Universidade de Brasília

DEPARTAMENTO DE ESTATÍSTICA

27 de dezembro de 2024

Lista 2

Prof. Drª. Thais Carvalho Rodrigues

Tópicos 2 - Aprendizagem de máquina

Aluno: Bruno Gondim Toledo | Matrícula: 15/0167636

1. Se você treinou 5 modelos diferentes com exatamente os mesmos dados de treino e todos alcançaram 95% de precisão, existe alguma forma de combiná-los para obter melhores resultados? Explique.

Sim, é possível. Com técnicas como *Bagging* e, principalmente, *Boosting*, é possível realizar o *ensembling* (combinação) dos modelos, afim de obter resultados ainda melhores que nos modelos individuais. Especialmente para o caso de *Boosting*, alguma técnicas como *XGBoost* irão focar justamente nos resíduos do modelo anterior, afim de obter uma precisão ainda maior e ainda assim sem incorrer em *overfitting*.

2. É possível acelerar o treinamento de um agrupamento com o método bagging? E boosting? Explique.

Sim e depende. No método Bagging, todos os procedimentos podem ser paralelizados. Portanto, o limite da complexidade do modelo será o poder computacional do usuário. Quanto mais poder, mais rápido o modelo será. No caso do Boosting, em geral não. Como em Boosting utilizaremos modelos sequenciais, dependeremos do modelo anterior para fazer o próximo, ou seja, essa modelagem não pode ser paralelizada. Entretanto, é possível tornar mais eficiente, como paralelizando atividades secundárias como seleção de *features*, ETL, e demais etapas intermediárias da modelagem. Algoritmos como *XGBoost* são conhecidos justamente por paralelizar tudo que é possível, tornando sequencial apenas a modelagem em sí. Além disso, é possível escrever em linguagens eficientes como Julia, Rust ou C, o que também potencialmente acelera o treinamento do modelo. Mas sempre haverá uma limitação do quão eficiente e rápido este método pode ser, por se tratar de uma modelagem sequencial.

3. Se o seu agrupamento do Adaboost se subajustar aos dados de treino, quais hiperparâmetros você deve ajustar e como?

Basicamente, o número de etapas. Vimos em aula que é matematicamente demonstrado que com um classificador fraco e modelos suficientes, chegaremos com probabilidade 1 a um modelo final de erro mínimo. Esta convergência pode não se dar com um ou dois *tree stump*, então teoricamente basta aumentar o número de modelos sequenciais do Adaboost que chegaremos a um modelo melhor.

4. Explique como o bagging e boosting impactam o viés e a variância do estimador.

O bagging foca na redução da variância a partir da técnica de reamostragem *Bootstrap*; esta pode levar a um aumento do viés caso a amostra seja pouco representativa da população, porém se estivermos em mãos de uma boa amostra, isso quase certamente não ocorrerá, e teremos uma redução de variância “limpa”.

O boosting foca na redução do viés, justamente combinando modelos afim de não permitir que um modelo específico viesado para alguma covariável ou escala se torne predominante, sendo assim bastante robusto a *overfitting*.

5. Ao conjunto de dados MNIST, ajuste uma floresta aleatória e compare o desempenho do modelo com o do exercício 8 da Lista 1.

```r
p_load(randomForest,keras,tidymodels,agua,tictoc,h2o,conflicted)

h2o::h2o.init()
```

```
 Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         13 minutes 49 seconds
    H2O cluster timezone:       America/Sao_Paulo
    H2O data parsing timezone:  UTC
    H2O cluster version:        3.46.0.6
    H2O cluster version age:    1 month and 24 days
    H2O cluster name:           H2O_started_from_R_Bruno_kro571
    H2O cluster total nodes:    1
    H2O cluster total memory:   3.92 GB
    H2O cluster total cores:    16
    H2O cluster allowed cores:  16
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    R Version:                  R version 4.4.2 (2024-10-31 ucrt)
```

```r
tidymodels_prefer()
conflict_prefer_all("h2o")
mnist <- dataset_mnist()

train <- mnist$train$x
train_labels <- mnist$train$y

train <- array_reshape(train, c(nrow(train), 784)) / 255
train <- data.frame(train, label = train_labels)

test <- mnist$test$x
test_labels <- mnist$test$y

test <- array_reshape(test, c(nrow(test), 784)) / 255
test <- data.frame(test, label = test_labels)

train$label <- as.factor(train$label)
test$label <- as.factor(test$label)

set.seed(150167636)
```

```r
data <- initial_split(train, prop = 0.8)
train <- training(data)
test <- testing(data)

rf_model = rand_forest(
  mode = "classification",
  engine = "h2o"#,
  # mtry = tune(),
  # trees = tune(),
  # min_n = tune()
) %>%
  translate()

recipe <- recipe(label ~ ., data = train)

wf <- workflow() %>%
  add_recipe(recipe) %>%
  add_model(rf_model)

# hp <- extract_parameter_set_dials(rf_model) %>%
#   finalize(train)
#
# grid <- grid_regular(hp,
#                      levels = 3)
#
# folds <- vfold_cv(sample_n(validation,1000), v = 5)
#
# tune <- wf %>%
#   tune_grid(resamples = folds, grid = grid)
#
# best_params <- select_best(tune, "accuracy")
#
# final_wf <- wf %>%
#   finalize_workflow(best_params)
#
# modelo <- fit(final_wf, data = train)
#
# pred <- predict(modelo, test) %>%
#   bind_cols(test) %>%
#   mutate(pred = factor(.pred_class, levels = levels(test$label)))
#
# accuracy <- accuracy(pred, truth = label, estimate = pred)

tic()
fit <- fit(wf,train)
```

```
toc()
```

```
116.49 sec elapsed
```

```
resultados = fit$fit$fit$fit

kable(h2o.confusionMatrix(resultados))
```

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	4673	0	7	5	6	8	16	1	33	3	0.016624	79 / 4.752
1	0	5289	32	12	11	6	6	10	14	4	0.017644	95 / 5.384
2	25	11	4612	35	18	2	12	25	43	11	0.037964	182 / 4.794
3	7	8	71	4572	12	59	7	40	69	34	0.062922	307 / 4.879
4	3	4	14	2	4474	3	26	18	18	92	0.038676	180 / 4.654
5	17	4	11	40	11	4133	59	6	41	20	0.048134	209 / 4.342
6	14	8	5	1	16	37	4578	0	30	1	0.023880	112 / 4.690
7	9	20	52	8	24	3	0	4831	12	70	0.039371	198 / 5.029
8	11	27	37	39	22	37	28	10	4469	53	0.055778	264 / 4.733
9	14	11	9	46	54	16	4	59	34	4496	0.052076	247 / 4.743
Totals	4773	5382	4850	4760	4648	4304	4736	5000	4763	4784	0.039020	1.873 / 48.000

OBS: Foi tentado inicialmente uma abordagem com o pacote ::randomForest para a construção da floresta aleatória com tune de parâmetros, porém rodou por horas sem resultado. Parti para uma modelagem mais eficiente, utilizando o frame h2o, com bons resultados.

Podemos observar que o modelo performou muito bem para o conjunto de dados, com erros variando de 1,6% para o dígito 0 para o máximo de 6,3% para o dígito 3. Além disso, esse frame h2o se mostrou extremamente parsimonioso, levando apenas 2 minutos para ajuste do modelo, utilizando abordagem paralelizada eficiente.

O erro foi extremamente compatível com o observado na questão 8 da lista 1, sendo que este modelo ajustou em pouco tempo para uma quantidade massiva de dados, enquanto o KNN tive de deixar a noite rodando para obter o ajuste!

Claro que para o KNN, a abordagem é muito mais simples, porém como está será realizada de forma maquínica de qualquer forma, e o custo envolvido é o custo de processamento basicamente, então em geral este modelo é muito melhor para um conjunto de dados tão grande ante ao KNN, se pensarmos por exemplo em sistemas embarcados ou modelagem em núvem em que o custo será o tempo de processamento basicamente.

6. Qual algoritmo de treinamento de regressão linear podemos utilizar para ajustar um conjunto de dados com milhares de covariáveis e 1 milhão de observações?

Teoricamente, qualquer um. Na prática, como vimos, quando a dimensionalidade do dado fica gigantesca, a maior parte dos algoritmos deixa de performar bem, não sendo possível aplicar qualquer técnica a um conjunto desta dimensionalidade. Possivelmente, algoritmos como random forest, gradiente descendente e técnicas de boosting para regressão podem funcionar para estes dados, mas podemos também aplicar por exemplo regressão sobre componentes principais do conjunto de dados.

Mas, mais especificamente, técnicas como Lasso e Ridge foram desenvolvidas justamente para lidar com este tipo de conjunto de dados, de dimensionalidade $n \to \infty; p \to \infty$, pois estas penalizam a complexidade do modelo, sendo capazes de deixar ínfimo ou até zerado o coeficiente de algumas covariáveis.

7. É importante escalonar as covariáveis para aplicação de regressão com regularização? Explique.

Sim, visto que a escala dos coeficientes das covariáveis irão impactar diretamente na penalização do algoritmo sobre elas. Portanto, não faria sentido aplicar penalidade a uma covariável não pela sua importância ou falta dela, mas tão somente pela escala da covariável. Claro que isso dificulta a interpretação do modelo, mas se estamos utilizando uma abordagem tão sofisticada em geral já estamos dispostos à abrir mão da explicabilidade do modelo de qualquer forma.

8. Suponha que você está ajustando um modelo utilizando o método do gradiente descendente em batch e plote o erro de validação em cada época. Se o erro de validação aumenta constantemente, o que pode estar acontecendo? Como corrigir?

Isso indica que o algoritmo está preso em algum mínimo local, ou seja, o algoritmo está viesado e irá ficar para sempre preso nesse viés. Justamente por isso, existem adaptações do algoritmo do gradiente descendente, como o gradiente descendente estocástico; visto que o algoritmo tradicional quando encontra um caminho descendente da derivada, irá ficar preso nesse “vale”. Isto não é um problema quando temos um plano convexo, com apenas um mínimo (que é o global). Entretanto, para problemas mais complexo, o que teremos é uma superfície de gradiente muito mais similar a um mapa topológico, com diversos mínimos locais. Portanto, para evitar o viés de um mínimo local, estratégias como utilizar uma variação do gradiente descendente, ou alterar o chute inicial do algoritmo, ou calibrar a *learning rate* e demais hiperparâmetros podem ser estratégias a serem tomadas para corrigir este problema.

9. Suponha que você esteja usando a regressão Ridge e nota que o erro do treino e o erro da validação são parecidos e altos. Então o modelo estimado tem um viés alto ou variância alta? Você deve aumentar o parâmetro α da regularização ou reduzí-lo?

Ambos os modelos tratam de reduzir variância ao introduzir um viés. Aprendemos em inferência estatística que o erro é formado pela soma da variância com o quadrado do viés, isto é, ao introduzir viés afim de diminuir variância, este deve reduzir ao menos a quantidade quadrática da variância para compensar a introdução do viés ao modelo para que o erro se torne ao menos constante. Neste caso, estamos talvez regularizando demais o modelo, introduzindo uma quantidade maior de viés ante a compensação na redução da variância. Portanto, neste caso seria importante reduzir o parâmetro α.

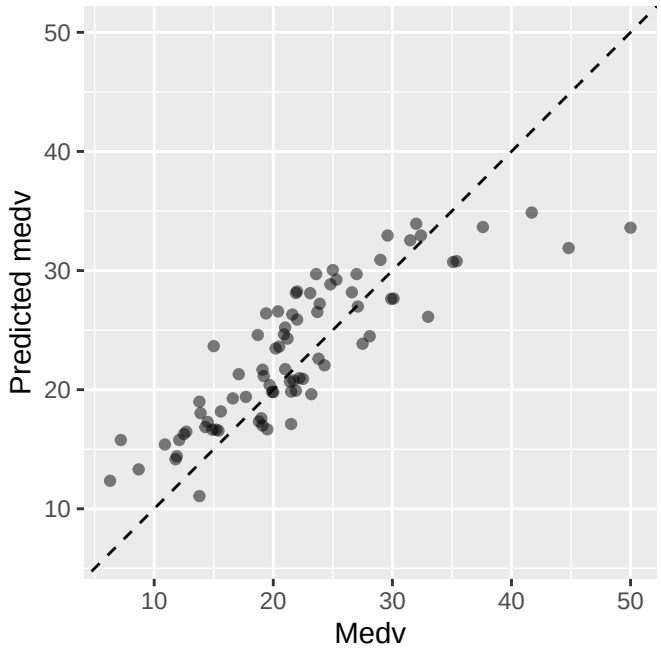
10. Qual é a principal vantagem da regressão LASSO em relação a regressão Ridge?

A principal vantagem é que a regressão Lasso ZERA coeficientes, sendo até indiretamente uma forma de realizar seleção de covariáveis importantes para a explicabilidade do modelo, removendo todas aquelas que não trazem contribuição significativa para a variável resposta.

11. Ajuste uma regressão polinomial com regularização ao banco de dados Boston (y= medv, x= lstat).

```
p_load(MASS,tidymodels)
rm(list=ls())

data(Boston)
# Boston

split <- initial_validation_split(Boston, prop = c(0.7, 0.15), strata = "medv")
train <- training(split)
val <- validation(split)
test <- testing(split)

recipe <- recipe(medv ~ lstat, data = train) %>%
  step_poly(lstat, degree = 4)

model <- linear_reg(penalty = tune(),
                    mixture = tune() # 0 = ridge; 1 = lasso
                    ) %>%
  set_engine("glmnet")

wf <- workflow() %>%
  add_recipe(recipe) %>%
  add_model(model)

hp <- extract_parameter_set_dials(model) %>%
  finalize(train)
```

```r
grid <- grid_regular(hp,
                     levels = 5)

folds <- vfold_cv(val, v = 5)

tune <- wf %>%
  tune_grid(resamples = folds, grid = grid)

best_params <- select_best(tune)

final_wf <- wf %>%
  finalize_workflow(best_params)

modelo <- fit(final_wf,train)

pred <- predict(modelo, test) %>%
  bind_cols(test) %>%
  dplyr::select(.pred,medv)

ggplot(pred, aes(x = medv, y = .pred)) +
  geom_abline(lty = 2) +
  geom_point(alpha = 0.5) +
  labs(y = "Predicted medv", x = "Medv") +
  coord_obs_pred()
```



```
kable(rmse(pred,  truth = medv, estimate = .pred))
```

.metric	.estimator	.estimate
rmse	standard	4.409019

Com dados de dimensionalidade menor, foi possível de executar toda a rotina de otimização de hiperparâmetros, que retornou o melhor modelo com penalty = 1 e mixture = 1 (Lasso). Vemos que as previsões parecem relativamente aderentes ao verdadeiro, e o erro quadrático médio é aceitável.

Pelo gráfico, vemos que a variável aparenta mais ter uma relação logarítmica do que linear.