Solution for Otto Group Product Classification Challenge
https://www.kaggle.com/c/otto-group-product-classification-challenge
3rd place, Dmitry Efimov and Davut Polat

1 Approach summary

The purpose of this challenge was to build a predictive model which is able to distinguish between main product categories of the Otto Group companies. The provided dataset contained 93 features and more than 200 000 products. Our solution contains the following steps:

1. Build models for stacking.
2. 1st level ensemble.
3. 2nd level ensemble.

2 Models for stacking

The following list of algorithms has been chosen to calculate models for stacking.

1. **xgbx**:

   model: Gradient Boosting Trees, xgboost package in Python [8];

   comments: extended dataset;

   objective: "binary:logistic";

   eta: 0.1;

   max_depth: 15;

   eval_metric: "logloss";

   subsample: 0.9;

   colsample_bytree: 0.7;

   num_round: 300.

2. **xgbz**:

   model: Gradient Boosting Trees, xgboost package in Python [8];

   comments: the several groups of features have been added to the original dataset; each group has been obtained by the transformations: $\text{floor}(\log_a(X+1))$, $\text{floor}(\sqrt{X+1})$, $\text{floor}((X+1)^2)$, where $a =$

$2, 3, 4, 5, 6, 7, 8, 9, 12, 13$, X is an original dataset; each column scaled to the mean 0 and the standard deviation 1;

eta: 0.05;

max_depth: 50, 50, 30, 50, 50, 50, 40, 14, 14, 19;

eval_metric: "mlogloss";

min_child_weight: 5, 5, 1, 3, 5, 8, 5, 5, 5, 5;

colsample_bylevel: 0.012, 0.024, 0.012, 0.012, 0.018, 0.012, 0.012, 0.024, 0.036, 0.012;

colsample_bytree: 1;

num_round: 500, 400, 550, 400, 400, 650, 500, 750, 650, 750;

epoch: 10.

3. **linear1**:

   model: Gradient Boosting Logistic Regression, xgboost package in Python [8];

   comments: each row in the dataset is divided by its mean and each column scaled to mean 0 and standard deviation 1;

   objective: "binary:logistic";

   eta: 0.1;

   max_depth: 12;

   eval_metric: "logloss";

   min_child_weight: 1;

   colsample_bylevel: 0.4;

   scale_pos_weight: 1;

   num_round: 200;

   epoch: 1.

4. **linear2**:

   model: Logistic Regression, scikit-learn package in Python [1];

   comments: each row in the dataset is divided by its mean and each column scaled to mean 0 and standard deviation 1;

C: 0.01;

penalty: "L1".

5. **linear3**:

   model: Logistic Regression, scikit-learn package in Python [1];

   comments: each row in the dataset is divided by its mean and each column scaled to mean 0 and standard deviation 1;

   C: 0.01;

   penalty: "L2".

6. **nne**:

   model: Neural Net, lasagne package in Python [4];

   comments: dataset is transformed with the function $z = \log_{10}(1 + x)$ and scaled to mean 0 and standard deviation 1;

   layer1: 500 units;

   dropout layer2: p = 0.5;

   layer3: 300 units;

   dropout layer4: p = 0.1;

   layer5: 100 units;

   dropout layer6: p = 0.1;

   output: "softmax";

   update: "nesterov_momentum";

   update_learning_rate: 0.01;

   update_momentum: 0.9;

   max_epochs: 70;

   epoch: 50.

7. **nnb**:

   model: Neural Net, lasagne package in Python [4];

   comments: all values greater than 10 replaced by 10 and dataset is scaled to mean 0 and standard deviation 1;

layer1: 500 units;

dropout layer2: p = 0.5;

layer3: 300 units;

dropout layer4: p = 0.1;

layer5: 100 units;

dropout layer6: p = 0.1;

output: "softmax";

update: "nesterov_momentum";

update_learning_rate: 0.01;

update_momentum: 0.9;

max_epochs: 70;

epoch: 30.

8. **nnc**:

   model: Neural Net, lasagne package in Python [4];

   comments: each row is divided by its maximum value and dataset is scaled to mean 0 and standard deviation 1;

   layer1: 500 units;

   dropout layer2: p = 0.5;

   layer3: 300 units;

   dropout layer4: p = 0.1;

   layer5: 100 units;

   dropout layer6: p = 0.1;

   output: "softmax";

   update: "nesterov_momentum";

   update_learning_rate: 0.01;

   update_momentum: 0.9;

   max_epochs: 70;

   epoch: 30.

9. **nnd**:

   model: Neural Net, lasagne package in Python [4];

   comments: TF-IDF transformation is applied and dataset is scaled to mean 0 and standard deviation 1;

   layer1: 500 units;

   dropout layer2: p = 0.5;

   layer3: 300 units;

   dropout layer4: p = 0.1;

   layer5: 100 units;

   dropout layer6: p = 0.1;

   output: "softmax";

   update: "nesterov_momentum";

   update_learning_rate: 0.01;

   update_momentum: 0.9;

   max_epochs: 70;

   epoch: 30.

10. **extratrees**:

    model: Randomized Decision Trees, scikit-learn package in Python [1], [2];

    comments: TF-IDF transformation is applied to the dataset;

    n_estimators: 300;

    criterion: "entropy";

    max_features: 30;

    max_depth: 25;

    epoch: 5.

11. **svc**:

    model: Support Vector Classifier, scikit-learn package in Python [1], [7];

comments: values greater than 10 replaced by 10 and dataset is scaled to mean 0 and standard deviation 1;

C: 10;

gamma: 0.01;

probability: True;

epoch: 1.

12. **knn2**:

    model: k-Nearest Neighbours, scikit-learn package in Python [1];

    comments: values greater than 10 replaced by 10 and dataset is scaled to mean 0 and standard deviation 1;

    n_neighbors: 5, 20, 40, 60, 80, 100, 5, 20, 50, 100, 70;

    weights: "distance";

    p: 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1;

    epoch: 11.

13. **knn4**:

    model: k-Nearest Neighbours, scikit-learn package in Python [1];

    comments: TF-IDF transformation is applied and dataset is scaled to mean 0 and standard deviation 1;

    n_neighbors: 5, 20, 40, 60, 80, 100, 5, 20, 50, 100, 70;

    weights: "distance";

    p: 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1;

    epoch: 11.

14. **knn5**:

    model: k-Nearest Neighbours, scikit-learn package in Python [1];

    comments: each row in the dataset is divided by its mean; the group of features obtained by transformation $\text{floor}(\log_{10}(X+1))$, where X is an original dataset, has been added to the dataset; each column scaled to mean 0 and standard deviation 1;

    n_neighbors: 5, 20, 40, 60, 80, 100, 5, 20, 50, 100, 70;

weights: “distance”;

p: 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1;

epoch: 11.

15. **knn6**:

    model: k-Nearest Neighbours, scikit-learn package in Python [1];

    comments: TF-IDF transformation and Singular Value Decomposition are applied; the dataset is recovered from the SVD with 60 eigenvectors;

    n_neighbors: 5, 20, 40, 60, 80, 100, 5, 20, 50, 100, 70;

    weights: “distance”;

    p: 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1;

    epoch: 11.

16. **knn8**:

    model: k-Nearest Neighbours, scikit-learn package in Python [1];

    comments: the transformation $2\sqrt{X + \frac{3}{8}}$, where X is an original dataset has been applied;

    n_neighbors: 5, 20, 40, 60, 80, 100, 5, 20, 50, 100, 70;

    weights: “distance”;

    p: 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1;

    epoch: 11.

17. **rgf**:

    model: Regularized Greedy Forest [3];

    comments: original dataset;

    reg_L2: 0.01;

    algorithm: “RGF”;

    loss: “Log”;

    num_iteration_opt: 7;

    num_tree_search: 5;

min_pop: 8;

max_leaf_forest: 10000.

18. **h2o**:

    model: Neural Net, h2o package in R [6];

    comments: the transformation $\sqrt{X + \frac{3}{8}}$, where X is an original dataset has been applied;

    classification: True;

    activation: "RectifierWithDropout";

    hidden: 1024, 512, 256;

    hidden_dropout: 0.5, 0.5, 0.5;

    input_dropout_ratio: 0.5;

    epochs: 50;

    l1: 10^{-5};

    l2: 10^{-5};

    rho: 0.99;

    epsilon: 10^{-8};

    train_samples_per_iteration: 2000;

    max_w2: 10.

Model **xgbx** has been calculated on the extended dataset constructed in the following way:

1. The 9 copies X_i, $i = 1...9$, of the original dataset are generated.

2. The target variable for copy X_i, $i = 1...9$, is a binary target variable, corresponding to the original target variable and class i. For example, for copy X_4, target variable equals 1 if the original target variable equals "Class_4", 0 - otherwise.

3. For each dataset copy X_i, $i = 1...9$, binary features $PredictedClass_i$, $i = 1...9$, have been added. For the copy X_i, $PredictedClass_j = 1$, if $j = i$, 0 - otherwise. For example, for copy X_4, $PredictedClass_4 = 1$, and $PredictedClass_j = 0$, for all $j \neq 4$.

4. All 9 copies are combined to one dataset.

3 1st level ensemble

We used 10 folds for the train dataset to get predictions for stacking models. We have calculated means, maxima and medians by the similar models (for example, for the model **nne** the mean, max and median of 50 outputs for each class have been calculated). The first level ensemble contains 3 different models for the second level ensemble:

1. **xgb.ens**

   model: Gradient Boosting Trees;

   comments: all predictions are transformed with the $\log\left(\frac{Z}{1-Z}\right)$, where Z is a set of predictions obtained in the previous section;

   objective: "multi:softprob";

   eta: 0.125;

   max_depth: 4;

   eval_metric: "mlogloss";

   subsample: 0.7;

   colsample_bylevel: 0.12;

   colsample_bytree: 1.0;

   min_child_weight: 1, 5, 10, 15, 20, 25, 30, 35, 40;

   num_round: 110;

   epoch: 9.

2. **xgbx.ens**

   model: Gradient Boosting Trees;

   comments: extended dataset; all predictions are transformed with the $\log\left(\frac{Z}{1-Z}\right)$, where Z is a set of predictions obtained in the previous section;

   objective: "binary:logistic";

eta: 0.03;

max_depth: 6;

eval_metric: "logloss";

subsample: 0.9;

colsample_bytree: 0.7;

min_child_weight: 5, 10, 15;

num_round: 400;

epoch: 3.

3. **nn.ens**

   model: Neural Net;

   comments: all predictions (except knn predictions) are transformed with the $\log\left(\frac{Z}{1-Z}\right)$, where Z is a set of predictions obtained in the previous section;

   layer1: 700 units;

   dropout layer2: p = 0.7;

   layer3: 100 units;

   dropout layer4: p = 0.1;

   output: "softmax";

   update: "nesterov_momentum";

   update_learning_rate: 0.007;

   update_momentum: 0.9;

   max_epochs: 50;

   epoch: 12.

Here the extended dataset has been generated the same way as in the previous section. The simple average of epochs has been calculated for each model.

4 2nd level ensemble

The final ensemble was a geometric linear combination:

$$\mathbf{xgb.ens}^{0.4} \cdot \mathbf{nn.ens}^{0.3} \cdot \mathbf{xgbx.ens}^{0.3}.$$

The private and public scores are summarized in the following table:

Model name	**Public score**	**Private score**
xgbx	0.43321	0.43419
xgbz	0.42465	0.42602
linear1	0.60832	0.60799
linear2	0.63106	0.63287
linear3	0.60937	0.61063
nne	0.43496	0.43554
nnb	0.43945	0.44261
nnc	0.45352	0.45555
nnd	0.44900	0.45102
extratrees	0.50535	0.50751
svc	0.54507	0.54550
knn2	0.57879	0.58532
knn4	0.63175	0.64093
knn5	0.64080	0.65739
knn6	0.58893	0.60820
knn8	0.55503	0.56841
rgf	0.45753	0.45818
h2o	0.44532	0.44731
xgb.ens	0.39186	0.39385
xgbx.ens	0.39428	0.39584
nn.ens	0.39480	0.39662
final ens	0.38946	0.39124

5 File list and training

Task	File name	Description
Build dataset	01-data.build.R	**Output:** data table **data.all** in the folder data\output-r
Models for stacking	02-models.build.R model.rgf.stack.R model.h2o.stack.R model.extratrees.stack.py model.knn2.stack.py model.knn4.stack.py model.knn5.stack.py model.knn6.stack.py model.knn8.stack.py model.linear.stack.py model.nnb.stack.py model.nnc.stack.py model.nnd.stack.py model.nne.stack.py model.svc.stack.py model.xgbx.stack.py model.xgbz.stack.py	**Output:** predictions in folders data\output-py\train, data\output-py\test data\output-r\train, data\output-r\test
1st level ensembles	03-models.ens.1level.R model.nn.ens.py model.xgb.ens.py model.xgbx.ens.py	**Output:** predictions in the folder data\output-py\ens_1level
2nd level ensemble	04-models.ens.2level.R	**Output:** predictions in the folder data\submission
Helper functions	utils.R utils.parallel.R	

To calculate the final ensembling:

1. Put all data files (extracted) into the "data\input" folder.
2. Open R session with folder "otto-r" set as working dir.
3. Run the R script files in the folder "otto-r" in the following order:

(a) 01-data.build.R

(b) 02-models.build.R

(c) 03-models.ens.1level.R

(d) 04-models.ens.2level.R

4. The predictions will be saved in the data\submission\model.ens.csv after running 04-models.ens.2level.R.

Pre-requisites:

All files consider the folder "otto-r" as the working dir. Using RStudio and loading the project inside will do it.

The model can be run on Linux Ubuntu 12.04 or Mac OS.

Dependencies:

Python: pandas 0.12.0, numpy 1.8.0, scikit-learn 0.16.1, sys, getup, xgboost, random, os, lasagne, nolearn, joblib, theano, statsmodels.

R: doSNOW, foreach, cvTools, data.table, parallel, rlecuyer, verification, Matrix, h2o.

Utitlities: rgf 1.2.

All the listed versions are the used ones. It will probably work with newer versions, but it was not tested. The R version used was 3.1.0, Python version used 2.7.3.

References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay *Scikit-learn: Machine Learning in Python.* 1991, Journal of Machine Learning Research, 12, pp. 2825-2830.

[2] J. Simm and I. Magrans de Abril. *Package for ExtraTrees method for classification and regression.* 2013.

[3] R. Johnson and T. Zhang. *Learning nonlinear functions using regularized greedy forest.* 2011, Technical report, Tech Report: arXiv:1109.0887.

[4] https://lasagne.readthedocs.org/en/latest/

[5] M. A. Nielsen. *Neural Networks and Deep Learning.* 2015, Determination Press.

[6] http://cran.r-project.org/web/packages/h2o/h2o.pdf

[7] I. Guyon, B. Boser and V. Vapnik. *Automatic Capacity Tuning of Very Large VC-dimension Classifiers.* 1993, Advances in neural information processing.

[8] https://github.com/dmlc/xgboost