Being successful on Kaggle using `mlr`

Achieving a good score on a Kaggle competition is typically quite difficult. This blog post outlines 7 tips for beginners to improve their ranking on the Kaggle leaderboards. For this purpose, I also created a *Kernel* for the *Kaggle bike sharing competition* that shows how the R package, `mlr`, can be used to tune a xgboost model with random search in parallel (using 16 cores). The R script scores rank 90 (of 3251) on the Kaggle leaderboard.

7 Rules

1. Use good software
2. Understand the objective
3. Create and select features
4. Tune your model
5. Validate your model
6. Ensemble different models
7. Track your progress

1. Use good software

Whether you choose R, Python or another language to work on Kaggle, you will most likely need to leverage quite a few packages to follow best practices in machine learning. To save time, you should use 'software' that offers a standardized and well-tested interface for the important steps in your workflow:

- Benchmarking different machine learning algorithms (learners)
- Optimizing hyperparameters of learners
- Feature selection, feature engineering and dealing with missing values
- Resampling methods for validation of learner performance
- Parallelizing the points above

Examples of 'software' that implement the steps above and more:

- For python: scikit-learn (http://scikit-learn.org/stable/auto_examples).
- For R: `mlr` (https://mlr-org.github.io/mlr-tutorial) or `caret`.

2. Understand the objective

To develop a good understanding of the Kaggle challenge, you should:

- Understand the problem domain:
    - Read the description and try to understand the aim of the competition.

    - Keep reading the forum and looking into scripts/kernels of others, learn from them!
    - Domain knowledge might help you (i.e., read publications about the topic, wikipedia is also ok).
    - Use external data if allowed (e.g., google trends, historical weather data).
- Explore the dataset:
    - Which features are numerical, categorical, ordinal or time dependent?
    - Decide how to handle *missing values*. Some options:
        - Impute missing values with the mean, median or with values that are out of range (for numerical features).
        - Interpolate missing values if the feature is time dependent.
        - Introduce a new category for the missing values or use the mode (for categorical features).
    - Do exploratory data analysis (for the lazy: wait until someone else uploads an EDA kernel).
    - Insights you learn here will inform the rest of your workflow (creating new features).

Make sure you choose an approach that directly optimizes the measure of interest! Example:

- The **median** minimizes the mean absolute error **(MAE)** and the **mean** minimizes the mean squared error **(MSE)**.
- By default, many regression algorithms predict the expected **mean** but there are counterparts that predict the expected **median** (e.g., linear regression vs. quantile regression).
- For strange measures: Use algorithms where you can implement your own objective function, see e.g.
    - *tuning parameters of a custom objective* or
    - *customize loss function, and evaluation metric*.

3. Create and select features:

In many kaggle competitions, finding a "magic feature" can dramatically increase your ranking. Sometimes, better data beats better algorithms! You should therefore try to introduce new features containing valuable information (which can't be found by the model) or remove noisy features (which can decrease model performance):

- Concat several columns
- Multiply/Add several numerical columns
- Count NAs per row
- Create dummy features from factor columns

- For time series, you could try
    - to add the weekday as new feature
    - to use rolling mean or median of any other numerical feature
    - to add features with a lag...
- Remove noisy features: *Feature selection / filtering*

4. Tune your model

Typically you can focus on a single model (e.g. *xgboost*) and tune its hyperparameters for optimal performance.

- Aim: Find the best hyperparameters that, for the given data set, optimize the pre-defined performance measure.
- Problem: Some models have many hyperparameters that can be tuned.
- Possible solutions:
    - *Grid search or random search*
    - Advanced procedures such as *irace* or *mbo (bayesian optimization)*

5. Validate your model

Good machine learning models not only work on the data they were trained on, but also on unseen (test) data that was not used for training the model. When you use training data to make any kind of decision (like feature or model selection, hyperparameter tuning, ...), the data becomes less valuable for generalization to unseen data. So if you just use the public leaderboard for testing, you might overfit to the public leaderboard and lose many ranks once the private leaderboard is revealed. A better approach is to use validation to get an estimate of performane on unseen data:

- First figure out how the Kaggle data was split into train and test data. Your resampling strategy should follow the same method if possible. So if kaggle uses, e.g. a feature for splitting the data, you should not use random samples for creating cross-validation folds.
- Set up a *resampling procedure*, e.g., cross-validation (CV) to measure your model performance
- Improvements on your local CV score should also lead to improvements on the leaderboard.
- If this is not the case, you can try
    - several CV folds (e.g., 3-fold, 5-fold, 8-fold)
    - repeated CV (e.g., 3 times 3-fold, 3 times 5-fold)
    - stratified CV
- `mlr` offers nice *visualizations to benchmark* different algorithms.

6. Ensemble different models (see, e.g. *this guide*):

After training many different models, you might want to ensemble them into one strong model using one of these methods:

- simple averaging or voting
- finding optimal weights for averaging or voting
- stacking

7. Track your progress

A kaggle project might get quite messy very quickly, because you might try and prototype many different ideas. To avoid getting lost, make sure to keep track of:

- What preprocessing steps were used to create the data
- What model was used for each step
- What values were predicted in the test file
- What local score did the model achieve
- What public score did the model achieve

If you do not want to use a tool like git, at least make sure you create subfolders for each prototype. This way you can later analyse which models you might want to ensemble or use for your final commits for the competition.

Written on March 9, 2017 by Giuseppe Casalicchio