

This project explores ensembles of decision trees, i.e., **decision forests**, for classification (and possibly regression). It consists of programming a particular type of forest, applying it to several datasets, exploring the forest's performance comprehensively and reporting your results (in a written report and a presentation). The more extensive your project, the higher the grade. You can use any programming language (Matlab, Python, C, etc.), but we suggest using Matlab to produce plots.

The project will help you become familiar with a specific but widely applied and very successful machine learning model (tree ensembles), and to face the many decisions and issues that arise when working with real data.

I Components of a decision forest

Consider K -class classification. A classification forest requires the following:

Diversity mechanism (ensembling algorithm) This is what determines how each tree is trained (what data subset or feature subset, for example), and it is the part you have to program. Choose one of the following:

1. Bagging and bagging + random feature subset (= random forests) [2, ch. 15] [3, ch. 8].
2. AdaBoost.M1 [2, ch. 10].
3. Gradient boosting [2, ch. 10].
4. Random rotations [5, ch. 6].
5. Other: suggest something else to us and we will consider it.

Make sure you understand in detail the algorithm you implement (see the references). If you have questions or want to implement a modified or simplified version, ask us (TA and/or instructor).

Tree learning algorithm Each tree in the forest should be axis-aligned (univariate) with each leaf outputting a constant label. Each tree must be trained using some algorithm; for example, random forests typically use CART [1]. You don't need to implement the algorithm to train individual trees, instead use an existing implementation. The following are convenient:

1. `rpart` [8]: this is an R implementation (with parts written in C for efficiency) of most of the functionality of CART described in [1]. `rpart` is widely used in practice.
2. `scikit-learn` [6]: this is a Python implementation of part of the functionality of CART.
3. `C50` [4]: this is an R implementation of the C5.0 algorithm, which itself is an improved version of C4.5 [7]. It is a port of the original C code of <http://rulequest.com>.
4. Matlab has functions `fitctree` and `fitrtree` for classification and regression, respectively.

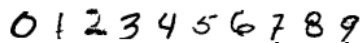
Unfortunately, different implementations use different computer languages. This is a reflection of the current state of machine learning. You will have to learn on the fly how to use them, as needed. [Engineering service learning at UC Merced](#) usually offers short courses on R, Python and Matlab.

Combination rule Just use majority voting (classification) or averaging (regression), possibly weighted (e.g. for boosting-based forests).

Hyperparameters (user parameters) All forests have a fundamental hyperparameter, the number of trees $T \geq 1$. Each ensembling algorithm may have its own hyperparameters (e.g. random forests need the number of features to sample for each tree). Also, each tree learning algorithm may have hyperparameters (e.g. how big to grow or prune the tree, what split criterion to use). Explore different settings in your experiments.

II Datasets

You have to apply your forest to the following, all classification problems:

1. The MNIST dataset of handwritten digits . Use our Matlab file [MNIST.mat](#), which uses as features the $28 \times 28 = 784$ grayscale pixel values in $[0, 255]$. It is partitioned into training (60k images) and test (10k images). Use the test set only to report test errors. Use the training set for training and (if needed) validation, in whichever way you want (e.g. a random split of 55k for training and 5k for validation).
2. The MNIST dataset but where each image is represented as an 800-dimensional feature vector, obtained from a pretrained LeNet5 convolutional neural net (the output of the neurons at layer conv2). Use our Matlab file [MNIST-LeNet5.mat](#). The images are the same as for the pixel-based images and appear in the same order.
3. Two other datasets of your own choice. Let us know in advance so we can check they are appropriate; they should be neither too simple nor too difficult (in terms of the number of instances, features and classes), and also somewhat diverse in terms of the subject matter (e.g. don't pick any other handwritten digit dataset). The course web page links to several web pages containing collections of datasets, but you don't have to be restricted to those.

III Experiments and evaluation

Having implemented the algorithm correctly, apply it to the datasets above. You will have to decide whether you want to apply some transformation to the data (e.g. making them zero-mean unit-variance), whether you want to apply cross-validation, how to select hyperparameters, etc. Evaluate the algorithm in a range of values of important hyperparameters, e.g. for the number of trees, you may want to vary it between 1 and 100 (say). Ask us if you have questions.

Your primary goal is to achieve a forest with the highest classification accuracy in the test set. This will likely require many, big trees, so in addition to accuracy, report:

- The number of parameters in the forest. For an axis-aligned tree, each decision node contains 2 parameters (the feature index and the real-valued threshold) and each leaf 1 parameter (the label).
- The inference time of applying the forest to an input instance. This should be given as the average over the training set, since the root-leaf paths followed by different instances may differ in length and so some instances will take longer than others.
- The training time. This depends on the algorithm and on its implementation (e.g. C can be many times faster than Matlab, depending on the computations).

The more extensive and insightful your experimental exploration, the more you will learn about the algorithm and the higher the grade in the project. Try to make your algorithm achieve the highest test accuracy possible by whatever means you can, and report exactly what you did and why (including negative results or things that didn't work). The winning group in MNIST will receive some extra grade.

IV Optional extensions

To get a good grade in the project, you have to do the minimum we require (as described above)¹. But you can go beyond that in any direction: trying more diversity mechanisms or tree learning algorithms (or modifying them), trying more datasets, manipulating the datasets (e.g. via image deskewing, data augmentation...), learning regression instead of classification trees, etc. We will value your effort and creativity in the project grade.

¹In order to have variability in the project presentations, the ensembling algorithm and pair of datasets should be unique to each group of students. Since every group will run their algorithm on MNIST (pixels and neural features), we can compare the performance (accuracy, number of parameters, inference time, training time, etc.) of each group's algorithm there.

V What you have to submit

Follow these instructions strictly. Email the TA the following packed into a **single** file (`project.tar.gz` or `project.zip`) and with email subject [CSE176] project:

- Your code to train a forest for each dataset with brief instructions of how to run it (include any external code needed, e.g. `rpart`). We will run it so make sure it has no errors. Don't include any data files, give a link to them as a comment inside the program file. For each dataset, create a directory with names like this: `MNIST`, `MNIST-LeNet5`, `dataset1`, `dataset2`. Have the code produce any necessary plots or tables.
- A report (max 10 pages) in PDF format (`report.pdf`) containing the following sections, in this order:
 1. A brief description of what each group member contributed to the project and of the sources you consulted (books, papers, web pages, code, etc.).
 2. A concise description of the algorithm you used and how you implemented it (including any decisions you may have taken to modify or simplify it).
 3. One section per dataset (named like the directories above) describing the dataset and your experimental results, carefully describing how you did things: selection of training, validation and test sets; choice of hyperparameters; etc.
 4. A conclusion section where you comment on the lessons learned about the algorithm based on your implementation and experiments.
- A presentation in PDF format (`presentation.pdf`) briefly describing the algorithm, experiments and results (as plots and/or tables). Each group will have about 10 minutes to present their work (plus about 5 minutes of questions from the audience), so don't make more than about 10 slides.

References

- [1] L. J. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, Calif., 1984.
- [2] T. J. Hastie, R. J. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning—Data Mining, Inference and Prediction*. Springer Series in Statistics. Springer-Verlag, second edition, 2009.
- [3] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Texts in Statistics. Springer-Verlag, 2013.
- [4] M. Kuhn, S. Weston, M. Culp, N. Coulter, and R. Quinlan. C50: C5.0 decision trees and rule-based models. R package version 0.1.2, May 22 2018. Available online at <https://cran.r-project.org/package=C50>.
- [5] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. John Wiley & Sons, second edition, 2014.
- [6] 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 É. Duchesnay. Scikit-learn: Machine learning in Python. *J. Machine Learning Research*, 12:2825–2830, Oct. 2011. Available online at <https://scikit-learn.org>.
- [7] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [8] T. Therneau, B. Atkinson, and B. Ripley. `rpart`: Recursive partitioning and regression trees. R package version 4.1-15, Apr. 12 2019. Available online at <https://cran.r-project.org/package=rpart>.