booster [default=gbtree]

* which booster to use, can be gbtree, gblinear or dart. gbtree and dart use tree based model while gblinear uses linear function.

We choose gbtree because that’s the one we learn in the class.

objective [default=reg:linear]

* “reg:linear” –linear regression
* “reg:logistic” –logistic regression
* “binary:logistic” –logistic regression for binary classification, output probability
* “binary:logitraw” –logistic regression for binary classification, output score before logistic transformation
* “gpu:reg:linear”, “gpu:reg:logistic”, “gpu:binary:logistic”, gpu:binary:logitraw” –versions of the corresponding objective functions evaluated on the GPU; note that like the GPU histogram algorithm, they can only be used when the entire training session uses the same dataset
* “count:poisson” –poisson regression for count data, output mean of poisson distribution
  + max\_delta\_step is set to 0.7 by default in poisson regression (used to safeguard optimization)
* “survival:cox” –Cox regression for right censored survival time data (negative values are considered right censored). Note that predictions are returned on the hazard ratio scale (i.e., as HR = exp(marginal\_prediction) in the proportional hazard function h(t) = h0(t) \* HR).
* “multi:softmax” –set XGBoost to do multiclass classification using the softmax objective, you also need to set num\_class(number of classes)
* “multi:softprob” –same as softmax, but output a vector of ndata \* nclass, which can be further reshaped to ndata, nclass matrix. The result contains predicted probability of each data point belonging to each class.
* “rank:pairwise” –set XGBoost to do ranking task by minimizing the pairwise loss
* “reg:gamma” –gamma regression with log-link. Output is a mean of gamma distribution. It might be useful, e.g., for modeling insurance claims severity, or for any outcome that might be [gamma-distributed](https://en.wikipedia.org/wiki/Gamma_distribution#Applications)
* “reg:tweedie” –Tweedie regression with log-link. It might be useful, e.g., for modeling total loss in insurance, or for any outcome that might be [Tweedie-distributed](https://en.wikipedia.org/wiki/Tweedie_distribution#Applications).

There are more than 14 objective’s inside of XGB method. We decide to focus on multi:softprob,binary:logistic and gpu:binary:logistic.

For evaluation method in the XGB, I pick “mlogloss” which means the multiclass logloss(negative of log-likelyhood), “rmse” root mean square error and “error” classification error rate.

eta [default=0.3, alias: learning\_rate]

* step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative.[might want increase num\_round]
* range: [0,1]
* try()

gamma [default=0, alias: min\_split\_loss]

* minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm will be.
* range: [0,∞]
* 3\*(1:4)

max\_depth [default=6]

* maximum depth of a tree, increase this value will make the model more complex / likely to be overfitting. 0 indicates no limit, limit is required for depth-wise grow policy.
* range: [0,∞]

We set it to be log(98359)

[1] 11.49638

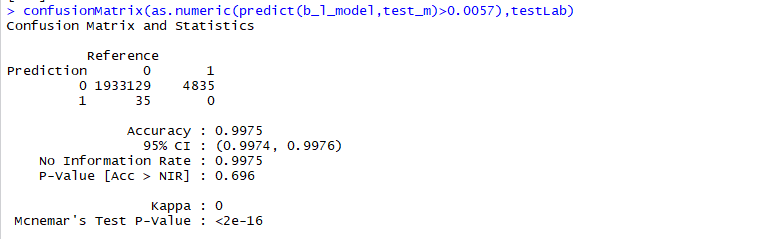
max\_delta\_step [default=0]

* Maximum delta step we allow each tree’s weight estimation to be. If the value is set to 0, it means there is no constraint. If it is set to a positive value, it can help making the update step more conservative. Usually this parameter is not needed, but it might help in logistic regression when class is extremely imbalanced. Set it to value of 1-10 might help control the update
* range: [0,∞]

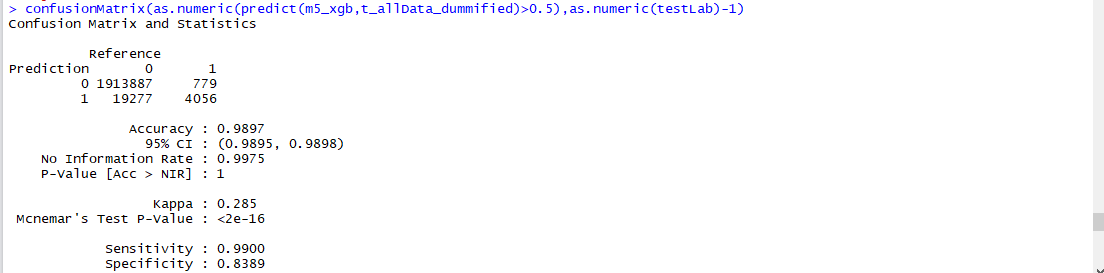
training threshold :

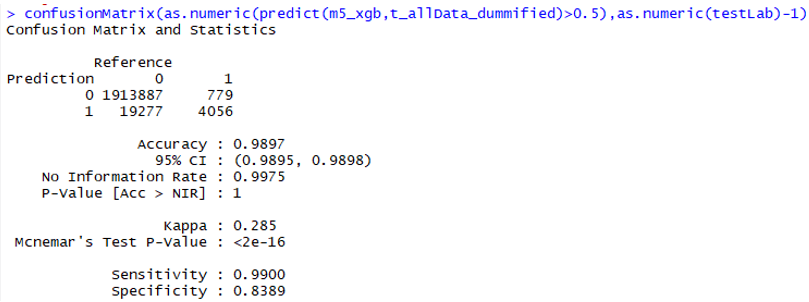


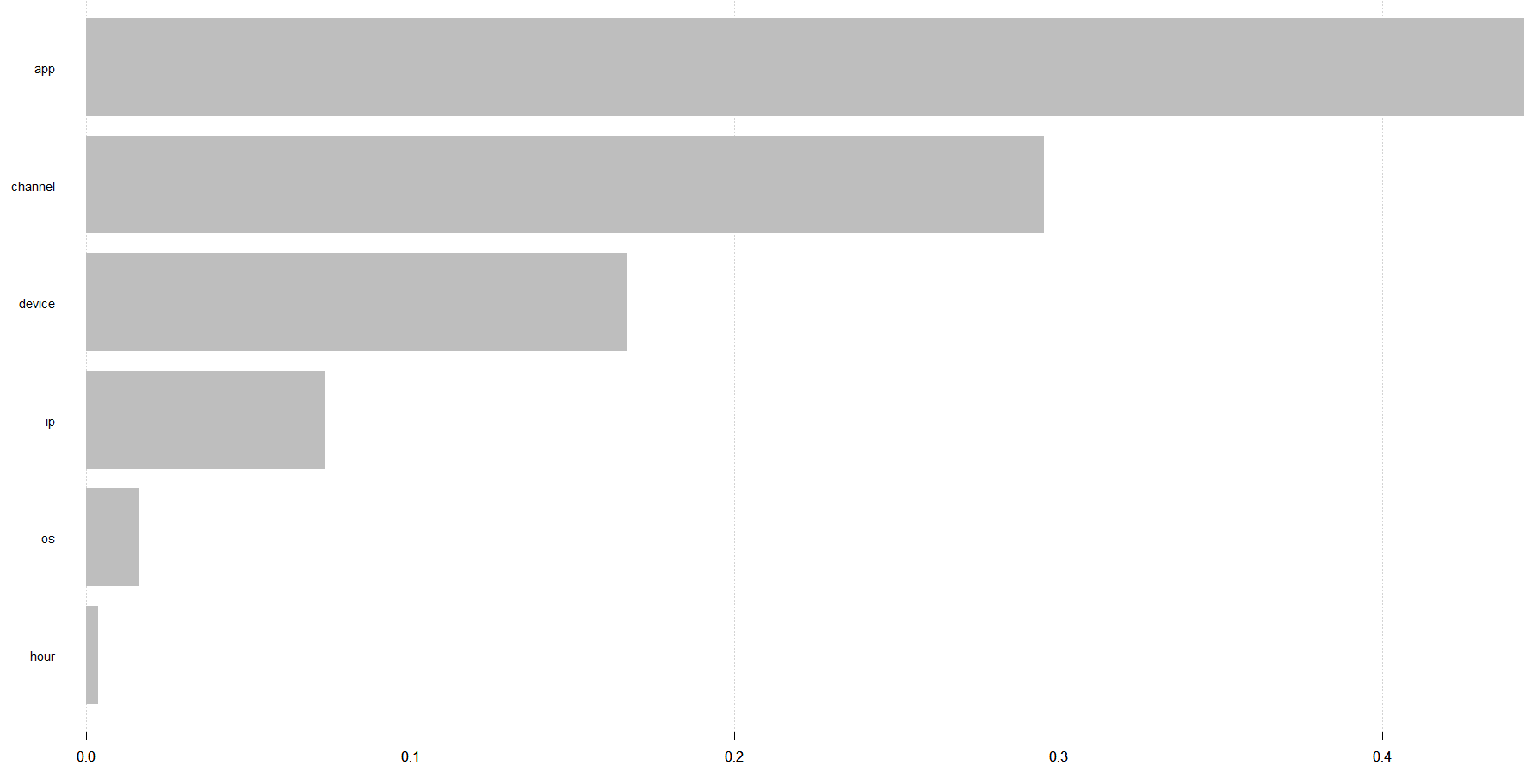
Test on test dataset.



trainData\_s







From numeric dataset