Team “Barneso” Report for the AUSDM Ensembling Challenge 2009

Jeremy Barnes JEREMY@BARNESO.COM

Abstract

Up to 1151 “black box” movie recommendation models were combined into an ensemble predictor. Significant success was achieved on the binary AUC task, using a deep neural network, a gated classifier and multiple logistic regression. Further improvement was achieved by adding hand-coded features, and by modelling the joint distribution of the movie models using a SVD and denoising auto-encoders.

On the large AUC task, the baseline[1] AOC[2] performance of 0.1635 was improved to 0.1461. On the more difficult medium task, the baseline performance of 0.3384 was improved to 0.3144, and on the small task, the already low baseline of 0.0597 was slightly improved to 0.0571.

Less success was achieved on the regression RMSE task. The best result, on the large task, reduced the baseline of 0.4419 to 0.4385[3].

The “black box” nature of the challenge and the underlying noise in the labels (to which the RMSE score is particularly sensitive) make progress difficult. An alternative framework for ensembling is discussed which, whilst placing more requirements on model builders, would likely lead to better improvement in the ensembles.

1. Introduction

The AUSDM ensembling challenge ran for approximately six weeks in October and November, 2009. The goal of the challenge was to combine existing personalised movie rating models into a more powerful ensemble predictor. The dataset was derived from work on the Netflix Prize (Netflix, 2007).

1.1. Netflix Prize

The goal of the Netflix Prize was to predict the rating (from 1 to 5) that a person would give to a film on a particular date, given a training dataset that contained examples of ratings that had already been made. These predictions were then compared to ratings collected from users to determine the efficiency of the predictions.

Three datasets were provided: A training dataset with 100 million (user, date, rating) triplets; a disjoint “probe” dataset with 1.4 million (user, date, rating) triplets that were not included in the training set, and a testing dataset with (user, date) pairs. The goal of the challenge was to predict the rating for each of these pairs.

The score was evaluated with the Root Mean Squared Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2} \tag{1}$$

where x_i is the user provided rating and $\hat{x}_i$ the model's prediction. This measure is quadratic in the magnitude of the errors, which makes it very sensitive to outliers and noise.

The leading entries in this challenge came from coalitions of collaborating teams. The teams would independently produce models that were trained only on the training set. These models would then be run to predict the values in both the probe and testing datasets. These predicted results would then be exchanged within the group of collaborators, and a final blended model would be produced, with the parameters for the blend learnt from the probe set.

Blending was necessary in order to achieve competitive performance, but most effort was expended in the component models.

[1]The average of the most accurate 20 models over a held-out set of 20% of the training set.

[2]AOC = 1 - AUC

[3]These RMSE scores are calculated on a ratings scale of [-1,1] not [1000,5000] as in the challenge, and should be multiplied by 2000 to be comparable.

Shameless plug: I'm currently looking for consulting work: Recommendation Engines, Machine Learning, Data Mining, Data Analysis, Computational Linguistics

Table 1. Problem sizes and row counts.

Size	Training	Testing	Models	Size
Small	15,000	15,000	200	15MB
Medium	25,000	25,000	250	25MB
Large	50,000	50,000	1151	300MB

Table 2. Label distribution for the large RMSE training set (50,000 samples). The mean label is 3.67. The Mean Model column gives the mean output over all models over all examples rated with the given label.

Label	Freq	Percent	Mean Model
1	2534	5.1	2.86
2	4877	9.8	3.09
3	12489	25.0	3.40
4	16422	32.8	3.76
5	13678	27.3	4.20

1.2. AUSDM Challenge

The AUSDM Challenge was designed to move the focus away from the component models and onto the blending algorithm. The organisers first approached the two leading coalitions from the Netflix Prize and obtained from them the probe set results of all of their models (some 1151 in total). From this large combined set of data, random sampling was used to produce twelve datasets, with three data sizes (Small, Medium and Large as described in table 1), two problem types (AUC and RMSE) and two datasets for each (a training set including target values, and a testing set with the target values removed).

No information about which movie, user or date a prediction applied was retained. As a result, the focus was entirely on the properties of the blending algorithms, as no side-channel information was available. This point will be discussed below.

1.2.1. RMSE Task

The RMSE task in the AUSDM Challenge was identical to that in the Netflix Prize: minimise the RMSE in equation 1.

Table 2 shows how the data is for this task is skewed towards the higher ratings, with far fewer 1 and 2 star ratings than the rest. As a result, the average model predictions are tightly clustered around the label mean of 3.67, and outliers are rare (the average model output for the 1 label is 2.86, nearly 2 stars away).

Table 3. AUC Task and correspondence between ±1 and star values, inferred from comparison of model means with RMSE dataset.

Size	-1	+1	AOC Top 20
Small	1	5	0.0597
Medium	2	3	0.3384
Large	2	4	0.1635

1.2.2. AUC Task

The AUC task was a binary ranking problem. Two ratings were selected (for example, 1 star and 5 stars), and rows with either one of these ratings were sampled. These two rows were then assigned the labels $+1$ and -1. The goal was to minimise the AUC score, which is a measure of the ability to separate the $+1$ from the -1 values via a real-valued confidence function. The AUC score is *linear* in the magnitude of the errors. It is possible to generate an AUC score from RMSE values.

Table 3 shows details of the three tasks and the inferred correspondence between the ±1 values and the number of stars. A baseline AOC score using the average RMSE of the 20 models with the highest AUC score is also provided[4] Due to the different selection of label values, the three tasks differ significantly in the baseline score and their potential for improvement over the baseline.

As a one-person team with limited time and labour, most effort was expended on this task. It is arguable that it represents better the real-world application of recommendation engines[5].

There should also be more improvement possible on the AUC task, as the cost of making an error is linear in the magnitude of the error, rather than quadratic as in the RMSE task. Larger errors are therefore less costly, and there should be some improvement possible simply by spreading the model predictions away from the mean.

[4]The AOC (area over the curve, AOC = 1 − AUC) was used so that the score could be interpreted as an error like the RMSE.

[5]For example, Netflix presumably wants to optimise the probability that someone who sees a recommendation rents the film (or rents the film and doesn't hate it): their revenue is increased by people renting more films (more active members have more profitable subscription levels and are less likely to let their subscription lapse).

2. Solution Strategy

Upon initial investigation of the problem, it became obvious that highly non-linear methods (boosting, decision trees, etc) were completely unsuitable to the task at hand due to their high sensitivity to noise[6]. Even the parameters for linear regression, one of the smoothest models possible, would vary wildly. These experiments led to the formulation of the following strategy:

1. Model the accuracy of the models over the different regions of the state-space;
2. Use a decomposition with an information bottleneck to reject noise and model the variation of models explicitly;
3. Add hand-coded smooth features derived from the model outputs to make model-building easier and to reject noise.
4. Reject noise as aggressively as possible:
   (a) Use ensembles of random samples of predictors;
   (b) Use noise-rejecting variants (such as ridge regression) wherever possible.
5. Produce multiple models with as much diversity as possible and merge them to produce the final result.

2.1. Modelling State-Space Accuracy

It is unlikely that ensembling algorithm would ever be sufficiently well informed to extrapolate outside the range spanned by its component models, especially in a black-box setting. A potential strategy is to interpolate between the models, weighting those that are more likely to be accurate *on a particular prediction* more heavily. A confidence function (a classifier for each model) can be used to provide these gating weights. Figure 1 illustrates this idea.

Several definitions of "accurate" were tried. For the AUC task, we said that a prediction of $1 \leq \hat{x} < 3$ was accurate for the -1 label, and that $3 \leq \hat{x} \leq 5$ for the $+1$ class. For RMSE, we tried to learn directly the error (difference between the label and the prediction) and a binary function of whether $|x - \hat{x}| < 1$ (whether the predicted value was within one star of the correct value). Crucially, each confidence function had the benefit of information about the other model's predictions in order to generate its value.

[6]It was rare that any of these techniques would even approach the baseline accuracy

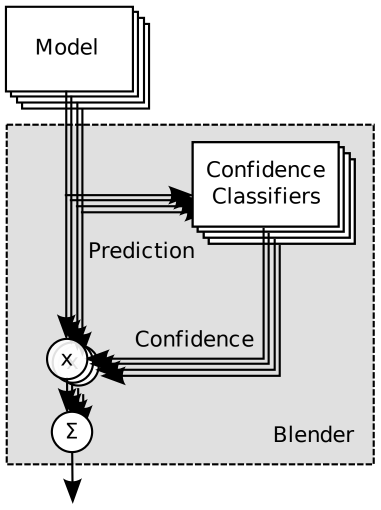


Figure 1. Gated Classifier Model. Note that *all* confidence classifiers receive the value of *all* models, not just the one for which it is generating a confidence value.

It proved to be extremely difficult to learn useful gating functions. The output of the function tended to be nearly identical over all input models, and thus the gating returned a value close to the average of the models.

2.2. Decompositions

By decomposition, we mean a way of reducing a set of 1151×50000 independent values (the outputs of all of the models over all examples) into a smaller dimensional space that preserves as much of the behaviour of the 1151 values[7].

These techniques are also known as "information bottleneck" methods or auto-encoders, tend to use some kind of a factorisation (explicit or implicit) to approximate a dataset with a large number of free parameters with a much smaller number of free parameters (for example, 50 or 200). They work by learning an (encoder, decoder) pair. The encoder reduces the 1151 dimensional input vector into a smaller internal representation. The decoder takes the encoded vector and reproduces the 1151 values as much as possible. A good encoder/decoder pair will do this without introducing much of an error. Frequently, the effect of the (encode, decode) sequence will be to remove noise from the input whilst keeping its essential characteristics. The encoded values can then be said to represent the deep structure of the data.

None of these techniques need to know the values of the

[7]In this section, we use numbers from the Large task for concreteness.

ratings: they model the joint distribution of the input models. They can be trained on an unlabelled set of data (for example, the testing set) to avoid biasing the training set.

2.2.1. Singular Value Decomposition

The simplest decomposition used was the SVD (Deerwester et al., 1990). It uses linear algebra to generate an optimal reduced rank representation of a series of models.

Applying the SVD to a matrix M breaks it down as follows:

$$M = U \Sigma V^T \tag{2}$$

where M (for the large contest) is the $50,000 \times 1151$ matrix of the model outputs, U is a $50,000 \times 1151$ matrix of left-singular orthonormal vectors, Σ is a 1151×1151 diagonal matrix with diagonal entries $[\sigma_1 \sigma_2 \dots \sigma_{1151}]$ and V is a 1151×1151 matrix of right-singular orthonormal vectors.

The σ values in Σ are all non-negative and are in decreasing order of magnitude. In order to reconstruct the best possible approximation of M of rank n, it is sufficient to set

$$M \approx \hat{M} = U_n \Sigma_N V_N^T \tag{3}$$

where $\hat{M}$ is a $50,000 \times 1151$ rank-n approximation to M, U_n is a $50,000 \times n$ matrix containing the first n columns of U, Σ_n is a $n \times n$ diagonal matrix containing the first n rows and columns of Σ, and V_n is a $1151 \times n$ matrix containing the first n columns of V. The error of any element in M is bounded by the magnitude of the excluded singular values $\sum_{i=n+1}^{1151} \sigma_i$.

This matrix can be used to reduce an 1151 element model vector $\mathbf{x}$ into a n-dimensional representation $\mathbf{z}$ which contains as much of the information in $\mathbf{x}$ as possible. Simply take

$$\mathbf{z} = \Sigma_n^{-1} V_n^T \mathbf{x}. \tag{4}$$

The features in $\mathbf{z}$ contain as much as possible of the information in $\mathbf{x}$, but in much fewer dimensions (typical values of n used in the challenge were 50, 100 and 200), and as a result have much of the noise removed. Even if the number of dimensions is not reduced, the $\mathbf{z}$ values tend to make better features for classification as they are orthogonal from each other.

We can also produce $\hat{\mathbf{x}}$, which is a reconstituted version of $\mathbf{x}$ produced from $\mathbf{z}$:

$$\hat{\mathbf{x}} = V_n \Sigma_n \mathbf{z} \tag{5}$$

and measure its error:

$$E = ||\mathbf{x} - \hat{\mathbf{x}}|| \tag{6}$$

If E is small, the information in $\mathbf{z}$ was sufficient to capture all of the information in $\mathbf{x}$. If E is large, it means that one or more of the dimensions excluded from the decomposition was important.

2.2.2. Denoising Auto-Encoder Decomposition

One problem with the singular value decomposition is that the the decomposition is entirely linear. A denoising auto-encoder (Vincent et al., 2008) can be used to generate a non-linear decomposition, which can approximate a much larger set of (non-linear) underlying phenomena.

The goal of an auto encoder is to learn the identity function: two functions $f(\cdot)$ and $g(\cdot)$ such that $f(g(\mathbf{x})) = \mathbf{x}$. Generally, in order to be interesting an auto-encoder will include some kind of restriction: for example, the number of dimensions in the range of f is smaller than the number of dimensions in its domain.

In neural networks, the most commonly used formulation shares an activation matrix $\mathbf{W}$ between the forward and reverse directions:

$$\mathbf{z} = f(\mathbf{x}) = t(W\mathbf{x} + \mathbf{b}) \tag{7}$$

$$\hat{\mathbf{x}} = g(\mathbf{z}) = t(W^T \mathbf{z} + \mathbf{c}) \tag{8}$$

When $t(\cdot)$ is a non-linear squashing function such as $t(\mathbf{x}) = \tanh(\mathbf{x})$, we can use back-propagation to learn a nonlinear decomposition.

Note that these auto-encoders can be stacked one on top of the other, so that we apply all of the forward functions one after the other and then all of the reverse directions in the opposite order.

In order to make the auto encoder generalise (and learn higher level features), we need to either a) add noise to the input (and train the auto encoder to remove the noise) and/or b) create an "information bottleneck", by reducing the number of neural network units lower down in the stack. This leads to the architecture used in the challenge, which is shown in figure 2:

In order to train the auto-encoders, a standard back-propagation algorithm can be used to perform gradient descent on the parameter space. Normally, a stack

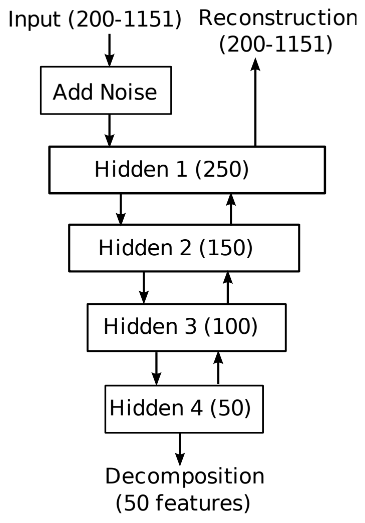


Figure 2. Denoising auto-encoder decomposition used in the challenge.

of auto encoders will be trained a layer at a time on the output of the previous layer (in a greedy manner), although it is possible to train the entire stack at once. Both methods were used in the challenge.

Improvements Several improvements were made to the auto-encoder model described previously.

Firstly, it is known that a linear neural network under the right conditions will approximate the singular value decomposition. It would make sense for the auto encoder described in the previous section to be able to do so. However, if we set t to the identity function and $W = \Sigma_n^{-1} V_n^T$ in 7 as in equation 4 (ignoring the bias terms), we get

$$\mathbf{z} = \Sigma_n^{-1} V_n^T \mathbf{x} \tag{9}$$

and so

$$\hat{\mathbf{x}} = V_n \left(\Sigma_n^{-1}\right)^T \Sigma_n^{-1} V_n^T \mathbf{x} = \Sigma_n^{-2} \mathbf{x} \tag{10}$$

where we rely on the fact that $V^T V = I$ due to V being orthonormal.

Thus, this auto-encoder can only reproduce its input, no matter the dimensionality, if all of the singular values of our data matrix are unitary, which is rarely true. The alternative of not including Σ_n in the encoder function is not satisfactory as this will cause the neurons corresponding to the high-valued singular values to dominate the training.

To rectify this problem, we added two extra terms to the decoding function 8:

$$\hat{\mathbf{x}} = t(D\mathbf{W}^T E\mathbf{z} + \mathbf{c}) \tag{11}$$

where D and E are diagonal matrices that control the input and output gain of the activation matrix W^T. Then, by setting $W = \Sigma_n^{-1} V_n^T$, $E = I$ and $D = \Sigma_n^{-2}$ we can achieve our goal of emulating the SVD.

A second improvement was made to the treatment of noisy inputs. In (Vincent et al., 2008), noisy inputs are simply set to zero. This causes problems with the auto encoder, as the same weight in W is simultaneously trying to reject noise, contribute to the hidden state and reproduce the output from the hidden state. Instead, as plenty of data was available, we used a separate activation matrix W_N for the noisy inputs. Those inputs which were chosen to be noisy were assumed to have an input value of 1 and connected via W_N instead of W to the hidden layer. This change significantly increased the accuracy of the reconstruction in the no-noise case[8].

The third improvement was made in the addition of noise. It was observed that the auto-encoders actually were better at reproducing the (noiseless) input when that input had noise added than when it was presented in a pristine state. This was because the auto-encoders were depending upon a certain number of their inputs having the value zero: if 20% noise is added and an internal state represents the mean of the inputs, then that state is going to be 125% the mean when no noise is present. In order to prevent the auto-encoders from expecting the noise to be present, every second example was presented with no noise added. Again, this change significantly increased the auto-encoder's reconstruction performance in the noiseless case.

2.3. Derived Features

In order to make it easier for the classifiers to work, the component models were augmented with several derived features:

- The minimum, maximum, mean and standard deviation of the model outputs;
- For of the 10 highest ranking models:
  - The minimum, maximum and mean value;
  - Variables comparing the spread of values over these 10 models and the spread of values over all of the models;
  - For each of the 10 models, the output of the model and the number of standard deviations

[8]It has not been proved that an auto-encoder needs to do a good job of reconstruction in order to provide a useful decomposition, but intuitively it seems necessary.

from the mean of all models;

  – The difference between the output of this model and the closest integer;
  – If a decomposition was used, the error of the reconstruction of this model by the decomposition;

- The output of the decomposition (SVD or Denoising Auto-Encoder), if there was any;

- The RMS error of the decomposition, as in equation 6.

2.4. Multiple Models

Highly non-linear algorithms such as decision trees, regression trees and especially meta-algorithms like Adaboost tend to have big problems dealing with noise. One way of mitigating this effect is by averaging models trained over random subsets of the data. The following techniques were used:

- Random decision trees and regression trees were used to train the final classifier in the gated models. The most successful model used 500 bags (with random selection of examples) of 10 iterations of boosted random decision trees (with a random subset of the features).

- Wherever regression (linear or logistic) was used, the regression was performed multiple (20-500) times and the average of the models taken. A random subset of examples and of features was chosen.

2.5. Ridge Regression

Ridge regression was used in place of linear regression in all circumstances, including within the Iteratively Reweighed Least Squares routines used to calculate the logistic regression coefficients (Komarek & Moore, 2005).

Ridge regression is a regularised form of linear regression, that penalises high weights in the model coefficients $\mathbf{x}$. The algorithm calculates the value of the vector $\mathbf{x}$ that minimises the following error:

$$E = ||W\mathbf{x} - \mathbf{b}|| + \lambda ||\mathbf{b}||^2 \leftarrow \text{Regularisation term} \quad (12)$$

The coefficient λ describes the trade-off between fitting the data and reducing the size of the parameters in $\mathbf{x}$. The optimal value of λ can be efficiently calculated using leave-one-out cross validation.

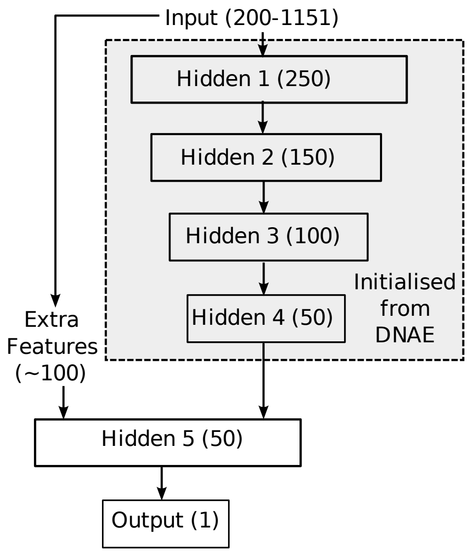


Figure 3. Deep Network Model.

Ridge regression also has the advantage of working well on rank-deficient or poorly conditioned problems, unlike standard linear regression. This is important in the context of this work: there are a lot of small, but not insignificant singular values in the data.

2.6. Deep Neural Network Model

The denoising auto-encoders are trained with no knowledge of the target feature. By adding an extra output layer or two and training the entire resulting network with back-propagation, the features that it has learnt in its internal representation can be fine-tuned to help produce a target output (here, the label for the AUC or RMSE model). Figure 3 illustrates the architecture.

3. Method

In order to generate results, a large number of predictors were trained implementing the ideas described above. Each of these predictors was trained on 80% of the training data, with the other 20% (always the same part) held out in order to train the final blending model. When each predictor was run, it created:

1. A blending results file containing the model's unbiased prediction for each entry of the 20% of final blending data held out;

2. A submission results file containing the model's prediction for each entry of the scoring set (for which labels weren't available).

Table 4. Denoising Auto-Encoder Decompositions. Layer Iter is the number of back-propagation iterations that each layer was trained by itself, Stack Iter is the number of back-propagation iterations that the entire stack was trained after each layer was added. Bug indicates the presence of a bug described in section 3.1.

Model	Layer Iter	Stack Iter	Bug
DNAE1	500	500	√
DNAE2	800	0	
DNAE3	400	200	

Once all of the files were available, they were combined using a final blending stage, the result of which was the submitted result.

3.1. Decompositions

A total of four decompositions were generated: one SVD decomposition and three denoising auto-encoder decompositions. Table 4 shows the parameters used. In each case, the layer sizes are as indicated in 2. Learning rates were set via manual tuning. On the small dataset, 80% of the examples were randomly selected on each iteration; 50% on the medium and 10% on the large. The presentation order of examples was random. The bug referred to in the table when training DNAE1 was the use of the non-noisy input vector when back-propagating, leading to noisy inputs being incorrectly updated.

3.2. Multiple Regression

The multiple regression predictors turned out to be the most powerful, particularly in the RMSE task. This is largely due to their ability to reject noise due to their inherent smoothness, the regularisation provided by ridge regression and the smoothing provided by the random selection of features and examples.

Table 5 describes the parameters for the different models. In all cases, 500 separate regression models were combined (linear regression for the RMSE task; logistic regression for the AUC task) on 6,000 randomly selected examples. On the small task, the number of features sampled and the decomposition order were set to 100; for the medium task 150 and for the large task 200. When extra features were used, they were sampled along with the model outputs.

3.3. Deep Neural Networks

Table 6 describes the deep network models used. Each of these had an architecture with 250, 150, 100 and 50 units (from the denoising auto-encoders) and another 50 hidden units feeding into the single unit output layer. The extra features are not fed in the top, but directly into the 50 unit hidden layer, bypassing the auto-encoder. Standard *tanh* units were used.

Table 5. Multiple Regression Models Used

Model	Decomposition	Extra Features
MR1		
MR2	SVD	
MR3	SVD	√
MR4	DNAE1	
MR5	DNAE1	√
MR6	DNAE2	
MR7	DNAE2	√
MR8	DNAE3	
MR9	DNAE3	√

Table 6. Deep Network Models Used

Model	Decomposition	Extra Features
DN1	DNAE2	√
DN2	DNAE2	
DN3	DNAE3	
DN4	DNAE3	√

3.4. Gated Merger

Several models of the "gated" merger described in 2.1 were tried. This model tended to perform reasonably well for the AUC task, but poorly for the RMSE task. Presumably, this is because the two-stage nature of the model caused the noise to be amplified between the stages.

The models differed in which decomposition they used (no decomposition, the SVD or the denoising auto-encoders), whether or not they used extra features, and the technique used for the final score once the confidence-modified values had been produced. Table 7 shows these parameters.

3.5. Classifier Models

For the RMSE data, two classifier models were used. One, `rtrees` used a random forest of 200 regression trees. The other, `mclass`, learnt a binary classifier for several discrete movie ratings (1, 2, 3, 4 or 5 stars; 2-5 stars; 3-5 stars; 4-5 stars) using a random forest of 5000 decision trees, and combined these predictions using linear regression. Neither model performed particularly well.

Table 7. Gated Merger Models Used

Model	Decomp.	Blend RMSE	Blend AUC
gated	SVD(200)	LR	Rand Forest
gated2		LR	Rand Forest
gated3	DNAE1	LR	Rand Forest
gated4	DNAE2	LR	Rand Forest
gated5	DNAE3	LR	Rand Forest
gated6	DNAE3	Rand Forest	N/A
gated7	DNAE3	Multi LR	N/A

3.6. Final Blending

Final blending was performed using a cross-validation training on the 20% held out data. The held out data was broken into 10 folds, and 10 different multiple regression blenders were trained, each training on 9 and leaving out one fold. The performance of the merged model on the entire 20% held out was then evaluated. The challenge submission files were generated by running the 10 multiple regression blenders over the entire testing set, and averaging the results.

This strategy was adapted in order to reduce the impact that a "rogue" model (with a high error rate, or over-fit on its training data) would have on the final blend. It is unlikely that performing (yet another) round of blending of already blended models would significantly improve the results; this bootstrapping with no additional leverage. Uniform linear blending appeared to work just as well.

3.7. Implementation

In practise, this challenge turned out to be as much about software engineering as about data mining.

The biggest reason for this was the amount of noise in the data. This necessitated that models be run many times (up to 500) and the results averaged, with a corresponding increase in compute time.

Due to the limited hardware resources available[9] and the large number of tasks, it was necessary that the software be both memory and CPU efficient. The entire code was vectorised to take advantage of the vector unit, multi-threaded[10] and the bottlenecks were profiled and carefully optimised. Single precision arithmetic was used wherever possible[11] due to its two-fold advantage in execution speed on modern hardware.

To save memory bandwidth, parameters were stored using as small a precision as possible and care was taken not to duplicate memory when splitting datasets into training and validation sets.

The fact that there were six different tasks (AUC and RMSE for the small, medium and large datasets) also increased the amount of CPU time and engineering work required, especially in manually tuning the back-propagation parameters.

In the end, about 5,000 lines of C++ code were written for the challenge directly, and about 10,000 lines added to the underlying machine learning library (primarily the code to perform Ridge Regression and the denoising auto-encoder routines). The entire set of results could be reproduced in about 24 hours on a consumer desktop PC.

The software was developed on Linux. The only significant external libraries used were LAPACK and BLAS for the linear algebra routines.

3.8. Open Source

The source code for this submission is available. The machine learning library used to perform the heavy lifting is available at http://bitbucket.org/jeremy_barnes/jml/. The source code of the actual AUSDM submission is available at http://github.com/jeremybarnes/ausdm. Both are available under the Affero GNU Public License version 3. The `ausdm` repository also contains some of the data files used in the building of the results.

4. Results

4.1. Diversity and Independence of Model Predictions

Table 8 shows the distribution of singular values over the six tasks. The top part lists values of the singular values; the bottom part lists counts of various categories.

The spread of the singular values give an idea of the diversity of the models in the data: the small singular values correspond to models that don't contain much more information over and above those with higher singular values. The small task appears to have little redundancy in the provided models, whereas the large task has significant redundancy, and potentially even

[9]One quad core "hyper-threaded" (8 virtual cores) desktop machine with 6GB of RAM, one dual core laptop with 2GB of RAM

[10]On the desktop machine, 8 threads were run to fully exploit the hyper-threaded processor

[11]It is frequently not possible. For example, whenever accumulating a series of numbers it is necessary to accumulate in double precision even if the numbers being accumulated are only in single precision.

Table 8. Independence and Conditioning of Models. The singular values of each data matrix were taken. The top half lists values (highest, second highest and lowest); the bottom half shows a histogram over orders of magnitude.

	Small		Medium		Large	
Type	AUC	RMS	AUC	RMS	AUC	RMS
Top	936	806	674	1040	2830	3538
2nd	69	55	83	77	257	253
min	0.9	0.9	0.4	0.8	10^{-5}	10^{-5}
> 100	1	1	1	1	9	9
> 10	72	59	88	87	433	428
> 1	199	197	247	249	1071	1074
> 0.1	200	200	250	250	1143	1143
≤ 0.1	0	0	0	0	8	8

models that were (pre-blended) linear combinations of others.

4.2. Decompositions

Table 9 shows the reconstruction accuracy of the decompositions of different orders over the training sets. The reconstruction accuracy increases as we move from DNAE1 to DNAE2 to DNAE3, but that they aren't as efficient at reproducing the data as the SVD. This is a disappointing result: the non-linearities were either not being exploited or were not useful. DNAE3 is particularly interesting, as it shows the effect of training the stack as a whole rather than each layer individually. Doing so reduces the efficiency of the individual layers as separate auto-encoders, but to improves the entire stack.

Looking at the results of the `gated` (SVD decomposition), `gated3` (DNAE1 decomposition), `gated4` (DNAE2 decomposition) and `gated4` (DNAE3 decomposition) algorithms (which differ only in the decomposition used), it appears that the SVD decomposition is the most useful, followed by the DNAE3, DNAE2 and DNAE1 decompositions. These results are disappointing. It is possible that allowing interactions between the features (as in (Larochelle et al., 2009)) would improve matters, but as they stand these results would have to be considered a failure.

4.3. AUC Results

We present the AUC results in table 10, showing the performance of both the component models and the blended result. The table for each task contains two columns of numerical information. The first describes the error score of the model, with the lift (reduction in error) as compared with the baseline model, multiplied by 1000. The second shows the average blending weight of the model (over 10 folds) as well as the standard deviation in this value. The blending weights give an idea of the importance of the model to the final result.

Table 9. Decomposition reconstruction accuracies. These are the total RMSE over all inputs for different decompositions as the order (dimensionality) of the decomposition varies.

Set	Order	SVD	DAE1	DAE2	DAE3
	250	0.00	0.86	0.81	6.51
S	150	0.20	1.48	1.15	3.51
AUC	100	0.44	1.99	1.30	1.92
	50	0.72	2.22	1.40	0.92
	250	0.00	0.62	0.62	3.58
M	150	0.30	1.09	1.02	3.44
AUC	100	0.48	1.52	1.42	2.87
	50	0.74	1.97	1.80	0.90
	250	1.00	2.08	2.18	17
L	150	1.31	3.24	3.05	4.88
AUC	100	1.52	4.18	3.74	2.98
	50	1.84	5.01	4.34	2.08
	250	0.99	2.31	2.51	23
L	150	1.30	3.58	3.42	5.52
RMSE	100	1.51	4.56	4.08	2.84
	50	1.82	5.38	4.49	2.08

The multiple regression models appear to be the most consistently accurate, followed by the gated models. The deep network models appeared to be used mostly to attenuate noise, as their blending weights were more negative than positive. The greatest lift was obtained on the medium task (which was also the hardest, and consequently had the most scope for improvement). A significant improvement was also obtained on the large task. The lift on the small task was small, but (as will be discussed below) there was not much scope for improvement due to the noise ceiling.

It is instructive to compare the multiple regression models to determine the effect of the various strategies. Recall from table 5 that `mr1` contained only the component models, `mr2` augmented this with an SVD and `mr3` with the extra features. Any improvement in `mr1` is therefore attributable to the calibration of the component scores from the RMSE (quadratic error) task to the AUC (linear error) task.

It seems that most of the improvement is due to the extra features, as `mr1` and `mr2` are not significantly different, but `mr3` is always significantly better than the others. On the other hand, it appears from the rest of the `mr` results that, especially on the large task, the

Table 10. AUC Blending Results. The lift is 1000 times the improvement over the baseline score. The weight values, which describe the blending weight in the final model, are reported as mean ± standard deviation.

Task	Small			Medium			Large		
Model	AOC	Lift	Weight	AOC	Lift	Weight	AOC	Lift	Weight
dn1	0.0585	1.2	-0.36±0.32	0.3299	8.5	0.04±0.23	0.1581	5.4	-0.41±0.20
dn2	0.0611	-1.4	0.20±0.19	0.3431	-4.7	-0.51±0.22	0.1634	0.1	-0.05±0.21
dn3	0.0611	-1.4	0.19±0.17	0.3432	-4.8	-0.49±0.19	0.1634	0.1	-0.60±0.23
dn4	0.0585	1.2	-0.34±0.30	0.3300	8.4	0.05±0.24	0.1581	5.4	-0.83±0.31
gated	0.0565	3.2	0.80±0.23	0.3239	14.5	0.60±0.25	0.1497	13.8	1.09±0.27
gated2	0.0584	1.3	0.26±0.24	0.3318	6.6	-0.08±0.27	0.1528	10.7	0.32±0.24
gated3	0.0592	0.5	-0.55±0.35	0.3282	10.2	0.09±0.29	0.1524	11.1	0.19±0.32
gated4	0.0586	1.1	-0.02±0.29	0.3303	8.1	-0.26±0.44	0.1528	10.7	-0.64±0.33
gated5	0.0580	1.7	0.37±0.14	0.3264	12.0	0.11±0.31	0.1519	11.6	0.32±0.13
mr1	0.0639	-4.2	0.23±0.33	0.3208	17.6	0.47±0.47	0.1581	5.4	-0.08±0.39
mr2	0.0633	-3.6	0.18±0.33	0.3203	18.1	0.46±0.49	0.1584	5.1	-0.88±0.30
mr3	0.0575	2.2	0.57±0.51	0.3174	21.0	0.03±0.51	0.1499	13.6	1.84±0.69
mr4	0.0573	2.4	0.39±0.27	0.3196	18.8	0.82±0.34	0.1508	12.7	-0.75±0.38
mr5	0.0574	2.3	-0.29±0.29	0.3128	25.6	0.80±0.44	0.1485	15.0	1.58±0.47
mr6	0.0573	2.4	0.51±0.30	0.3201	18.3	0.61±0.22	0.1505	13.0	0.13±0.29
mr7	0.0573	2.4	-0.21±0.25	0.3135	24.9	0.65±0.42	0.1485	15.0	0.74±0.43
mr8	0.0572	2.5	0.90±0.60	0.3186	19.8	-0.06±0.30	0.1500	13.5	2.04±0.57
mr9	0.0574	2.3	-0.04±0.16	0.3146	23.8	0.24±0.24	0.1484	15.1	0.58±0.66
combined	0.0571	2.6		0.3144	24.0		0.1461	17.4	

DNAE models provide a useful substitute for the extra features, whereas the SVD does not. Or in other words, the DNAE decompositions manage to implicitly capture most of the information in the extra features whereas the SVD does not.

The deep network models (`dn1` to `dn4`) did not perform well, but were improved by adding the extra features.

In all cases, the `gated` model worked better than `gated2` through `gated4`. As these models always had extra features available, this shows that the SVD decomposition is more useful than the DNAE decompositions.

Overall, the AUC results successfully provided a modest lift. It was particularly encouraging to see that the merged model on the large dataset was significantly more accurate than any of its component predictors.

4.3.1. Noise

The small AUC task was particularly noisy, and only a small improvement seems possible. One explanation for this would be the selection of target values for this task, which are ($-1 \rightarrow 1$) and ($+1 \rightarrow 5$). These two values, and particularly the value 1, are associated with extreme emotional reactions to films by users, much of which cannot realistically be modelled[12]. The relative scarcity of 1 rankings in the dataset also makes them more susceptible to noise (accidental mouse clicks, distraction, cats walking on keyboards, etc), which are probably uniformly distributed over the dataset. When adapting to the large and particularly the medium task at the end of the challenge, it was observed that noise was much less of a problem.

4.4. RMSE Results

The RMSE task results presented in table 11 are not encouraging. The deep network results were all very poor and the gated results only rarely beat the baseline. The multiple regression results were nearly uniform which means that the effect of the extra features and the decompositions was minimal. The final lift obtained is far too small to be detectable by a user of such a system. The inescapable conclusion is that we failed to achieve a significant improvement in the RMSE task.

Unlike the AUC task, final blending was ineffectual on the RMSE task: the score of the blended result was slightly worse than the best individual result[13]. This is probably due to there not being enough diversity in

[12]How is a computer to know that this was the favourite film of a hated ex and is thus terrible by association?

[13]The submitted results were still the blended ones, however, as they should be more resistant to the selection bias in the validation set

Table 11. RMSE Blending Results. The lift is 1000 times the improvement over the baseline score. The weight values, which describe the blending weight in the final model, are reported as mean ± standard deviation.

Task	Small			Medium			Large		
Model	RMSE	Lift	Weight	RMSE	Lift	Weight	RMSE	Lift	Weight
dn1	0.4422	-2.4	0.02±0.01	0.4332	-2.5	0.04±0.02	0.4442	-2.3	-0.06±0.03
dn2	0.4485	-8.7	0.02±0.03	0.4407	-10.0	0.03±0.02	0.4495	-7.6	-0.02±0.04
dn3	0.4486	-8.8	0.02±0.04	0.4409	-10.2	0.03±0.03	0.4497	-7.8	0.02±0.07
dn4	0.4421	-2.3	0.02±0.01	0.4332	-2.5	0.03±0.02	0.4442	-2.3	-0.06±0.04
gated	0.4500	-10.2	0.07±0.04	0.4462	-15.5	0.02±0.04	0.4418	0.1	0.09±0.04
gated2	0.4409	-1.1	0.08±0.04	0.4315	-0.8	0.07±0.04	0.4414	0.5	0.09±0.04
gated3	0.4554	-15.6	-0.04±0.03	0.4420	-11.3	0.02±0.04	0.4428	-0.9	0.05±0.04
gated4	0.4490	-9.2	0.03±0.03	0.4393	-8.6	0.00±0.03	0.4419	0.0	0.04±0.05
gated5	0.4620	-22.2	0.03±0.03	0.4487	-18.0	-0.04±0.04	0.4460	-4.1	0.01±0.02
gated6	0.4421	-2.3	0.03±0.04	0.4317	-1.0	0.03±0.04	0.4421	-0.2	0.02±0.05
gated7	0.4394	0.4	0.04±0.02	0.4295	1.2	0.04±0.03	0.4405	1.4	-0.05±0.10
mr1	0.4379	1.9	0.07±0.02	0.4277	3.0	0.10±0.02	0.4386	3.3	0.13±0.04
mr2	0.4377	2.1	0.08±0.01	0.4275	3.2	0.10±0.04	0.4387	3.2	0.10±0.02
mr3	0.4377	2.1	0.07±0.02	0.4276	3.1	0.09±0.03	0.4386	3.3	0.11±0.03
mr4	0.4382	1.6	0.06±0.03	0.4278	2.9	0.08±0.03	0.4386	3.3	0.10±0.06
mr5	0.4379	1.9	0.06±0.02	0.4277	3.0	0.08±0.04	0.4386	3.3	0.10±0.06
mr6	0.4379	1.9	0.07±0.02	0.4277	3.0	0.09±0.03	0.4386	3.3	0.11±0.02
mr7	0.4377	2.1	0.07±0.02	0.4278	2.9	0.06±0.04	0.4385	3.4	0.15±0.09
mr8	0.4377	2.1	0.07±0.02	0.4275	3.2	0.10±0.04	0.4389	3.0	0.07±0.02
mr9	0.4379	1.9	0.06±0.01	0.4278	2.9	0.05±0.05	0.4389	3.0	0.05±0.03
mclass	0.4398	0.0	0.06±0.05	0.4319	-1.2	0.02±0.03	0.4417	0.2	-0.00±0.04
rtrees	0.4411	-1.3	0.05±0.05	0.4321	-1.4	0.02±0.04	0.4427	-0.8	-0.05±0.05
combined	0.4381	1.7		0.4277	3.0		0.4385	3.4	

the models blended: there were only a few really distinct models with reasonable performance, and these frequently used the same features (from the DNAE, the SVD and the derived features).

4.4.1. Criticism of quadratic metrics on noisy data

This phenomena is probably explained by the use of a RMSE metric, and the amount of noise in the data. The RMSE metric penalises very heavily incorrect predictions at the extreme ends of the scale: a prediction of 1 has 4 times the potential MSE error (16 points) than a prediction of 3 (4 points). As the noise in the data increases, it becomes more and more costly to deviate significantly from the 3 prediction.

Assuming that random noise is added equally to each label, label 1 already becomes difficult to predict. If we also consider that label 1 would most likely be used by people for emotional reactions to films (likely, some proportion of the 1 ratings are due to vitriol), they become even more difficult to model.

Row Breakdown It is instructive to break the dataset into different types of rows, based upon the ease with which a prediction can be made:

- *easy*: all models are within one star of the correct answer;
- *possible*: at least one model is within one star of the correct answer;
- *impossible*: no model is within one star of the correct answer[14].

Table 12 shows the result of this breakdown on the large RMSE dataset, and the contribution of the different row types to the total MSE for the baseline model. Considering that it is very unlikely that any improvement could be made on these positions, it is still very important to pick a good middle value for them, as they account for 1/3 of the error.

The upshot is that it is nearly impossible to make progress on the RMSE metric, as the noise on the outliers is amplified significantly by the RMSE metric. It

[14]It is could happen that the impossible values be clustered on both sides of the correct answer and that their mean be a good predictor, but this was not observed in the data.

Table 12. Large RMSE training set broken down by row difficulty. In the last column, we see that 1/3 of the MSE comes from the impossible positions, which account for just 5.7% of the data.

Category	Freq	Avg MSE	Total MSE
Easy	0.329	0.019	0.0063
Possible	0.614	0.198	0.1216
Impossible	0.057	1.189	0.0678
Total	1.000	0.195	0.1953

would be more useful to either a) use a linear metric such as the mean error, or b) remove the impossible entries from the evaluation dataset.

4.5. Cracking Open the Black Box

There is absolutely zero side-channel information obtainable for this task[15]. Even information that could normally be used to determine the accuracy of the underlying models (such as the amount of information about the given movie and user in the training data) was not available. This information would be useful to the blender, and it is worth considering how it could be provided.

In addition, every model made a prediction for every data point, irregardless of whether or not that prediction was likely to be useful. In other words, the models were designed to maximise recall.

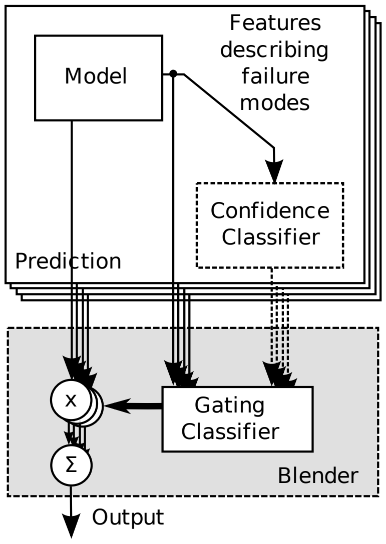
The author's previous work on ensembling in computational linguistics has shown that this strategy is suboptimal: precision is far more important than recall, and it is better for a model to be highly accurate on a tiny subset of reliably identified examples than be mediocre on many.

One way to allow for a precision/recall trade-off to be made is for models to provide both a prediction and a confidence in that prediction. The confidence gives the probability that the prediction is correct (for example, the probability that the output of the model is within one star of the correct rating). The blender can then improve the precision of a given model by thresholding on this confidence value.

The features provided by each model for the confidence function need to provide information about the failure modes of the algorithm. For example, a statistical model might perform poorly when there is little data available about the user; in this case, the amount

[15]Contrast with the Netflix Prize where the identities of the films were known, which allowed the possibility to obtain further information about the film from the Internet.

Figure 4. Ensembling with confidence information.



of data available would be provided to the confidence classifier. An iterative algorithm would provide the number of iterations required to reach convergance, and so on. There is not necessarily more work to create a (model, confidence function) pair like this: the algorithm is allowed to make really bad predictions, so long as its confidence function can predict them. Algorithms can become more focused on modelling exactly one phenomenon, instead of the Swiss Army Knife that is necessary when no confidence is provided. If the selection of algorithms is significantly diverse, it will be possible to predict most examples using mostly information from accurate models *for that example.*

Figure 4 shows one way to implement such a scheme using a gating function. The confidence classifiers could either be part of the black box, trained externally from the features, or learnt implicitly as part of the gating function (which would receive only the features).

4.6. Deep Networks

The final submission for the small AUC task that is presented in table 10 is not in fact the best results that were obtained for this task. During development of the denoising auto-encoders, an initial auto-encoder was produced that, when trained into a deep network using supervised back-propagation, produced results significantly better than these. However, due to numerical issues and thread-order non-determinism, these results could not be duplicated nor even closely matched. There are two possible explanations. The first is that, in speeding up the code to run at an acceptable speed for the large task, an error was introduced (nearly 2,000 lines of test cases that tested a lot of the invariants in the system). The second is that these methods are very sensitive and one needs to train many models to fall on a good one[16].

A large amount of effort was also expended to improve the speed of back-propagation, which was were most of the time was spent in training the denoising auto-encoders and deep network models. The second-order methods indicated in (LeCun et al., 1998) were implemented; however they tended to be over-enthusiastic about the learning rate, leading to divergence or oscillation. Even the simpler methods to generate an overall learning rate failed, and it was necessary to fall back onto manual tuning of parameters.

Perhaps the most important conclusion is that these models are difficult to get right in practise, and would require much experience to use successfully: especially when moved out of the image domain (where most of the successful work has come from) where it is simple to visualise what has been learnt and verify that the models make sense.

5. Conclusion

The AUC task proved to be rich and enjoyable, and a significant improvement was obtained on this task, especially on the medium and large datasets. Models based upon gating of the input models and multiple regressions were successfully used. Further improvement was achieved by hand-coding features.

The use of unsupervised decompositions to model the joint distribution of the input variables led to some success. Both a linear SVD decomposition and non-linear denoising auto-encoder decompositions were tried. The denoising auto-encoder decompositions however did not end up providing good pre-initialisation for deep neural networks, except for on one model which could not be reproduced. These models appear to be difficult to use well and more experience would have been necessary to use them effectively.

No significant improvements were made on the RMSE task, due to the interplay of a skewed dataset, the presence of noise and the quadratic nature of the RMSE metric. A less severe metric should be adopted or noise removed from the dataset. Judging from the leader board on the small task, it is unlikely that any team managed to achieve a significant improvement over the baseline.

[16]Of course, one can make one's own luck. This is why practitioners of deep networks suggest to make them both wide and deep: for each useful generalisation, there is likely to be a neuron somewhere within a wide enough network that will learn it. Unfortunately, the computational resources were not available to train significantly wider networks than the ones described here.

An improved model of ensembling was proposed, whereby each model in the ensemble provides not only a prediction of the target function, but a list of features that can be used to determine when the model is likely to be inaccurate. This model requires further work on the part of the ensemble builders to provide this information, but would allow the ensembling method to have significantly more leverage.

Acknowledgements

Thanks to Phil Brierley for organising and running the challenge.

The LaTeX style file for this report is based upon that for ICML 2009, by P Langley, Terran Lane, Jennifer Dy, Kristian Kersting and Ricardo Silva.

References

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., & Harshman, R. A. (1990). Indexing by latent semantic analysis. *JASIS*, *41*, 391–407.

Komarek, P., & Moore, A. (2005). *Making logistic regression a core data mining tool: A practical investigation of accuracy, speed, and simplicity* (Technical Report CMU-RI-TR-05-27). Robotics Institute, Pittsburgh, PA.

Larochelle, H., Erhan, D., & Vincent, P. (2009). Deep learning using robust interdependent codes. *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)* (pp. 312–319). Clearwater (Florida), USA.

LeCun, Y., Bottou, L., Orr, G., & Muller, K. (1998). Efficient backprop. *Neural Networks: Tricks of the trade*. Springer.

Netflix (2007). Netflix prize web site. Available at http://www.netflixprize.com/; accessed 23 November, 2009.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). *Extracting and composing robust features with denoising autoencoders* (Technical Report 1316). Département d'Informatique et Recherche Opérationnelle, Université de Montréal.