# Some Methods of Blending Multiple Models for the AusDM Challenge 2009

October 2010
Ed Ramsden

www.edscave.com
EdR@sensorlytics.com

## 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.

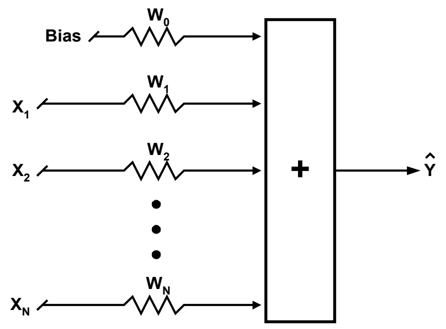
The AusDM 2009 challenge organizers provided sets of predictions obtained from the two leading teams in the Netflix Prize competition; Belkor's Pragmatic Chaos, and The Ensemble. For the RMSE portion of the challenge, three data sets were provided, a small set with 30,000 sets of predictions from 200 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.

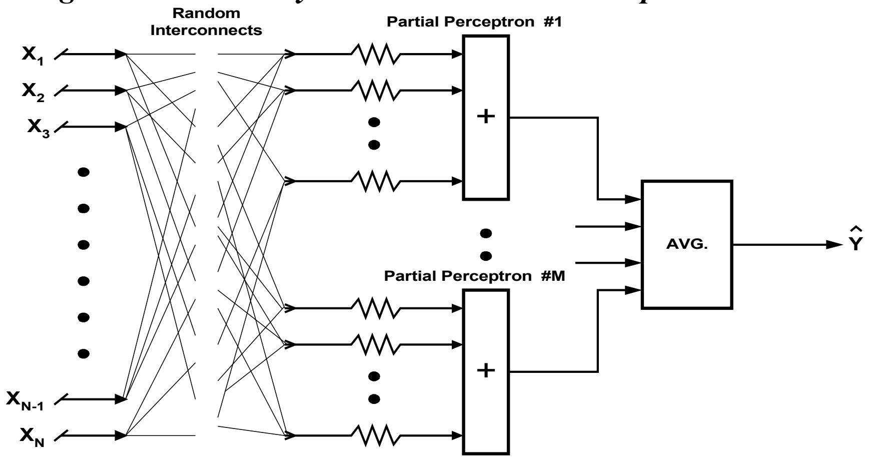
## 2. Variations:

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).

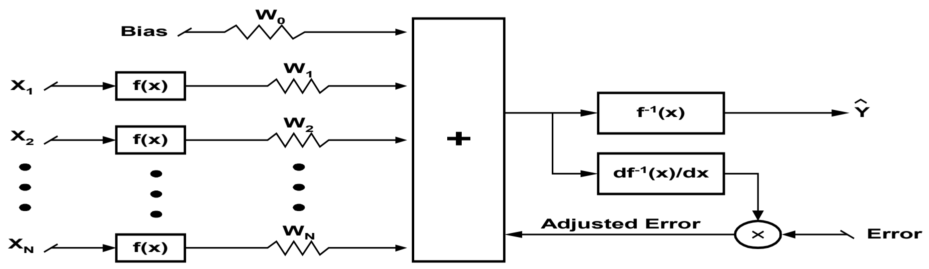
***Figure 2 – Randomly Connected Partial Perceptron Ensemble***



**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***

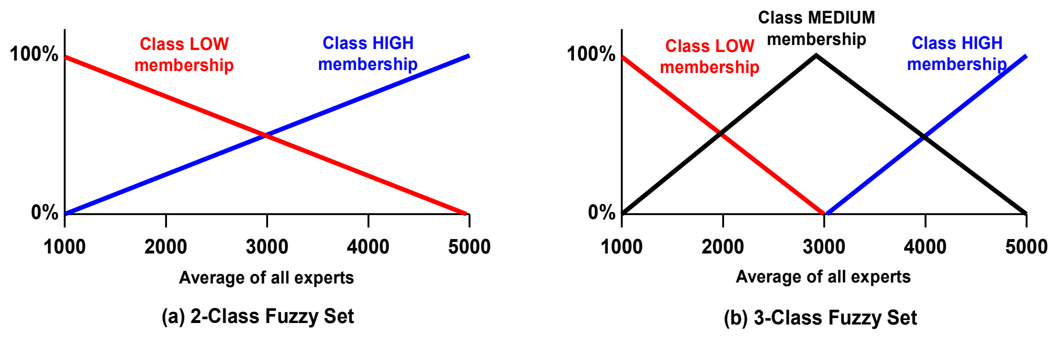


To ensure that the perceptron was training to minimize RMSE referenced to the original data (and not the transformed data – which could be significantly different), the inverse transformation was applied to the output. To train the weights, this also meant that the derivative of the inverse transform 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:

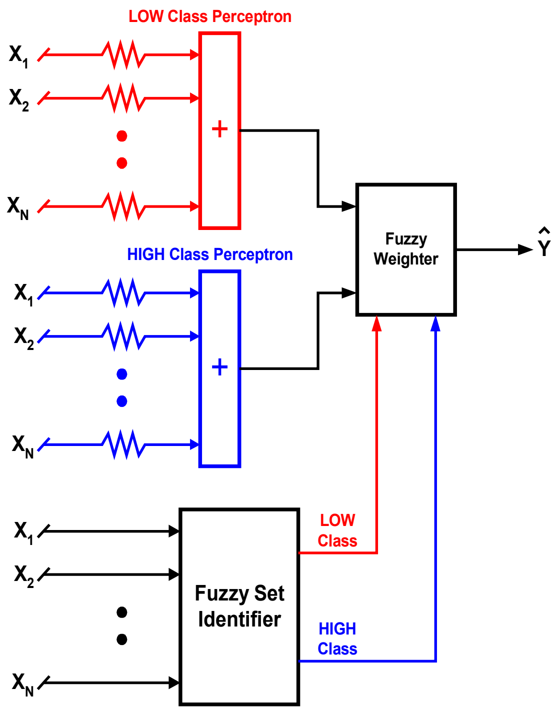
This ensembling method was based on the hypothesis that a single perceptron model would not be optimal over the entire range of input values (1000-5000). A membership function was defined that categorized fuzzy membership into Low and High classes (for a 2-class model) and Low, Medium, and High classes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

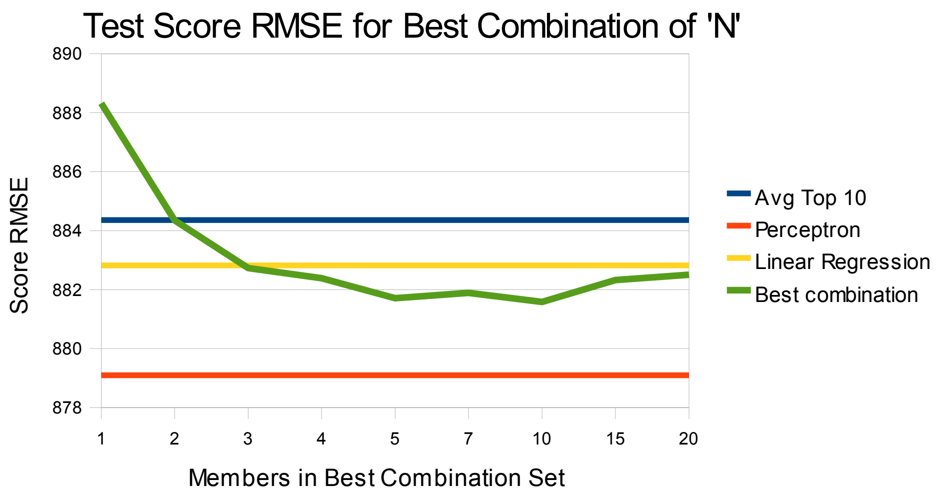
```
1) Select initial random set of experts
2) evaluate RMSE (of their combined ratings)
3) select an expert 'X'  within the set to replace
4) select an expert 'Y' outside the set as a replacement
5) re-evaluate RMSE with expert 'Y' substituted for expert 'X'
6) if new RMSE is better, then let 'Y' remain 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%** |

A report describing methods used by the various AusDM 2009 Challenge participants can be found at http://www.tiberius.biz/ausdm09/AusDM09EnsemblingChallenge.pdf

Copyright © 2009,2010 Ed Ramsden