Report code

Hannah Sansford, Ettore Fincato, Harry Tata

This markdown file contains the code used to produce all plots and conduct all statistical analysis contained within the report.

Setup

First load the important packages.

```
library(tidyverse)
library(reshape2)
library(ggplot2)
library(glmnet)
library(ROCR)
library(caret)
library(cvms)
library(tibble)
library(pROC)
library(checkmate)
library(BBmisc)
library(testit)
library(devtools)
library(mlr)
```

Next, install and load the `higgsboson` package.

```
install_github("https://github.com/hsansford1/higgsboson")
```

```
##
##      checking for file ‘/tmp/Rtmpc2FceE/remotes13c25ad9f575/hsansford1-higgsboson-5b77e2b/DESCRIPTIO
##   -  preparing ‘higgsboson’:
##    checking DESCRIPTION meta-information ...  v  checking DESCRIPTION meta-information
##   -  checking for LF line-endings in source and make files and shell scripts
##   -  checking for empty or unneeded directories
## -  looking to see if a ‘data/datalist’ file should be added
##        NB: this package now depends on R (>= 3.5.0)
##        WARNING: Added dependency on R >= 3.5.0 because serialized objects in  serialize/load version
## -  building 'higgsboson_0.1.0.tar.gz'
##
##
```

```
library(higgsboson)
```

Access the data included in the package. First load the data used to train the models.

```
train <- higgsboson::training
```

Put the data in the format required for training:

```
df_train <- train[,2:33] #remove eventid
df_train <- df_train[,-31] #remove weights

df_train$Label <- ifelse(df_train$Label=="s",1,0) #encode "s" and "b" to 1 - 0 (resp.)
df_train$Label <- as.factor(df_train$Label) #need this as factor for caret package
```

Use the `reweight` function to normalise the weights (use `??reweight` to see function help). `Ns()` and `Nb()` are hardcoded values of N_s and N_b for the higgsboson dataset (see 'Problem Formulation' section of report for explanation of these values).

```
weights <- reweight(train$Weight, df_train$Label, Ns(), Nb())
```

Set all missing values equal to zero.

```
df_train[df_train==-999] <- 0
```

To get a standardised data set `st_train`, run the following.

```
st_train <- df_train
st_train <- as.data.frame(scale(df_train[,1:30]))
st_train["Label"] <- df_train$Label
```

Weighted Logistic Regression with custom AMS metric

The `mlr` package allows the use of custom metrics. First, define the classification task and make the logistic learner

```
trainTask <- makeClassifTask(data = st_train,
                             target = "Label",
                             positive = 1,
                             weights = weights
                             )

logistic.learner <- makeLearner("classif.logreg", predict.type = "response")
```

Use the custom AMS measure created in the `higgsboson` package `AMS_measure()` to conduct cross-validation on fitting the weighted logistic regression (WLR). This gives us an idea of the AMS we can expect to get from a test set.

```
set.seed(22)
AMS_mlr <- AMS_measure()
cv.logistic <- crossval(learner = logistic.learner, task = trainTask, iters = 5,
                        stratify = TRUE,
                        measures = AMS_mlr,
                        show.info = F, models=TRUE)
cv.logistic$aggr
```

```
## AMS_weighted.test.mean
##              0.2326592
```

```
cv.logistic$measures.test
```

```
##   iter AMS_weighted
## 1    1    0.4093149
## 2    2    0.1659211
## 3    3    0.2190233
## 4    4    0.2178583
## 5    5    0.1511781
```

Now, we fit a WLR model using the whole training data set and check how it performs on the test data.

```
train_control <- trainControl(method = "cv", number = 10)

fmodel <- caret::train(Label ~ .,
                       data = st_train,
                       trControl = train_control,
                       method = "glm",
                       weights = weights,
                       family=binomial()
)

test <- higgsboson::test

df_test <- test[,2:33] #remove eventid
df_test <- df_test[,-31] #remove weights
df_test$Label <- ifelse(df_test$Label=="s",1,0) #encode "s" and "b" to 1 - 0 (resp.) for logistic regre
df_test$Label <- as.factor(df_test$Label) #need this as factor for caret
# set misssing values to 0 and standardise data
df_test[df_test==-999] <- 0
st_test <- as.data.frame(scale(df_test[,1:30]))
st_test$Label <- df_test$Label

weights_test <- reweight(test$Weight, st_test$Label, Ns(), Nb()) # extract weights

truth <- st_test$Label
response <- predict(fmodel, newdata = st_test[,-length(st_test)])

AMS_weighted(truth, response, weights_test)
```

```
## [1] 0.1818793
```

Two-stage maximisation

Below is the code for conducting the two-stage maximisation of the AMS.

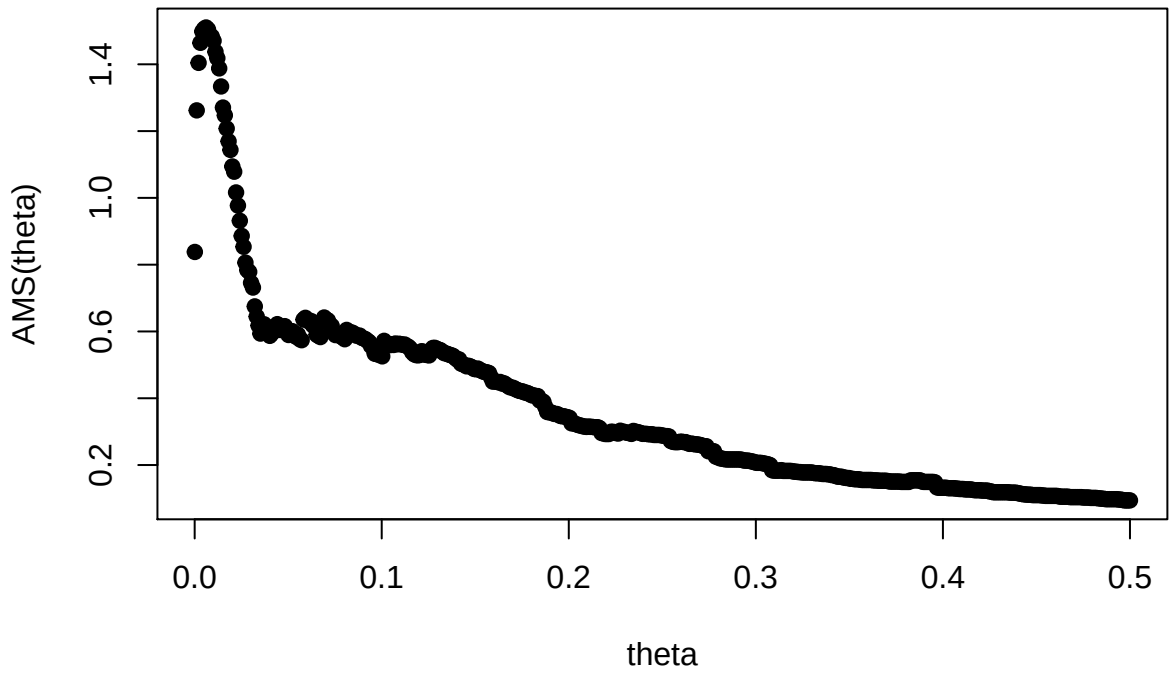
First we create a training/validation split of the training data that preserves the overall class distribution.

```
trainIndex <- createDataPartition(st_train$Label, p = .8, list = FALSE, times = 1)

Train <- st_train[ trainIndex,]
Valid  <- st_train[-trainIndex,]
```

We then fit a WLR model using the new training set `Train`, before visualising how the AMS varies with threshold `theta`. This informs the users decision of how to set the parameters `theta_0` and `theta_1` in the function `threshold_CV` in the `higgsboson` package (use `??threshold_CV` for help on this function). We can see that the peak definitely occurs in the range 0 to 0.1.

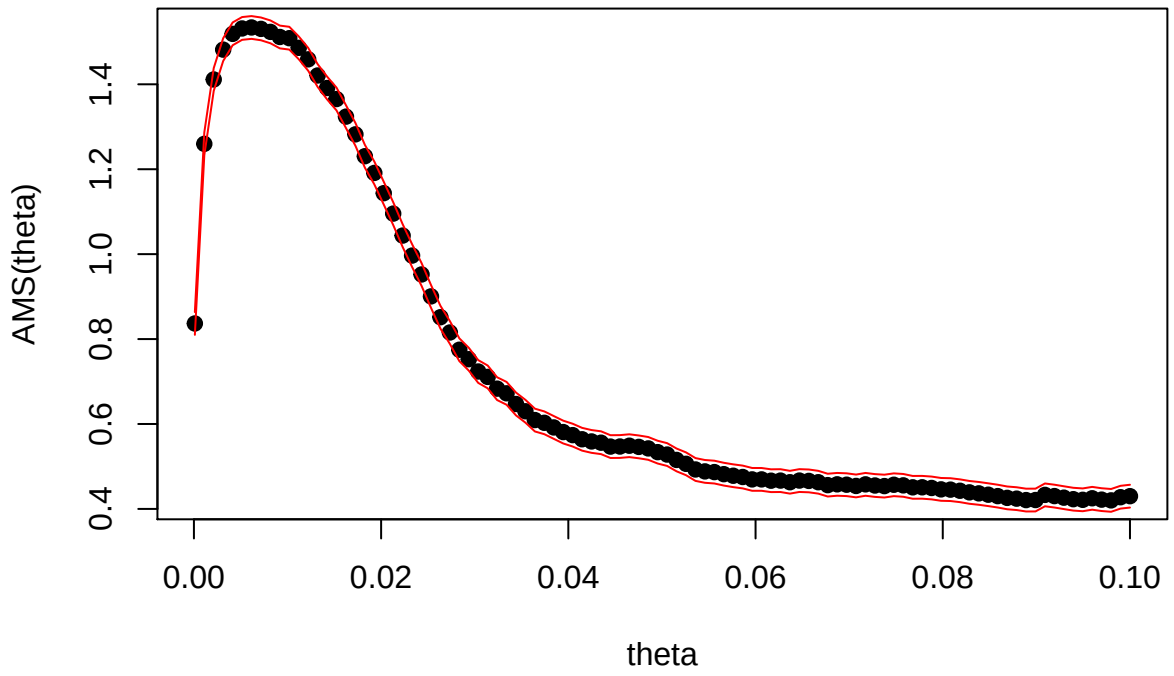
```
weights_Train <- reweight(weights[trainIndex], Train$Label, Ns(), Nb())
weights_Valid <- reweight(weights[-trainIndex], Valid$Label, Ns(), Nb())

logreg_weighted2 <- caret::train(Label ~ .,
                       data = Train,
                       method = "glm",
                       weights = weights_Train,
                       family=binomial()
```

```
)


#Plot AMS for different values of threshold theta
theta_vals <- as.data.frame(seq(0.0001, 0.5, length.out=500)) # generate small sample thresholds theta
AMS_vals <- apply(theta_vals, 1, AMS(logreg_weighted2,Valid[,1:30],Valid[31], weights_Valid)) #compute
plot(as.array(unlist(theta_vals)), AMS_vals, xlab="theta", ylab="AMS(theta)", pch=19) #plot it
```



Now, using the plot above, we can set `theta_0 = 0.0001` and `theta_1=0.1` in the `threshold_CV` function.

```
theta_CV <- threshold_CV(st_train, st_train$Label, weights, theta_0=0.0001, theta_1=0.1, n=100)
```



```
theta_CV
```

```
## $max_theta
## [1] 0.006154545
##
## $max_AMS
## [1] 1.533845
##
## $mean_AMS_sd
## [1] 0.0268266
##
## $max_thetas
## [1] 0.007163636 0.007163636 0.005145455 0.006154545 0.005145455
```

We can now use the threshold found in the cross-validation to find our predictions on the test set and the resulting AMS.

```
theta <- theta_CV$max_theta

probabilities <- predict(logreg_weighted2$finalModel, st_test[,1:30], type = "response")

predicted.classes <- ifelse(probabilities > theta, 1, 0)

Label_valid <-  as.array(unlist(st_test[,31]))
Label_valid <- as.numeric(levels(Label_valid))[Label_valid] #convert from factor to numeric

AMS_weighted(Label_valid, predicted.classes, weights_test)
```

```
## [1] 1.526392
```

Principal-Component Analysis

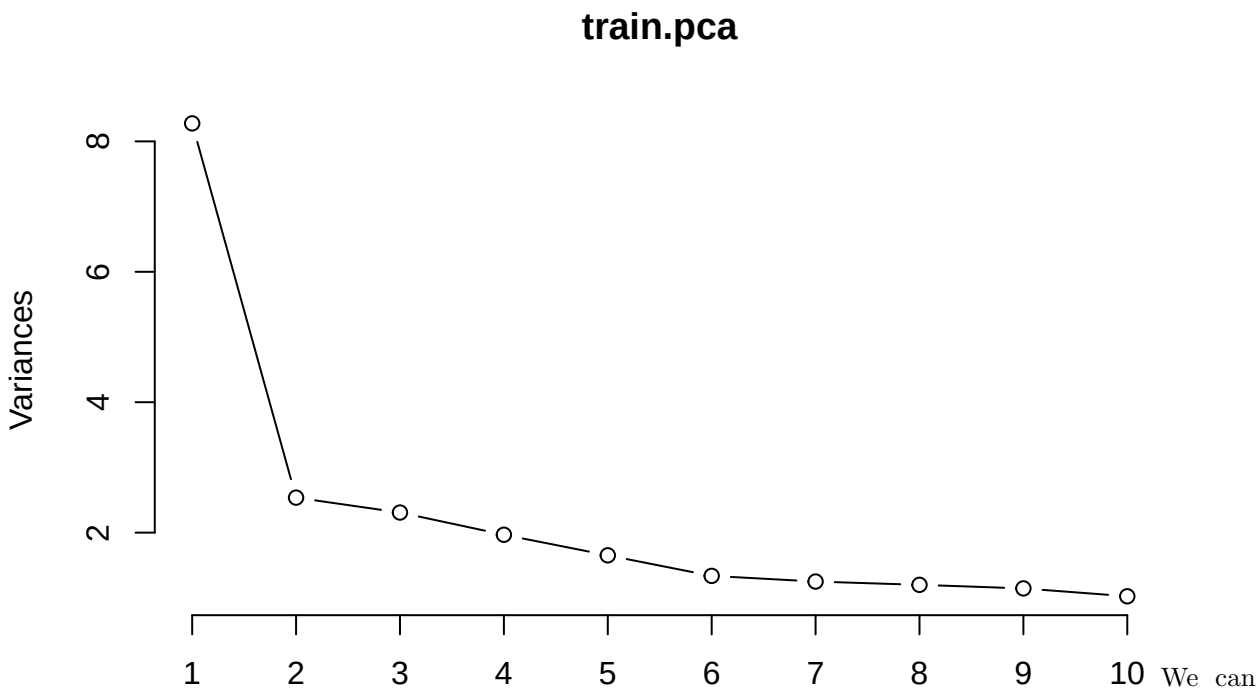
Alternatively, we can perform PCA on the data set before fitting a model and tuning the threshold. This will reduce the dimension of the data set, improving the speed of the function and, hopefully, reduce any overfitting of the more complicated model.

```
train.pca <- prcomp(st_train[,1:30])
summary(train.pca)
```

```
## Importance of components:
##                            PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.8768 1.59275 1.51937 1.40279 1.28531 1.15624 1.11806
## Proportion of Variance 0.2759 0.08456 0.07695 0.06559 0.05507 0.04456 0.04167
## Cumulative Proportion  0.2759 0.36042 0.43737 0.50296 0.55803 0.60259 0.64426
##                            PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     1.09531 1.07001 1.01201 1.00306 0.93818 0.87197 0.86820
## Proportion of Variance 0.03999 0.03816 0.03414 0.03354 0.02934 0.02534 0.02513
## Cumulative Proportion  0.68425 0.72242 0.75656 0.79009 0.81943 0.84478 0.86990
##                           PC15    PC16    PC17    PC18    PC19   PC20    PC21
## Standard deviation     0.80557 0.74824 0.71231 0.66525 0.60098 0.5198 0.48412
## Proportion of Variance 0.02163 0.01866 0.01691 0.01475 0.01204 0.0090 0.00781
## Cumulative Proportion  0.89153 0.91020 0.92711 0.94186 0.95390 0.9629 0.97072
##                           PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     0.46176 0.39368 0.37755 0.35851 0.30716 0.25633 0.24541
## Proportion of Variance 0.00711 0.00517 0.00475 0.00428 0.00314 0.00219 0.00201
## Cumulative Proportion  0.97782 0.98299 0.98774 0.99203 0.99517 0.99736 0.99937
```

```
##                             PC29      PC30
## Standard deviation     0.13754 3.534e-06
## Proportion of Variance 0.00063 0.000e+00
## Cumulative Proportion  1.00000 1.000e+00
```

```
screeplot(train.pca, type="lines")
```



We can see that the first two PCA's account for the majority of the vaiance, but we shall keep the first three as a precaution.

```
st_train_dimred <- data.frame("PCA1"=train.pca$x[,1],"PCA2"=train.pca$x[,2],"PCA3"=train.pca$x[,3])
st_train_dimred["Label"] <- st_train$Label
```

Now, we can perform the first stage of the two-stage procedure as above.

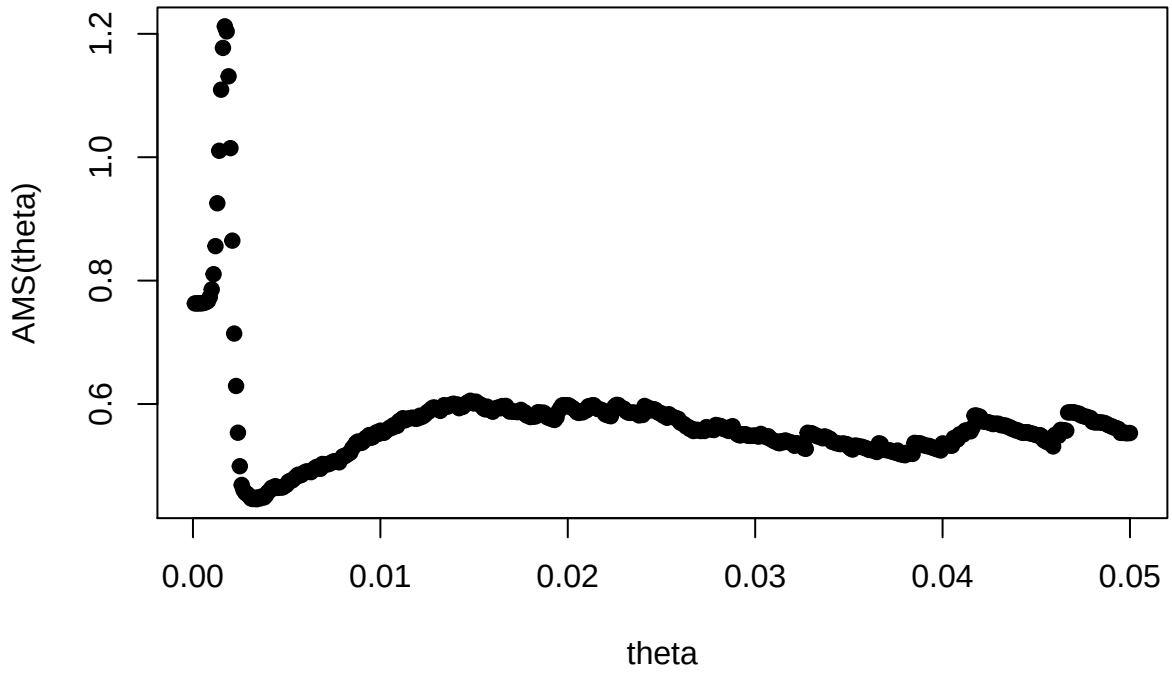
```
trainIndex <- createDataPartition(st_train_dimred$Label, p = .8,
                                  list = FALSE,
                                  times = 1)

Train <- st_train_dimred[ trainIndex,]
Valid  <- st_train_dimred[-trainIndex,]


weights_Train <- reweight(weights[trainIndex], Train$Label, Ns(), Nb())
weights_Valid <- reweight(weights[-trainIndex], Valid$Label, Ns(), Nb())

logreg_weighted_pca <- caret::train(Label ~ .,
                                    data = Train,
                                    method = "glm",
                                    weights = weights_Train,
                                    family=binomial()
)
print(logreg_weighted_pca)
```

```
## Generalized Linear Model
```

```
##
## 200001 samples
##      3 predictor
##      2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200001, 200001, 200001, 200001, 200001, 200001, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.6580688  0.0006967816
```

Now we can plot the AMS for varying values of the threshold theta.

```
theta_vals <- as.data.frame(seq(0.0001, 0.05, length.out=500)) # generate small sample thresholds theta
AMS_vals <- apply(theta_vals, 1, AMS(logreg_weighted_pca,Valid[,1:3],Valid[4], weights_Valid)) #compute
plot(as.array(unlist(theta_vals)), AMS_vals, xlab="theta", ylab="AMS(theta)", pch=19) #plot it
```




```
max_theta <- theta_vals[which.max(AMS_vals),1]
max_theta
```

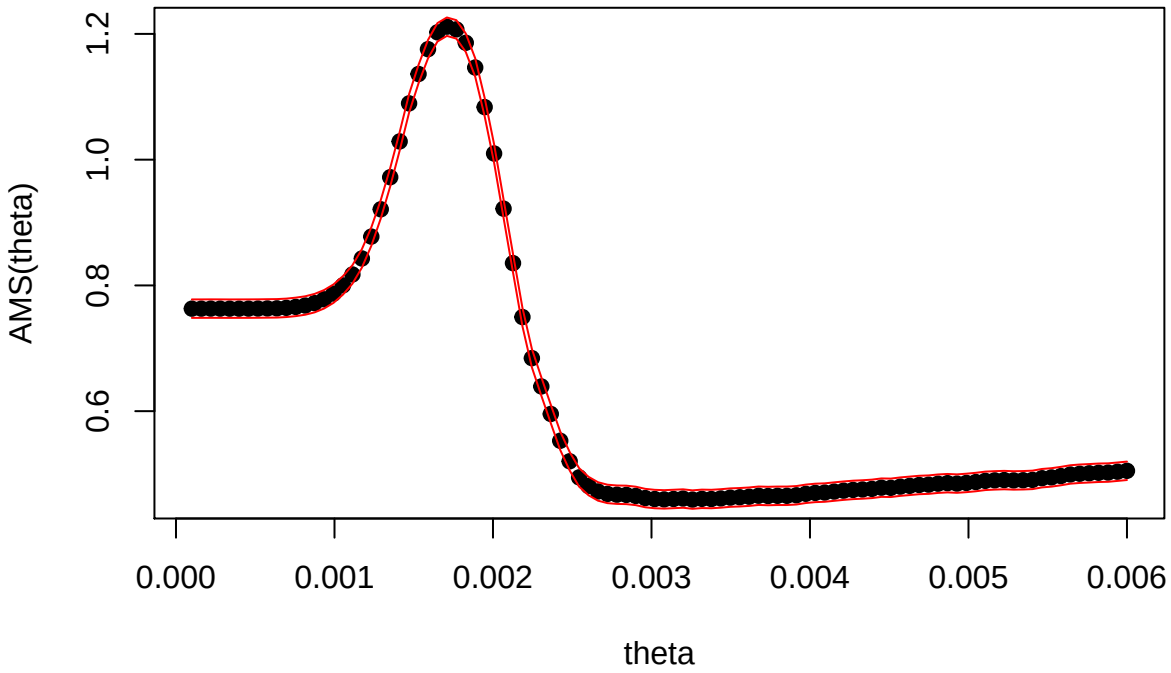
```
## [1] 0.0017
```

```
max_AMS <- AMS_vals[which.max(AMS_vals)]
max_AMS
```

```
## [1] 1.212122
```

We can see that the peak occurs between very small values of theta, hence we try `theta_0=0.0001` and `theta_1=0.006` in the `threshold_CV` function.

```
theta_CV <- threshold_CV(st_train_dimred[,1:3], st_train_dimred$Label, weights=weights, theta_0=0.0001,
```




```
theta_CV  #less variance, both of theta and of AMS
```

```
## $max_theta
## [1] 0.001709091
##
## $max_AMS
## [1] 1.211464
##
## $mean_AMS_sd
## [1] 0.01466137
##
## $max_thetas
## [1] 0.001709091 0.001709091 0.001709091 0.001709091 0.001709091
```

Finally, we can now use the threshold found in the cross-validation to find our predictions on the test set and the resulting AMS.

```
theta <- theta_CV$max_theta

probabilities <- predict(logreg_weighted2$finalModel, st_test[,1:30], type = "response")

predicted.classes <- ifelse(probabilities > theta, 1, 0)

Label_valid <-  as.array(unlist(st_test[,31]))
Label_valid <- as.numeric(levels(Label_valid))[Label_valid] #convert from factor to numeric

AMS_weighted(Label_valid, predicted.classes, weights_test)
```

```
## [1] 1.370724
```