Predicting gender from OKCupid profiles using ensemble methods

Isaac Backus

Abstract

In this paper I explore a method for predicting the sex of OKCupid users. Only male/female are options in this dataset. Missing data is replaced through mean imputation, although other methods are tried. 8 models (7 coded by me, 1 from scikit-learn) are trained on the data. Their results are combined with the ensemble method of stacking which uses a weighted sum of the models' predictions. This method is able to predict the sex of users with an 89% success rate.

1 Introduction

Ensemble methods in machine learning can present a simple but powerful method of combining the predictions of several, or many, machine learning algorithms into an ensemble model which is a better predictor than every individual model used.

In this paper I explore using the ensemble method of stacking, which combines predictions through a weighted sum, to predict the sex of people on OKCupid from a limited set of profile questions[1]. All the machine learning code was written by me, except one model (a random forest from scikit-learn) which I used primarily to verify my methods. In §2 I outline the dataset and the methods I used for generating features from it. In §3 I explain how I handled missing data, since this dataset is a questionnaire with an 87% response rate to questions. In §4 I outline the 8 models trained and explain the ensemble method to combine their predictions in §5. Results are presented in §6 and discussed, along with possible improvements to my methods, in §7. I find that I am able to predict sex (from one of male or female) with an 89% success rate.

2 Generating Features

The dataset is a text .csv document available on GitHub[2], made freely available by OKCupid. The entries are responses to a 31 questions by 59946 people within 25 miles of San Francisco with at least one profile picture who were online during 2011-2012, all of whom were over 18 (at least according to their profiles). The respondents in this dataset are 40.2% female (class 0) and 59.8% male (class 1). The ages have a range of 18-69 with a mean of 32.3 and a standard deviation of 9.4. This is sample is clearly not representative of the population at large.

Generating usable features for this dataset required parsing. Two methods were used which generated 217 and 101 features. I explain those methods here. After features were generated, the data set was split up into training and test sets of 49946 and 10000 entries, respectively, by randomly choice.

[1]In my original proposal I was going to predict incomes, but I realized that the response rate to incomes was very low (1/6) and from linear regression it appeared there were many outliers who may have been misreporting their income. This leaves too small of a usable dataset. For this reason I switched to predicting sex.

[2]`https://github.com/rudeboybert/JSE_OkCupid.git`

2.1 Full feature set

10 of the profile questions were short responses to prompts such as *What I'm doing with my life* or *On a typical Friday night I am...*. Since natural language processing was outside of the scope of this project, these were discarded. The remaining 21 responses were numbers, binary choices, multiple choice, multiple choice-multiple response, or ranked multiple choice responses and were all handled differently.

Numbers, such as height, age, or income, were left unaltered. Binary choices such as *sex* were converted to 0/1 according to the (alphabetized) options. So female (male) go to 0 (1). Multiple choice responses such as *pets* were converted multiple binary features (one for each choice). A response to *pets* of *has dogs and likes cats* would be converted (under the assumption that dog owners like dogs) to: (*has dogs, likes dogs, has cats, likes cats*) = (1, 1, 0, 1).

Certain questions, such as languages spoken, allowed multiple responses to be given, with possible modifiers. Someone who speaks English fluently and Chinese alright might respond *English (fluently), Chinese (okay)*. Speaking a given language poorly was counted as not speaking it, otherwise you are counted as a speaker. A binary feature was then made for each language, marking a 1 for each language a person speaks. This generates many (76) features which is addressed in §2.2. The two ranked multiple-choice questions were about drugs and drinking. These were assigned numbers 0, 1, 2, ... indicating the amount of use.

Features with a response rate lower than 25% were dropped (including income). Finally, the data was conditioned by subtracting the mean of each feature and normalizing by the standard deviation of each feature. This procedure generated 217 features which I call the full feature set.

2.2 Pared feature set

Certain questions created many features (because of their many possible responses), which all had very small regression weights (see §4). To see if this is an issue, I created a pared down feature set. I reduced the number of language of features from 76, one for each language), to 3: (i) a binary choice for speaking English; (ii) the number of languages a person speaks; (iii) and I retained *speaks_c++* since that was a good predictor of sex (being male).

I retained responses to how much astrological sign matters (*it doesn't matter*, *it matters a lot*, or *it's fun to think about*) and recorded whether or not the person reported their sign and discarded the astrological sign features. Education was also a multiple choice question with many possible responses, such as *high school* or *dropped out of college*. I created a ranking by hand of education with numbers of 1-28 with 1 being *dropped out of high school* and 28 being *graduated from ph.d program* and replaced the education features by a single feature.

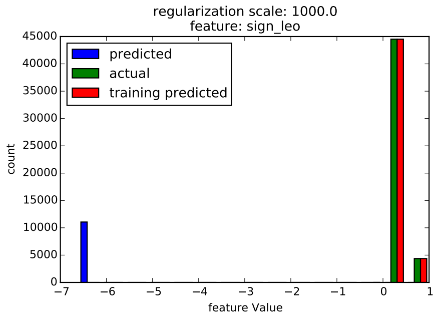
As with the full set, the data was conditioned by subtracting the mean of each feature and normalizing by the standard deviation of each feature. This created a set of 101 features.

3 Missing Data

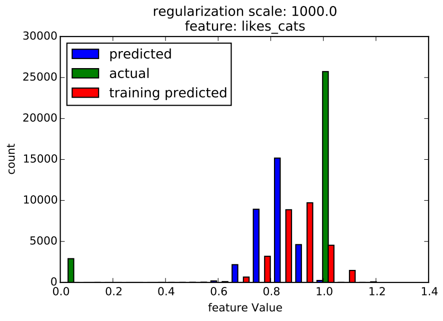
Around 13% of these features had no response. Restricting the dataset to only people/questions will full responses would have narrowed the dataset to a tiny fraction of its initial size and may have biased the data. Therefore, 3 imputation methods were tested.

The simplest technique—which was ultimately used—was mean value imputation. Missing entries were replaced with their feature means. This technique underestimates feature variances and does ignore any possible correlations between among features and ignores the possible importance of choosing to not answer a question. I also tested random selection, whereby missing entries are replaced by randomly chosen entries (from the same feature). This was as successful as mean value imputation under logistic regression.

I also tried using regression models for missing data imputation, but this failed with my dataset. First, missing values were replaced by mean value imputation. A feature was then selected and treated as training labels. A linear model was trained on the data with responses, which could be used to predict missing values (see §4.1). This was repeated for all features and the missing



(a)



(b)

Figure 1: Regression imputation. Histograms of predictions from linear regression for features compared to actual values. Under the regression imputation, these models would be used to predict missing values. For features which are uncorrelated with the rest of the data and have mainly one response, such as astrological signs (1a), predictions can be wildly off. Other features fair better and capture the mean well but don't separate the data into multiple populations well and are not very good predictors (1b).

values were replaced by their predictions. Examining the predictions of the models showed many poorly predicted features (see Figure 1). This method was therefore dropped in favor of mean value imputation.

4 Models

A total of 8 different models were trained on the data. 7 of these were written by me in python using numpy and scipy. Additionally, a random forest model was trained using scikit-learn, which was chosen primarily as a verification that my methods were indeed working and were performing well given the data. Results for these models are presented in table 1. These models were then combined using an ensemble method, detailed in §5.

Here I explain the models used and present their results.

4.1 Linear (ridge) regression

A standard ridge regression was very useful to implement, since implementation is fast and training/prediction are very quick. Under ridge regression, y values are predicted according to a linear model:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}} + \hat{w_0} \tag{1}$$

$$\hat{\mathbf{w}} = \left(\mathbf{X^T X} + \lambda\,\mathbb{I}\right)^{-1} \mathbf{X}^T \mathbf{y} \tag{2}$$

Where $\mathbf{X}$ is the training data and $\mathbf{y}$ is the training labels. The regularization I used here was $\lambda = 10./mean(\mathbf{X}^2)$ where the square is element-wise, chosen by selecting the largest value which gave a good squared training loss.

Classification is performed by classifying data points with $\hat{\mathbf{y}}$ bigger than some threshold as class 1 and the others as class 0. The threshold was found by choosing a threshold that minimizes the 0/1 loss on the training set by considering many threshold values within the range of $\hat{\mathbf{y}}$.

For the ensemble method (§5) it is good to have some way of quantifying the confidence (or the probability) of these predictions. This is not inherent to a linear model such as this, so I used a simple heuristic. The confidence of being in class 1 is given by:

$$C_0(y) = \frac{n_{0>}}{n_{1<} + n_{0>}} \tag{3}$$

where $n_{0>}$ is the number of training points of class 0 greater than y and $n_{1<}$ is the number of training points in class 1 less the y. The confidence of being in class 0 is $1 - C_0$. This gives a confidence of 1 in regions at the extremes of $\hat{\mathbf{y}}$ where all training points are in one class or the other. The confidences are then rescaled to C_1 such that at the threshold $C_1 = 0$. This method seems to over-predict the confidences, with a strong emphasis on $C_1 = 1$, but it is still better than just a binary prediction for the ensemble method used below.

4.2 Logistic regression

The logistic model predicts the probability that a data point $\mathbf{x}$ belongs to class 1 according to:

$$p(\mathbf{y_i} = 1|\mathbf{x_i}) = \frac{1}{1 + \exp[-(\mathbf{x_i} \cdot \mathbf{w} + w_0)]} \tag{4}$$

The weights $\mathbf{w}, w_0$ were trained using a stochastic gradient descent (SGD) algorithm that I wrote. In my algorithm, the data is randomly shuffled after each epoch and weight updates are performed in mini-batches. As in any SGD, the weights were updated by calculating the gradient of the loss function:

$$\mathbf{w}' = \mathbf{w} - \eta \nabla_{\mathbf{w}} L \tag{5}$$

$$L = \lambda \left\|\mathbf{w}\right\|^2 - \frac{1}{N}\sum_{i=1}^{N} \ln(p_i) \tag{6}$$

where η is the learning rate (a parameter). The gradient of the loss is can be calculated analytically from this and eq.4. Since the SGD is effectively regularized, I used $\lambda = 0$. I used a mini-batch size of 100. I set the convergence criterion to be when the log-loss had a fractional decrease of less than 10^{-6}. A powerlaw learning rate was used according to:

$$\eta = \frac{\eta_0}{(1 + k/\tau)^\kappa} \tag{7}$$

where$k$ is the total number of data points that have traversed. For the logistic regression I used $\eta_0 = 0.1/mean(\mathbf{X}^2)$, $\tau = 10^5$, and $\kappa = 0.55$. The algorithm converged after 14 epochs.

4.3 L2 Boosted Ridge Regression

Since the ridge regression ended up being as good as the best algorithms (see §6), I decided to attempt a boosted ridge regression. The algorithm for this L2 boosted ridge regression is as follows.

1. Initialize a while loop by setting a predictor $f^{(0)}(\mathbf{X}) = 1$ and setting $m = 0$.
2. Train a linear model $g^{(m)}(\mathbf{X}) = \mathbf{X}\mathbf{w}^{(\mathbf{m})} + w_0^{(m)}$ via ridge regression on the residuals vector $\mathbf{U} = f^{(m)}(\mathbf{X}) - \mathbf{y}$.
3. Update the model by setting $f^{(m+1)}(\mathbf{X}) = f^{(m)}(\mathbf{X}) + \nu g^{(m)}(\mathbf{X})$ where $\nu \in (0, 1]$ is a tuneable parameter (I chose 0.5). Then increment m and loop over steps 2-3 until the maximum number of iterations is reached

I stopped iterating when the 0/1 loss on the training set of $f^{(m)}$ converged (10 iterations). This is a linear model which weights misclassified points more heavily by fitting the residuals. Classification and confidences of predictions use the same method as the linear model outlined in §4.1.

4.4 Gaussian Mixture Model (GMM)

A mixture of two gaussians on the was trained by splitting the training data into 2 datasets according to their label and fitting a multivariate normal distribution to each dataset. Note that this method differs from GMM with hidden variables, i.e. unlabeled data. The parameter estimates for the mean μ and the covariance matrix $\mathbf{\Sigma}$ are given by:

$$\hat{\mu}_k = \frac{1}{N_k}\sum_{i,\mathrm{class_i}\in k} \mathbf{x_i}, \quad \hat{\mathbf{\Sigma}}_k = \frac{1}{N_k - 1}(\mathbf{X^{(k)}} - \hat{\mu}_k)^T(\mathbf{X^{(k)}} - \hat{\mu}_k) \tag{8}$$

where k is the class, $\mathbf{X^{(k)}}$ are the data points in that class, N_k is the number of points in that class, and the mean $\hat{\mu}_k$ is subtracted from each data point in $\mathbf{X^{(k)}}$.

In high dimensions, calculating $\det(\mathbf{\Sigma})$ and exponentiating the term involving $\mathbf{\Sigma}^{-1}$ in the multivariate normal are poorly behaved numerically. I implemented principle component analysis (PCA) and reduced dimensionality by projecting $\mathbf{X}$ onto 20 principle components.

Class predictions are given by

$$\hat{y}_i = \text{argmax}_{k \in \{0,1\}} \mathcal{N}_k(\mathbf{x}_i) \tag{9}$$

where $\mathcal{N}_k$ are the normal distributions fit to the two classes and $\hat{y}_i \in \{0, 1\}$. The confidence of the prediction (used for the ensemble method in §5) was estimated as:

$$c_i = 2\frac{\mathcal{N}_{\hat{y}_i}}{\mathcal{N}_0 + \mathcal{N}_1} - 1 \tag{10}$$

This ensures a $c = 1$ in the limit that the value of one normal distribution is much larger than the other $c = 0$ in the limit that the two normal distributions are equal.

4.5 K-Means

Similar to the GMM in §4.4, k-means can perform better in lower dimensional data, as I observed through testing. Additionally, it runs faster. To reduce dimensionality I performed PCA, projecting onto 25 principle components. I then normalized the data long each dimension by the standard deviation along that axis. This is an unsupervised method, which in general may not perform as well as supervised methods when labels are available. However, as with the GMM method, it may add predictive power to the ensemble method.

The k-means algorithm used here is as follows:

1. Initialize K cluster centroids $\mathbf{C}_k$ by random selecting K data points from the training data. In this case, I used $K = 256$.
2. Assign each data point to a cluster k according to the cluster that is nearest to each data point.
3. Update cluster centroids by taking the mean of the data points belonging to them: $\mathbf{C}_k := \frac{1}{n_k}\sum_{i=1}^{n_k} \mathbf{x}_i$
4. Repeat steps 3-4 until all of the centroids move less than some δ_C along every axis. Here I set $\delta_C = 0.1$ which was also sufficient for the 0/1 loss to converge.
5. Finally clusters are assigned labels l_k according to the class with the most number of members assigned to cluster k.

Classification is then performed by finding the centroid $\mathbf{C}_k$ that a data point $\mathbf{x}$ is closest and classifying it according to the label l_k. Confidence of the classification is taken as the fraction of members (in the training set) of cluster k belonging to class l_k.

4.6 Random Map + logistic

This method uses a feature map and then performs logistic regression on the mapped data, using the method of §4.2. First, k random vectors $\mathbf{v}_j$ are generated, where $k = N_{train} = 49,946$ and $\mathbf{v}_j \in \mathcal{R}^d$ where d is the number of features. The elements of the vectors $\mathbf{v}_j$ are drawn at random from a normal distribution. The mapping is $H_{ij} = h_j(\mathbf{x}_i) = \max\{\mathbf{v}_j \cdot \mathbf{x}_i, 0\}$

The logistic regression algorithm of §4.2 was then performed using $\mathbf{H}$ in place of $\mathbf{X}$. Due to memory constraints (this generates a feature matrix of shape $N_{train} \times N_{train}$), the feature mapping is applied on the fly whenever data points are accessed during the logistic regression. The mini-batch size used was 50. No regularization was used. Training rate parameters (see eq.7) were $\eta_0 = 5 \times 10^{-4}/\overline{(\mathbf{H}^2)}$, $\tau = 10^5$, $\kappa = 0.55$. Convergence was determined by a fractional decrease in the loss of less than 10^{-6}.

4.7 RBF Kernel + logistic

Similar to the model in §4.6, this method applies a feature map to the data and then performs the logistic regression of §4.2. This feature map is the radial-basis function (RBF) kernel. This gives a feature mapping of:

$$H_{ij} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right) \tag{11}$$

This is calculated on the fly whenever data points are accessed. The same learning rate scheme and fit parameters were used as in §4.6. The bandwidth parameter σ was set as the median of the distances between 100 randomly selected pairs of points.

4.8 Random Forest (scikit-learn)

In order to verify the success of my methods and to include a distinct model in the ensemble predictions, I used an out of the box random forest model implemented in scikit-learn. A random forest is made by training decision trees on random subsets of the data and averaging their predictions. Random forests help avoid the overfitting that decision trees are prone to. Since the goal of this project is to present models that I've implemented, I will mainly treat the random forest as a black box. I used 70 estimators (70 decision trees were trained on 70 random subsets of the data) and required a minimum of 25 data points per tree leaf to help prevent overfitting. Splitting the tree further seemed to overfit the data (as determined by cross-validation).

5 Ensemble Predictions

To leverage the predictive power of all the 8 models in §4, I used the ensemble method of stacking to combine the predictions of each model. Stacking will always perform as well or better on the training set than any individual model, and often performs better on the test set. The approach is simple.

Let $f^{(m)}(\mathbf{X})$ be a training model which returns numbers $\hat{\mathbf{y}} \in [-1, 1]$ where $c_i = \|\hat{y}_i\|$ is the confidence of the prediction and the predicted class is 1 for $\hat{y}_i > 0$ and 0 for $\hat{y}_i \leq 0$. This is equivalent to $\hat{y}_i = (2k_i - 1)c_i$ where k_i is the class of $\mathbf{x}_i$ predicted by model m.

The ensemble model is then given by:

$$f(\mathbf{X}) = \sum_{m=1}^{N_{models}} \alpha_m f^{(m)}(\mathbf{X}) \tag{12}$$

The predicted class is then 1 for $f(\mathbf{X}) > 0$ and 0 for $f(\mathbf{X}) \leq 0$. The factors $\alpha_m \geq 0$ are weights for the various models and are found by minimizing the loss. Here, I determined the weights by minimizing a regularized 0/1 loss $L = L_{0/1} + \lambda \|\alpha\|^2$ using a simplex minimizer (scipy.optimize.fmin). This method can cause overfitting if any of the models $f^{(m)}(\mathbf{X})$ are overfit by giving large weight to overfit models, but I was careful to avoid overfitting the models (as can be seen in §6). Since only the sign of $f(\mathbf{X})$ matters, the weights can be normalized without loss of generality. Table 2 lists the normalized classifier weights used for both (full and pared) feature sets.

6 Results

The ensemble method trained here does a fairly good job of predicting the sex of people in this sample. The full fit loss results are listed in table 1. The baseline 0/1 loss is 0.40 since 60% of the population is male and guessing male would give a loss of 0.40. All the individual methods achieve a lower 0/1 loss than this on the test set for both the full and pared feature sets. The ensemble stacking model fares the best, achieving test losses of 0.116 and 0.118 for the full and pared feature sets, respectively. That these losses are so close indicates that the procedure used to reduce the number of features (§2.2) was effective. Removing astrological signs does not seem to have removed predictive power from the model.

Surprisingly, the simple logistic regression model is the best individual model across the board. The incredibly simple linear ridge regression performs nearly as well! Perhaps unsurprisingly, the GMM and the unsupervised K-means models perform the worst. The GMM is a very simple model and is fitting data which is by no means gaussian in nature. The K-means is unsupervised and no attempt was made to scale dimensions according to how important they are for predicting sex.

Table 1: Fit 0/1 losses on the training and test sets. Across the board, the best model is the ensemble stacking method and the best individual model is the logistic regression. Note that the ensemble only improves slightly (and possibly not significantly) over the logistic regression. The ridge regression is competitive with this. Also note that the test losses are only slightly larger than the training losses, indicating that overfitting is not a large issue.

		Full set		**Pared Features**	
Method	**Abbr.**	**Training**	**Test**	**Training**	**Test**
Gaussian Model	gm	0.316	0.316	0.246	0.255
K-means (256 clusters)	kmeans	0.301	0.307	0.291	0.311
L2 Boosted Ridge Regression	l2boost	0.113	0.119	0.115	0.123
Ridge Regression	lin	0.113	0.119	0.115	0.123
Logistic Regression	log	0.112	0.119	0.114	0.121
Linear Random Map	lrm	0.143	0.156	0.124	0.133
Random Forest (sklearn)	random forest	0.116	0.128	0.114	0.128
RBF Kernel Logistic	rbf	0.171	0.185	0.160	0.162
Ensemble		0.106	0.116	0.106	0.118

Table 2: Classifier weights (α_m in eq.12) for the ensemble stacking model.

Method	**Full Set**	**Pared features**
Random Forest (sklearn)	0.520	0.448
RBF Kernel Logistic	0.137	0.227
K-means (256 clusters)	0.129	0.106
L2 Boosted Ridge Regression	0.080	0.026
Linear Random Map	0.060	0.030
Ridge Regression	0.052	0.019
Logistic Regression	0.016	0.098
Gaussian Model	0.006	0.045

Table 2 lists the weights (α_m in eq.12) used for each classifier for both the full and the pared feature sets. While the random forest was not the best predictor overall, it has the largest weights. This may indicate that the other successful predictors give similar predictions. If the other predictors are very similar, giving them many small weights will still give their predictions large total weight.

7 Discussion

Linear regression results are fairly interpretable. Since the ridge regression was nearly as good a predictor as the ensemble model, it is a good place to begin any discussion. Table 3 shows the features with the largest positive weights and those with the largest negative weights. Since the data was normalized by the standard deviations of per feature, this should give a decent measure of how important these weights were for the model. Positive weights correspond to belonging to class 1 (male) and negative weights correspond to class 0 (female).

By far the most important feature is *height* (which correlates with being male), with a weight 4 times bigger than the next largest *body_type_curvy* (which correlates with being female). This should be unsurprising, that males are as a population taller than females. Having a computer or science/tech/engineering job are strong male features, while having a job in medicine/health is a strong female predictor. Interestingly, being gay is a strongly male feature and being bisexual is a strongly female feature. This is reflected in this dataset where about 15% of both males and females report an orientation other than straight. About 11% of males report as gay and about 2% as bisexual, whereas 6% of females report as gay and 8% as bisexual.

Two features on these lists indicate that this sample is not representative of the population as a whole: *ethnicity_hispanic / latin* is a strongly male feature and *ethnicity_white* is a strongly female feature. Of course, in the general population half of each of these ethnicities are female. Therefore, I would not expect these models to perform as well on a person randomly selected from the general population.

Table 3: Absolute value of the largest feature weights normalized by the largest weight for the linear ridge regression of §4.1 on the full feature set. The male traits are the features with the 5 biggest postive weights. The female traits are the features with the largest negative weights. By a factor of 4, height is the largest weight. Strangely, *ethnicity_white* is one of the most female traits.

Male		**Female**	
Feature name	$\|\mathbf{w}\|/\|\mathbf{w}\|_{\mathbf{max}}$	**Feature name**	$\|\mathbf{w}\|/\|\mathbf{w}\|_{\mathbf{max}}$
height	1.00	body_type_curvy	0.25
body_type_athletic	0.15	ethnicity_white	0.14
job_computer / hardware / software	0.13	body_type_full figured	0.12
orientation_gay	0.09	orientation_bisexual	0.11
ethnicity_hispanic / latin	0.09	has_cats	0.10
job_science / tech / engineering	0.08	job_medicine / health	0.08

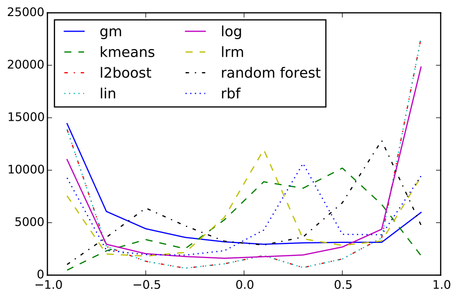


Figure 2: Histograms of the training models $f^{(m)}$ (as in §5) on the training set for the single models. Abbreviations are defined in table 1. Some models cluster at $+/-1$. These may be over-estimating the confidence of predictions.

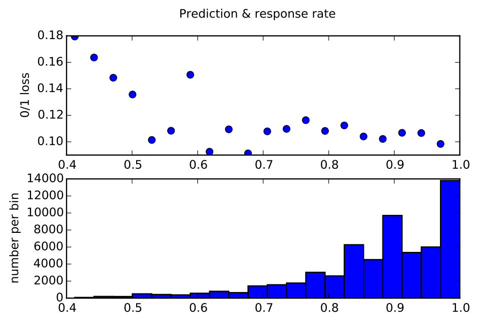


Figure 3: 0/1 loss and response rate for the full feature set. The x-axis on both panels is the response rate to features (fraction of answered questions averaged over all people). (top panel) 0/1 loss on the training set as a function of individual's response rate. (bottom panel) Histogram of people's response rates. 0/1 loss decreases with increasing response rate.

Given that several different models do no better than 0.11 0/1 loss and that the ensemble method barely improves on them, it may be possible that I have approached the best possible results for my feature set, and possibly for my dataset. There are, however, a few possible ways to improve upon the methods presented here.

There may be better ways of generating features. In particular, I discarded the written responses, but these might hold a lot of information which is orthogonal to the rest of the dataset. Generating features from these would require some natural language processing and perhaps some intuition.

The confidences used to generate the predictors in the ensemble stacking method may be a little bogus (see figure 2). Certainly the confidences of the predictions of the K-means and linear models are a little ad-hoc. The logistic model also may over-estimate its confidence, skewing model predictions $f^{(m)}$ to $+/-1$. The random forest is naturally suited to generating confidence predictions and this may be one of the reasons is receives such a large weight in the stacking (see table 2).

The missing data imputation might be handled in a more sophisticated manner. Figure 3 shows the relation between individual response rate to the questions and 0/1 loss. Low response rates lead to poor 0/1 loss. This decreases the overall 0/1 loss directly and indirectly by training the model on "worse" data points.

A final place for improvement in the model is allowing gender options other than m/f. It's quite possible that there are outliers in this feature set which are being mislabeled as m/f.