AusDM 09

Analytic Challenge

‘Ensembling’

www.tiberius.biz/ausdm09

This document contains the challenge details, results and selected contestant reports.

Sponsored by

Tiberius

data mining

AusDM 2009

Analytic Challenge

The Challenge

In Brief:

You have asked several experts to independently build you some mathematical models that will predict how your customers would rate particular movies. All the experts, who were given the same information on how previous movies were rated, provided predictions with very similar accuracies. While trying to determine which experts predictive methodology to use in your recommender system, you noticed that by simply averaging the predicted ratings of each expert, you ended up with a more accurate solution than any particular individual expert could produce alone.

As predictive accuracy is paramount, you now want to determine if there is a better way of combining the predictions than simple averaging.

The challenge is to find the most accurate method of combining your experts.

More Detail:

Ensembling, *Blending*, *Committee of Experts* are various terms used for the process of improving predictive accuracy by combining models built with different algorithms, or the same algorithm but with different parameter settings. It is a technique frequently used to win predictive modelling competitions, but how it is actually achieved in practice maybe somewhat arbitrary.

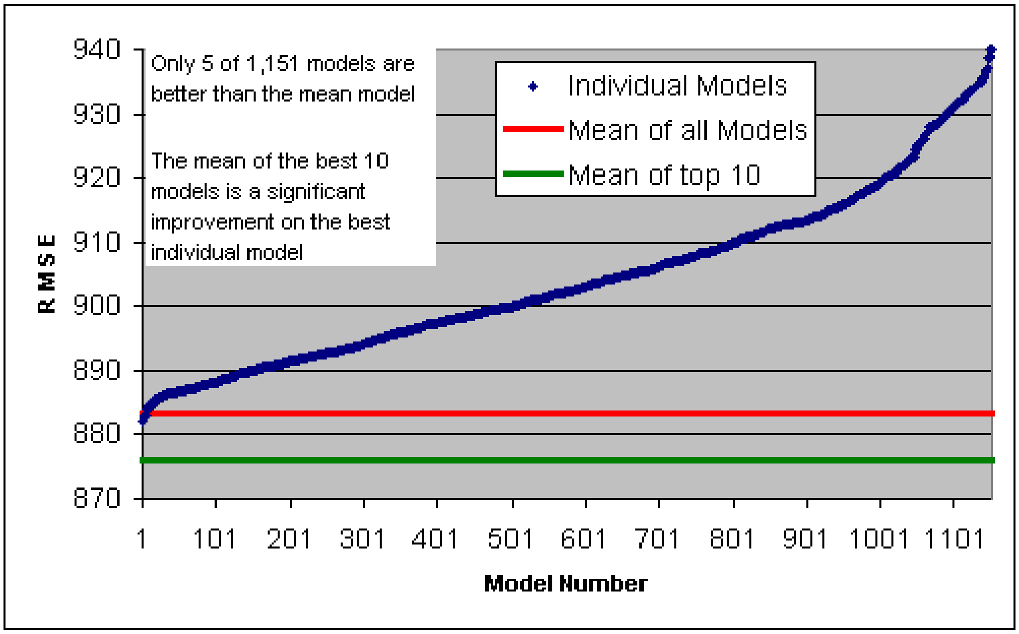
One of the drawbacks in researching the problem is that you first have to generate a lot of models before you can even start. There have been numerous predictive modelling competitions that could potentially provide good data sets for such research - many models built by many experts using many techniques.

The Netflix Prize is one such competition that has been on going for nearly 3 years. It recently finished, and the eventual winners were an amalgamation of several teams that somehow combined their individual model predictions.

Over 1,000 sets of predictions have been provided by the two leading teams (who actually ended up with the same score), The Ensemble and BellKor's Pragmatic Chaos. The primary goal of this challenge is to stimulate research into 'Ensembling' techniques... but also answer the question many of us want to know - *how low can Netflix go?*

By way of an example, the chart below shows the individual RMSE of over 1,000 models provided. Taking the mean prediction over all these models is only slightly worse than the best individual model. The mean of the best 10 models is significantly better than any individual model.

The purpose of this challenge is to somehow combine the individual models to give the best overall model performance.



Only 5 of 1,151 models are better than the mean model

The mean of the best 10 models is a significant improvement on the best individual model

The Data

The data provided for this challenge comes from 1,151 sets of predictions of integer ratings in the range (1-5). Each set of predictions was generated by mathematical models with the objective (generally) to minimise the 'root of the mean squared error' (RMSE) over the full set of over 1.4 million ratings. To those familar with the Netflix Prize, each of these sets of predictions would be what is known as a probe file, and we collected 1,151 probe files altogether.

All the individual probe files were merged into one large file (1,151 columns, 1.4 million rows). Form this file, random sampling was used to generate 6 data sets for the challenge. Each of these data sets is split into 2 files, one for Training (Target/Rating provided) and one for Scoring (Target/Rating withheld). We will refer to the Rating as the 'Target' - the thing we are trying to predict.

There are 2 small, 2 large and 2 medium data sets. You can use the small data sets for developing your techniques. Your predictions of the Targets for the small data set Scoring files can be uploaded and and feed back will be given on the performance. It is the predictions of the Targets in the medium and large data set Scoring files that will be used to determine the winner - no feedback is available for these.

There are 2 tasks, one to develop a method to predict a continuous value and the other to predict a binary value. The metric used to detemine accuracy are different for each task.

In each of the data sets, the Training and Scoring files are of the same size:

RMSE Small - 200 sets of predictions for 15,000 ratings. The objective is to minimise the RMSE.

AUC Small - 200 sets of prediction for 15,000 ratings. Only ratings of 2 particular values are included (eg 1s and 5s), sampled with different weightings on each value. The objective is to minimise the AUC.

RMSE Large - 1,151 sets of predictions for 50,000 ratings. The objective is to minimise the RMSE.

AUC Large - 1,151 sets of predictions for 50,000 ratings. Again, these only include ratings of 2 particular values that may or may not be the same values used in the small set. The objective is to minimise the AUC.

RMSE Medium - 250 sets of predictions for 20,000 ratings. The objective is to minimise the RMSE.

AUC Medium - 250 sets of predictions for 20,000 ratings. Again, these only include ratings of 2 particular values that may or may not be the same values used in the small/large set. The objective is to minimise the AUC.

The data sets are csv files with header rows. The first 2 columns are rowID and Target (actual rating) and then either 1,151 or 200 columns of predicted rating values. In the Scoring files the Target values are zeros.

For both the RMSE and AUC tasks, the predicted ratings values have been converted to integers by rounding to 3 decimal places and multiplying by 1,000. Thus each predicted rating will be an integer and should lie in the range 1,000 to 5,000.

For the RMSE tasks, the same transformation was applied to the actual (Target) rating. Your supplied predictions should also be on this scale (ie the scale 1,000 - 5,000) but can also have decimal places.

For the AUC tasks, the Target values are 1 and -1 (zeros in the Scoring files). The actual values for your supplied predictions is not important, it is the rank order that matters.

The solution files should be a text file with no header row and 1 number per row (ie 15,000 or 50,000 rows). They should be sorted in order of rowID but rowID should not be included in the file.

-

Evaluation and Prizes

You can submit entries for the AUC, RMSE or both tasks - there will be winners in each category (only on the large and medium data sets). You can submit multiple times, but your last submission will be the one that counts.

To be in contention to win, you must also provide a report (pdf format) outlining your methods. This can be as long or short as you like but should provide enough detail to give others a reasonable idea about how you approached the problem. If you submit entries for both tasks then only 1 report is required. Please include in the report your team name, team members and any affiliation. These reports will be made available on this site.

There will be also be a 'Grand Champion'. This will go to the team with the lowest average rank over both the AUC and RMSE submissions (where only the ranks of teams who have entered both tasks for both large and medium data sets are taken into account, with the large having twice the weighting of the medium). In the event of a tie, only the large rankings will be considered, if still a tie then the team with the lowest error on the large AUC problem will be declared the winner.

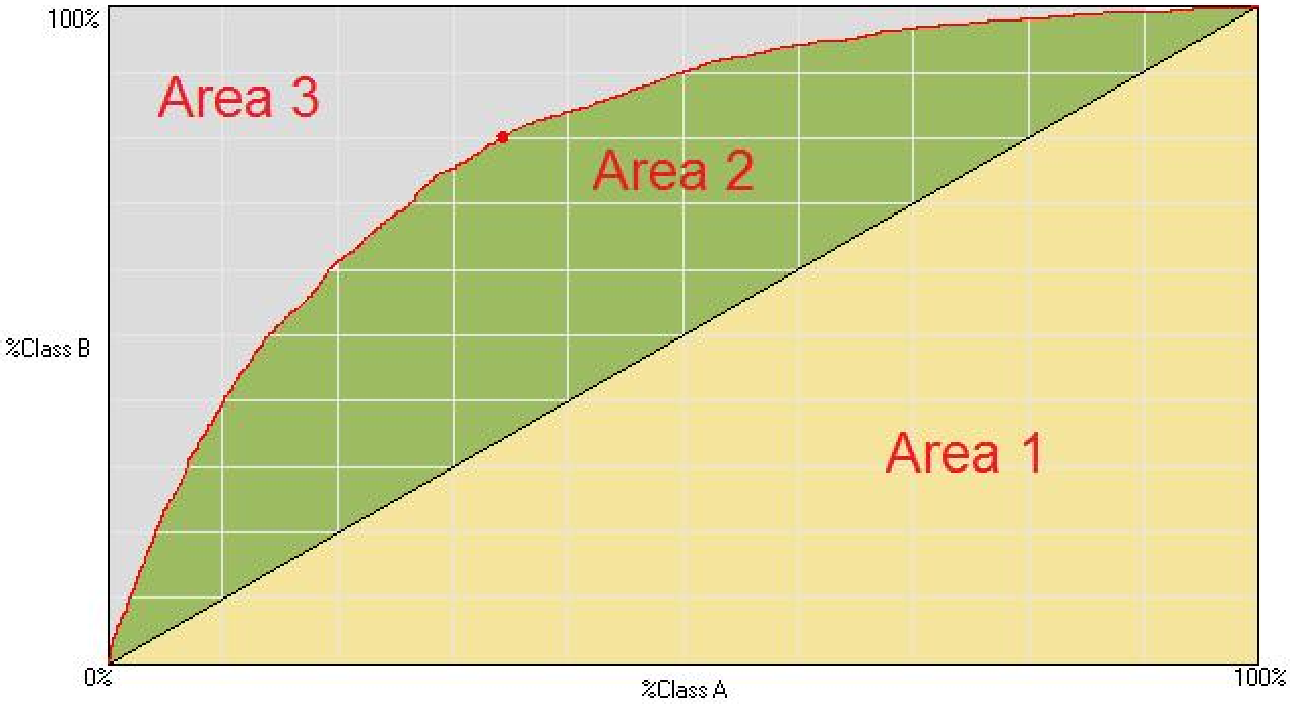
The 'Grand Champion' will receive $1,000 Australian Dollars.

Bragging rights only to the rest!

AUC - *what is it?*

The Area Under the Curve (AUC) is one metric to quantify how well a model performs in a binary classification task. Given that the thing we are trying to predict has only 2 values (eg Class A/Class B, 1/0 , red/not red, rating 1/rating 5), how well does the model separate these values. In the data provided, the two values are coded as 1 and -1. AUC is a pretty simple concept and easy to calculate...

Your model is trying to predict a value of 1 or -1. Sort your models predictions in descending order (that is the only role the predictions play!). Now go down the sorted list and for each prediction, plot a point on a graph that represents the cummulative % of class A v the cummulative % of class B that have actually been encountered upto and including that particular prediction. Join up all the points to form a curve. The AUC is the area under this curve.



Area 1 + Area 2 + Area 3 = 1 (100%)

AUC = Area 1 + Area 2

Note that if the curve followed the diagonal line then the model has no ability to separate the 2 classes - it is no better than random. This would have an AUC of 0.5. An AUC of 1 is a perfect model (Area 3 disappears - 100% of class B have been found before any of class A) and an AUC of 0 is also a perfect (but backward) model.
A similar metric is the Gini coefficient, which is essentially the same but scaled so that 1 represents a perfect model, 0 a random model and -1 perfect but backward.

Gini = 2 * (AUC - 0.5)

For this challenge we will be calculating the absolute value of the Gini coefficient, which means you do not have to worry about whether your model attempts to assign higher scores to class A or class B.

We have include an Excel file on the download page that contains macro code to calculate the AUC.

Important Dates

September 21, 2009
Bellkor's Pragmatic Chaos announced as winners of the Netflix Prize receiving $1million.

September 26, 2009
Challenge begins. Teams can register, download the small data sets and submit solutions to the leaderboard.

October 7, 2009
Medium data sets made available.

October 18, 2009
Large data sets made available.

November 8, 2009
Final Solutions and Reports can start to be submitted.

November 22, 2009

Challenge closes. Deadline for submitting solutions on the large and medium data sets and uploading a report.
The submission page will close when it is no longer Nov 22nd anywhere in the world.

December 1-4, 2009
Results announced at AusDM 2009 Conference.

Other Information

Who can enter?

- Anyone, even the experts who supplied us with the rating predictions (and we hope they do).

What about teams?

- You can enter either individually or as a member of a team - but not both. No individual can be in more than 1 team. Any large commercial organisations submitting multiple entries via single person teams will be disqualified.

Supporting code

- There is an Excel file on the download page that will calculate the AUC with VBA code showing how it's done.
- We have also prepared a Visual Basic 2008 project that you can use as a framework. It prepares the data for fast loading and demonstrates very simple hill climbing and gradient descent methods. It will also automatically generate a submission file for the built models, so you can easily make submissions using the given algorithms and included executable. VB2008 Express Edition is available for free here. You might need to install the latest version of the .net framework if you just want to use the executable.

Literature

Please contact us with any relevant online research literature that may be useful and we will add it to the list below.

Y. Koren, "The BellKor Solution to the Netflix Grand Prize", (2009) [see Section VII].

A. Toscher, M. Jahrer, R. Bell, "The BigChaos Solution to the Netflix Grand Prize", (2009). [see Section 6]

M. Piotte, M. Chabbert, "The Pragmatic Theory Solution to the Netflix Grand Prize", (2009) [see Section 4].

Its already been calculated what a 50/50 blend of BPC and The Ensemble submissions would have achieved. Read more here.

IBMs winning entry in the 2009 KDD Cup reference the following:
Ensemble Selection from Libraries of Models
Getting the Most Out of Ensemble Selection

Team Gravity made their linear regression code available to all Netflix participants.

Analysis of the PAKDD 2007 Competition entries

Acknowledgments

Many thanks to Netflix for organising the Netflix Prize and allowing us to redistribute their rating values.
(Note that this challenge is not endorsed by Netflix in any way. They gave permission to use some of their data but are otherwise uninvolved.)

To Gabor Takacs, Greg McAlpin and Andreas Toscher for organising the file collection and being enthusiastic about this challenge.

Your expert model builders - for 3 years of hard work:

The Ensemble
BellKor's Pragmatic Chaos

Gabor Takacs, Istvan Pilaszy, Bottyan Nemeth, Domonkos Tikk, Lester Mackey, David Lin, David Weiss, Joe Sill, Ces Bertino, Gavin Potter, William Roberts, Willem Mestrom, Bob Bell, Chris Volinsky, Yehuda Koren, Andreas Toscher, Michael Jahrer, Martin Piotte, Martin Chabbert, Xiang Liang, Jeff Howbert, Chris Hefele, Larry Ya Luo, Aron Miller, Craig Carmichael, Bo Yang, Bill Bame, Nicholas Ampazis, George Tsagas.

Contact

If you have any questions please contact Phil Brierley via Email

FAQs

If you have any questions, please post them in the Forum.
Note that this is a free forum board and has a bit of downtime, so please be patient.

Please explain this AUC problem a bit more?
The AUC is just a common metric for determining how good a model is when the outcome is binary, we could quite easily have chosen the RMSE as the metric used to determine the winner.
Not many 'off the shelf' algorithms seek to minimise the AUC directly, but generally speaking, improving the performance of one metric will tend to improve on all metrics. You will see in a previous competition that there were 4 metrics to optimise. The leading Tiberius entry submitted the same solution for all 4 tasks. So if you treat it as a linear regression problem and try to minimise the RMSE, then you should also be getting a decent AUC at the same time.
In the supplied VB code we provide a hill climbing algorithm that seeks to maximise the AUC directly.

Does the AUC submission file just have to contain 1s or -1s ?
No, it can contain numbers of any value.
You are trying to rank order them, not assign a particular value.

Do I need to submit the actual rank orders in the AUC file ?
No, the calulation algorithm sorts all this out. Just submit a file where the higher the number means the greater propensity. You don't even have to worry about whether high numbers mean greater propensity to be 1 or -1, this is all dealt with by the calculation algorithm.

Sponsors

This challenge has been arranged and sponsored by Dr Phil Brierley of Tiberius Data Mining.

If you are interested in the outcome of this research and would like to encourage the competitors by contributing to the prize fund (and getting your name here) then please get in touch.

RESULTS

Grand Champion and $1,000 prize winner:
NosferatoCorp (Andrzej Janusz)
University of Warsaw, Faculty of Mathematics, Informatics and Mechanics

Runner Up:
UniQ (Vladimir Nikulin)
University of Queensland, Department of Mathematics

Bronze:
barneso (Jeremy Barnes)
Barneso Consulting

Commendations:
LatentView (C. Balakarmekan, R. Boobesh)

ADM1 (Tom Au, Rong Duan, Guangqin Ma, Rensheng Wang)
AT&T Labs, Inc.-Research, USA

Champions League - Final Standings

	Medium				Large				
Team Name	RMSE	Rank	Gini	Rank	RMSE	Rank	Gini	Rank	Points
NosferatoCorp	881.650	2	0.392	1	865.861	1	0.6941	2	9
UniQ	881.884	5	0.3879	2	866.582	3	0.6972	1	15
barneso	882.125	8	0.387	3	866.514	2	0.6923	4	23
LatentView	881.322	1	0.3693	6	866.809	7	0.692	6	33
Kranf	882.383	10	0.3819	4	867.172	9	0.6924	3	38
hugojair	881.722	3	0.3643	10	866.738	5	0.6884	9	41
Baseline1	881.98	6	0.3687	8	866.785	6	0.6903	8	42
Innovative analysts	882.879	12	0.3691	7	868.541	11	0.6918	7	55
axct	881.985	7	0.3674	9	870.801	15	0.6921	5	56
tkstks	881.755	4	0.3613	13	867.273	10	0.6831	10	57
EnsembleMaster09	882.656	11	0.2703	19	866.643	4	0.522	19	76
Edr2	882.211	9	0.3353	16	869.127	13	0.6671	15	81
DMLab	883.038	13	0.3153	18	866.985	8	0.635	18	83
The final say	889.101	19	0.3694	5	873.55	18	0.6759	12	84
Team_EXL	886.394	18	0.3421	15	868.555	12	0.6686	14	85
Baseline2	885.299	15	0.3635	12	873.219	17	0.6702	13	87
DreamTeam	885.412	16	0.3635	11	873.638	19	0.6778	11	87
TULIP	884.128	14	0.3217	17	869.216	14	0.662	17	93
albert2	894.071	20	0.3606	14	883.029	20	0.6638	16	106
Green Ensemble	886.367	17	0.1327	20	872.25	16	0.3572	20	109
TUB09	7858.22	21	0.0315	21	8120.22	21	0.1234	21	126

medium RMSE

Rank	Team Name	RMSE	Method
1	LatentView	881.322	
	The Champions	881.627	Average - Nosferato,UniQ,barneso
2	NosferatoCorp	881.650	crippledGMB
3	hugojair	881.722	bipso_blr_so12-Nov-2009091014_MEDIUM_RM
4	tkstks	881.755	Ensemble Selction
5	UniQ	881.884	lm_m
6	Baseline	881.98	Linear Regression Ensemble
7	axct	881.985	s150
8	barneso	882.125	Merged model
9	Edr2	882.211	Single Linear Perceptron
10	Kranf	882.383	elasticnet
	The Ensemble	882.457	Average All Entries
11	EnsembleMaster09	882.656	LASSO
12	Innovative analysts	882.879	
13	DMLab	883.038	
14	ISMLL	883.109	
15	BusinessResearch	883.852	MLP ensemble after stepwise selection
16	TULIP	884.128	
17	Combador	884.643	Please refer to the report
18	Baseline	885.299	Linear Regression
19	DreamTeam	885.412	M_RMSE_ScoreMY1column.txt
20	Green Ensemble	886.367	
21	Team_EXL	886.394	IE with random sets
22	Baseline	888.649	Average Top 10 Experts
23	The final say	889.101	
24	KKUI TU Kosice	889.844	avg some models
25	Baseline	892.249	Best Expert
26	albert2	894.071	
27	Baseline	894.827	Average All Experts
28	Baseline	947.88	Worst Expert
29	snustat	1461.93	random forest
30	snustat_mk	1461.93	rf
31	TUB09	7858.22	ComSUM

medium AUC

Rank	Team Name	Gini	Method
	The Champions	0.3928	Average Rank - Nosferato,UniQ,barneso
1	NosferatoCorp	0.392008	GeneticMetaBlender
2	UniQ	0.387888	glm_m
3	barneso	0.386991	Merged model
4	Kranf	0.381865	elasticnet
5	ADM1	0.381485	M_ALL5_Score
	The Ensemble	0.3769	Average Rank All Entries
6	The final say	0.369404	
7	LatentView	0.369343	
8	Innovative analysts	0.369093	
9	Baseline	0.3687	Logistic Regression Ensemble
10	axct	0.367385	s50
11	hugojair	0.36432	bipso_blr_so19-Nov-2009135305_auc_MEDIU
12	DreamTeam	0.363502	LogReg1version
13	Baseline	0.3635	Logistic Regression
14	tkstks	0.361324	
15	albert2	0.360562	
16	NeuroTech RDI	0.369282	Zoomed MLP
17	Team_EXL	0.342139	NN+BLASTING+IE
18	Edr2	0.335274	Best Combo of 10 Experts by GINI
19	Baseline	0.3284	Average Top 10 Experts
20	Baseline	0.3243	Best Expert
21	Baseline	0.3218	Average All Experts
22	TULIP	0.321704	
23	DMLab	0.315261	
24	EnsembleMaster09	0.270304	Gradient Boosting
25	Baseline	0.2522	Worst Expert
26	Green Ensemble	0.13271	
27	TUB09	0.0315204	Comb

large RMSE

Rank	Team Name	RMSE	Method
1	NosferatoCorp	865.861	crippledGMB
	The Champions	865.969	Average - Nosferato,UniQ,barneso
	The Ensemble	866.160	Average All Entries
2	barneso	866.514	Merged model
3	UniQ	866.582	lm_L
4	EnsembleMaster09	866.643	LASSO
5	hugojair	866.738	bipso_blr_so19-Nov-2009131111_LARGE_RMS
6	Baseline	866.785	Linear Regression Ensemble
7	LatentView	866.809	
8	DMLab	866.985	General linear
9	Kranf	867.172	elasticnet
10	tkstks	867.273	RSS003
11	ISMLL	867.875	
12	BusinessResearch	868.312	MLP ensemble after stepwise selection
13	Innovative analysts	868.541	
14	Team_EXL	868.555	Coefficient Blasting and bootstrapping
15	Edr2	869.127	Single Linear Perceptron
16	TULIP	869.216	
17	axct	870.801	s300
18	Green Ensemble	872.25	
19	Baseline	873.204	Average Top 10 Experts
20	Baseline	873.219	Linear Regression
21	The final say	873.55	
22	DreamTeam	873.638	Reg1version
23	Baseline	880.269	Best Expert
24	Baseline	880.766	Average All Experts
25	albert2	883.029	
26	Baseline	933.278	Worst Expert
27	TUB09	8120.22	

large AUC

Rank	Team Name	Gini	Method
1	ADM1	0.698372	L_ALL7_Score
	The Champions	0.697159	Average Rank - Nosferato,UniQ,barneso
2	UniQ	0.697161	glm_L
3	NosferatoCorp	0.69407	GeneticMetaBlender
	The Ensemble	0.69240	Ave Rank All Entries
4	Kranf	0.692358	elasticnet
5	barneso	0.692293	Merged model
6	axct	0.692122	s6
7	LatentView	0.691992	
8	Innovative analysts	0.691773	
9	Baseline	0.6903	Logistic Regression Ensemble
10	hugojair	0.688352	bipso_blr_so19-Nov-2009171830_auc_LARGe
11	tkstks	0.683087	es006
12	NeuroTech RDI	0.681153	MLP
13	DreamTeam	0.677822	L_AUC_ScoreMY1column.txt
14	The final say	0.675946	
15	Baseline	0.6702	Logistic Regression
16	Team_EXL	0.668552	Blasting + PCA + NN
17	Edr2	0.667128	Best Combo of 10 Experts by GINI
18	albert2	0.663776	
19	TULIP	0.661976	
20	Baseline	0.6592	Average Top 10 Experts
21	Baseline	0.6532	Average All Experts
22	Baseline	0.6526	Best Expert
23	DMLab	0.635049	
24	Baseline	0.5564	Worst Expert
25	EnsembleMaster09	0.521953	LASSO
26	Green Ensemble	0.357228	regression tree
27	TUB09	0.123444	Comb

AusDM 09 Ensembling Challenge

Phil Brierley – Tiberius Data Mining (http://www.tiberius.biz/)

Baseline Entries

The results tables contain some ‘Baseline’ entries for each data set. Here we briefly describe how these were generated.

Baseline2

In the final standings table, Baseline2 is basic linear regression for the RMSE sets and logistic regression for the AUC sets. All data in each of the training sets was used to build then models and then the coefficients applied to the scoring sets.

Baseline1

Baseline1 is similar in that it only uses logistic regression for the AUC sets or linear regression for the RMSE sets. Rather than a single model, multiple models are built on random sup populations of the data and then averaged to give a final prediction. This method and accompanying R source code was made available to all competitors via the competition forum.

Fig 1 and Fig 2 demonstrate the benefit of this technique in preventing over fitting. The small RMSE train data was split 50/50 to create train and test sets. A linear regression model was built on the train set and then applied to the test set. The red and blue horizontal lines show the two RMSEs. Over multiple random 50/50 splits, it was always found that the test error was much worse then the train error.

Numerous models were built using a random x% of the variables and y% of the cases of the train set data. Each new model was added to the ensemble with equal weighting, and the resulting ensemble model errors calculated on both the train and test sets.

Fig 1 and 2 both show that the train set error never reaches the level achieved when all the available data is used in a single model, but the test set error does improve, which is the name of the game.

In order to find the best settings, a simulation was performed that iterated through all combinations of %variables and %cases (in 5% bands), building ensembles containing 25 individual models. This is shown in Fig 3. It can be seen that the model error is more sensitive to the % of variables than the % of cases. The best error on the train set is when all variables are used, but this gives the worst error on the test set. The best test set error is achieved when less than 40% of the variables are used per model.

Fig 4 shows a similar plot for the AUC data.

In the reported scores on the results tables, 50% of variables and 100% of cases were used for the medium and large RMSE sets (Linear Regression Ensemble), and 50% of variables and 50% cases for each model in the AUC sets (Logistic Regression Ensemble). 100 models per ensemble were used in all cases except the large AUC, which consisted of 50 models. It is not assumed that these are the optimal settings.

For the RMSE task, the final result of the linear regression ensemble is a single linear regression model, as the coefficients are just averaged. This technique is just a way of arriving at a different set of coefficients.

For the logistic ensemble, it is not as simple as just averaging the coefficients to obtain a single logistic model.

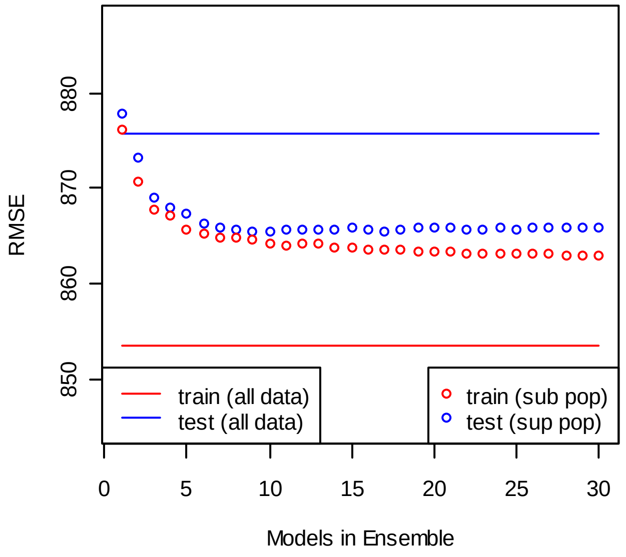


Fig 1 – each model in the ensemble as built using 30% of the variables and 30% of the cases

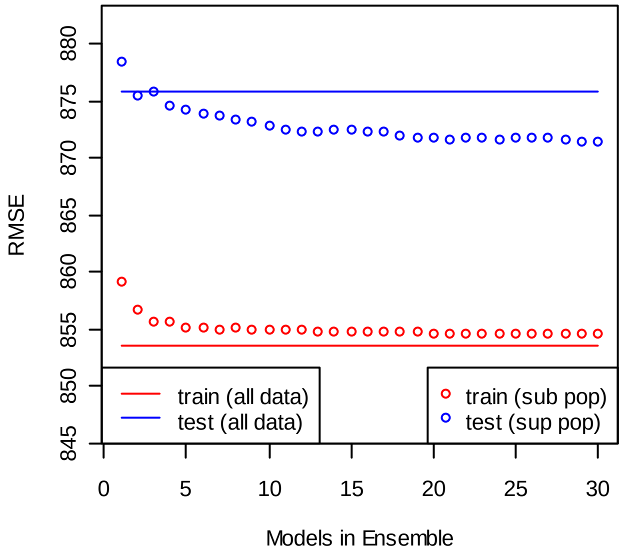


Fig 2 – each model in the ensemble as built using 80% of the variables and 80% of the cases

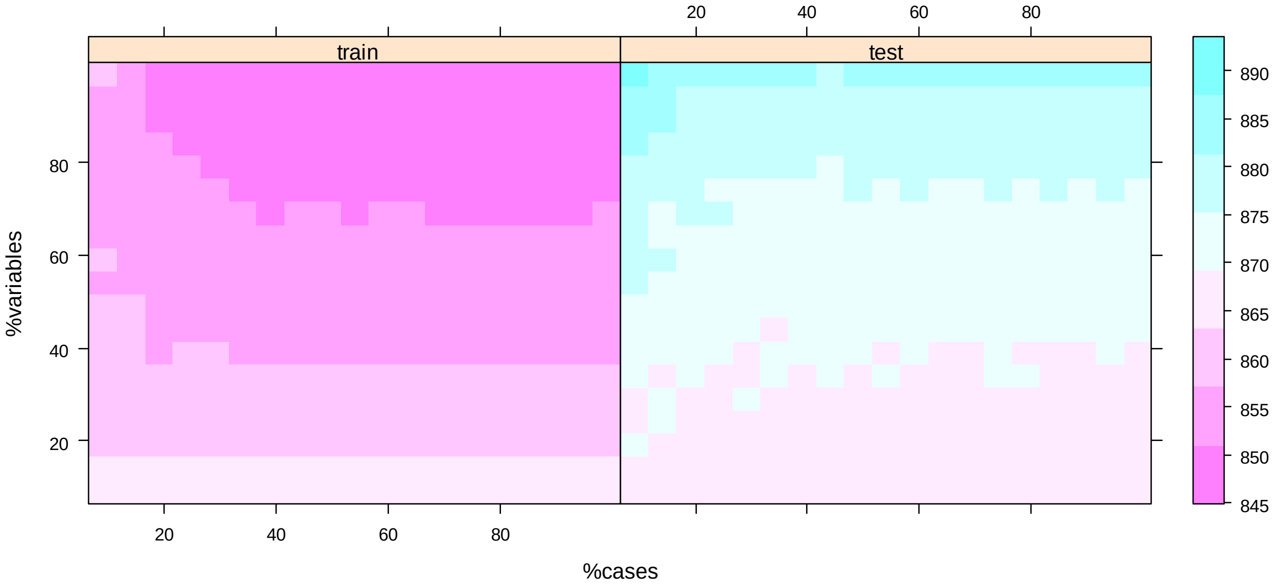


Fig 3 – Choosing the best percentages of variables and cases for each model in the RMSE ensemble. Each ensemble contained 25 models. It can be seen that the errors are more sensitive to the %variables than the %cases. These plots were generated using the small RMSE train data set.

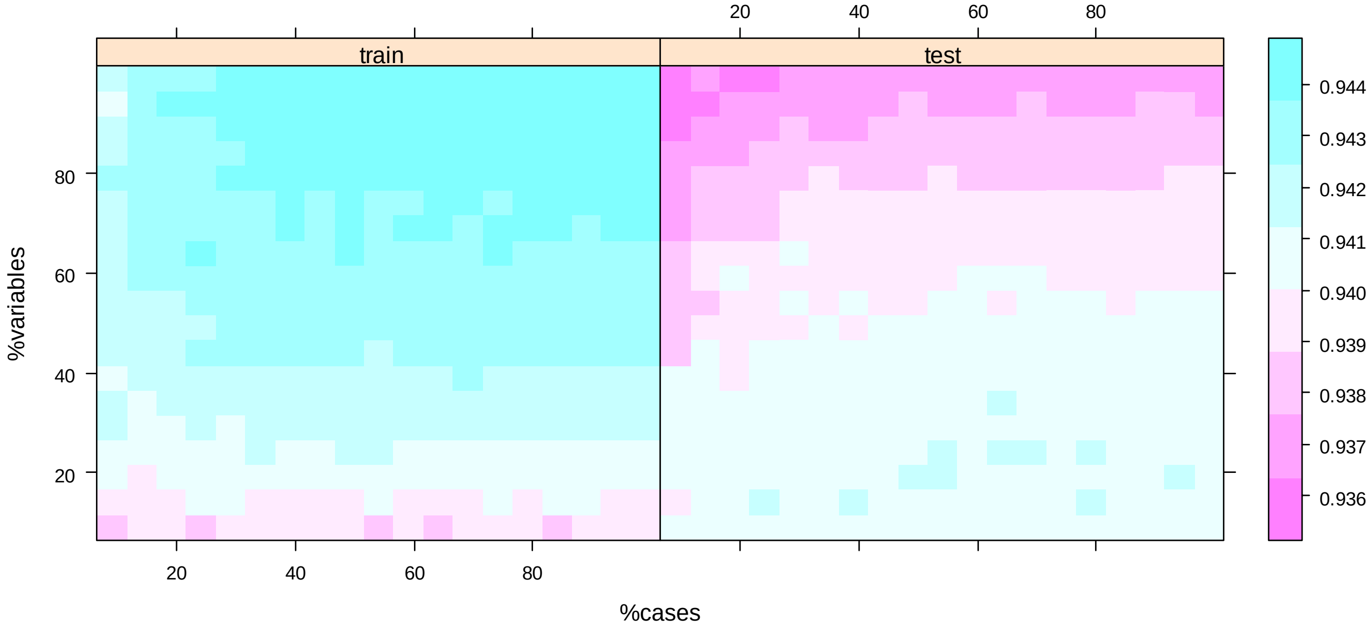


Fig 4 – Choosing the best percentages of variables and cases for each model in the AUC ensemble. Each ensemble contained 25 models It can be seen that the errors are more sensitive to the %variables than the %cases. These plots were generated using the small AUC train data set.

AusDM09 Analytic Challenge Report

TEAM NAME: NosferatoCorp

TEAM MEMBERS: Andrzej Janusz (andrzejanusz@gmail.com)

AFFILIATION: University of Warsaw, Faculty of Mathematics,
Informatics and Mechanics

USED METHOD: **G**enetic **M**eta-**B**lender

SUMMARY OF THE APPROACH:

GMB is a method for constructing ensembles of multiple classifiers which utilizes the idea of a genetic algorithm to optimize the proportions between the models in the final blend.

The assessment of samples from the score data sets was performed in two steps. First, for each training set a wide range of predictive models were constructed. Popular models available in standard R System libraries were used for the AUC task: linear and logistic regression models (library stats), neural networks (library nnet), recursive partitioning trees (rpart), k-nearest neighbors (class), the random forest (randomForest) and boosted regression models (gbm). The linear, logistic and neural network models were additionally averaged over multiple runs on different attributes subsets. The neural networks had one hidden layer which contained 1 to 5 neurons. The recursive partitioning trees were bagged and a few values of the complexity parameter were tried. The k-NN algorithm was used as a regression model, several k values between 50 and 150 were used. The boosted regression models were fitted with the bernoulli, gaussian and the adaboost loss functions. Due to lack of time for experiments with the parameters settings, only linear regression models and a simple neural networks were used for the RMSE task.

Each model's prediction values for samples from the training sets were acquired by the cross-validation test and used in the second step as an input for the blending algorithm. The genetic algorithm, which was implemented in R for the purpose of the Challenge, used different scoring functions for each of the tasks. It tried to directly maximize the AUC or minimize the RMSE by assigning significance levels to models in the ensemble. The total number of models included for the GA optimization in the final submission was dependent on the size of the datasets and the task. It was limited to 20 for the small and medium AUC data, to 25 for the large AUC data and to 10 for all sizes RMSE data. The restriction on number of models was introduced to avoid over-fitting. Some other precautions, such as a restriction on the granularity of the weights of the models, were also taken. The algorithm was stopped when the averaged quality of the population members didn't change significantly in 5 consecutive generations.

Below, there is a comparison of the results of the final GMB model and the straight average of models from which it was constructed. The scores were computed after the challenge by the organizers and were included to this report at their request:

Medium AUC
GA = 0.3920
Ave = 0.3859

Medium RMSE
GA = 881.650
Ave = 881.616

Large AUC
GA = 0.69407
Ave = 0.69208

Large RMSE
GA = 865.861
Ave = 866.224

The GA optimization led to better results in 3 out of 4 datasets. The differences between the optimized and the averaged models are generally less significant for the RMSE task, which perhaps is due to lower diversification of the utilized models.

Ensemble Learning with Random Sampling

Vladimir Nikulin (Team: UniQ)

Department of Mathematics, University of Queensland
{v.nikulin@uq.edu.au}

Ensemble (including voting and averaged) classifiers are learning algorithms that construct a set of many individual classifiers (called base-learners) and combine them to classify test data points by sample average. It is now well-known that ensembles are often much more accurate than the base-learners.

The data provided for this challenge[1] comes from 1,151 sets of predictions of integer ratings in the range (1-5). Each set of predictions was generated by mathematical models with the objective (generally) to minimise the 'root of the mean squared error' (RMSE) over the full set of over 1.4 million ratings[2].

There were 2 small, 2 medium and 2 large data sets, which were divided into 2 equal parts for training and for testing.

Assuming that the distributions of the training and test datasets are the same, the following approach appears to be the most natural: 1) select a few top-performing and most uncorrelated predictions, and 2) build the final decision function as a sample average. Using above approach, it will not be difficult to optimise the optimal number of input predictions or predictors.

Let us consider Fig. 1(a) which may be regarded as an illustration to the fact that the distribution within the training set is not stable.

The following method was used in order generate Fig. 1(a). We split randomly large training set into two equal parts and computed

$$d_{it} = RMSE_{it1} - RMSE_{it2}, t = 1, \ldots, 100,$$

where t is the number of iteration, $RMSE_{it1}$ and $RMSE_{it2}$ correspond to the first and second parts, i is the index of the predictor. As an outcome, we found standard deviations (STD) of the differences d for any particular predictictor, see Fig. 1(a). It is interesting to note that all STDs are within the range between 6 and 7.5. On the one hand, low margin indicates very high level of volatility, on the other hand, the difference between upper and low margins is rather small (means that the level of volatility is a quite stable within the range of the given set of predictors).

Based on our experimental results against small test set, we concluded that the distributions of the medium and large test sets are also differ from the corresponding training distributions.

In order to overcome the problem with 'volatile' distributions we decided to use a popular method of random feature selection. We considered 4 main models: 1) *GBM* and *GLM* in R for the AUC-task; 2) *LM* in R and *RLR* (regularised linear regression - our own code written in C) for the RMSE-task.

[1] http://www.tiberius.biz/ausdm09/index.html

[2] http://www.netflixprize.com/

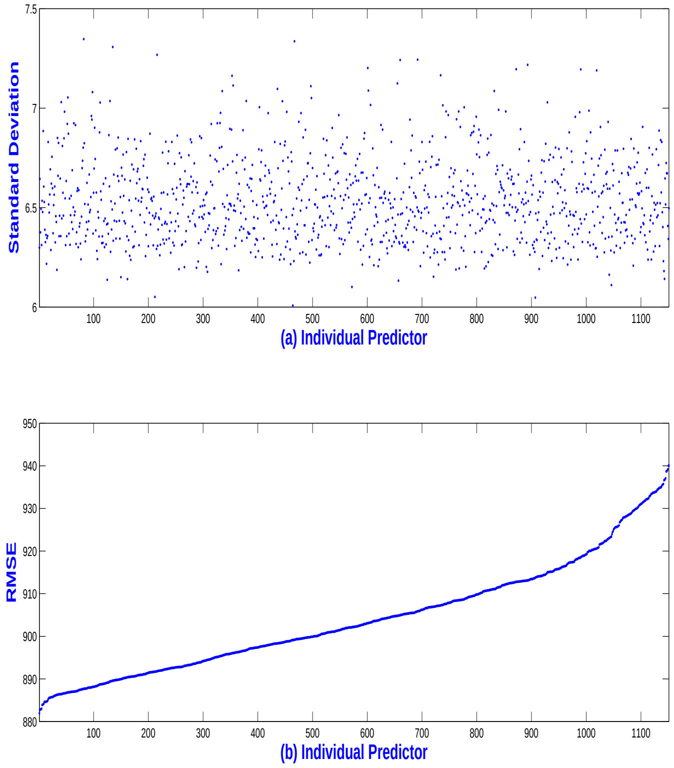


Fig. 1. (a) standard deviations of the particular predictions; (b) RMSE-scores of the particular predictors in a sorted order.

Any particular decision function was computed as a sample average of a few hundreds of base-learners, each of which was based on 30% - 50% of randomly selected features.

Different decision functions were linked together using an ensemble constructor as it is described in [1].

References

[1] Nikulin, V., McLachlan, G.J.: Classification of imbalanced marketing data with balanced random sets. JMLR: Workshop and Conference Proceedings **7** (2009) 89–100

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, $\text{AOC} = 1 - \text{AUC}$) 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.

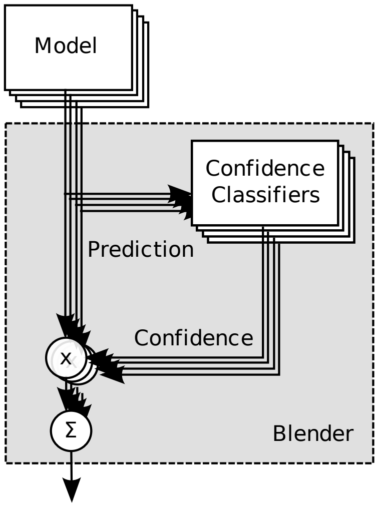


Figure 1. Gated Classifier Model. Note that *all* confidence classifiers receive the value of *all* models, not just the one for which it is generating a confidence value.

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

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

[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 $[\sigma_1 \sigma_2 \ldots \sigma_{1151}]$ 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

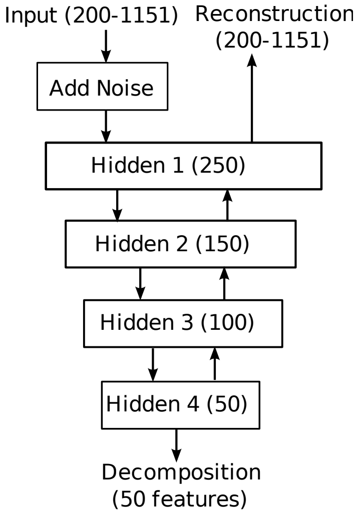


Figure 2. Denoising auto-encoder decomposition used in the challenge.

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.

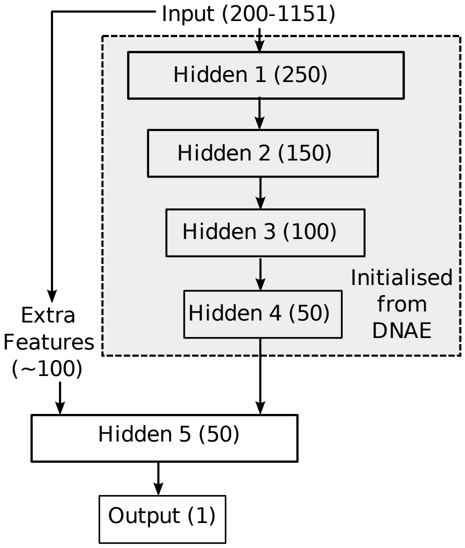


Figure 3. Deep Network Model.

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
S AUC	250	0.00	0.86	0.81	6.51
	150	0.20	1.48	1.15	3.51
	100	0.44	1.99	1.30	1.92
	50	0.72	2.22	1.40	0.92
M AUC	250	0.00	0.62	0.62	3.58
	150	0.30	1.09	1.02	3.44
	100	0.48	1.52	1.42	2.87
	50	0.74	1.97	1.80	0.90
L AUC	250	1.00	2.08	2.18	17
	150	1.31	3.24	3.05	4.88
	100	1.52	4.18	3.74	2.98
	50	1.84	5.01	4.34	2.08
L RMSE	250	0.99	2.31	2.51	23
	150	1.30	3.58	3.42	5.52
	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 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. Ensembling with confidence information.

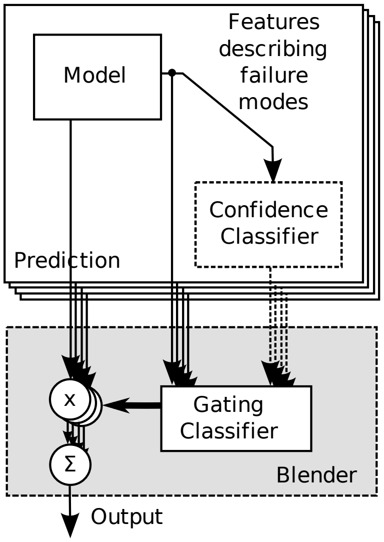


Figure 4 shows one way to implement such a scheme using a gating function. The confidence classifiers could either be part of the black box, trained externally from the features, or learnt implicitly as part of the gating function (which would receive only the features).

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

[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.

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.

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.

[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.

Combining Experts by Modeling and Boosting Score Characteristics

Team ADM1: Tom Au, Rong Duan,
Guangqin Ma, and Rensheng Wang

AT&T Labs, Inc.-Research, USA

Abstract. Based on the same information, subjects are classified into two categories by many experts, independently. The overall accuracy of prediction differs from expert to expert. Most of the time, the overall accuracy can be improved by taking the vote of the experts committee, say by simply averaging the ratings of the experts.

More sophistic softwares and models may be created trying to ensemble the experts ratings and produce a better overall predictive accuracy (i.e., higher AUC), but often they are not apparent to users.

In this exercise, we propose a method to summarize the subject-wise characteristics of experts scores and combine them with experts' rating to boost the overall predictive accuracy. The advantages of this approach are less computing intensive, easy to implement and apparent to user, and most of all, it produces much better result than the simple averaging, say.

For an application with a base consists of hundreds of millions of subjects, 1% improvement in predictive accuracy will mean a lot. Our method which requires less efforts and resources will be one more plus to practitioners.

Key words: Score Characteristics, LogitBoost

1 Data Structure

We consider the following data structure

$$\{(Y_i, f_{i,1}, f_{i,2}, ...f_{i,m}), i = 1, 2, 3, ..., n\}$$

where $Y_i = 1$ *or* 0 indicating the category that subject i belongs to or not, n is the number of subjects, $f_{i,j}$ is the rating score assigned to subject i by expert j, $j = 1, 2, ..., m$. Without loss of generality, we assume that $Y = 1$ associates with higher rating scores.

2 Invariant Property of ROC Curve

The performance of a classifier is measured by its ROC curve. Two classifiers have different ROC curves not because they have different values, but because they have different ranking of subjects under study. Therefore, the ROC curve is invariant under a monotone transformation of the decision variables (i.e., $f_{i,j}$), for given j.

3 Ranks for Classifiers and Its Characteristics

For classifier j, we sort $\{f_{i,j}, i = 1, 2, ..., n\}$ in ascending order, let $\{r_{i,j}, i = 1, 2, ..., n\}$ be the corresponding percentage rank scores ($0 \leq r_{i,j} \leq 1$).

Based on the rank scores $\{r_{i,j}, i = 1, 2, ..., n\}$, we created the following variables:

The Blom [1] normal rank scores

$$x_{i,j} = \Phi^{-1}((n * r_{i,j} - \frac{3}{8})/(n + \frac{1}{4})), i = 1, 2, ..., n,$$

where $\Phi()$ is the cumulative standard normal distribution function. Based on the invariant argument, the ROC curves based on $f_{i,j}$, $r_{i,j}$ and $x_{i,j}$ are the same for a given expert j.

If subject i's true status is $Y_i = 1$, then we would expect that majority of the experts will assign higer ranking scores to the subject, and vise versa.

The characteristics of the score distribution of $\{r_{i,j}, j = 1, 2, ..., m\}$ for individual i is characterized by the shape parameters (α_i, β_i) of beta distribution $beta(\alpha_i, \beta_i)$ fitted to the percentage scores. When $\alpha_i < \beta_i$, the distribution of the scores is skewed to the left(in the sense that smaller values become more likely); when $\alpha_i > \beta_i$, the distribution of the scores is skewed to the right; when $\alpha_i = \beta_i$ the score distribution is symmetric about 0.5.

A 3-mean cluster analysis based on the Blom rank scores are performed and we find that these clusters are well captured by the beta shape parameters, see Figure 1 to Figure 3, they displayed how these characteristics associated with individual subjects true status.

Other characteristics of the percentage scores based on $beta(\alpha_i, \beta_j)$ are the mean $(\alpha_i/(\alpha_i + \beta_i))$, mode $((\alpha_i - 1)/(\alpha_i + \beta_i - 2), \alpha_i, \beta_i > 1)$ and variance $((\alpha_i\beta_i/(\alpha_i + \beta_i)^2(\alpha_i + \beta_i + 1))$

To further summarize the characteristics of the score distribution for subject i, we created 21 quantiles of $\{r_{i,j}, j = 1, 2, ..., m\}$: $q_{i,1}, q_{i,2}, ..., q_{i,21}$. Quantile $q_{i,1}$ and quantile $q_{i,21}$ are the 1 and 99 percentiles, respectively; the other quantiles are 5% apart starting with $q_{i,2}$ being the 5th percentile. Figure 4 shows rank scores based on raw data and their three quartiles.

Last, we created all the descriptive statistics based on $x_{i,j}, j = 1, 2, ..., m$, for subject i. These descriptive statistics include mean, variance, max, min, range, IQR, skewness and kurtosis; in addition, the entropy measure based on normalized percentage rank scores for each subject.

4 Regression and Boosting

So far, for each suject, we have created Blom normal scores, characteristics of scores based on beta distribution, quantiles and descriptives statistics of scores based on the m experts rating.

Our next step is to use these created rank scores and characteristics as explanatory variables (X) to perform a LogitBoost [2]. At the regression stage a backward selection is used. More precisely, let $\frac{1}{2}ligit\big(P(Y=1|X)\big)=F(X)$, where $F(X)$ is a linear combination of explanatory variables. Then we start with

1: Assign each individual X with the same weight $w(X)=\frac{1}{N}$, N is the number of observation in the data, $F(X)=0$ and $P(Y=1|X)=\frac{1}{2}$;

2: Repeat from $m=1,2,...,M$:

– (a) Compute the working response and weights

$$f_m(X)=\frac{Y-P(Y=1|X)}{P(Y=1|X)\big(1-P(Y=1|X)\big)},$$

$$w(X)=P(Y=1|X)\big(1-P(Y=1|X)\big);$$

– (b)Fit the function $f_m(X)$ by weighted least-squares regression to X using weights $w(X)$ and backward selection;

– (c) Update $F(X)\leftarrow F(X)+\frac{1}{2}f_m(X)$ and $P(Y=1|X)\leftarrow e^{F(X)}/\big(e^{F(X)}+e^{-F(X)}\big)$;

3: Output $F(X)=\sum_{m=1}^{M}f_m(X)$ and $P(Y=1|X)\leftarrow e^{F(X)}/\big(e^{F(X)}+e^{-F(X)}\big)$.

5 Model Assembly

We set out to build four nested models and hope each model will capture some aspects of the subjects' true status:

- Model 1: The Blom's Rank Scores and the 21 percentiles;
- Model 2: Model 1 + the descriptive statistics (mean, stdev, skewness, kurtosis, min, max, IQR and Range);
- Model 3: Model 2 + the fitted beta parameters;
- Model 4: Model 3 + other fitted beta parameters such as the mode, mean and variance of the fitted beta density.

Initially, we split the training data randomly into 50/50 training and testing samples. We then use the testing sample to validate the result at each iteration. The testing yields the highest AUCs between the third and the seventh iterations. Therefore, we used the average scores of these models from iterations 3 to 7 as final result for each model. Instead of using simple average of all these four final results, we searched the best weights for the weighted average of these four final results.

References

1. Blom, G., Statistical Estimates and Transformed Beta Variables, New York: John Wiley & Sons, Inc (1958)
2. Friedman, J., Haste, T., Tibshirani, R.,"Additive Logistic Regression: A Statistical View of Boosting," The Annals of Statistics. Vol. 28, No. 2, 337–407 (2000)

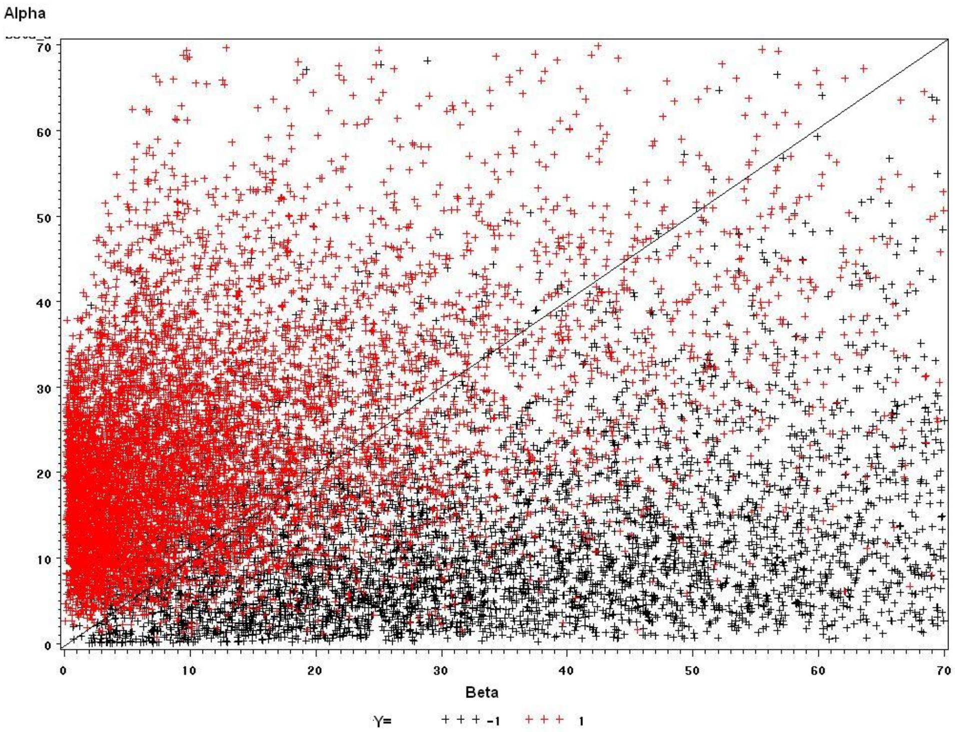


Fig. 1. Scatter plot of $\{(\alpha_i, \beta_i), i = 1, 2, ..., n\}$,red and black crosses indicating Y=1 and Y=-1, respectively.

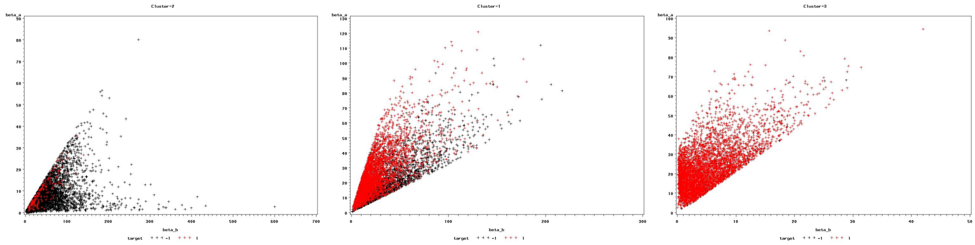


Fig. 2. Scatter plot of $\{(\alpha_i, \beta_i), i = 1, 2, ..., n\}$ by clusters based on rank scores.

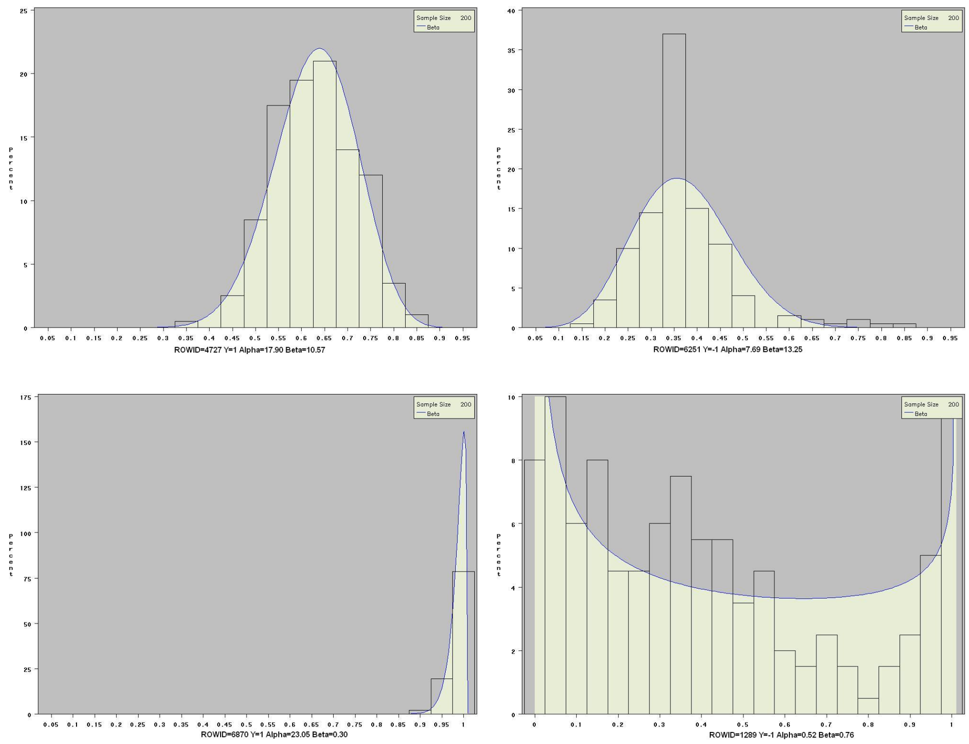


Fig. 3. Percentage rank score distributions based on the small training data set for ROWIDs =4727, 6251, 6870 and 1289. The scores are shown in histograms and the continuous curves are fitted beta densities.

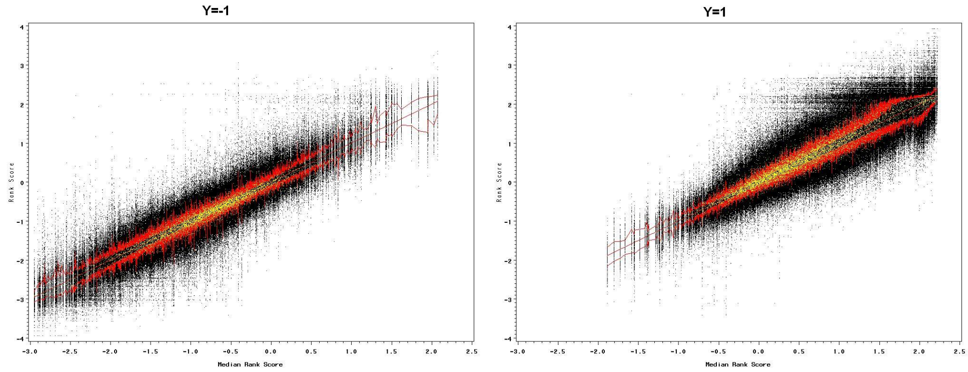


Fig. 4. Blom rank scores (black dots), quartiles (red lines) and the average of 10 best classifiers (yellow dots).

Team Name : Latentview

Team Members: C. Balakarmekan, R. Boobesh

Abstract:

Method – AUC:

In order to ensemble the scores, we generated 50 random samples from the population and built stepwise logistic model with the random sample of variables in the single logistic model using log-link function. The scores were combined by averaging the scores.

Method - RMSE:

- Decision tree technique was used to built model using treenet tool.
- We built CART models to predict the target variables using the model scores
- Ensemble of the results from the above two techniques was done to arrive at the final scores.

Kranf Team

All the models were created using in a straight forward fashion, the «Elastic Net »[1] linear regression technique $X\beta = Y$ where X is the learning dataset, β is the "predictive model" and $Y$is the "target to predict".

This technique is from the type of "penalized" regression techniques where there are two regularization parameters: α and γ. The loss function to optimize is:

$$loss\_function(\beta) = square\_of\_residual\_error(\beta) + \ \alpha\|\beta\|_1 + \gamma\|\beta\|_2$$

The parameters α and γ were estimated using a 6-fold-cross-validation technique.

For the binary prediction model, I used, as a target, Y =0 or 1

For the continuous prediction model, I used, as a target, Y =1000,2000,3000,4000 or 5000.

Reference

1. "Regularization and variable selection via the elastic net", Hui Zou and Trevor Hastie, J.R. Statist. Soc. B (2005) 67, Part 2, pp. 301-320

Particle Swarm Optimization and Meta-Ensembles for the AusDM2009 Analytic Challenge

Hugo Jair Escalante

Computer Science Department
National Institute of Astrophysics, Optics and Electronics
Luis Enrique Erro # 1, Santa María Tonantzintla, Puebla, 72840, México
E-mail: hugojair@inaoep.mx

Abstract

This paper describes our solution to the AusDM2009 Analytic Challenge. The challenge is to find an effective strategy for combining the outputs of a set of experts so that a performance measure is optimized. Accordingly, we proposed an effective way of selecting subsets of experts and then simply aggregated their outputs. Additionally, we further combined the outputs obtained from the multiple selection of subsets, that is, adopting an ensemble-of-ensembles approach. For the expert selection process we considered the geometric version of particle swarm optimization (PSO) for Hamming spaces. This formulation allows us to effectively explore the search space by creating new solutions as convex combinations of previous ones; we also included a mutation term into the PSO algorithm. For merging the outputs of experts (resp. ensembles) we simply took the average of their outputs. Despite being simple, our approach has obtained very promising results in the small data sets for both tracks of the challenge. Our results are encouraging and motivate further research and applications for the proposed technique.

Team name: hugojair; **Team member:** Hugo Jair Escalante.

1 Introduction

This paper describes the participation of my team at the AusDM2009 analytic challenge. The considered scenario is as follows. We are given the response of a set of N experts for T instances of particular problem and we want to determine what experts to use and how to combine their output such that the performance in the specific problem is maximized [1]. Clearly, there are (at least) two aspects that may have an impact on the performance of methods that face this task: $1)$ selecting the experts that are to be combined; $2)$ determining the way in which the outputs of the selected experts will be combined. Our approach to the AusDM2009 analytic challenge, depicted in Figure 1, attempts to have a positive impact on both of these aspects.

On the one hand, we propose the application of particle swarm optimization (PSO) for the selection of the experts that are to be combined. Specifically, we developed a modified version of the geometric PSO algorithm (GPSO) for Hamming spaces [5]. The fitness function of the GPSO algorithm minimizes the evaluation metrics that are evaluated in the challenge (i.e. the gini coefficient, GINI, and the root-mean-square error, RMSE), see [1] for details. Our implementation allows us to influence the number of experts that are to selected, which may be important for reducing the dimensionality of the problem according to the available resources. Once that a set of experts is selected we merge their output by using a rather simple strategies, namely, averaging the expert's outputs (for minimizing the RMSE) and building a binary classification models (for minimizing 1-GINI).

On the other hand, we propose the combination of the outputs of multiple selections of experts (see Figure 1), in what we called the ensemble-of-ensembles (or the meta-ensembles) approach. The intuition behind this formulation is that whereas the combination of a single selection of experts may be biased (because of the initialization of GPSO or because of a local minima) the average over multiple selections of experts is a more robust strategy. Our preliminary results, as reported in the AusDM2009 challenge leaderboard [1], in the SMALL data set are encouraging and motivate further research and applications for the proposed technique. The rest of this document briefly describes our approach. Notice that the methodology described in this working note will be described in detail in a forthcoming publication [3].

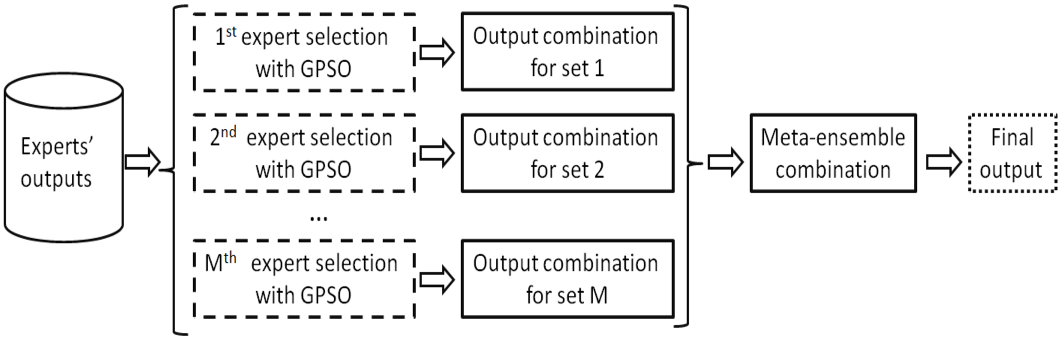

Experts' outputs
1st expert selection with GPSO
2nd expert selection with GPSO
...
Mth expert selection with GPSO
Output combination for set 1
Output combination for set 2
Output combination for set M
Meta-ensemble combination
Final output

Figure 1. Diagram of the methodology we adopted for the AusDM2009 challenge.

2 Geometric PSO

PSO is a bio-inspired search technique that has shown outstanding performance in a wide variety of domains [2]. In PSO, the solutions to the problem at hand are called particles and the population of $M-$solutions is known as the swarm; at time t each particle i has a position $\mathbf{x}_i^t \in \mathbb{R}^d$, with d the dimensionality of the problem, in the search space. The aptitude of solutions is evaluated by a fitness function, which returns for each particle a scalar value indicated how far the corresponding particle is from the optimal solution. The PSO algorithm (as well as the GPSO version) proceeds as follows. At the beginning of the optimization process the solutions (i.e. particles' positions) are initialized randomly and their aptitude is evaluated, then the following iterative process starts: based on previous solutions and on the fitness of that solutions, particles' positions are updated (i.e. new solutions are generated); next, the fitness of the new particles' positions are evaluated and the process is repeated for a fixed number of iterations. The way in which new positions are generated indicates how the search space is to be explored.

For expert selection (resp. feature selection) each solution $\mathbf{x}_i^t$ is coded as a binary vector (i.e. $\mathbf{x}_i^t \in \{0,1\}^N$) with length ($N$) equal to the number of experts (resp. features) available; the values of this vector indicate the presence or absence of a specific expert (resp. feature) in the current solution. In its original form, PSO was defined for working in Euclidean spaces. For problems in Hamming spaces the continuous version of PSO is adapted for such problems [4, 5], e.g. by considering scaling functions that map real numbers into the $[0, 1]$ interval and using a threshold to determine if the value is either 0 or 1. The latter formulation, however, presents a number of limitations that make it not suitable for the problem (e.g. how to define the threshold).

GPSO is an equivalent formulation for PSO in which the representation of solutions does not need to be adapted. Instead, GPSO provides a mathematical generalization of the notion (and motion) of particles for a general class of spaces, see Moraglio et al. for details [5]. In GPSO positions are updated as follows:

$$\mathbf{x}_i^{t+1} = (1 - \phi_1 - \phi 2) \times \mathbf{x}_i^t + \phi_1 \times \mathbf{p}_i + \phi_2 \times \mathbf{g} \tag{1}$$

where $\mathbf{x}_i^{t+1}$ is the new position for a particle i, $\mathbf{x}_i^t$ is the current position for particle i, $\mathbf{p}_i$ is the best solution found by particle i at time t, $\mathbf{g}$ is the best particle found so far in the swarm and $\phi_1 \geq 0$, $\phi_2 \geq 0$, $\phi_1 + \phi_2 < 1$. This means that the new position for a particle is a convex linear combination of its previous position, the best position found by the particle and the best solution found by any particle in the swarm, regardless of the space in which the solutions lie. For expert selection we consider the GPSO version for Hamming spaces as described in [5]. In the rest of this section we describe the fitness function we used for minimizing the RMSE and for maximizing the GINI coefficient for the two tasks in the AusDM2009 challenge.

2.1 GPSO for RMSE

For each particle i, let $E_i = \{j : \mathbf{x}_{i,j}^t = 1\}$ (i.e. each j is an expert index, $j \in \{1, \ldots, N\}$); let $N_E^i = |E_i|$ and consider the matrix $\mathbf{X}_i^E$ formed by concatenating the outputs of the N_E^i selected experts E_i; under these settings, the fitness function of GPSO for a particle $\mathbf{x}_i^t$ with the goal of minimizing the RMSE is given by:

$$f(\mathbf{x}_i^t) = \sqrt{\frac{1}{T}\sum_{k=1}^{T}(\mathbf{o}_k - avg(\mathbf{X}_k^E))^2 + \left(\lambda \times q(N_E^i, N_{max})\right)} \tag{2}$$

with $avg(\mathbf{X}_k^E) = \frac{1}{N_E^i}\sum_{j=1}^{N_E^i} \mathbf{X}_{k,j}^E$ is the average of the N_E^i experts for the instance k; $\mathbf{o}_k$ is the ground-truth label of instance k; N_{max} is the maximum number of experts that we want to consider and where:

$$q(N_E^i, N_{max}) = \begin{cases} \log(N_E^i - N_{max} + 1) & \text{if } N_E^i > N_{max} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The first term in Equation (2) is just the RMSE obtained by combining (via simple average) the outputs of the considered experts. The second term in Equation (2) penalizes the incorporation of more than N_{max} experts in the current solution, the scalar λ weights the importance of the latter term.

2.2 GPSO for AUC

Similarly, the the fitness function of GPSO for a particle $\mathbf{x}_i^t$ with the goal of maximizing the GINI coefficient is given by:

$$f(\mathbf{x}_i^t) = \left(1 - 2. * (g_{auc}(b(\mathbf{X}^E, \mathbf{o}), \mathbf{o})) - 0.5)\right) + \left(\lambda \times q(N_E^i, N_{max})\right) \tag{4}$$

where λ, and $q(N_E^i, N_{max})$ as above and with $b(\mathbf{X}^E, \mathbf{o})$ being the output of a binary classifier (for efficiency we considered a Naïve Bayes classifier) that uses as training set to the outputs of the N_E^i experts (as training patterns) and the ground-truth outputs for the instances (as outputs of the training patters); $g_{auc}(avg(\mathbf{X}^E), \mathbf{o})$ is a function that computes the area under the ROC curve for the predictions of $b(\mathbf{X}^E, \mathbf{o})$ given the ground-truth outputs $\mathbf{o}$. The minimization of Equation (4) is equivalent to maximizing the GINI coefficient. The first term in Equation (4) is just the GINI coefficient for the a classifier trained on the outputs of the experts; whereas the second term (defined as above) accounts for the number of experts considered in the solution.

3 Meta-ensembles

For each task (i.e. RMSE or GINI) we applied the GPSO implementation outlined in Section 2 multiple times, using different initializations each time, selecting different subsets of experts. For each of the resultant sets of experts we kept the output generated by combining the outputs of the corresponding experts as follows. For the RMSE task, we kept the result of fitting a linear regression model (using the least squares method) by using the outputs of the selected experts and the ground-truth outputs from the training set. For the GINI task, we build a binary classifier (using a more effective classification method than that used for computing the fitness function in Equation (4)) using the outputs of the selected experts and the ground-truth outputs from the training set.

As a result of the latter process, we obtain multiple outputs (generated by the combination of the outputs of experts) which we just simple average to generate a final output for the corresponding task. This setting, that we called the meta-ensembles approach, aims at obtaining more robust estimators than when the output of a single combination of features is used. In preliminary experimentation we found that satisfactory results were obtained with our methodology.

4 Parameter settings

The methodology described in this document, requires the specification of several parameters, including the number of iterations of GPSO, the number of particles for the swarm, the mutation probabilities (see [5]), λ, N_{max}, the learning algorithm for building the classifiers for the GINI task, etcetera. For our submissions to the challenge we set this parameters by trial and error using as evaluation measure to the score provided by the organizers through the leaderboard [1]. We would like to emphasize that our approach to the AusDM2009 analytic challenge will be properly described in detail in a forthcoming publication [3], the code implementing our approach will be publicly available from *http://ccc.inaoep.mx/∼hugojair*.

References

[1] P. Brierley. Website of the AusDM analytic challenge. http://www.tiberius.biz/ausdm09/, 2009.

[2] A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley, 2006.

[3] Hugo Jair Escalante. Geometric particle swarm optimization and meta-ensembles for output aggregation and feature selection. *In Preparation*, 2009.

[4] J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In *IEEE*, 1997.

[5] A. Moraglio, C. Di Chio, and R. Poli. Geometric particle swarm optimisation. In M. Ebner et al., editor, *Proceedings of EuroGP'07*, volume 4445 of *LNCS*, pages 125–136. Springer, 2007.

Team name : tkstks

Member name : tsukasa moritoki

Method :

Small Medium

RMSE : Random SubSpace[LInear Regression]
# models : 50-80
iteration : 20-40

AUC : Random SubSpace[Logistic Regression]
# models : 100-150
iteration : 30-50

Large

RMSE : RSS[LIR](m1) Ensemble Selection(m1)
m1 : 100 LInear Regression models from randomly
selected 70-80 base models

AUC : RSS[LR](m2) Ensemble Selection(m2)
m2 : 100 Logistic Regression models from randomly
selected 150 base models

Software :

R , RapidMiner , Weka

From the team **axct** with warm regards:

Used some mild modifications of the recommended R-code: LM function was used for RMSE and GLM function was used for AUC.

```
ensemblerContinuous(num=100,rowpercent=0.99,colpercent=0.5,df_train=trainset
,df_test=testset,mytarget=Target,fname= "c:/mydata/file.csv")
```

where

```
ensemblerContinuous <-
function(num,rowpercent,colpercent,df_train,df_test,mytarget,fname)
{
mytarget <- deparse(substitute(mytarget))
ensemble_lr <- 0
colprob <- array(dim=NCOL(df_train))
colprob[] <- colpercent
colprob[which(names(df_train)==mytarget)] <- 0
for(i in 1:num)
{
cat(i, " of ", num)
print("")
flush.console()
rows <-sample(NROW(df_train), NROW(df_train) * rowpercent)
cols <- sample(ncol(df_train),ncol(df_train) * colpercent, prob = colprob)
cols <- c(which(names(df_train)==mytarget), cols)
subpop <- df_train[rows,cols]
model_lr <- lm(as.formula(paste(mytarget, " ~ . ")) , data=subpop)
scores_lr <- predict(model_lr, df_test)
scores_lr <- ifelse(scores_lr > 5000, 5000, scores_lr)
scores_lr <- ifelse(scores_lr < 1000, 1000, scores_lr)
ensemble_lr <- scores_lr + ensemble_lr
}
ensemble_lr <- ensemble_lr / num
write.table(as.numeric(as.character(ensemble_lr)), file=fname,col.names=FALSE,
row.names=FALSE, sep=",")
}
```

AusDM 2009

Analytic Challenge

Team: BusinessResearch

Country: Russia

City: Saint-Petersburg

Members:

Evgeny Antipov ("Evgeny Antipov's Center for Business Analysis", President; "Comcon Research Company", Lead analyst; State University - Higher School of Economics, MS student)

Elena Pokryshevskaya ("Evgeny Antipov's Center for Business Analysis", Lead project manager; State University - Higher School of Economics, MS student)

Dataset analyzed: M_RMSE, L_RMSE

Approach: MLP ensemble after stepwise selection procedure

1. Select predictors based on stepwise regression procedure with PIN=0.05, POUT=0.05 (for L_RMSE PIN=0.01, POUT=0.01). This approach is not as computationally intensive as best subset search, but works almost as well, if we look at adjusted R^2 values (only slightly smaller than R^2). At each step, the independent variable not in the equation that has the smallest probability of F is entered, if that probability is sufficiently small. Variables already in the regression equation are removed if their probability of F becomes sufficiently large. The method terminates when no more variables are eligible for inclusion or removal.

2. Build neural networks using the set of variables selected at Step 1 (for M_RMSE – 31 predictors). We recommend using the following input information:

Training sample =70%

Test sample=20%

Validation sample=10%

MLP:

Min hidden units=7

Max hidden units=23

(for L_RMSE Min hidden units=8 Max hidden units=26)

Networks to train: 20

Networks to retain: 5

Error function: Sum of squares

3. Take the average of 5 best models and recode values greater than 5000 to 5000 and those less than 1000 to 1000.

Methodology:

Our methodology finds out the best top predictors from training sets using Hill Climbing Approach and we implement those top predictors in the score (test) sets for deriving mean values or results.

Our approach of Hill Climbing uses the following steps to find the best model predictors from the Training Set data:

1. Finding the Individual RMSE value of every predictor and sorting them out on the basis of ascending order of RMSE value. Then we select the best top 10 model predictors.
2. Starting with every predictor not in the top 10 predictor list, temporarily replace the top predictor by it, and then calculate the new RMSE. We compare this new RMSE value with the one calculated in the step 2. If the new RMSE is greater than the RMSE calculated in step 2 then we stop the replacement for that non top predictor. Otherwise, the replacement takes place.
3. Re-arranging the new top 10 predictors after the replacement process.
4. Repeating the steps 2 and 3 for all other remaining non-top predictors. Stop when there is no replacement candidate.
5. After the end of iteration, selecting the newly generated top 10 best predictors

After getting the best top 10 model predictors from the training data after the replacement process, we use those model predictors with the Medium Score Data to calculate the mean of those top 10 predictors which is the final result of our methodology.

Team Members:

Dr. Sumanta Guha

Mr. Chansophea Chuon

Mr. Suresh Raj Bhattarai

Dataset beneficiation based on statistical matching

Technical report for AusDM 2009 Analytic Challenge

Csaba Gáspár-Papanek, Gábor Fodor

Department of Telecommunications and Media Informatics,
Budapest University of Technology and Economics,
2nd Magyar Tudosok Krt., Budapest, H-1117,

gaspar@tmit.bme.hu, fodgabor@math.bme.hu

Introduction

AusDM 2009 Analytic Challenge was aimed at creating an ensembling method for improving the predictive accuracy of different algorithms based on 1,000 sets of predictions. This problem is made simpler as a model-aggregation problem where the goal is to combine results of two predictive models according to a predetermined evaluation function.

In this paper our novel intuitive approaches are proposed in order to enrich statistical matching method based datasets and thus increase available predictions.

Our solution

The CRISP-DM data mining methodology is used in the below solution, therefore the number of possibilities have been tried and rejected due to its nature.

The final solution consists of the following steps:

- creating transformed and enriched datasets;
- modelling by subproblems and by datasets;
- ensembling the predictions of above models.

Datasets

Available datasets were considerably noisy and there were no confidence values for predictions so the preprocessing of datasets were regarded important. There are 2 small, 2 large and 2 medium data sets for training and testing. In the training phase of the final solution , the following datasets were used:

- Original datasets supplemented an average variable based on all other predictions
- Distribution-based datasets

    - Firstly from each of the six datasets was selected the 100 most accurate models.
    - Thereafter, we sorted predictions of each models: the highest value of a given row was the first input field, the lowest in the last field. Thus the predictions of different models were in the same position of given rows. In this way we lost a lot of information (the error of models based on this dataset was greater) but these models can be learned other characteristics of datasets. So the models based on this dataset proved to be effective in the combination of predictions.
    - In case of AUC type datasets, the target attributes were appreciated based on distribution of input variables.
    - The original six training datasets were transformed to one large uniform dataset with 100 input attributes.
- Datasets based on statistical matching for variables
    - We tried to find the connection between the attributes of the different datasets. First of all, we computed the empirical distribution function (conditional and marginal) for the attributes.
    - Then we calculated the distance between the distribution functions in different norms (sup, L_1, L_2) for each pair.
    - We generated random weights for the differences and chose the best 30. Then we searched the closest pairs of attributes.
    - Finally we accepted that two attributes are the same (or very similar) if they were supported by enough weight vector and the second closest attribute was far enough.

Modelling

By evaluation and comparison of the models, a cross-validation framework was created. The cross-validation random sample was carried out repeatedly, while it was possible to tell exactly which the most effective model combination method for each subproblem is. For the sake of enriched datasets our model combination methods used more predictions but in the validation step the original set of records were selected (typically 25% of the original dataset).

On the training dataset more than 17 different models have been tried, but the final solution contained only the following:

- linear regression with different variable selection methods

    - enter
    - backward
    - forward
    - stepwise
- Generalized linear models
- Feedforward neural networks

According to cross validation results, the following models were used in our final solutions.

Task on dataset M_RMSE_Score

Dataset	Model	Performance based on crossvalidation (RMSE)
Distribution-based datasets	Regression – Enter mode	871.3
Distribution-based datasets	Regression – Backward mode	871.1
Distribution-based datasets	Regression – Forward mode	871.3
Distribution-based datasets	Regression – Stepwise mode	871.4
Statistical Matching on L_RMSE_Train and M_RMSE_Train	Regression – Enter mode	869.8
Statistical Matching on L_RMSE_Train and M_RMSE_Train	Regression – Backward mode	869.8
Statistical Matching on L_RMSE_Train and M_RMSE_Train	Regression – Forward mode	869.8
Statistical Matching on L_RMSE_Train and M_RMSE_Train	Regression – Stepwise mode	868.7
Statistical Matching on L_RMSE_Train and M_RMSE_Train	Neural network (topology: two hidden layer with 20 and 16 nodes	867.3
Statistical Matching on L_RMSE_Train and M_RMSE_Train	Generalized Linear Regression	869.4
	Ensamble method: average (final solution)	867.11

Task on dataset L_AUC_Score

Dataset	Model	Performance based on cross-validation (Gini)
Distribution-based datasets	Generalized Linear Regression – rating prediction	0.6642
Statistical Matching on L_RMSE_Train and L_AUC_Train	Regression – Backward mode	0.6683
Statistical Matching on L_RMSE_Train and L_AUC_Train	Neural network (topology: two hidden layer with 20 and 16 nodes	0.6614
Statistical Matching on L_RMSE_Train and L_AUC_Train	Generalized Linear Regression	0.6685
	Ensamble method: average (final solution)	0.6711

References

[1] McCullagh, Peter, and Nelder, John (1989). *Generalized Linear Models*, Second edition, London: Chapman & Hall. ISBN 0-412-31760-5

[2] Nelder, John and Wedderburn, Robert (1972). *Generalized Linear Models*. Journal of the Royal Statistical Society, Series A, 135, pp. 370–384.

TEAM NAME: DreamTeam

TEAM MEMBERS: Gábor Szűcs

Affiliation: Budapest University of Technology and Economics

Medium Data Sets
medium RMSE:
Linear regression of the whole set of prediction variables

medium AUC
Logistic regression (logit) of the whole set of prediction variables

Large Data Sets
large RMSE:
Linear regression of the whole set of prediction variables

large AUC:
Variable selection, and after this:
Logistic regression (logit) of the selected prediction variables

AusDM Challenge 2009

Team EdR2 Final Report

November 17, 2009

Ed Ramsden

Principal
Sensorlytics LLC
Hillsboro, Oregon, USA
www.sensorlytics.com

earamsden@verizon.net

Background:

The goal of the AusDM 2009 challenge was to encourage the discovery of new algorithms for ensembling or 'blending' sets of expert predictions. Ensembling is the process of combining multiple sets of expert predictions so as to result in a single prediction of higher accuracy than those of any of the individual experts. From previous data mining competitions such as the Netflix Prize, it has become apparent that for many predictive analytics problems, the best approach for maximizing prediction accuracy is to generate a large number of individual predictions using different algorithms and/or data, and ensembling these sub-results for a final prediction.

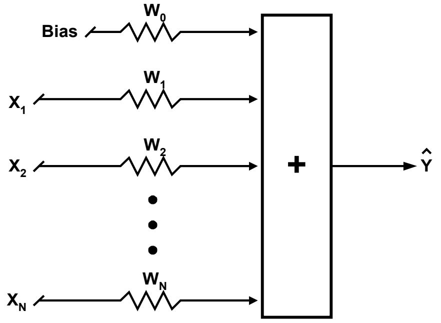
The AusDM 2009 challenge organizers provided sets of predictions obtained from the two leading teams in the Netflix Prize competition; Belkor's Pragmatic Chaos, and The Ensemble. For the RMSE portion of the challenge, three data sets were provided, a small set with 30,000 sets of predictions from 200 experts (different algorithms or variations), a medium set with 40,000 sets of predictions from 250 experts, and a large set with 100,000 sets predictions from 1151 experts. The three data sets each were evenly divided into a scoring subset containing only the individual expert predictions and a training subset containing both the expert predictions and the actual values for training. The training values were obtained from the Netflix Prize dataset by the organizers of the AusDM Challenge.

This report mainly describes on my experiments and results using the RMSE data sets and scoring criterion. A simple blending model was also applied to the AUC data sets.

1. Linear Perceptron Blender:

My initial focus was on using neural network (NN) methods for blending results. I first tried using a classical two layer NN (Multilayer Perceptron) with sigmoid first-layer units and a linear output unit, trained using back-propagation, but was unable to find sets of learning parameters that resulted in a 'good' blend (better than the 'best 10 experts' benchmark). At this point I tried an even simpler NN, the single-layer linear perceptron. This is perhaps the simplest possible NN structure:

Figure 1 – Linear Perceptron



The single-layer linear perceptron provides an output that is a linear weighting of the various inputs, and provides a model structurally identical to a linear regression, although parameters may vary depending on the details of the training process. The algorithm for training a single-layer linear perceptron is straightforward:

Linear Perceptron Training Algorithm:

```
'  X(i,j) is array of inputs, by exemplar and predictor
' Y(i) is array of training outputs, by exemplar
' Eta is the learning rate (Initially set to 6E-10 / # of predictors)
' Eta_Decay is decay rate for learning constant after each epoch (0.999)
' W(j) is a weight vector that defines the linear mix of the predictors
' W(0) is a bias term (weighted by 3000, a 'typical' input value

For Epoch = 1 to Max_Epochs
          For I = 1 to Max_Exemplars

                     ' estimate output for exemplar
                     Est = W(0) * 3000
                     For J = 1 to Max_Predictors
                               Est +=  W(J) * X(I, J)
                     Next J
                     Err = Y(I) - Est

                     ' train weights from error
                     W(0) += Err * Eta * 3000
                     For J = 1 to Max_Predictors
                               W(J) += Err * Eta * X(I, J)
                     Next J
          Next I
          Eta *= Eta_Decay
Next Epoch
```

Initially, the weights were set to 1/MAX_PREDICTORS, except for W(0), which was set to zero. This resulted in an initial model of averaging all predictors.

Some experimentation was needed to find appropriate values for Eta and Eta_Decay, as well as the maximum number of epochs to train for. The number of epochs was finally chosen so that a total of 50,000,000 exemplars would be presented, regardless of the number of exemplars in the training set (ranging from 15,000 for the small set to 50,000 in the large set). One key to obtaining good generalization performance seemed to be to train the network relatively quickly at first, and more slowly in successive iterations. Having the W(0) constant bias provided a slight improvement in both triaining and scoring accuracy.

The following table shows the training RMSEs seen for each data set as well as the scoring RMSE for the small data set:

Data Set	Bias Term?	Train RMSE	Score RMSE
Small	N	861.72	879.19
	Y	861.63	879.10
Medium	Y	860.75	-
Large	Y	864.63	-

While the perceptron yielded good results compared to other simple methods such as linear regression, I found it to be unsatisfying in that the coefficients do not yield a great deal of insight into the blending process and seem to be numerically ill-formed, in that a great deal of the signal from the individual predictors mutually cancels out. For example, in the fully trained network for the small dataset, approximately 80% of the numerical value of the predictors is expended in cancellation! In a classical linear regression, this situation would make a model's results highly suspect.

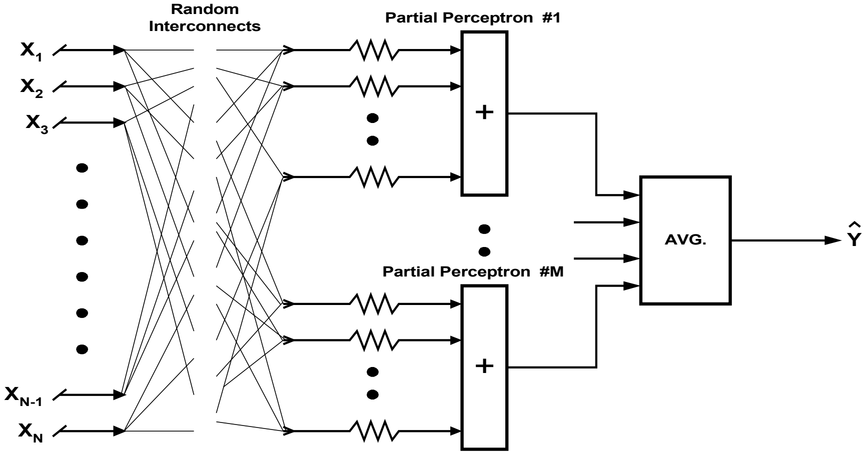
2. Variations:

I also tried a number of variations on the basic perceptron technique described above. In a few cases, these attempts yielded marginal RMSE improvements, but more often resulted in a loss of predictive accuracy. The best improvements hardly justified the additional model complexity.

Randomly-connected Partial Perceptron Ensemble:

Inspired by Phil Brierley's technique of combining a number of linear regressions each covering only a fraction of the total predictors, I performed an analogous experiment using perceptrons. I averaged the result of an ensemble of 6 perceptrons, each with its inputs connected to roughly 50% of the available experts, as illustrated schematically in Figure 2. Each perceptron was trained independently of the others, and their outputs were averaged together only when used for producing blended predictions. The results were not significantly different than those yielded by the single perceptron blending model (879.11 vs. 879.10 for test score).

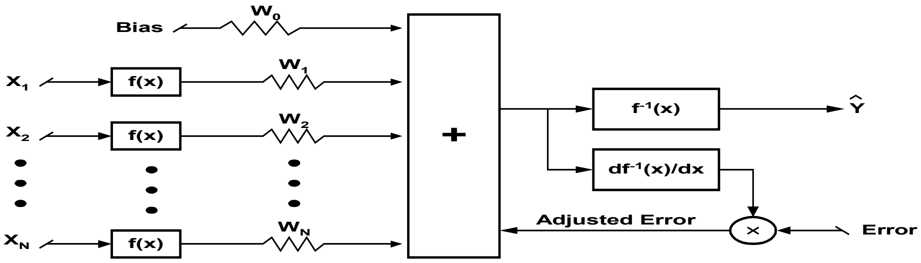
Figure 2 – Randomly Connected Partial Perceptron Ensemble



Transformation of Variables:

Another idea was to transform the data so that the perceptron might have a different view of it. This might be used in two ways. First, the transformed data might enable a perceptron to yield a better blend. Second, by transforming the data in a number of different ways, and training different perceptrons on each transformation, the ensemble of the results might generalize better than those of a single perceptron. A schematic representation of a perceptron with data transformation is shown in Figure 3.

Figure 3 – Variable Transformation

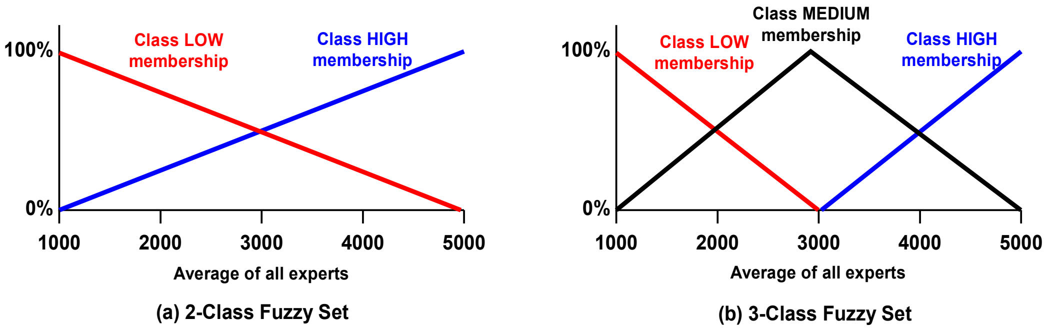


To ensure that the perceptron was training to minimize RMSE referenced to the original data (and not the transformed data – which could be significantly different), the inverse transformation was applied to the output. To train the weights, this also meant that the derivative of the inverse transform needed to be calculated to back-propagate the error signal. The need to select transformation functions that were both continuous and had inverses that were both continuous and continuously differentiable limited the selection of functions. To test this idea, I implemented this scheme using X^2 and sqrt(X) as transform functions. Neither yielded significantly different test score RMSEs than the original basic linear perceptron. When these two results were averaged with those of the linear perceptron, however, the ensembled results got a test score RMSE of 878.91, about a 0.19 improvement over that of the basic perceptron.

Fuzzy Perceptron:

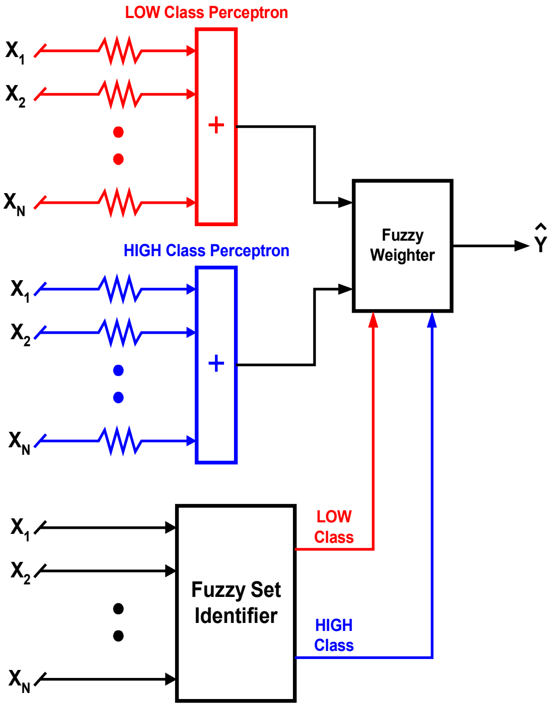
This ensembling method was based on the hypothesis that a single perceptron model would not be optimal over the entire range of input values (1000-5000). A membership function was defined that categorized fuzzy membership into Low and High classes (for a 2-class model) and Low, Medium, and High classes (for a 3-class model).

Figure 4 – Fuzzy Sets, 2-class (a) and 3-class(b)



Each membership class was assigned its own perceptron to be trained and to be used for evaluation. Figure 5 shows the arrangement for the two-class case.

Figure 5 – Fuzzy Perceptron (2-set)



The degree of membership in each class was used to control both evaluation and training. In the evaluation phase, the outputs of the perceptrons representing each class were linearly combined proportional to the identified class membership. Class membership of an input vector (X) was identified on the basis of its average value, although other metrics (e.g. variance) could conceivably be used. During training, the class membership information would be used to proportion the back-propagated error among the various class perceptrons. For example, an low-valued input exemplar, with for example an average of 2000, would generate a 75% LOW class membership and a 25% HIGH class membership (assuming 2 classes). The total output would be 0.75 x LOW_class_perceptron + 0.25 x HIGH_class_perceptron. During training, the low class perceptron's training rate would be 0.75 x ETA while the high class perceptron's training rate would be 0.25 x ETA for that particular exemplar.

This elaboration allowed for a better fit to training data than the simple perceptron model, but unfortunately did not generalize as well. The following table shows training and test score RMSEs on the small dataset.

Model	Training RMSE	Score RMSE
2-Class Perceptron	857.14	880.69
3-Class Perceptron	854.74	881.67

3. A More Intuitive (But Less Accurate) Blending Model:

I also explored other methods, none of which beat the prediction accuracy of the linear perceptron. While most of these attempts yielded were total dead-ends, in terms of both results and insight, one method stood out as providing an intuitively and simple model while providing results better than vanilla linear regression. This method was to search for a *best combination of experts*.

Selecting a best combination of experts is a different task than selecting the set of best experts in that the objective function being optimized during 'training' is a function (the average) of the entire set. For example, an expert who is consistently low could be an optimal combination for an expert who is consistently high despite the case that neither expert considered individually may be very good. While finding *the* optimal set is a difficult problem because of the number of potential sets that could exist (2.25E16 possible sets of 10 experts can be selected from a group of 200), I found that a simple greedy optimization technique based on RMSE provided reasonable results.

Greedy Best-Combination Algorithm:

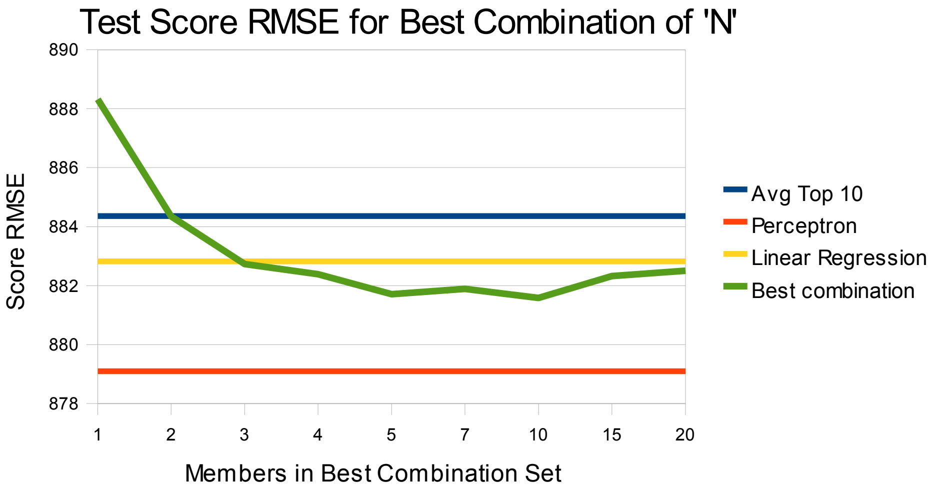
```
1)  Select initial random set of experts
2)  evaluate RMSE (of their combined ratings)
3)  select an expert 'X'  within the set to replace
4)  select an expert 'Y' outside the set as a replacement
5)  re-evaluate RMSE with expert 'Y' substituted for expert 'X'
6)  if new RMSE is better, then let 'Y' remain in set, otherwise return 'X' to set.
7)  Repeat from 3 until no changes occur for a while
```

Because the above algorithm employs a greedy replacement strategy, it has a good chance of becoming trapped in local minima. Despite this drawback it delivers good predictive performance with a very simple resulting model – namely that of averaging a small number of expert predictions.

One surprising result of this algorithm is the size of the set of of experts needed to produce substantial improvement. A combination of only 3 experts was needed to provide a predictive accuracy exceeding that of linear regression on the entire set of experts. The performance of 'best' combinations of various number of experts, can be seen in the table below. Figure 6 graphically depicts the performance. Note that accuracy decreases (RMSE increases) when the number of experts is increased bast 10.

Method	Train RMSE	Score RMSE
Average of 10 Best Experts	878.45	884.36
'Vanilla' Linear Regression	n/a	882.82
Perceptron	861.63	879.1
Best Combination of 1 (Best Expert)	878.49	888.32
Best Combination of 2	872.29	884.35
Best Combination of 3	870.08	882.73
Best Combination of 4	869.90	882.39
Best Combination of 5	869.47	881.71
Best Combination of 7	869.14	881.89
Best Combination of 10	869.01	881.58
Best Combination of 15	868.98	882.32
Best Combination of 20	869.10	882.51

Figure 6 – Graphical Summary of Best-Combination-of-N Blender Performance



Although several thousand iterations may be needed to converge on a final result, each iteration can be performed quickly, especially if the model is updated incrementally with entering and leaving experts. Additionally, the majority of the model convergence occurred within the first few hundred trials.

Because the model evaluation and expert selection processes are independent, it is straightforward to use this technique with prediction combination methods other than simple averaging. Some alternate (but untried) possibilities include:

- Median
- Geometric Mean
- Root-mean-squared average
- 'Trimmed' Average (toss out some number of high and low expert predictions)

4. Application to AUC Criteria

I also tried a few experiments on the AUC criteria that were based directly on my RMSE approaches. The first was to try training the linear perceptron using the [1,-1] target values in the AUC data sets. This did not work very well, resulting in a test score GINI of only 0.8727, which was actually worse than the average of the top 10 experts. I then adapted the 'average of best set of predictors approach described in the previous section, using GINI as an optimization objective function instead of RMSE. This worked better, with the average of a 'best set' of 10 predictors yielding a 0.8760, a slight improvement over the hill climbing benchmark. Although the performance was hardly optimal, this exercise demonstrated the generality and flexibility of the 'best set' approach to blending.

5. Summary:

The linear perceptron, an extremely simple neural network/machine learning technique provided a competitive degree of ensembling performance, exceeding that of linear regression, and approaching the best demonstrated publicly to date in the AUSDM 2009 challenge. This method has the advantage of simplicity, as the core training routine was coded in fewer than 25 lines of VB.NET 2008.

Despite the perceptron's effectiveness and simplicity, it does not provide simple insights into the model it develops internally. For this reason I explored conceptually simpler blending techniques and found that a greedy combinatorial search could be used to identify small sets of predictor variables that provided good results when simply averaged. While the accuracy of this technique was significantly less than that of the perceptron method, it exceeded more sophisticated methods such as blending on the basis of a linear regression model, and the resulting model is easy to understand.

Appendix – Summary of Ensembling Performance (RMSE):

The following table lists results of methods I tried that are described in this report, along with a few benchmark methods from the leaderboard. Each method also has an 'relative performance' rating assigned where 0% is defined as the performance of the overall average of all experts (naïve blending) and 100% is defined as the performance of the high scoring method at the time of writing (Team Optibrebs, Nov 15, 2009).

Method	Description	Train Score	Test Score	Relative Performance
BEST	**Top-of-Leaderboard (Team Optibrebs)**	**-**	**877.91**	**100.00%**
A	Average of methods **B,C,E**	-	878.91	93.25%
B	Perceptron, sqrt(x) transform	862.51	879.07	92.18%
C	Perceptron w/bias term	861.63	879.10	91.97%
D	Perceptron ensemble (6 x 50% cover)	-	879.11	91.91%
E	Perceptron, x^2 transform	863.68	879.11	91.91%
F	Perceptron no bias term	861.72	879.19	91.37%
G	2-Class Fuzzy Perceptron	857.14	880.69	81.28%
H	Best Combination of 10	869.01	881.58	75.29%
I	3-Class Fuzzy Perceptron	854.74	881.67	74.68%
J	Best Combination of 5	869.47	881.71	74.41%
K	Best Combination of 7	869.14	881.89	73.20%
L	Best Combination of 15	868.98	882.32	70.31%
M	Best Combination of 4	869.90	882.39	69.84%
N	Best Combination of 20	869.10	882.51	69.03%
O	Best Combination of 3	870.08	882.73	67.55%
P	Linear Regression	-	882.82	66.94%
Q	Best Combination of 2	872.29	884.35	56.65%
R	Average of Best 10 Experts	871.02	884.36	56.58%
S	Best Expert	878.49	888.32	29.94%
BASELINE	**Naïve (Average all Experts)**	**878.45**	**892.77**	**0.00%**

The Green Ensemble Report for AusDM 2009

Ashkan Sami
asami [at] ieee.org

Ehsan Khoddam Mohammadi
Mohammad J. Mahzoon
Ali Askari
Ashkan Ghaffari Nejad
{khoddam,mahzoon} [at] cse.shirazu.ac.ir
ashk365 [at] gmail.com
ali_askari [at] yahoo.com

CSE and IT Department
School of Electrical and Computer Engineering
Shiraz University
I. R. IRAN

November 2009

1 Introduction

This report contains a quick review of investigations and methods which are applied to **AusDM 2009** challenge by Green Ensemble team. **Green Ensemble** is group of undergraduate students at school of electrical and computer engineering at Shiraz university under supervision of Dr. Ashkan Sami, Assistant Professor of CSE and IT Department at School of Electrical and Computer Engineering.

The objective of AusDM challenge is to combine results of predictor models to obtain better results than any individual model. Three sizes of data sets, small, medium and large are available for two tasks, RMSE and AUC challenge. Each data set splits into train data set and score data set. Train data set has predictions of models. In AUC challenge, the goal is to classify the movies into two classes; 1 and -1.

2 Data Analyze

In this part we discuss some important statistical properties of the data sets. In RMSE challenge, each data set presents results of different prediction models for movies. Ranks are 1, 2, and 5 which are multiplied by 1000. Figure 1 shows the actual distribution of ranks for movies in the medium data set for RMSE challenge. Most ranks are distributed over ranks 3 to 5. In fact predicting these ranks will have direct effect on accuracy of our ensemble. Other data sets have the same form of distribution. Figure 2 contains the same type of information for small data set.

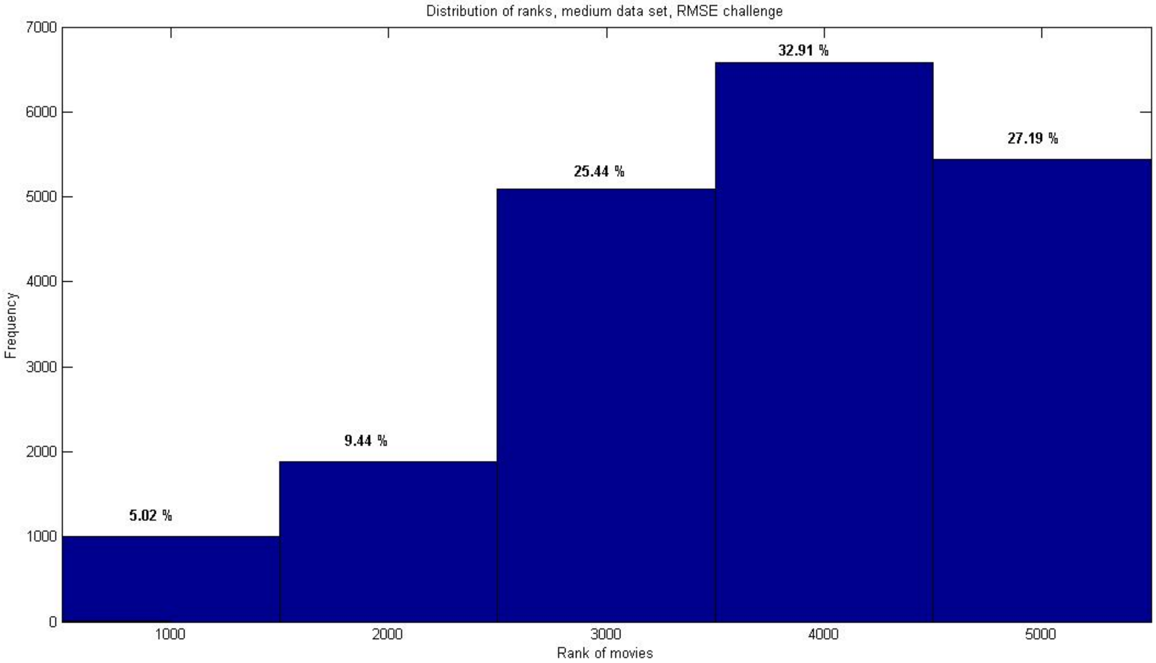


Figure 1: Actual Distribution of ranks for medium data set

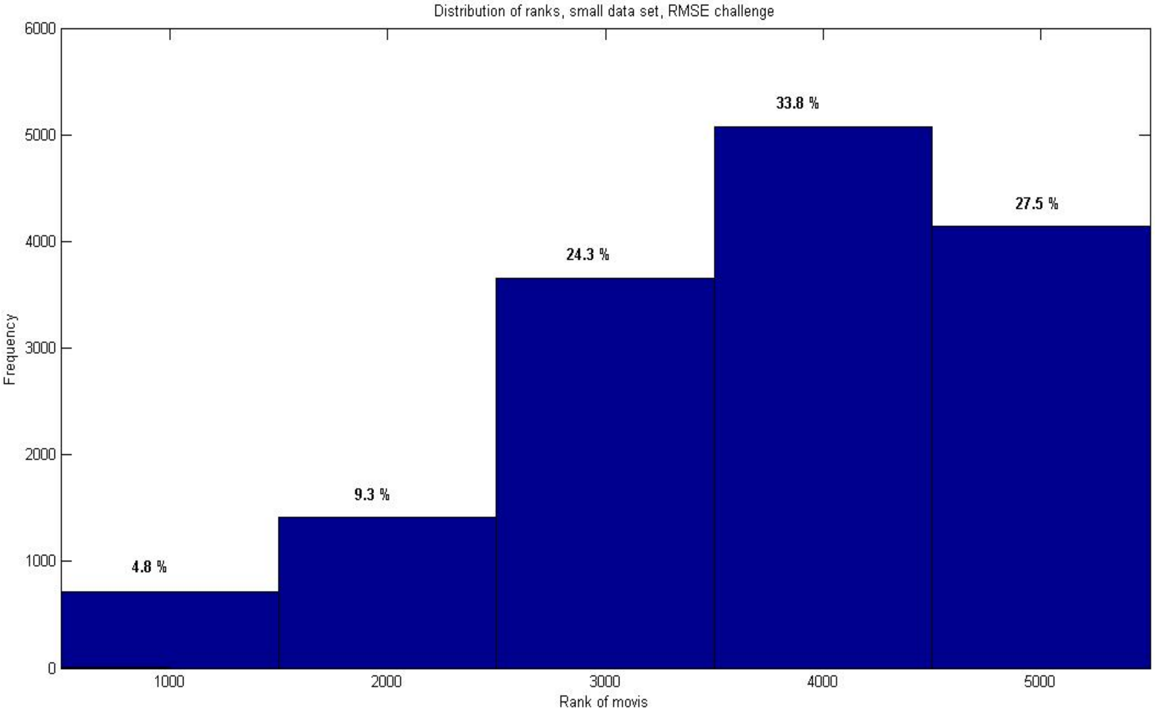


Figure 2: Actual Distribution of ranks for small data set

Next property which is investigated is how good models predict ranks. The distribution of RMSE of models for predicting each rank is shown in Figure 3 to Figure 7. Figure 3 shows that models are predicting with high order of magnitude of error when they are predicting movies with rank 1000 (small data set for RMSE). Generally models give prediction 3000 to movies with rank 1000 and 2000. In other words, it is not clear how to distinguish among 1000, 2000 and 3000 when the output of the model is around 3000. Stated differently, it is difficult to classify among 1000, 2000 and 3000 ranked movies. In addition to RMSE, some statistical tests and measures like K-S test, KL divergence and mutual information also show that these three classes are overlapping and hard to distinguish. So we investigate regression methods.

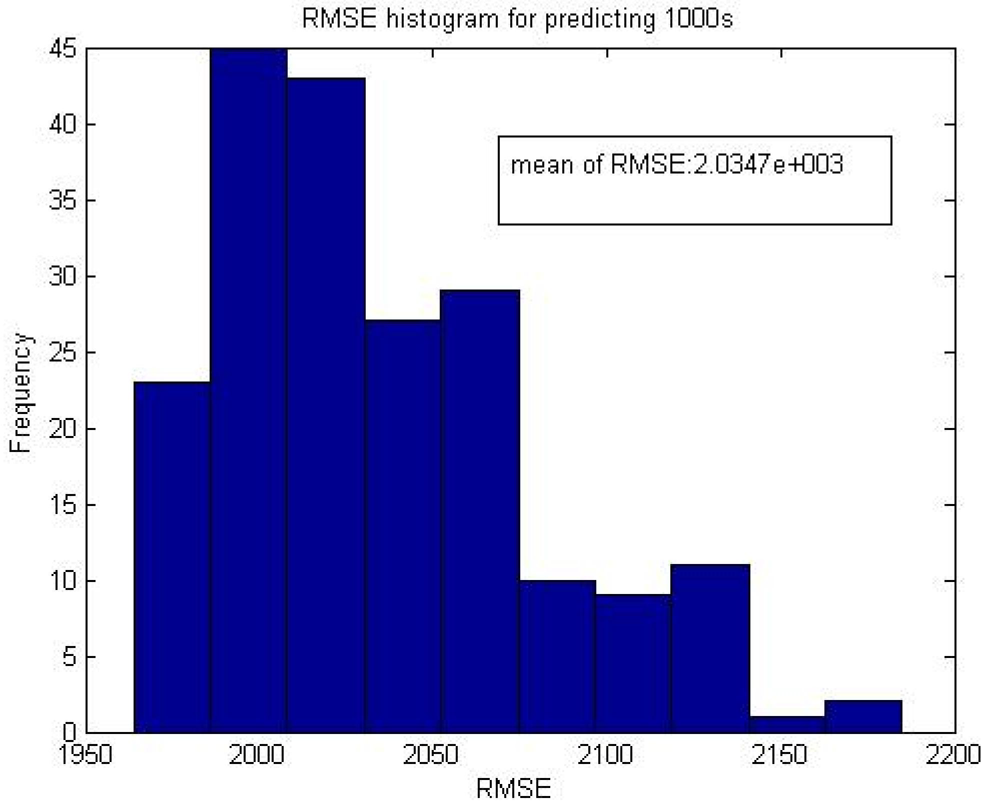


Figure 3: Distribution of RMSE when actual rank is 1000

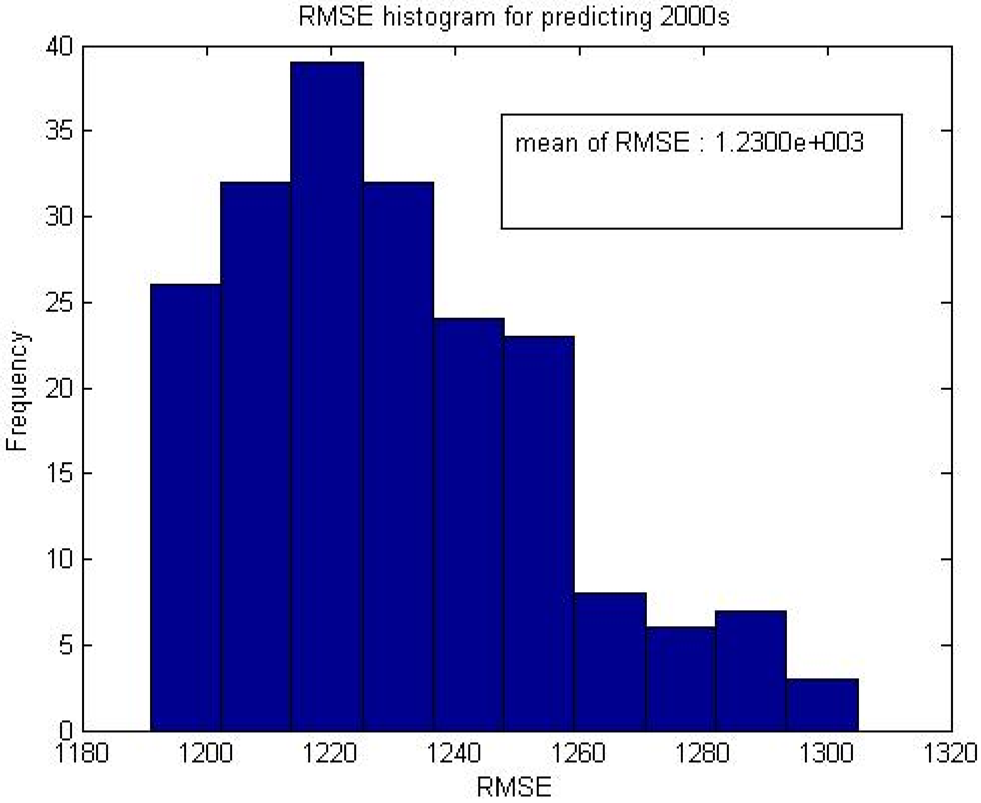


Figure 4: Distribution of RMSE when actual rank is 2000

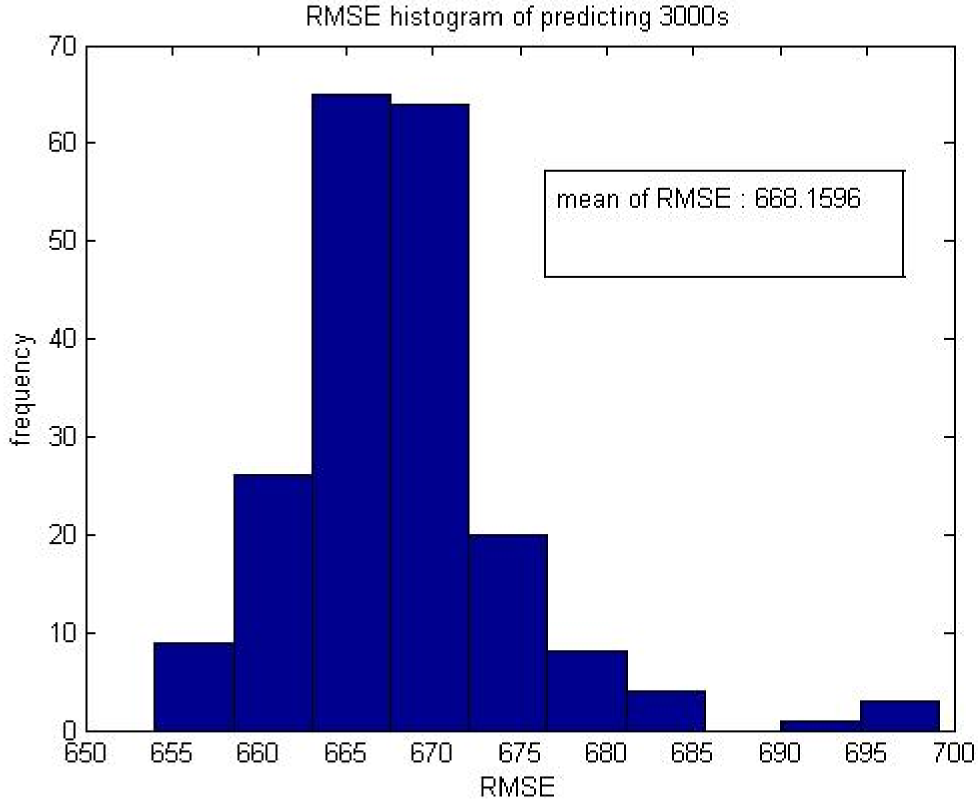


Figure 5: Distribution of RMSE when actual rank is 3000

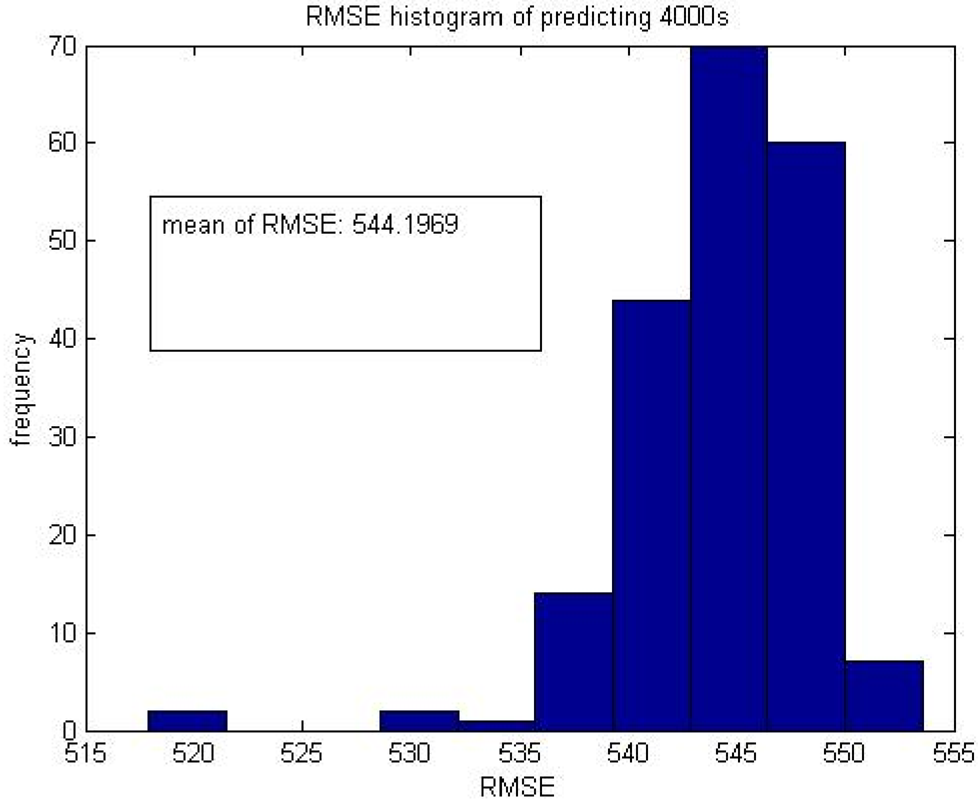


Figure 6: Distribution of RMSE when actual rank is 4000

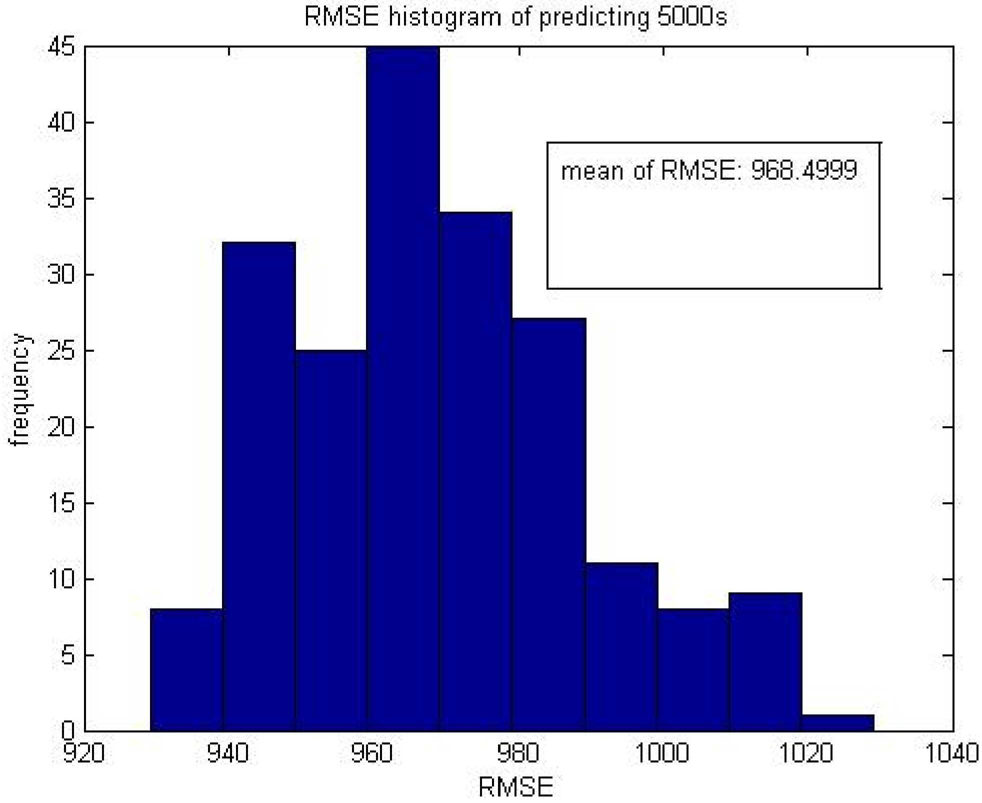


Figure 7: Distribution of RMSE when actual rank is 5000

3 Review of Methods

We have tried various methods. The below list presents a sample of the tools and techniques deployed to build ensemble.

RMSE CHALLENGE

I. Genetic Algorithm: finding optimum set of weights for averaging predictions using cross over and mutation heuristics applied to randomly generated set of weight. (Davis, 1991)

II. Ensemble Selection: hill-climbing approach for finding optimum bag of models (Caruana, Niculesco-Mizil, Crew, & Ksikes, 2004)

III. Feed-Forward ANN: fitting with two-layered FF-ANN (Menhaj & Hagan, 1994)

IV. Least-squares regression: using pseudo-inverse matrix of predictions to find best weights which could model train data set, same weights apply to test data set. (Chatterjee & Hadi, 1986)

V. Stepwise regression: same as Ensemble Selection but each model added to bag according to p-value and F-statistic. Stepwise regression is our best approach according to feedback of small size data set. (Draper & Smith, 1981)

AUC CHALLENGE

I. Regression Tree (Breiman, 1993)

II. Multinomial logistic regression: our best approach (McCullagh & Nelder, 1990)

Methods and algorithms are implemented in C#, JAVA and MATLAB. Built-in functions, Statistics toolbox and Curve Fitting toolbox of MATLAB 7.6 were used.

REFERENCES

Breiman, L. e. (1993). *Classification and Regression Trees.* Boca Raton: Chapman & Hall.

Caruana, R., Niculesco-Mizil, A., Crew, G., & Ksikes, A. (2004). Ensemble Selection from Libraries of Models. *Proceeding of ICML '04* .

Chatterjee, S., & Hadi, A. S. (1986). Influential Observations, High Leverage Points, and Outliers in Linear Regression. *Statistical Science* , 379- 416.

Davis, L. (1991). *Handbook Of Genetic Algorithms.* Van Nostrand Reinhold Company.

Draper, N., & Smith, H. (1981). *Applied Regression Analysis* (2 ed.). John Wiley and Sons.

McCullagh, P., & Nelder, J. (1990). *Generalized Linear Models* (2 ed.). Chapman & Hall/CRC Press.

Menhaj, M. b., & Hagan, M. (1994). Training Feedforward Networks with Marquardt Algorithm . *IEEE TRANSACTION ON NEURAL NETWORKS* , 989-993.

Team Name: Innovative Analysts

Team Member: 1. Priya
2. Navin

AUC:

Introduction:

Contest problem is to predict the target variable using the model scores as independent variables in both medium and large datasets. We ensembled the scores of 50 samples using logistic regression technique.

Ensemble Logistic Model:

- Generate 50 random samples of 30,000 records, say S_1, S_2, …, S_{50}
- For each of the samples, we developed stepwise logistic model with the key features identified in the single logistic model using log-link function. Let the probabilities be P_1, P_2, …, P_{50}
- The probabilities were combined to obtain the ensemble logistic score using methodologies like
    - Average(P_1, P_2, …, P_{50})
    - Average (Min(P_1, P_2, …, P_{50}), Max(P_1, P_2, …, P_{50}))

RMSE:

- Logistic regression model was built for hundreds of samples with selection = cp criteria. The score with best cp was selected from each interation and then we ensembled the scores.

A Simple Ensemble Technique

Team: ISMLL
Members: Krisztian Buza and Lars Schmidt-Thieme
buza@ismll.de, schmidt-thieme@ismll.de
Information Systems and Machine Learning Lab
University of Hildesheim, Germany, European Union

November 21, 2009

Abstract

In this paper we present our solution for the *AusDM Analytic Challenge 2009*. We applied a simple approach based on a sophisticated variable selection technique. The basic idea is looking at the pairs of models and searching for those pairs where the errors of the two models compensate each other.

As the final results of the challenge are unknown at the time of writing this paper, we can only report preliminary results on the small data set: according to this, despite its simplicity, our technique is less than 0.1% worse than the currently best method[1].

1 Introduction

The topic of the *AusDM Analytic Challenge 2009* was ensembling: “Ensembling, Blending, Committee of Experts are various terms used for the process of improving predictive accuracy by combining models built with different algorithms, or the same algorithm but with different parameter settings.”[2] This technique is frequently used to improve predictive models, see e.g. [4, 3, 2]. On the one hand some fundamental reasons are known, why ensembles work better than single models (these reasons are described for example in [1]), on the other hand, how ensembles are “actually achieved in practice maybe somewhat arbitrary. One of the drawbacks in researching the problem is that you first have to generate a lot of models before you can even start. There have been numerous predictive modelling competitions that could potentially provide good data sets for such research - many models built by many experts using many techniques. The Netflix Prize is one such competition that has been on going for nearly 3 years.”

[1] 17th November 2009, 11:40 GTM

[2] All the cited text in the Introduction is from http://www.tiberius.biz/ausdm09/

In the Netflix challenge the task was to predict how users rate movies on a 1 to 5 integer scale (5=best, 1=worst). Participants of the challenge delivered real numbers as predictions and the RMSE (Root Mean Squared Error) between the predictions and the actual ratings was used as evaluation metric.

The Netflix challenge "recently finished, and the eventual winners were an amalgamation of several teams that somehow combined their individual model predictions. Over 1,000 sets of predictions have been provided by the two leading teams (who actually ended up with the same score), The Ensemble and BellKor's Pragmatic Chaos."

In the AusDM Analytic Challenge 2009 the task was to build an ensemble over these provided predicitions. There were two subtasks "one to develop a method to predict a continuous value" (RMSE-task) "and the other to predict a binary value" (AUC-task). We only participated in the RMSE-subtask.

For the participants of the challenge 6 datasets were provided: for both tasks there were small, medium and large datasets. During the challenge feedback was given on the small dataset, but the final results were determined based on the performance on the medium and large datasets (for which no feedback was provided during the challenge).

2 A Simple Ensemble Technique

Ensembles work better than a single predictive model in those situations when different predictive models have different error characteristics and their errors can compensate each other. We build our simple ensembling technique based on this observation. In fact, we applied a variant of stacking with linear regression as meta learner.

For simplicity we will describe our technique in context of linear regression (as meta learner) and RMSE as evaluation score, but the same technique work with arbitrary classification or regression models (like SVMs, Bayesian Networks, Decision Tree, etc.) as meta learner and various evaluation scores (like accuracy, AUC, etc.), thus our technique is quite general in spite of the current description which is very specific. The only assumption we make, is that *each predictive model* assisting in the ensemble *delivers a prediction for the target.*

While learning, we use a technique similar to 10 fold crossvalidation: first we devide the train data into 10 splits, which are numbered 0, 1, 2. . . , 9. In the 1st fold the splits 0, 1, 2, 3 and 4 serve as *basic train set* and the rest as *internal evaluation set.* In general, in the kth fold, the *basic training set* contains the splits k mod 10, $(k + 1)$ mod 10, $(k + 2)$ mod 10, $(k + 3)$ mod 10 and $(k + 4)$ mod 10, and the *internal evaluation set* contains the rest.

What we describe from now on, is done for each fold. Our ensembling technique looks at the pairs of models and searches for those pairs where the errors of the two models compensate each other. For the simplicity of the description, we introduce the *model-pair graph.* The *model-pair graph* is a weighted, undirected, graph. Each vertex v of this graph corresponds to one of the models that assist in the ensemble. All the vertexes are connected, i.e. the graph is a complete graph. Each edge $\{v_i, v_j\}$ has a weight reflecting how "good" is the *combination* of the models v_i and v_j, that is how well they compensate each other's errors. The weight of $\{v_i, v_j\}$ is determined on the *basic train set*: we regard the average of the outputs of the models v_i and v_j as prediction for the target variable, and we calculate the RMSE between this average and the actual value of the target. This RMSE score will be the weight of the edge $\{v_i, v_j\}$.

We process the edges in order of their scores, beginning with the best edge. (As in case of RMSE the smaller values indicate the better predictions, we process the edges in *ascending* order with respect to their weights.) Let M denote a set of models, initially M is the empty set. Let s denote the estimated quality of our ensemble based on the models in M. Initially s is the worst possible quality score, i.e. $s = +\infty$ (positive infinity), as in case of RMSE scores. Let $\epsilon = 0.25$.

For each edge $\{v_i, v_j\}$ the followings have to be done:

1. If both $v_i \in M$ and $v_j \in M$, then proceed for the next edge,

2. else

   (a) $M' = M \cup \{v_i, v_j\}$.

   (b) Train a multivariate *linear regression* over the outputs of the models contained in M' using the *basic train set*, and evaluate it on the *internal evaluation set*. Let s' denote the evaluation score. If s' is better than s at least by ϵ (i.e. if $(s' + \epsilon) < s$ for RMSE), than $M \leftarrow M'$ and $s \leftarrow s'$.

   (c) Proceed for the next edge.

This way for each fold we select a set of models M, or being more exact: we get M_0, M_1,..., M_9 for the folds 0,1,...,9. Let N denote the set of such models that are contained at least $n = 4$ times among the selected models, i.e. N is a set of such models that are contained in at least $n = 4$ sets among the sets M_0,M_1,...,M_9.

Finally we train a *linear regression* over the output of the models in N on the whole training set and apply this for the unlabeled data. We built our implementation on the linear regression of WEKA[3] software package [5].

The hyperparameters (ϵ and n) are to be learned on a hold-out subset of the train data that is disjoint both from the basic train set and from the internal evaluation set.

3 Preliminary Evaluation and Outlook

As the challenge is not yet finished at the time of writing this paper, we only report preliminary results that were achived on the small data set, where feedback was given to the participants. On the score set of the small data set, the currently[4] best method (Optibrebs) achieved RMSE of 877.907. Our technique

[3] http://www.cs.waikato.ac.nz/~ml/index.html

[4] 17th November 2009, 11:40 GTM

Table 1: Experiments

Method	Performance (RMSE)
Optibrebs (currently best)	877.907
Simple Ensemble Technique (Our method)	878.612
Average Top 10 Experts (Baseline)	884.359
Best Expert (Baseline)	888.32
Average All Experts (Baseline)	892.77
Worst Expert (Baseline)	994.118

had RMSE of 878.612, which is less than 0.1 % worse, than the RMSE of the best model, whereas the best baseline method (average top 10 experts) achived RMSE of 884.359. The performances of our technique and the baseline methods and the currently best method are summarized in Table 1.

The simple technique presented in this paper can be improved in several directions, for example: (i) one can consider to check not only pairs, but also triples of variables, (ii) new edge weighting strategies can be introduced, (iii) the technique can simply be applied for other evaluation measures than RMSE. As future work, we would also like to explore how general this simple ensembling technique can be applied, as a first step on can measure the performance on other data sets.

Acknowledgements. This work was co-funded by the EC FP7 project MyMedia under the grant agreement no. 215006. For inquiries please contact info@mymediaproject.org.

References

[1] T. G. Dietterich (2000): *Ensemble Methods in Machine Learning*, MCS 2000, LNCS 1857, pp. 1-15., Springer

[2] Y. Peng (2006): *A novel ensemble machine learning for robust microarray data classification*, Computers in Biology and Medicine, Volume 36, Issue 6, Pages 553-573

[3] C. Preisach, L. Schmidt-Thieme (2008): *Ensembles of Relational Classifiers* Knowledge and Information Systems Journal 14(3)

[4] A. C. Tan, D. Gilbert (2003): *Ensemble machine learning on gene expression data for cancer classification*, New Zealand Bioinformatics Conference

[5] Ian H. Witten, Eibe Frank (2005): *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)* Morgan Kaufmann, ISBN 0-12-088407-0

Aggregating Zoomed MLP Neural Networks Combines Experts Scores for the NetFlix Binary Decision Problem

Paulo Adeodato[1,2] , Lucas Gallindo[1], Víctor Braz[1,2], Adrian Arnaud[1]

[1] NeuroTech Ltd., Recife-PE, Brazil
[2] Center for Informatics, Federal University of Pernambuco,
Recife-PE, Brazil
{Paulo,Lucas,Victor,Adrian}@neurotech.com.br,

Abstract. This manuscript presents an approach based on *n*-fold cross validation and zooming carried out to solve the AusDM 2009 Challenge for combining experts' scores for the NetFlix binary classification problem. The approach consisted of aggregating zoomed MLP systems trained in an *n*-fold cross-validation process. The zooming approach is the decomposition of the decision in two levels: the first level scored the whole set and the central tercile from the first level (most mixed) passed through another decisor trained only on that part of the labeled sample. Several constraints on the data processing are still to be overcome.

1 Introduction

The combination of the opinion of experts has been used in human committees for improving the quality of decisions for a long time and the idea has been formally introduced to the decision support systems by Wolpert [1] back in 1982. Few years later, Leo Breiman [2] systematized the combination approaches. In recent years, the winner approaches in data mining competitions have confirmed that committees are the far better than single solutions. This year the NetFlix winners have consolidated this approach and motivated the current challenge [3].

2 Proposed Approach

The approach proposed by our team combines the scores of several instances of MLP architectures as had been done in previous competitions [4], [5]. Here the MLP systems are trained via an *n*-fold cross validation process.

Each MLP system consists of two cascaded MLPs: the first level MLP is trained on a random sample of the labeled data, while the second level MLP is trained on the central tercile of another sample. The idea is to separate the easiest parts of the examples (top and bottom terciles) with the first level system and score the central tercile with another system trained focused on these mixed examples.

The sequence of steps of data processing for the proposed approach is:

Stage-1: Training

1. Variables selection
2. Variables creation
3. *n*-fold cross-validation
    a. Training of first level MLPs with ¼ of the modeling sample
    b. Scoring the other ¾ of the sample
    c. Training the second level MLPs with the central tercile (¼)
4. Performance evaluation on the *n* test sets: first and third terciles scored on the first level only while the second tercile scored also on the second level and re-scaled between the bottom and the top terciles

Stage-2: Usage

1. Score generation on the *n* trained Systems for the challenge data: First and third terciles scored on the first level only while the second tercile scored also on the second level and re-scaled between the bottom and the top terciles
2. Score rank calculation on each of the n systems
3. Selection of the median of the *n* systems for each example

To select the most adequate variables, first their univariate contributions were measured with the AUC_ROC metrics, once that all were numerical variables. The top ten were selected and from the remaining ones, the five most dissimilar ones were preserved. Due to computational constraints Variance Inflation Factor (VIF) could only be used for variable selection within the LeaderBoard dataset. The other sets had the similarity calculated by their correlation.

Some new input variables were created for capturing information from the existing experts' scores. Average, standard deviation, median and asymmetry of the scores were some of these variables. All had their AUC_ROC calculated and were preserved totaling 24 input variables.

To do the two-level training, the small and medium datasets were small for having completely statistically independent sets, considering that only MLPs had been implemented in the software platform. Therefore, training data were re-used in the two levels for these sets; only the large data set had statistically independent samples for training each level. The idea was to use logistic regression instead of MLPs because of the smaller need of examples and the speed of calculation but the software was not implemented yet.

3 Conclusion

Unfortunately, the software implementation needed for simulations took longer than expected and several constraints in data processing had to be faced for the challenge, just a few days from the deadline.

The dimension of the input space was quite large for executing the analyses needed in reasonable time. Therefore, correlation had to be used for redundancy detection.

MLPs demand too much data and forced the re-use of data in the two-level training.

Applying the approach above in the challenge, NeuroTech Research, Development and Innovation team (NeuroTech RDI) reached an AUC_ROC=0.877742 on the LeaderBoard dataset, slightly better than the hill climbing baseline. Despite some drawbacks, the results might be better for the larger datasets.

References

1. Wolpert, D.H.: Stacked generalization. Neural Networks, 5, 241–259, (1992)
2. Breiman, L.: Bagging Predictors. Technical Report No. 421, Department of Statistics, University of California at Berkeley, September (1994)
3. Australian Conference on Data Mining Challenge – AusDM 2009.: Last accessed November 2009, http://www.tiberius.biz/ausdm09/
4. Adeodato, P. J. L., Vasconcelos, G. C., Arnaud, A. L., Cunha, R. C. L. V., Monteiro, D. S. M. P., Oliveira Neto, R. F.: The Power of Sampling and Stacking for the PAKDD-2007 Cross-Selling Problem. International Journal of Data Warehousing and Mining (IJDWM), 4, 22-31, (2008)
5. Adeodato, P. J. L., Vasconcelos, G. C., Arnaud, A. L., Cunha, R. C. L. V., Monteiro, D. S. M. P.: A Systematic Solution for the NN3 Forecasting Competition Problem Based on an Ensemble of MLP Neural Networks. 19th International Conference on Pattern Recognition (ICPR), Tampa, FL, Dec (2008)

AUSDM 2009 Analytic Challenge

1. Team name: EnsembleMaster09
2. Team members: Seungho Huh, Taiping He
3. Company: SAS Institute
4. Description of the method:

   Since there are too many input variables, the main focus was to reduce the number of them. After experimenting with the SMALL datasets, we decided to use the following methods.

a) AUC - Gradient Boosting

Gradient Boosting is well known for its accuracy in classification. It also has intrinsic variable selection capability based on the Decision Tree feature. SAS Enterprise Miner has a node implementing this Gradient Boosting method. We used this node with the number of boosting iterations=50.

b) RMSE - LASSO

The LASSO is known as a good technique that naturally removes unimportant variables. SAS Enterprise Miner also has a node implementing the LASSO. We used this node with 5-fold cross validation.

EXL Team report on AUSDM 2009 competition submission

Team Structure

Sassoon Kosian (Lead)
Sahil Manocha
Preetesh Shukla
Anita Sachdeva

Blending Approaches

All the submitted scores are the result of blending using several approaches as outlined below. Each submission has a different set of approaches combined

1. Linear and Logistic Regressions on Bootstrapped sets
2. Linear and Logistic Regressions with coefficient blasting
3. Neural Networks
4. Hill climbing technique on random subsets

Blending of different approaches on final sets for submission used simple average on standardized scores.

Name of the team: The final say

Team members: Padmasini

Abstract: AUC

Joint Score:

The Joint score technique was to combine the predicted scores from various techniques to obtain the final prediction for each of the models.

- Logistic regression model was built to predict the target variable using model scores as independent variables in both medium and large datasets.
- Decision tree models were built for both medium and large datasets using the model scores as independent variables. TreeNet software was used to develop these models.
- Logistic regression model was built to predict the target variable using the top 10 variables in the order of AUC of model scores.
- The above 3 models were combined by averaging the scores and averaging the minimum and maximum of scores.

Abstract: RMSE

Decision tree model was built to predict the target variables using the independent variables. Treenet was used to build decision tree model.

albert2

Wojciech Stach and Lukasz Kurgan

Department of Electrical and Computer Engineering

University of Alberta, Canada

Introduction

We addressed both problems using an advanced kernel classifier/predictor. The unique characteristics of our solutions are customized two-step parameterization of the classifier/predictor and best first search based feature selection that was used to improve both efficiency and predictive quality. We utilized out-of-sample cross validation (CV) tests for the feature selection and parameterization on the training dataset to minimize overfitting. The task 1 was implemented using Support Vector Regression (SVR) predictor and task 2 using Support Vector Machine (SVM) classifier. The experiments were performed using WEKA platform (http://www.cs.waikato.ac.nz/ml/weka/) and Java-based scripting.

Task 1

Step 1. Initial SVR parameterization

- Small dataset was used
- Top 20 attributes were chosen based on ranking using their individual RMSE error with respect to the target ranking
- Using the 20 attributes, 4-fold cross-validated SVR learning with different parameters (cost parameter C, kernel type, kernel parameters) was carried out to minimize RMSE
    - best setup: C=0.1 ,RBF kernel, gamma=0.01 → RMSE = 873.5

Step 2.   Feature selection

- Features were ranked based on the RMSE error for the 4-fold CV on the training set (for medium and large datasets 10000 samples were randomly chosen) using individual attributes with SVR parameterized in Step 1
- Starting with the top ranked attribute, one attribute at a time was added and evaluated using RMSE (based on the 4-fold CV on the training set with SVR parameterized in Step 1). If a given attribute improved the RMSE then it was added to the selected attributes, otherwise it was rejected. A single scan through all ranked attributes was performed.
- The following number of attributes was selected for the corresponding datasets:
    - Small: 41
    - Medium: 57
    - Large: 109

Step 3. SVR parameterization

- For each training dataset using the selected features 4-fold CV based experiments were carried out to parameterize SVRs; the best setups for the corresponding datasets follow:
  - Small: C=0.1 ,RBF kernel, gamma=0.1
  - Medium: C=1, RBF kernel, gamma=0.01
  - Large: C=1, RBF kernel, gamma=0.01

Step 4. Building predictive models

- Predictive models were established for each training dataset using features found in step 2 and parameters found in step 3
- The models were used to perform predictions on the corresponding test files. Predictions above 5000 were rounded down to 5000, whereas all predictions below 1000 were rounded up to 1000

Task 2

Step 1. Initial SVM parameterization

- Each dataset was parameterized separately, though for the large dataset the number of attributes was reduced to 250 by random selection and for the medium and large datasets the number of samples was reduced to 10000 by random selection
- The 4-fold cross-validated SVM learning with different parameters (cost parameter C, kernel, kernel parameters) was carried out with the objective to minimize the Gini index
  - Small: C=30, RBF kernel, gamma=0.001 → Gini = 0.879
  - Medium: C=3, RBF kernel, gamma=0.1 → Gini = 0.375
  - Large: C=1, RBF kernel, gamma=0.01→ Gini = 0.675

Step 2. Feature selection

- Features were ranked based on the Gini index value for the 4-fold CV on the training set (for medium and large datasets 10000 samples were randomly chosen) using individual attributes with SVM parameterized in Step 1
- Starting with the top ranked attribute, one attribute at a time was added and evaluated using the Gini index value (based on the 4-fold CV on the training set with SVM parameterized in Step 1). If a given attribute improved the Gini value then it was added to the selected attributes, otherwise it was rejected. A single scan through all ranked attributes was performed; we added the top 10 ranked attributes for the medium dataset and the top 20 ranked attributes for the large dataset
- The following number of attributes was selected for the corresponding datasets:
  - Small: 24
  - Medium: 30
  - Large: 38

Step 3. SVM parameterization

- For each training dataset using the selected features 4-fold CV based experiments were carried out to parameterize SVMs; the best setups for the corresponding datasets follow:
  - Small: C=10,RBF kernel, gamma=0.001
  - Medium: C=3, RBF kernel, gamma=1
  - Large: C=0.1, RBF kernel, gamma=0.01

Step 4. Building predictive models

- Predictive models were established for each training dataset using features found in step 2 and parameters found in step 3
- The models were used to perform predictions on the corresponding test files. Probability values generated by SVMs for all instances were submitted