This assignment makes you more familiar with boosting in general and AdaBoost and Gradient Tree Boosting in particular. You will apply the concepts from the lecture, implement the algorithms and experiment with them to see them in action. The assignment consists of two IPython notebooks and some questions to be answered with pen and paper.

1. **Coding Exercise**
   In this exercise, you implement AdaBoost and Gradient Boosting. If done correctly, your implementation yields very similar results as the implementation in scikit-learn (or even exactly the same, depending on the numpy version used). With that implementation, you can then explore the impact of some of the important hyperparameters.

   (a) Work through the 'adaboost_exercise.ipynb' notebook found on ILIAS.

   (b) Work through the 'gbm_exercise.ipynb' notebook found on ILIAS.

   (c) Apply these algorithms to the Kaggle competition we're running for the course.

2. **Boosting vs Bagging**
   This question aims to give you a better understanding of the algorithms you just implemented.

   (a) You implemented Gradient Boosting with shrinkage. Explain how this parameter influences the performance and how it interacts with the number of iterations.
   Solution: *Shrinkage slows down learning and thus reduces overfitting and achieves a better test error. But, as the contribution of each estimator is decreased, it requires more iterations to achieve the same training error as without shrinkage. Furthermore, too much shrinkage can prevent convergence.*

   (b) Explain *in words* the difference between bagging and boosting (see also the slides).
   Solution: *Bagging fits models independently of each other, while Boosting iteratively trains models. In Bagging all models are trained on bootstrapped data from the same dataset, while in Boosting the training data is altered such that each model corrects the errors its predecessor made.*

   (c) Name one advantage of bagging over boosting and vice versa.
   Solution: *In contrast to boosting, bagging can easily be parallelized as it fits models independently. In contrast to bagging, boosting pushes the training error to zero such that underfitting is not a problem.*