It took me 3 full working days to reach the #3 rank in the leader board (as of 11/04/2017). I decided to stop mostly because I had to take care of other tasks, even though I had many ideas to go further. I will attempt to describe shortly the logical process that led me to a leading estimator. The reasoning is simple (I tried practically the entire sklearn package), though the final estimator is rather complex.

1. **Understanding the dataset**

The first step was to understand what we were dealing with. Obviously, the data really starts to get "big": the number of samples was starting to get heavy for the average desktop computer. For this reason, I decided to test first the quicker methods, then I thought of reducing the number of samples studied through ensemble methods studies or by simply cutting it short, and eventually I went on jogging while I tested the most time-consuming models. The number of features is still reasonable, and they happen to be all quite meaningful. 128 features for a picture… Some serious signal processing was done before. I also tried to run analysis with only the most significant features (after a PCA). Also, it is important to notice that it is a slightly modified binary classification problem, with balanced class (no need to consider weights or things like that).

1. **Linear models**

I tried them all. They were rather fast, and gave rather satisfactory results as the score was already below 2. The best models were the Logistic regression and SGDClassifier. I tried with the full dataset, as it was or standardized… Well standardization happened to be bad for these classifications. Ridge classification was not very efficient even with optimal regularization because, well, all variables are significant. When there was room for customization, I used a grid search, or a series of grid searches to find the best parameters. By the end of this step, my highest training score was 0.63.

1. **Dimensionality reduction**

Before I tried other methods, I wanted to test LDA (and I did not understand how it worked). For such a fast technique, the score was not too shabby. Then I tried to study the impact of the variables on the variance with a PCA… And it appears that in the optimal projection space, no more than 15 features (out of 128) can be removed without consequences… When keeping 60% of the most significant features, the SGD score was nearly doubled…

1. **SVM + PCA**

Then I realized I could have shuffled the data (which was organized in 2 separate halves) before running a classifying task, to improve performance. I did not come back though, as SVC gave very good results. I kept only 50% of the samples and 60% of the features, and yet obtained a score of 0.116 after optimization of the hyperparameters… and approximately 10 min. I had to split the search into several grids, and use detailed reports to understand their relative significance.

Because of the good results, I had the idea of introducing the 0 label. SVC have a "predict=True" mode, which provides probability that a sample belongs to a specified class. Unfortunately, it also triples the computational time. When the probability was too close to 0.5 (below a threshold - say 085), I attributed the label 0… It gave me the amazing training score of 0.083. Unfortunately, I faced overfitting because the test score was no smaller than 0.4… It would have been great to train the SVC on the entire dataset, as this classifier is supposed to be robust to overfitting. But as it would have taken hours, I decided to spend more time finding other classifiers, and eventually never came back to SVC.

As the notebook was growing rather long, I defined a set of functions to help me reduce the amount of code in each cell.

* Displaying any amount of time measured more nicely.
* Automatizing the 0-labeling process, as well as customizing it by adding a threshold parameter.
* Preprocessing of data by automatizing standardization, PCA, feature selection, sample selection, and shuffling all together.
* Saving the test predictions without overwriting previous predictions.

1. **Nearest neighbors**

Both very time consuming, and inconclusive, I quickly gave up. Data is not separable after all, I should have expected the outcome.

1. **Gaussian processes**

For some reason, I could not make it work. Never mind.

1. **Bayesian methods**

Rather unsatisfactory… Maybe because features were not that independent.

1. **Trees**

Because many techniques rely on trees, I spent quite some time on it. I tried simple trees (which overfit), then I tried a personalized cross validation of trees, using a pipe to evaluate the best dimensionality reduction, and the best hyperparameters for trees, so that they were not to be too big or to small (overfitting vs. no ML). The pipe allowed me to make a combined gridsearch. I am quite confident I found good parameters… and yet the test score was not satisfactory…

1. **Ensemble methods based on Trees**

I tried Extra Trees, Random Forests, and Gradient Boosting Classifier which happened to be the best classifier so far, once correctly tuned (training score of 0.088, but test score of 0.33).

1. **Neural Networks**

It was an enlightenment: by far the fastest and most accurate classifier. After optimization, I obtained a test score of 0.197. I am however uncertain how I could have improved the score by playing with hidden layers. Despite my research, I could not find any relevant guidance, and despite my trials, the default parameter performed best.

As I had reviewed all the classifiers, it was high time I tried ensemble methods with other classifiers than trees. I decided to start by bagging the incredibly good Multi-Layer Perceptron, and after a few tests, I was in the top 20 of the leader board.

Then I went crazy, and tried to average the average. I combined several bagging of MLP with slight variations in a voting classifier. And OMG, I was #3 in the leaderboard with my current score.

1. **Ideas for further improvement**

Then I had to stop, because I had spent quite some time on the challenge, and had to do some other stuff. But Could I continue, I would have tried to make several bags of SVC trained on the full set, and Neural Networks, to intertwin them in a voting classifier. It might have run for days, as SVC are not a scalable technique, but could have scored even better.

1. **EDIT (22/04/2017)**

After letting the challenge aside for a while, some echoes from my friends led me to try being simpler and play with the parameters of the MLPClassifier, and a simple bagging. It led me to a new higher score…

**References:**

I used a lot of documentation. First, I read almost all scikit-learn guidance and tutorials, and then I also used other sources… in particular:

* <http://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/> to understand that too many features may not be useful.
* <https://jmetzen.github.io/2015-01-29/ml_advice.html> for the choice of loss functions.
* <http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html#sphx-glr-auto-examples-svm-plot-rbf-parameters-py> for the choice of γ and C in a SVC.
* <http://scikit-learn.org/stable/modules/computational_performance.html#prediction-latency> to improve speed of classification, and choose more appropriately the techniques.
* <http://stackoverflow.com/questions/10119913/pca-first-or-normalization-first> To use jointly PCA and standardization.
* <http://scikit-learn.org/stable/modules/neighbors.html#choice-of-nearest-neighbors-algorithm> To better tune the k-neighbors algorithm.
* <https://www.quora.com/Would-using-too-many-neighbors-in-the-k-nearest-neighbor-pattern-classification-algorithm-lead-to-overfitting-since-it-would-smooth-out-the-data> to assess the impact of the number of neighbours for k-neighbours classifier. (So as to reduce overfitting).
* <http://scikit-learn.org/stable/modules/neighbors.html#effect-of-leaf-size> to understand the impact of the leaf\_size in the trees.
* <http://scikit-learn.org/stable/modules/ensemble.html#loss-functions> To assess the fact that ensemble methods are supposed to work better with weak learners.