Notes Otto

* Tried **PLSDA** con performance **1.70806**
* Tried to adjust PLSDA submission so that **all class probs < maxProb are recalibrated so that the sum is 1 - maxProb**: performed **1.08942on leaderboard** (mySub\_PLSDA\_2.csv) – **moved up 103 positions** on the leaderboard
* Tried to identify cases with same feature string: does it happen? Perhaps, you can consider neighbors voting with closer feature string …

XGBOOST + feature scaling : 0.46 (tried many combinations with PCA/ICA/Spatial sign – all worst )

SVM (imbalanced class managed / no feature scaling / C and gamma estimated analytically): 0.8

SVM (imbalanced class not managed / no feature scaling / C and gamma estimated analytically): 6.5

SVM (imbalanced class managed / feature scaling / gamma estimated analytically / C estimated by cross validation): 0.76

NN (imbalanced class managed / feature scaling / p, h, lambda estimated by cross validation): xx

NN (imbalanced class managed / feature scaling / p, h, lambda estimated by cross validation / only 40 most important features according to xgboost): xx

# XGBoost

Feature scaling

"objective" = "multi:softprob"

"eval\_metric" = "mlogloss"

"num\_class" = 9

"nthread" = 8

**cross validation (**nrounds = 175 **/** nfold = 3)

* train.mlogloss.mean = 0.198990
* train.mlogloss.std = 0.001306
* test.mlogloss.mean = 0.491164
* test.mlogloss.std = 0.008580

**train** (nrounds = 175)

* [174] train-mlogloss:0.233554

the submission scored 0.469 (0.491164 - 0.008580 = 0.482584)

**early stop**

> which(bst.cv$test.mlogloss.mean == min(bst.cv$test.mlogloss.mea) )

[1] 163

>

> early.stop = which(bst.cv$test.mlogloss.mean == min(bst.cv$test.mlogloss.mea) )

> bst.cv[early.stop,]$test.mlogloss.mean

[1] 0.490686

> bst.cv[early.stop,]$test.mlogloss.std

[1] 0.008311

## variants

* booster [default=gbtree] = gblinear
* eta [default=0.3] -- 0.05
  + 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 shrinkage the feature weights to make the boosting process more conservative
* gamma [default=0] -- 0.5
  + 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
* max\_depth [default=6] -- 10
  + maximum depth of a tree
* min\_child\_weight [default=1] -- 0.1
  + minimum sum of instance weight(hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min\_child\_weight, then the building process will give up further partitioning. In linear regression mode, this simply corresponds to minimum number of instances needed to be in each node. The larger, the more conservative the algorithm will be
* 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
* subsample [default=1] -- 0.5
  + subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees and this will prevent overfitting.
* colsample\_bytree [default=1] –
  + subsample ratio of columns when constructing each tree.

## Variant #1

* eta [default=0.3] 🡨 0.05
* so, early stop passed from 163 to 998 with test.mlogloss.mean ~ 0.48
* submitting got 0.461 (improving my score of 0.008)

## Variant # 2

* eta [default=0.3] 🡨 0.05
* gamma [default=0] 🡨 0.5
* max\_depth [default=6] 🡨 10
* subsample [default=1] 🡨 0.5

>> early.stop: 462 [test.mlogloss.mean: 0.467299 ]

submitting got 0.44

## Variant # 3

* eta [default=0.3] 🡨 0.05
* gamma [default=0] 🡨 0.5
* max\_depth [default=6] 🡨 15
* subsample [default=1] 🡨 0.5
* min\_child\_weight [default=1] 🡨 1 (tried 0.1 but with bad results)
* colsample\_bytree [default=1] 🡨 0.5
* max\_delta\_step [default=0] 🡨 1

early.stop = 471 - train-mlogloss:0.116288+0.000647 test-mlogloss:0.463285+0.005166

submitting got 0.43

## Variant # 4

* eta [default=0.3] 🡨 0.05
* gamma [default=0] 🡨 0.5
* max\_depth [default=6] 🡨 15
* subsample [default=1] 🡨 0.5
* min\_child\_weight [default=1] 🡨 1 (tried 0.1 but with bad results)
* colsample\_bytree [default=1] 🡨 0.5
* max\_delta\_step [default=0] 🡨 0

hai avuto un crash con sul Mac e il document e’ andato perso.

In sostanza, avevi fatto delle prove in base a cui

* min\_child\_weight [default=1] 🡨 1
* colsample\_bytree [default=1] 🡨 0.5
* max\_delta\_step [default=0] 🡨 5

Ora stavi vedendo che effetto fa max\_depth = 25 …

[392] train-mlogloss:0.094077+0.000576 test-mlogloss:0.467277+0.006738

riprovo settando max\_delta\_step = 1

[420] train-mlogloss:0.092276+0.000830 test-mlogloss:0.462888+0.003991

## solo le prime 40 features importanti

riprovo settando sulle prime 40 variabili piu’ importanti settando nfold = 5 (anzicche’ 3) ..

[380] train-mlogloss:0.128543+0.000375 test-mlogloss:0.523652+0.008701

*per la serie, alcune features non sono molto importanti, ma togliele e poi vedi come si incazza!!*

## max\_delta\_step = o nella best configuration

riprovo settando max\_delta\_step = 0 ..

[406] train-mlogloss:0.092734+0.000444 test-mlogloss:0.454927+0.002632

scored 0.43836 on leaderboard

*fenomeno interessante … ha performato peggio che settando max\_delta\_step a 1 ma con performance sul test set di 0.463285*

*… forse conviene settare max\_delta\_step a 1 …*

> head(importance)

Feature Gain Cover Frequence

1: feat\_60 0.03998089 0.02144361 0.01127387

2: feat\_11 0.03593321 0.01924782 0.01003048

3: feat\_25 0.03485804 0.02743988 0.03433026

4: feat\_15 0.03307437 0.02711514 0.02378688

5: feat\_34 0.03298646 0.02177694 0.01421420

6: feat\_14 0.03138750 0.02898105 0.03127124

*Notice that the order (and Gain) of the most important features is not equal to below*

## encoding di feat2 come variabile categorica

riprovo facendo l’encoding delle features categoriche (per ora solo feat2) ..

>> early.stop: 419 [test.mlogloss.mean: 0.454406 ]

performed as 0.43797

Notare che ancora una volta i predittori importanti sono ordinate diversamente e hanno diverso Gain

> head(importance)

Feature Gain Cover Frequence

1: feat\_34 0.04620828 0.02216267 0.014305881

2: feat\_11 0.04465141 0.01929734 0.009551308

3: feat\_60 0.03868275 0.02192880 0.011909554

4: feat\_14 0.03659382 0.03011047 0.033766177

5: feat\_40 0.03071641 0.03149970 0.030113207

6: feat\_15 0.02882586 0.02678399 0.023514468

## encoding di feat2 e 73 (le uniche 2 classificate come sicuramente categoriche) come features categoriche

>>Params:

$objective

[1] "multi:softprob"

$eval\_metric

[1] "mlogloss"

$num\_class

[1] 9

$eta

[1] 0.05

$gamma

[1] 0.5

$max\_depth

[1] 25

$subsample

[1] 0.5

$nthread

[1] 10

$min\_child\_weight

[1] 1

$colsample\_bytree

[1] 0.5

$max\_delta\_step

[1] 1

test.perf ~ 0.454

*… per la serie fare l’encoding di possibili variabili categiriche sembra spostare niente ..*

## Feature Importance xgboost



param <- list("objective" = "multi:softprob",

"eval\_metric" = "mlogloss",

"num\_class" = 9,

"eta" = 0.05, ## suggested in ESLII

"gamma" = 0.5,

"max\_depth" = 10,

"subsample" = 0.5 , ## suggested in ESLII

"nthread" = 8)

nround = 400

> head(importance,40)

Feature Gain Cover Frequence

1: feat\_60 0.074076321 0.023691403 0.016112250

2: feat\_11 0.068568018 0.020075976 0.010451034

3: feat\_34 0.058982370 0.024869059 0.015507431

4: feat\_15 0.039898385 0.031592516 0.023590787

5: feat\_90 0.037285351 0.009622065 0.006277500

6: feat\_14 0.036123053 0.032046207 0.027979304

7: feat\_40 0.031226952 0.037841410 0.030220859

8: feat\_86 0.029974054 0.029554622 0.029059952

9: feat\_67 0.026136324 0.027394372 0.034093417

10: feat\_42 0.025574460 0.021047474 0.017995500

11: feat\_36 0.024828460 0.015269956 0.013821966

12: feat\_25 0.021812540 0.027651953 0.030014476

13: feat\_39 0.020931005 0.013439497 0.010508363

14: feat\_62 0.020775992 0.016800847 0.021710404

15: feat\_75 0.019130816 0.017967921 0.013902226

16: feat\_24 0.018410323 0.016963791 0.032476746

17: feat\_68 0.018309937 0.016208009 0.010029668

18: feat\_9 0.016691167 0.024259627 0.019093346

19: feat\_43 0.015532814 0.021461088 0.019529044

20: feat\_48 0.015128709 0.021321675 0.027987904

21: feat\_64 0.014857017 0.021905081 0.025385178

22: feat\_26 0.014043636 0.015202321 0.010256116

23: feat\_88 0.013926588 0.021934636 0.024562510

24: feat\_69 0.013318248 0.009301331 0.006974044

25: feat\_53 0.013219447 0.012743469 0.011766729

26: feat\_59 0.011298740 0.012890193 0.009894945

27: feat\_16 0.011006482 0.010151477 0.021426626

28: feat\_47 0.010706836 0.011380669 0.006994109

29: feat\_72 0.010619623 0.017055599 0.017155633

30: feat\_17 0.010096351 0.015497853 0.009854815

31: feat\_8 0.009473515 0.010800775 0.014329325

32: feat\_32 0.009356325 0.013168849 0.018006965

33: feat\_85 0.008903082 0.013710329 0.016608144

34: feat\_50 0.008565093 0.009986303 0.009304459

35: feat\_30 0.008482331 0.007742610 0.004202199

36: feat\_78 0.008435530 0.009193416 0.006647270

37: feat\_54 0.008346802 0.007080457 0.017743253

38: feat\_56 0.008274922 0.018188159 0.011288034

39: feat\_70 0.007786013 0.009127248 0.018442664

40: feat\_35 0.007645322 0.007291299 0.008003096

Feature Gain Cover Frequence

# Neural Nets

## Neural Nets with top 40 most important predictors according to xgboost



Figure Train curve and cross validation curve for Class9 (p, h, lambda are indexed by i)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **p** | **h** | **lambda** | **accuracy\_xval** |
| 1 | 60 | 1 | 1 | 89.11917098 |
| 2 | 60 | 1 | 0.3 | 87.47092572 |
| 3 | 40 | 1 | 0.003 | 85.19675203 |
| 4 | 60 | 2 | 0 | 87.37233055 |
| 5 | 40 | 1 | 1 | 98.54014599 |
| 6 | 60 | 2 | 1 | 95.70215776 |
| 7 | 40 | 3 | 0.03 | 89.70070423 |
| 8 | 60 | 1 | 1 | 95.18606025 |
| 9 | 60 | 1 | 1 | 95.66094854 |

On the leaderboard 3.35915 – forse non normalizzando … ma per ora non spreco una submission per togliermi il dubbio …

## Neural Nets full

Given the bad score of reduced NN, perhaps it’s better not to normalize probabilities … The submission on SVM was actually not normalized and scored 0.76 on leaderboard. Probably, if it were normalized it would have scored worst.



Figure Train and cross validation curve per Class9

Performed as 0.94695

# Eta 0.01 (10^-1 ~ 2173)

> source('Predict\_3\_XGBOOST\_BOOST.R')

**Loading required package: xgboost**

>>Params:

$objective

[1] "multi:softprob"

$eval\_metric

[1] "mlogloss"

$num\_class

[1] 9

$eta

[1] 0.01

$gamma

[1] 0.5

$max\_depth

[1] 25

$subsample

[1] 0.5

$nthread

[1] 10

$min\_child\_weight

[1] 1

$colsample\_bytree

[1] 0.5

$max\_delta\_step

[1] 1

>> early.stop: 2173 [test.mlogloss.mean: 0.450558 ]

**leaderboard 0.43265**

so,

* 0.437 – 0.432 = 0.005 (leaderboard)
* 0.454 – 0.450 = 0.005 (cross-validation)

per vincere devi quindi arrivare a 0.40 in cross-validation

# Eta 0.005 (10^-3 ~< 4500)

[3999] train-mlogloss:0.095585+0.000595 test-mlogloss:0.448368+0.011393

train.mlogloss.mean train.mlogloss.std test.mlogloss.mean

1: 2.191420 0.000098 2.192351

2: 2.187135 0.000050 2.187438

3: 2.181811 0.000089 2.182559

4: 2.176216 0.000090 2.177697

5: 2.170728 0.000077 2.172698

---

3996: 0.095637 0.000596 0.448370

3997: 0.095623 0.000598 0.448373

3998: 0.095612 0.000597 0.448369

3999: 0.095598 0.000596 0.448369

4000: 0.095585 0.000595 0.448368

test.mlogloss.std

1: 0.000089

2: 0.000094

3: 0.000182

4: 0.000254

5: 0.000248

---

3996: 0.011387

3997: 0.011388

3998: 0.011389

3999: 0.011390

4000: 0.011393

>> early.stop: 4000 [test.mlogloss.mean: 0.448368 ]

>> redo-cv [early.stop == cv.nround= 4000 ] with 2 \* cv.nround ...

>> cv.nround: 8000

interotto e fatta submission impostando nround = 4500 …. got 0.43197

a 5000 got **0.43176**

# Eta 0. 00001 (?10^-6 ~< 11000?)

Run 12.00 15 May

# Eta 0. 000001 (?10^-5 ~< 9000?)

Run 12.00 15 May

# Eta 0. 0001 (?10^-4 ~< 7000?)