Mini Project: part II Samantha Maticka

**Recap:**

My project is looking at the relationship between covariates that describe different physical activities, namely swimming, biking, running, and walking. The covariates consist of variables detailing speeds, distance, elevation change, etc. throughout the respective activity. Each activity is a single observation. *Note: my part I submission said I would be predicting total distance, but I realized this is too simple since it’s an exact calculation (average speed/total time). My response variable is now the start time of the activity.*

**1 Baseline Models**

A/C

Before you launch into creating your predictive models, start with some baselines to give you some sense of what you are trying to achieve. Make sure you have an evaluation strategy in mind; recall that if you’re not sure what to do, cross validation is always a reasonable way to compare model performance. In particular, answer the following questions:

A) For your regression problem, predict the mean outcome; for your classification problem, predict whichever label (zero or one) occurs more frequently in the data. What is the perfor- mance of each approach?

B ) For your classifiation problem, is 0-1 loss the right objective, or something else?

C ) Fit a linear regression (resp., logistic regression) model with only a few covariates that you think are likely to be important. How does your model compare to the baseline model?

*For the regression model (response = Tstart = start time of activity):*

The mean of Tstart is 11.92. When this value is used as a prediction, the MSE is 14.10. The RMSE is 3.75 hr, indicating, on average, the activities start just before noon, and the dummy model predicts on-average within 4 hours of the actual start time.

*­­*

*For the classification problem (4 classifications: walk, swim, run, bike):*

Since there are 4 separate activities, the activity with the highest empirical frequency is predicted as the dummy model. ‘Run’ accounts for 51.58% of the observations. When ‘run’ is predicted every time, the Accuracy is 51.58%.

B

For the case of selecting among 4 activities, accuracy is a good description of how well the

model performs, where the sum of true ‘positives’ and true ‘negatives’ (which only makes sense for binary data) becomes the sum of correct classifications across all cases (i.e. true ‘case 1’ + true ‘case 2’ +…). This tells us what percent of the time the model is able to accurately predict an activity.

However, we care about the accuracy of prediction per activity as well (e.g. TPrun/(total # of runs)). One thing to strive for is maximum average accuracy (i.e. in 1A, total accuracy is 51%, but the model is 0% accurate at predicted 3 of the 4 activities and 100% at predicting 1, therefore the average accuracy is 25%).

C

|  |  |  |  |
| --- | --- | --- | --- |
|  | Model Name | MSE/Accuracy | Model |
| Linear Regression | Base 1 (1a) | 14.10 |  |
| Base 2 (1c) | 12.64 |  |
| Logistic Regression | Base 1 (1a) | Total: 51.58%  Walk: 0 %  Swim: 0 %  Run: 100%  Ride: 0 %  Average Accuracy: 25 % |  |
| Base 2 (1c) | Total: 92.49%  Walk: 0 %  Swim: 0 %  Run: 100%  Ride: 96.73 %  Average Accuracy: 49.18 % |  |

**2 Building your Model**

A

Start by improving on the linear regression (resp., logistic regression) baseline model you built above, however you can: transformations, interactions, regularization, etc. Keep track of the best model you can build for your chosen objective. Make sure you record the model you have selected.

B) Next, feel free to try any other methods you wish on your data; among other things, you can try k-nearest-neighbors, or naive Bayes classifiers, or any other methods you may be interested in investigating. You are welcome to do as much or as little as you like for this part of the project, but if you do try out other methods, please (concisely) describe the following: Which method(s) did you try? Which methods worked well, and which ones failed? Can you explain their performance, e.g., in terms of bias and variance?

Linear Regression:

Transformations: All covariates that had a skewed PDF were log transformed, and the correlation with the response variable was compared to the correlation of the non-transformed covariates to decide which ones to apply a transform to. Only 1 covariate was better transformed, but gave errors when I used it in the linear model; for now, *no transformations were made.*

Model Selection: Both, the backward BIC and AIC methods were applied to determine a model with the lowest MSE and a low complexity. The total set of covariates included all individual covariates and all two-way interaction terms.

Note: Some of the initial covariates were possibly unnecessary, but were ‘thrown in’ to see which would be a better metric. For example, the max, median, and mean speeds were included, but I’m only keeping one in order to keep the model simple.

The following covariates were ‘thrown out’:

|  |  |
| --- | --- |
| **Covariate** | **Reason for removal** |
| Umax | Poor correlation, and the median appears to be a better metric. |
| Umean | Moderate correlation, but median speed will be more robust to outliers – which may be common if user doesn’t stop recording activity when finished. |
| Turnsmean | Poor correlation, and the median/min appears to be a better metrics. |
| Ibike (not shown) | The threshold was too strict. All activities registered 0. |

Chosen model: The model was chosen from BIC which yielded the lowest MSE (aside from a comparable AIC MSE) and less terms than AIC selection.

Logistic Regression: Multinomial Classification

To determine the best multinomial classification model, the R-package ‘glmnet’ was used. The model-selection function, ‘cv.glmnet’ determines the ‘best’ model through an iterative process. Different lambdas are used to penalize larger coefficients in the maximum likelihood calculation. The model corresponding to a lambda that yields the lowest mean CV error is chosen. I used 10 folds, a ridge regression method for lambda, and the iteration was conducted 100 times. The error appeared to converge (see below). For now, only individual terms were used in the model, but no interaction terms. The results were pretty good, so it will be interesting to see if the interaction terms improve the prediction.

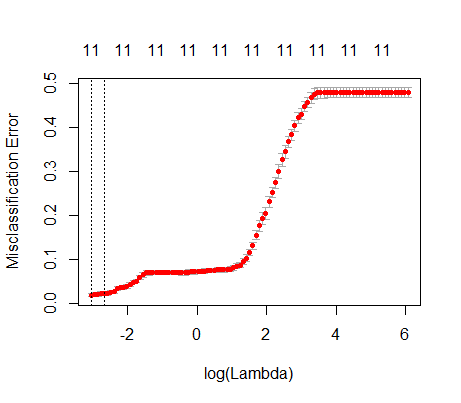
C

Finally, report on the best of all the models you tried. Most importantly, give an estimate of what you think the test error will be when you run your model on the previously held out test set; explain how you arrived at your estimate.

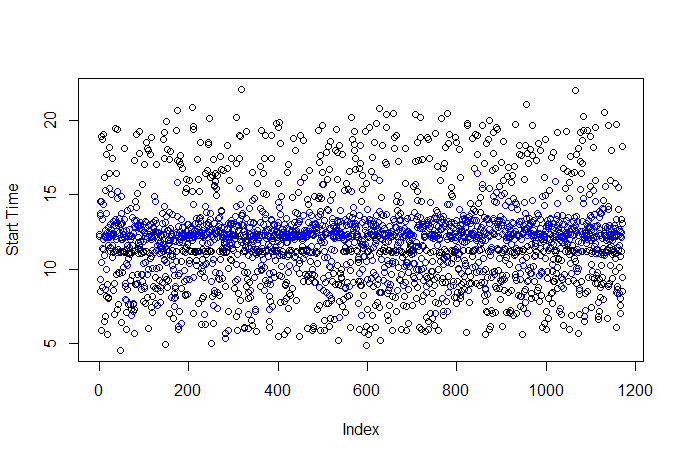
MSE and accuracy were both calculated by performing cross-validations across 10 folds and the average error metric is what is listed in the table. These should both be unbiased estimates of prediction error, since cross-validation was used; hence, the errors are calculated on different data than what was used to train the models.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Model Name | MSE/Accuracy | Chosen Model |
| Linear Regression | Chosen model 2 A/B/C | 11.45 |  |
| Logistic Regression | Chosen model 2 A/B/C | Total: 98.21%  Walk: 80 %  Swim: 91.23 %  Run: 99.01%  Ride: 98.57 %  Average Accuracy: 98.57 % |  |

**Convergence of Error wrt Lambda**

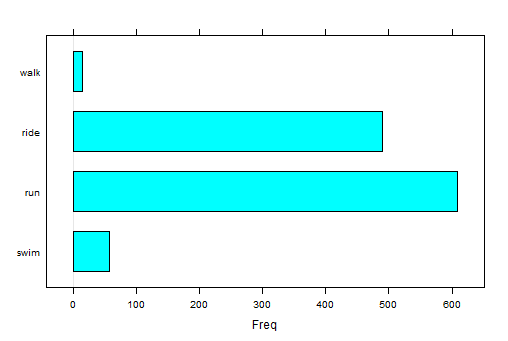
**Model Performance**

Linear Regression (model fit)

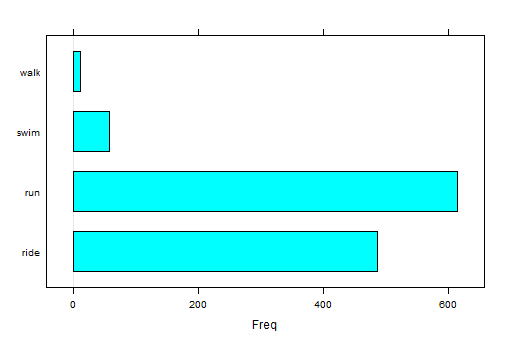


Logistic Regression (In-model comparison)

**Observed**



**Modeled**



Reflection: Even though CV was used, I do think the estimate for error is optimistic, since my training data set may not be the best representation of the population. More training data would improve this estimate

**Code:**

**‘BuildingModels\_partII.R’**

# this script creates different model templates

# read in training data

setwd**(**'C:\\Users\\smaticka\\Box Sync\\stanford\\Classes\\MS&E 226 small data\\mini project'**)**

.libPaths**(**"C:\\Users\\smaticka\\Documents\\R\\win-library\\3.2"**)**

library**(**gdata**)** # to read in data

library**(**arm**)**

library**(**ggplot2**)**

library**(**GGally**)**

library**(**cvTools**)**

library**(**nnet**)**

library**(**glmnet**)**

source**(**"PreProcess\_partII.R"**)**

# part 1A

fm.base1 **=** mean**(**df**$**Tstart**)**

MSE.base1 **=** sum**(** **(**df**$**Tstart **-** fm.base1**)^**2 **)/**length**(**df**$**Tstart**)**

RMSE.base1 **=** sqrt**(** MSE.base1 **)**

# part 1C

## run linear regression with a few covariates

fm.base2 **<-** lm**(**df, formula **=** Tstart **~** 1 **+** dEmax **+** Dtotal **+** Ttotal **+** Turnsmin**)**

fm.base2.cv **=** cvFit**(**fm.base2, data **=** df, y **=** df**$**Tstart, K **=** 10**)**

MSE.base2 **=** fm.base2.cv**$**cv**^**2

## run logistic regression with a few covariates

# use Tstart, Ttotal, Turnsmin, Umed

# set a reference level (chose slowest activity, swim)

df**$**Activity **=** relevel**(**df**$**Activity, ref **=** "swim"**)**

# perform cv via a penalized maximum likelihood iterative scheme

# and regularization. 'class' returns classification error.

# ridge = 0 < alpha < 1 = lasso

mat **=** data.matrix**(**data.frame**(**df**$**Tstart, df**$**Ttotal, df**$**Turnsmin, df**$**Umed**)**,rownames.force **=** **NA)**

#mat = data.matrix(df,rownames.force = NA)

fm.base2.cat **=** cv.glmnet**(**mat, df**$**Activity, family **=** "multinomial",

nfolds **=** 10, alpha **=**0, type.measure **=** "class"**)**

# use the lambda that yields the minimum mean cv error

fit.base2.cat **=** predict.cv.glmnet**(**object **=** fm.base2.cat, newx **=** mat,

s **=** "lambda.min", type **=** "class"**)**

# convert to factor to compare to observations

fit.base2.cat **=** factor**(**fit.base2.cat**)**

# mean cv error (misclassification error)

cvErr **=** fm.base2.cat**$**cvm**[**fm.base2.cat**$**lambda **==** fm.base2.cat**$**lambda.min**]**

# compart to observation

df.check **=** data.frame**(**Accuracy **=** **(**fit.base2.cat **==** df**$**Activity**)**,

Activity **=** df**$**Activity**)**

Acc **=** sum**(**df.check**$**Accuracy**)/**nrow**(**df.check**)**

# calculate accurace per activity

df.sub **=** df.check**[(**df.check**$**Activity **==** 'run'**)**,1**]**

Acc.run **=** sum**(**df.sub**)/**length**(**df.sub**)**

df.sub **=** df.check**[(**df.check**$**Activity **==** 'ride'**)**,1**]**

Acc.ride **=** sum**(**df.sub**)/**length**(**df.sub**)**

df.sub **=** df.check**[(**df.check**$**Activity **==** 'walk'**)**,1**]**

Acc.walk **=** sum**(**df.sub**)/**length**(**df.sub**)**

df.sub **=** df.check**[(**df.check**$**Activity **==** 'swim'**)**,1**]**

Acc.swim **=** sum**(**df.sub**)/**length**(**df.sub**)**

## part 2

# part a: linear regression

## Transform some variables with log

#df.cov.tran = df.cov

#df.cov.tran$dEmax = log(df.cov.tran$dEmax + .005) # non-transformed is better correlated

#df.cov.tran$Grademed = log(df.cov.tran$Grademed + .005)

#df.cov.tran$Turnsmin = log(df.cov.tran$Turnsmin)

#df.cov.tran$Ttotal = log(df.cov.tran$Ttotal)

#df.cov.tran$Dtotal = log(df.cov.tran$Dtotal)

#df.cov.tran$Dispmed = log(df.cov.tran$Dispmed)

coef.tran **=** rep**(NA**,Ncov**)**

**for** **(**i **in** 1**:**Ncov**-**1**)** **{**

coef.tran**[**i**]** **=** cor**(**df.cov.tran**[**,i**]**, df**$**Tstart**)**

**}**

plot**(**factor**(**covar**)**, coef, las **=** 2**)**

points**(**factor**(**covar**)**, coef.tran, las **=** 2**)**

# try a complex model. include all single covariates and all 2-way interactions

fm **<-** lm**(** data **=** df, formula **=** Tstart **~** . **+** .**:**.**)**

fm.cv **=** cvFit**(**fm, data **=** df, y **=** df**$**Tstart, K **=** 10**)**

MSE.fm.all **=** fm.cv**$**cv**^**2

# perform AIC and BIC to see which results in low MSE and complexity

# AIC backward

fm.AIC.bac **=** stepAIC**(**fm, direction **=** "backward", k **=** 2**)**

# BIC backward

fm.BIC.bac **=** stepAIC**(**fm, direction **=** "backward", k **=** log**(**nrow**(**df**)))**

# cross-validate each model

fm.AIC.bac.cv **=** cvFit**(**fm.AIC.bac, data **=** df, y **=** df**$**Tstart, K **=** 10**)**

fm.BIC.bac.cv **=** cvFit**(**fm.BIC.bac, data **=** df, y **=** df**$**Tstart, K **=** 10**)**

# calculate MSE

MSE.AIC.bac **=** fm.AIC.bac.cv**$**cv**^**2

MSE.BIC.bac **=** fm.BIC.bac.cv**$**cv**^**2

## part 2

# part a: logistic regression

# perform cv via a penalized maximum likelihood iterative scheme

# and regularization. 'class' returns classification error.

# ridge = 0 < alpha < 1 = lasso

mat **=** data.matrix**(**df,rownames.force **=** **NA)**

fm.cat **=** cv.glmnet**(**mat, df**$**Activity, family **=** "multinomial",

nfolds **=** 10, alpha **=**0, type.measure **=** "class"**)**

# use the lambda that yields the minimum mean cv error

fit.cat **=** predict.cv.glmnet**(**object **=** fm.cat, newx **=** mat,

s **=** "lambda.min", type **=** "class"**)**

# convert to factor to compare to observations

fit.cat **=** factor**(**fit.cat**)**

# mean cv error (misclassification error)

cvErr **=** fm.cat**$**cvm**[**fm.cat**$**lambda **==** fm.cat**$**lambda.min**]**

# compart to observation

df.check **=** data.frame**(**Accuracy **=** **(**fit.cat **==** df**$**Activity**)**,

Activity **=** df**$**Activity**)**

Acc **=** sum**(**df.check**$**Accuracy**)/**nrow**(**df.check**)**

# calculate accurace per activity

df.sub **=** df.check**[(**df.check**$**Activity **==** 'run'**)**,1**]**

Acc.run **=** sum**(**df.sub**)/**length**(**df.sub**)**

df.sub **=** df.check**[(**df.check**$**Activity **==** 'ride'**)**,1**]**

Acc.ride **=** sum**(**df.sub**)/**length**(**df.sub**)**

df.sub **=** df.check**[(**df.check**$**Activity **==** 'walk'**)**,1**]**

Acc.walk **=** sum**(**df.sub**)/**length**(**df.sub**)**

df.sub **=** df.check**[(**df.check**$**Activity **==** 'swim'**)**,1**]**

Acc.swim **=** sum**(**df.sub**)/**length**(**df.sub**)**

# plot things

barchart**(**df**$**Activity**)**

title**(**main **=** "Observed"**)**

barchart**(**fit.cat**)**

title**(**main **=** "Modeled"**)**

plot**(**df**$**Tstart, ylab **=** "Start Time"**)**

points**(**fm.BIC.bac**$**fitted.values, col **=** "blue"**)**

**‘PreProcess\_partII.R’**

# preprocessing for part II

df <- readRDS("TrainingData.rds")

ID <- as.data.frame(cbind(df$Athlete, df$FileID))

## create covariate data frame without response variable

# reassign df without IDs

df = df[,colnames(df)!="Athlete"]

df = df[,colnames(df)!="FileID"]

# remove some covariates

df = df[,colnames(df)!="Umean"]

df = df[,colnames(df)!="Umax"]

df = df[,colnames(df)!="Turnsmean"]

df = df[,colnames(df)!="Ibike"]

# convert Iswim integer to numeric to apply mean to data frame as matrix

df$Iswim = as.numeric(df$Iswim)

df.cov = df[,colnames(df)!="Tstart"]

## rerun correlation coefficients for kept covariates

Ncov = length(df.cov)

coef = rep(NA,Ncov)

for (i in 1:Ncov-1) {

coef[i] = cor(df.cov[,i], df$Tstart)

}

# associate variable name to the coefficient

covar = colnames(df.cov)

plot(factor(covar), coef, las = 2)

# mutual correlations

#ggpairs(df, las =1)