Measures of Prediction Accuracy and Ranking

Kyle Payne, Tangwei Li, Manzi Qin

March 1, 2016

Introduction

The following report consists of the Author's Recommendation for Measures of Prediction Quality for the TerraRef Project at the University of Illinois. In this preliminary report we will focus on two particular problems that have been addressed so far:

- How to determine the accuracy of a continuous prediction on a continous target value (e.g. phenotype).
- How to Score Predicted Rankings for some subset of lines (e.g. genotypes).

While we only address these problems in a relatively closed sense, the measures that we propose may be applicable to other settings as well. We will define the our measures, describe their respective numerical and statistical properties, make recommendations for using these measures in practice, and produce functions in both the Python and R languages for their implementation.

Measures

Continuous Phenotype Prediction

An example of the types of problems that fall under 'How to determine the accuracy of a continuous prediction on a continous target value (e.g. phenotype)' would be to demonstrate that predicted values are within 20 percent of ground truth values. Thus, we we need a measure that accounts for the difference between the predicted and ground truth (the true observed phenotype values), while also accounting for the *relative degree* by which the predictions are different from the ground truth values. One metric that appears in the literature (citation) is the Relative Mean Squared Prediction Error, or the RMSPE, which we define in equation (*)

Let $Y_1, ..., Y_n$ be a set of ground truth phenotype values, and let $\hat{Y_1}, ..., \hat{Y_n}$ be the set of corresponding predictions for the ground truth phenotype values, then the RMSPE is defined as.

$$RMSPE = \frac{\sqrt{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}}{\sqrt{\sum_{i=1}^{n} Y_i^2}}$$

The square of the numerator $\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$ is a well-studied function within the machine learning and statistics community, known as the Mean Squared Prediction Error, or (MSPE). This is an easily computed, and numerically stable quantity that provides several desirable large sample properties. The denominator of $RMSPE$ can be viewed as the difference of the continuous ground-truth phenotype values from 0. Thus, $RMSPE$ acts a relative measure of the difference between the ground truth continuous phenotype values, expressed in units of the continuous phenotype. Using this measure, an experimenter could make a statment such as, "I demonstrated that the predicted values are within 0.2 of the ground truth values". This type of measure could be helpful for determining prediction accuracy for both Terminal Biomass and 3D Structural Models. Moreover, the $RMSPE$ could also be used to compare the prediction accuracy between competing models, such as comparing algorithm predictions v.s. the Lemnatec Software.

Examples

If a prediction model produces a set of predictions $\hat{Y_1}, ..., \hat{Y_n}$, for a set of ground truth continuous phenotype values $Y_1, ..., Y_n$, then let's state that

$$RMSPE \leq 0.20 \Rightarrow \sqrt{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2} \leq 0.2\sqrt{\sum_{i=1}^{n} Y_i^2}$$

The equation above describes the case where the $RMSPE$ being less than or equal to 20 percent is equivalent to stating that the mean squared prediction error is bounded by 0.2 the size of the ground truth phenotype values. The quantity on the left-hand side of the inequality is an example of a commonly used measure of distance in mathematics, engineering, statistics, computer science, etc. known as the *Norm* (Weisstein, Eric W. "Norm." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Norm.html).

Let $RMSPE_L$ be the relative mean squared prediction error of Lemnatec software predictions on the ground truth continuous phenotype values $Y_1, ..., Y_n$, which we will denote as $Y_1^*, ..., Y_n^*$. Let $RMSPE_M$ be the relative mean squared prediction error of some prediction model or algorithm. Then, making a determination such as 'algorithm predictions are no less accurate than values predicted via LemnaTec software', would require comparing $RMSPE_L$ and $RMSPE_M$, thus

$$RMSPE_M \leq RMSPE_L$$

Thus, if we compare two prediction models in $RMSPE$ on the same set of data, the comparision is equivalent to just comparing the average squared difference between the predictions and the ground truth values.

Regularized RMSPE

Performance

The Relative Mean Squared Prediction Error Performs well in situations in which there is additional noise in the continuous phenotype values $Y_1, ..., Y_n$. For the following example, let's assume that some prediction algorithm has been fitted to a set of training data. In this case, we chose a Random Forest Model to predict Stem Biomass using the plot identifier and the precipitation data on a sub-sample of simulated data. We trained the random forest model on a sub-sample of simulated data. Predictions were then made using a test sample of the data, with additional mean 0 gaussian random noise applied to the Stem Biomass data. The RMSPE measure increases like a polynomial with increasingly variable noise. However, the RMSPE remains relatively robust to deviations from the true Prediction Error, and only increases to very large values as the

```
library(randomForest)
```

```
## randomForest 4.6-12
```

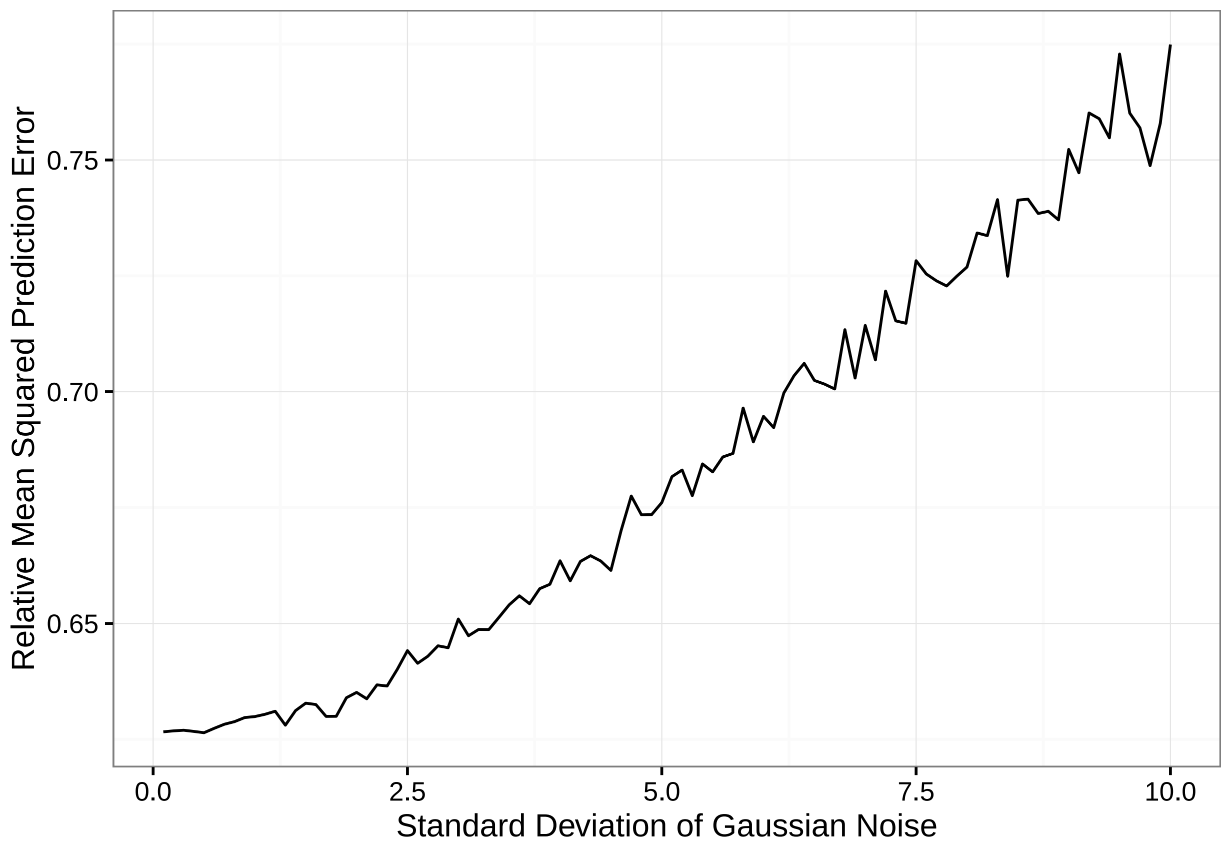
```
## Type rfNews() to see new features/changes/bug fixes.
```

```
obs_sub <- read.csv("~/Desktop/terra_ref/obs_sub.csv")

ind <- sample(2, nrow(obs_sub), replace = TRUE, prob=c(0.8, 0.2))
obs_sub$genotype <- factor(obs_sub$genotype)
obs_sub$plotid <- factor(obs_sub$plotid)
```

```r
RMSPE <- function(y, prediction){
  return(norm(as.matrix(y-prediction), type="F")/norm(as.matrix(y), type="F"))
}

obs_rf <- randomForest(Stem ~ plotid + precip,
                       data=obs_sub[ind == 1,])

rmspe <- numeric(100)
for(k in 1:100){
  test <- obs_sub[ind==2,]
  test['Stem'] <- test['Stem'] + rnorm(n=nrow(obs_sub[ind==2,]),
                                        mean = 0, sd=k/10)
  obs_pd <- predict(obs_rf, test)
  rmspe[k] <- RMSPE(test[,'Stem'], obs_pd)
}
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':
##
##     margin
```

```r
results <- data.frame(rmspe, noise_sd =(1:100)/10)
ggplot(results, aes(x=noise_sd, rmspe))+geom_line()+theme_bw()+
  xlab("Standard Deviation of Gaussian Noise") +
  ylab("Relative Mean Squared Prediction Error")
```



Ranking

Another problem of interest is assigning a score to rank predictions, e.g. given a set of lines of sorghum, and we score the predicted rank orders from some prediction algorithm? The literature is replete with measures of rank ordering. We will focus here two measures of rank ordering correlation, Kendall's Tau and the Normalized Discounted Cumulative Gain.

Kendall's Tau

Kendall's Tau is a measure of the correlation between two sets of rankings. Let $Y_1, ..., Y_n$ be a set of observations and let $\hat{Y}_1, ..., \hat{Y}_n$ be a set of predicted observations. Then Any pair of observations $(Y_i, \hat{Y}_i)$ and $(Y_j, \hat{Y}_j)$, where $i \neq j$, are said to be concordant if the ranks for both elements agree: that is, if both $Y_i > Y_j$ and $\hat{Y}_i > \hat{Y}_j$ or if both $Y_i < Y_j$ and $\hat{Y}_i < \hat{Y}_j$. They are said to be discordant, if $Y_i > Y_j$ and $\hat{Y}_i < \hat{Y}_j$ and $Y_i < Y_j$ and $\hat{Y}_i < \hat{Y}_j$. If $Y_i = Y_j$ and $\hat{Y}_i = \hat{Y}_j$, the pair is neither concordant nor discordant.

Kendall's Tau is defined as:

$$\tau = \frac{(\text{number of concordant pairs}) - (\text{number of discordant pairs})}{1/2n(n-1)}$$

If both sets, the ground truth and predicted values are completely concordant in rank, then $\tau = 1$. Likewise, if both sets are in perfect disagreement, then $\tau = -1$. Kendall's Tau is analogous to a correlation coefficient in this manner. While Kendall's τ is easy to interprete, τ does not weight discordance differently depending on where in the rankings the discordance occurs. Therefore, one may be interested in a different ranking score, which we describe below. For examples of Kendall's Tau, please see the github repository.

Normalized Discounted Cumulative Gain

Normalized Discounted Cumulative Gain (NDCG) has gained prominence in the Learning to Rank literature across many fields as a robust and power answer to the problem of measuring the performance of ranking functions. First we start with a explation of the discounted cumulative gain, then we will introduce the normalize version of the measure.

$$DCG_{n,p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log_2(i+1)}$$

where n is the total number of items to rank. rel_i is the relevancy of the predicted rank, which is a value that can take on varying interpretations depending on the context. For this work, we set $rel_i = n - |p_i - t_i|$, where p_i is the predicted rank for the i^{th} ranked value and t_i is the true rank for the i^{th} value. Obviously $t_i = i$.

$$DCG_p = \sum_{i=1}^{p} \frac{2^{n-|p_i-i|-1}}{log_2(i+1)}$$

Examples

Let's say we wish to predict the ranks of the top 10 lines of sorhgum, based on the overall biomass. Then,

```r
library(knitr)
true_rank <- 1:10
pred_rank <- c(1,2,4,3,5:10)

ranks <- matrix(c(true_rank, pred_rank), byrow=T, nrow=2)
row.names(ranks)<-c("Ground Truth", "Predicted")

kable(ranks)
```

Ground Truth	1	2	3	4	5	6	7	8	9	10
Predicted	1	2	4	3	5	6	7	8	9	10

In the table, we have the true ranks for the top 10 lines of sorghum out of 20 total lines, which is obviously just a sequence of the first 10 integers. At the 3rd and 4th positions, our prediction algorithm get's the predicted ranks wrong, and swaps 3 and 4. Therefore, the term in the summand of the discounted cumulative gain, that is

$$(2^{rel_3} - 1)/log_2(3+1) = (2^{20-|(3-4)|} - 1)/log_2(4) = 262144$$
$$(2^{rel_4} - 1)/log_2(3+1) = (2^{20-|(4-5)|} - 1)/log_2(5) = 225798.6$$

The mis-ranking at the 4th position had a slightly smaller effect on the scoring than that of the misprediction at the 4th position.

Let's look at another example.

```r
library(knitr)
true_rank <- 1:10
pred_rank <- c(1,2,3,4,5,6,7,10,9,8)
```

```r
ranks <- matrix(c(true_rank, pred_rank), byrow=T, nrow=2)
row.names(ranks)<-c("Ground Truth", "Predicted")

kable(ranks)
```

Ground Truth	1	2	3	4	5	6	7	8	9	10
Predicted	1	2	3	4	5	6	7	10	9	8

In this example, the prediction algorithm mixed up the bottom 3 in our list of the top 10 ranks. Let's look at the first term in the summands that correspond to those positions in the list:

$$(2^{rel_8} - 1)/log_2(9) = (2^{20-|(2)|} - 1)/log_2(9) = 82696.91$$

As the mis-ranking occurs much farther down in the list of the rankings, it contributes much less to the overall score. Thus, if the objective is the maximize the overall ranking score, then mis-rankings in positions far along in the list cost less than mis-rankings that occur in the front of the list.

A perfect prediction ranking would lead to a maximized DCG, for this example if would be

$$(2^{20} - 1)\sum_{k=1}^{10} \frac{1}{log(k+1)}$$

The Normalized Discounted Cumulative Gain is the ratio of a prediction algorithm's Discounted Cumulative Gain over the maximum possible Discouted Cumulative Gain.

$$NDCG = \frac{DCG_p}{max\{DCG_p\}}$$

One reasonable question to ask is why do define $rel_i = n - |p_i - i|$? instead of $rel_i = k - |p_i - i|$, where k is the length of the list of rankings compared. We chose this form as it allows for mis-rankings outside of the "top k many lines" list. E.g.

```r
library(knitr)
true_rank <- 1:10
pred_rank <- c(1,2,3,4,50,6,7,8,9,10)

ranks <- matrix(c(true_rank, pred_rank), byrow=T, nrow=2)
row.names(ranks)<-c("Ground Truth", "Predicted")

kable(ranks)
```

Ground Truth	1	2	3	4	5	6	7	8	9	10
Predicted	1	2	3	4	50	6	7	8	9	10

We can still score the predicted ranking relative to the true ranking even if the $5th$ position of the list of predicted ranks has a value that is way outside of the top 10 values.