pages). Their creation is an 'Art'
  - 1 is the always guess 1 feature, it exists for constant factors
  - 3 and 6 are the log of the number of reviews a user made or movie got. This is included to show how much data this user or movie has
  - 12 and 24 are the log of number of ratings the rater has given or the log of the number of ratings the movie the rater rated has. (exists for similar reasons to 3/6)
  - 10 and 14 characterize available correlation information
  - 4, 13 and 15 are included to include time data
  - Standard Deviations are a good match for confidence level in a prediction where high Std Dev implies low confidence

# Results

- Accuracy using each meta-feature on cross validation (CV) and the test set
- Yes the final test set, these results were not known to the during contest
- CV and test set are 'reasonably close'

| Meta-Feature | Probe CV RMSE | Test RMSE |
|---|---|---|
| 1 | 0.869889 | 0.863377 |
| 2 | 0.869513 | 0.862914 |
| 3 | 0.869092 | 0.862508 |
| 4 | 0.868734 | 0.862237 |
| 5 | 0.868612 | 0.862210 |
| 6 | 0.868348 | 0.862060 |
| 7 | 0.868271 | 0.862028 |
| 8 | 0.868238 | 0.861994 |
| 9 | 0.868227 | 0.861978 |
| 10 | 0.868163 | 0.861935 |
| 11 | 0.868023 | 0.861834 |
| 12 | 0.867967 | 0.861786 |
| 13 | 0.867890 | 0.861695 |
| 14 | 0.867861 | 0.861657 |
| 15 | 0.867846 | 0.861580 |
| 16 | 0.867773 | 0.861507 |
| 17 | 0.867745 | 0.861532 |
| 18 | 0.867707 | 0.861552 |
| 19 | 0.867636 | 0.861532 |
| 20 | 0.867618 | 0.861494 |
| 21 | 0.867575 | 0.861504 |
| 22 | 0.867547 | 0.861475 |
| 23 | 0.867543 | 0.861498 |
| 24 | 0.867512 | 0.861456 |
| 25 | 0.867501 | 0.861405 |

# Feature Weighted Linear Stacking
# End

# The BellKor Solution to the Netflix Grand Prize
# Koren

- The winners.
- This paper is the one the contest said was the representative one, there was another one put out as well though:
  - The BigChaos Solution to the Netflix Grand Prize
- First I'll explain some constituent parts, then the ensemble (there are a LOT)

# The BellKor Solution to the Netflix Grand Prize
# Koren

- Notations:
  - Users referred to with letters u and v
  - Movies referred to with letters i and j
  - Rating $r_{ui}$ is the preference of user u to movie i
  - Rating $\hat{r}_{ui}$ is a predicted preference
  - Ratings are in range [1,5]
  - $t_{ui}$ is the time of the rating (in days)
  - $K = \{(u,i) \mid r_{ui} \text{ is known}\}$
    - R(u) → all movies user u rated that we have the rating for
    - R(i) → all users who rated movie I
    - N(u) → R(u) + the unknown ratings movies

# Constituent part 1: Baseline Predictors

- The data used for this prediction falls in to two broad groups
  1) Things inherent to the user or movie
     - User: This user rates 5s on everything
     - Movie: Even people who don't like subtitled films and don't like fantasy films might like Pan's Labyrinth
       - Note you don't actually have genre info
  2) Things about the user and movie
- Baseline Predictors are category 1
- These are 'much of the observed signal'

# Baseline Predictors

- $B_{ui} = \mu + B_u + B_i$
  - $\mu$ – overall average rating
  - B – Baseline value related to the subtitle (u for user i for movie ui for total)
- For example:
  - Say all movies have an average rating of 3.7 → $\mu = 3.7$
  - Pan's Labyrinth has .8 above average → $B_i = .8$
  - Arthur is a cinical husk of a human being and hates movies so he rates .3 lower → $B_u = -.3$
  - B(Pan's Labyrinth)(Arthur) = 3.7 + .8 - .3 = 4.2

# Baseline Predictors

$$b_i = \frac{\sum_{u \in R(i)} (r_{ui} - \mu)}{\lambda_1 + |R(i)|}$$

- For all reviews given to this movie:
  - Sum the difference between all the scores this movie received and the average score for all movies
  - Divide by the total number of reviews it got + a regularizing constant
  - $\lambda 1 = 10$, decided by results on probe data

$$b_u = \frac{\sum_{i \in R(u)} (r_{ui} - \mu - b_i)}{\lambda_2 + |R(u)|}$$

- For all reviews this person gave to all movies they reviewed:
  - Sum the difference between the review they gave for all movies they reviewed and that movies' baseline score
  - Divide by the total number of reviews this person gave + a regularizing constant
  - $\lambda 2 = 25$, decided by results on probe data

# Baseline Predictors

- That last slide was just an idea of what they were going for. What they actually did was this:

$$\min_{b_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_3 (\sum_u b_u^2 + \sum_i b_i^2) .$$

- Term 1 was included to find $b_i$s and $b_u$s that fit the intended rating
- Term 2 penalized magnitude to reduce over-fitting
- $b_*$ is “all user and item biases”

# Time Changing Baseline Parameters

- Not done yet! Now take in to account time of the prediction.
  - Maybe the economy was good this year and people were happier
  - Maybe a war happened and patriotic movies got a higher rating
    - Note again no genre info
  - etc

# Time Changing Baseline Parameters

- $B_{ui} = \mu + B_u(T_{ui}) + B_i(T_{ui})$
  - User biases are more fickle and require higher resolution than movie biases
- Item baseline is split in to stationary and variable parts. Variable part is binned
  - $b_i(t) = b_i + b_{i,bin(t)}$
  - Many bin sizes worked well, they used 30 bins each about 10 weeks long spanning all 6 years
- Humans are too fickle for binning

# Time Changing Baseline Parameters

- Idea: Use a linear function to predict value
  - $b^{(1)}_u(t) = b_u + \alpha_u * dev_u(t)$
    - $\alpha$ and b are learned
  - $dev_u(t) = sign(t - tu) * |t - tu|^{\beta}$
    - $t-t_u$ is the number of days between
    - $\beta$ is .4, from probe set
- Not quite enough because users sometimes have spikes where they rate everything the same
  - They add a daily bias per user called $b_{ut}$
  - $b^{(3)}_u(t) = b_u + \alpha_u \cdot dev_u(t) + b_{ut}$
- All together now the bias term is:
  - $b_{ui} = \mu + b_u + \alpha_u * dev_u(t_{ui}) + b_{u,tui} + b_i + b_{i,Bin(tui)}$
  - By itself this has a RMSE = .9605

# Time Changing Baseline Parameters

- Still not done with this:
- Users respond to ratings they see, and change their own personal scale with time
  - $b_{ui} = \mu + b_u + \alpha_u * dev_u(t_{ui}) + b_{u,tui}$ $+ (b_i + b_{i,Bin(tui)}) * c_u(t_{ui})$
  - They don't go in to much detail on the c term other than to say it is trained similarly to b
  - This lowers RMSE to .9555

# Frequency

- The Baseline term isn't done yet!
- How many ratings a user gives in a day is indicative of what their bias will be as well
  - $F_{ui}$ is the number of movies u reviewed on day $t_{ui}$
  - This number is not used, instead they take a rounded down logarithm:
    - $f_{ui} = \log_a F_{ui}$ rounded down
      - a is 6.76
    - Oddly enough this effects the item bias rather than user bias
  - $b_{ui} = \mu + b_u + \alpha_u * dev_u(t_{ui}) + b_{u,tui} + (b_i + b_{i,Bin(tui)}) * c_u(t_{ui}) + b_{i,fui}$
  - RMSE is now .9278
  - This is now officially better than what Netflix was using before the competition

# Predicting Future Days

- Waaaaaait a second… Why were we talking about daily parameters when we're predicting for days that have not occurred yet?
  - They were trying to get background info for the ratings that are being fed in
  - On days in the future, the terms for daily baseline information are left out
  - This allowed these parameters to absorb short term fluctuations in the training data.

# What's in the Blend?

- In this section they talk about how they get the values for parameters and that the values of b et al are trained with stochastic gradient descent with weight decay and 40 iterations.

$$\min_{b_*, c_*, \alpha_*} \sum_{(u,i) \in \mathcal{K}} \Big( r_{ui} - \mu - b_u - \alpha_u \cdot \mathrm{dev}_u(t_{ui}) - b_{u,t_{ui}} - \big(b_i + b_{i,\mathrm{Bin}(t_{ui})}\big) \cdot \big(c_u + c_{u,t_{ui}}\big) \Big)^2 + \lambda_a b_u^2 + \lambda_b \alpha_u^2 + \lambda_c b_{u,t_{ui}}^2 + \lambda_d b_i^2 + \lambda_e b_{i,\mathrm{Bin}(t_{ui})}^2 + \lambda_f (c_u - 1)^2 + \lambda_g c_{u,t_{ui}}^2 . \quad (12)$$

| | $b_u$ | $b_{ut}$ | $\alpha_u$ | $b_i$ | $b_{i,\mathrm{Bin}(t)}$ | $c_u$ | $c_{ut}$ | $b_{i,f_{ui}}$ |
|---|---|---|---|---|---|---|---|---|
| lrate $\times 10^3$ | 2.67 | 2.57 | 3.11e-3 | .488 | .115 | 5.64 | 1.03 | 2.36 |
| reg $\times 10^2$ | 2.55 | .231 | 395 | 2.55 | 9.29 | 4.76 | 1.90 | 1.10e-6 |

# Matrix Factorization with Temporal Dynamics

$$\hat{r}_{ui} = b_{ui} + q_i^T \left( p_u(t_{ui}) + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

- $b_{ui}$ is the one from before
- Each movie is associated with a $q_i$ and each user is associated with a $p_u$. $y_j$ is a movie variable associated with the $p_u$
  - q is an f dimensional vector common to all models
  - y is also f dimensional a user factor based upon the movies rated by that user. It's summed for all ratings of the user gave and divided by the square root of the number of ratings
  - $p_u(t_{ui})^T = (p_{u1}(t),..., p_{uf}(t))$ also f dimensional and time dependent
    - $p_{uk}(t) = p_{uk} + \alpha_{uk} * dev_u(t) + p_{ukt}$
    - This is treated the same as the baseline stuff from before, sometimes the last term is omitted for memory's sake
  - f is a number

# Matrix Factorization with Temporal Dynamics

- Not quite done, they added a frequency term like before

$$\hat{r}_{ui} = b_{ui} + (q_i^T + q_{i,f_{ui}}^T)\left(p_u(t_{ui}) + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j\right)$$

- From here they go over many parametrizations of f and iterations. The one they go with has a RSME of .8777
- They don't go in to the full detail of how it's trained, they just say (for the final one they picked)
  - “Their respective learning rate is 2e-5, with regularization of 0.02.”
  - f is picked to be 200, and the number of iterations is 40

# Linear Blending

- For a data set of size N:
  - let y be a $R^N$ vector of unobserved true ratings
  - let $x_1,...x_p$ in $R^N$ be p vectors of known predictions
  - Let X be the N by p matrix with columns x1,..xp
  - The mean has been subtracted from y and each column of X
  - The goal is to find the linear combination of x1,…,xp that best predicts y
  - If y were known, linear regression would be use do estimate y by $X\beta$

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{y})$$

# Linear Blending $\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{y})$

- $(X^TX)^{-1}$ is easy enough but what about $(X^Ty)$?
- It is possible to estimate each component of this vector with 'high precision'
  - Consider the j-th element

$$\sum_u x_{ju} y_u .$$

  - Rewrite as

$$\frac{1}{2}\left[\sum_u y_u^2 + \sum_u x_{ju}^2 - \sum_u (y_u - x_{ju})^2\right]$$

# Linear Blending

$$\frac{1}{2}\left[\sum_u y_u^2 + \sum_u x_{ju}^2 - \sum_u (y_u - x_{ju})^2\right]$$

- Because the mean was subtracted from all terms, the first term can be closely approximated to N * the quiz set variance 1.274
- Term 2 can be computed strait
- Term 3 is N times the Mean Squared Error associated with $x_j$ for the quiz data
- This is guaranteed to be accurate within .01% except that N isn't known. Luckily it cancels next step!

# Linear Blending

$$\hat{\beta} = \left(\mathbf{X}^T\mathbf{X} + \lambda N\mathbf{I}\right)^{-1}\left(\mathbf{X}^T\mathbf{y}\right)$$

- They changed the formula for Beta to this because it somehow gets rid of problems with data being correlated
  - $\lambda$ = .0014
- Because $\lambda$ messed with things, MSE is now:

$$\mathrm{MSE}(\mathbf{X}\hat{\beta}) = \sum_i \hat{\beta}_i \left(\mathrm{MSE}(\mathbf{x}_i) - \mathrm{Var}(\mathbf{x}_i)\right) + 1.274\left(1 - \sum_i \hat{\beta}_i\right) + \mathrm{Var}(\mathbf{X}\hat{\beta})$$

# Linear Blending – Overfitting Estimation

- For ordinary least squares regression, the bias of the quiz MSE as an estimate of test MSE is -2p/N
  - p is the number of predictions
  - N is the size of the quiz data
- This bias is called optimism
  - Part of this optimism stat is random, but it is small
  - Part is that the test set might be just harder or easier than the quiz set

# Linear Blending - Overfitting Estimation

- The reason the equation below was used (Ridge Regression) is because it:
  - Regularizes
  - Deals with colinearity
  - Helps reduce the uncertainty of $X^T y$ from rounding

$$\hat{\beta} = \left(\mathbf{X}^T\mathbf{X} + \lambda N\mathbf{I}\right)^{-1}\left(\mathbf{X}^T\mathbf{y}\right)$$

  - This gives degrees of freedom:

$$\mathrm{df}(\lambda) = \mathrm{tr}\left[\mathbf{X}\left(\mathbf{X}^T\mathbf{X} + \lambda N\mathbf{I}\right)^{-1}\mathbf{X}^T\right]$$

  - And $\lambda$ = .0014 reduces optimism by .0003 with little penalty if any, and now you know why $\lambda$ was .0014

# Gradient Boosted Decision Trees

- An ensemble learner that uses many learners.
- Similar in concept to Random Forest
- Here's a few used:
  - Neighborhood Models with Temporal Dynamics
  - 3 Extensions to Restricted Boltzmann Machines
  - Another Gradient Boosted Decision Tree
  - 2 SVD++'s
  - A bunch of things they call predictor #(some number)
- In total there were 3 different ensembles used
  - One of 454 learners
  - One of 75 learners picked from those
  - And one of 24 “BellKor predictors” of similar quality to the ones I described earlier

# Gradient Boosted Decision Trees

- GBDT are “an additive regression model consisting of an ensemble of trees, fitted to current residuals in a forward step-wise manner.”
  - Sequentially fit a series of decision trees to the data
  - Each tree predicts the errors of the previous tree
  - Often uses perturbed versions of the data
- 4 parameters:
  - Number of trees
  - Size of each tree
  - Shrinkage (learning rate)
  - Sampling rate

# Gradient Boosted Decision Trees

- GBDTs can handle skewed data, so they added these data points as well:
  - User support (number of rated movies)
  - Movie support (number of rating users)
  - Frequency of rating
  - Day of rating (number of days past since earliest rating)

# Gradient Boosted Decision Trees

- So… they don't actually describe in detail how GBDTs work in this paper
- So I had to look elsewhere
  - The rest of my explanation is largely based upon http://tullo.ch/articles/gradient-boosted-decision-trees-primer/

# GBDT - Background

- In a normal supervised learning problem, we take a labeled feature vector (x,y) and seek an estimation function F(x) that approximates the mapping F*
- Minimize expected loss function
- In gradient boosting, the model assumes an additive expansion
  - $F(x,\beta,\alpha) = \Sigma\ \beta_i h(x,\alpha_i)$
    - Where h is the weak learners and this is a linear combination of weak learners
  - $B_m$ is the weight a classifier has in the ensemble
  - Weights the training examples to compute the ith weak classifier

# GBDT Pseudocode

- Initialize list of weak learners to a singleton list with simple prior
- For each round in 1 to numRounds
  - Reweight examples (x,y) to (x,y') by upweighting examples that the existing forest poorly predicts
- Estimate new weak classifier $h_i$ on weighted examples
- Compute $\beta_i$ of the new weak classifier
- Add the pair ($h_i$, $\beta_i$) to the forest
- Return forest

# GBDT

- In gradient boosting, iteratively build a sequence of predictors, and the final predictor is a weighted average of these predictors
- At each step, focus on adding an incremental classifier that improves the performance of the entire ensemble
  - “Gradient descent in functional space”

# GBDT Results

- Using:
  - #trees = 200
  - Tree size = 20
  - Shrinkage = .18
  - Sampling rate = .9
- RMSE for 454 ensemble = .8603
- RMSE for 75 users = .8606

# GBDT Results

- Using
  - #trees = 150
  - Tree size = 20
  - Shrinkage = 0.2
  - Sampling rate = 1.0
- RMSE for the ensemble of 24 = .8664

# List of non- ensemble blended things

- Automatic Parameter Tuning
- Movie KNN
  - Correlations
  - KNNMovieV3
  - KNNMovieV3-2
- Time Dependence Models
- Restricted Boltzmann Machines
- Global Effects
- Global Time Effects
- Weekday Effect
- Integrated Model
- Maximum Margin Matrix Factorization
- NSVD
  - NSVD1
  - NSVD2
  - NSVD1 Discrete
- SBRAMF
- SVD++
- SVD-Time
- SVD with Adaptive User Factors
- SVD-AUF with Kernel Ridge Regression
- SVD Trained with Alternating Least Squares
- Rating Matrix Factorization – SVD
- Neighborhood Aware Matrix Factorization
- Regression on Similarity

# List of Blends Blended in to the Blend

- Linear Regression
- Polynomial Regression
- Binned Linear Regression
  - Support Based Bins
  - Date Based Bins
  - Frequency Based Bins
  - Clustering Based Bins
- Subset Generation
  - Forward Selection
  - Backward Selection
  - Probe-Quiz Difference Selection
- Neural Network Blending
- Ensemble Neural Network Blending
- Bagged Gradient Boosted Decision Tree
- KRR on a Probe Subset
- SVD Feature Predictor Extraction
- RBM Feature Predictor Extraction
- KNN Predictor Extraction

The End!