NEWS ML report sketch

Artem Golovatiuk

January 2018

1 Gaussian fit

First of all, we fitted each image with a 2D-Gaussian to get some physical features describing the grain for further training of Machine Learning algorithms.

Here are some examples of the fits:

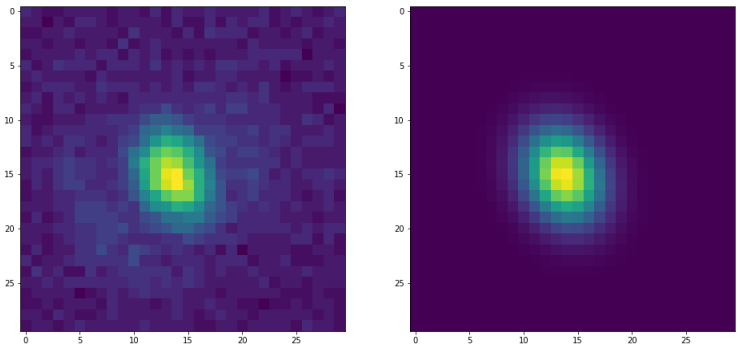


Figure 1: Random image of the signal event of C nuclei at 100 keV.

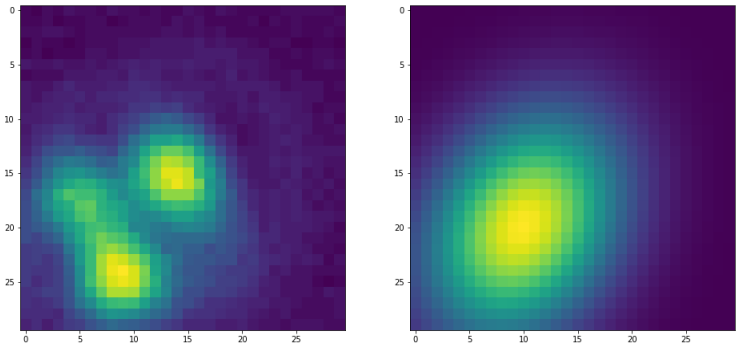


Figure 2: Random image of the gamma background event.

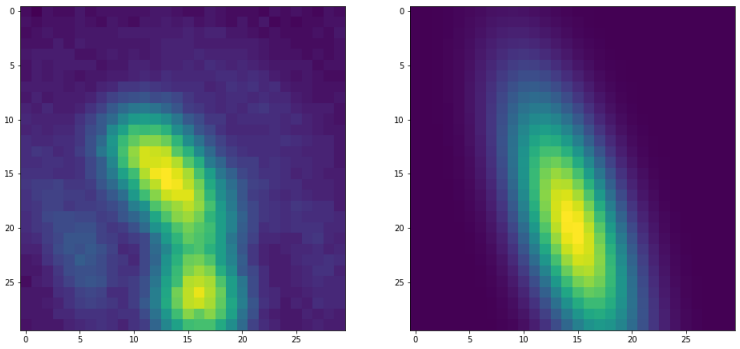


Figure 3: Random image of the dust-fog background event.

However, the fit diverged on lots of images and that is, most probably, not a problem of the method, but of the images themselves. If one looks at the divergent images, he will find that they are hard or even impossible to fit with 2D-Gaussian:

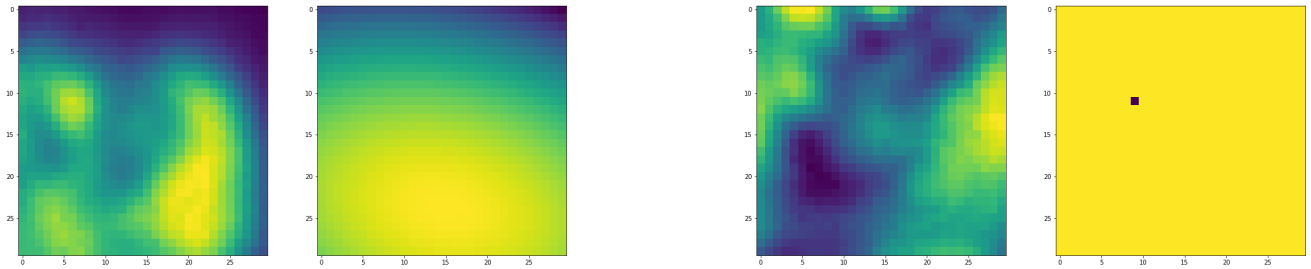


Figure 4: Examples of the fit divergence.

That's why we sorted out the most divergent images after the fit: the ones where parameters became extremely large or unphysical (like negative amplitude or negative diagonal elements of covariance matrix).

To show the rate of divergent images, here are numbers of the images remaining after discarding the divergent ones, for 160000 images of each class at the beginning:

- C60keV: 131882
- C80keV: 103120
- C100keV: 147112
- gamma: 145463
- dust-fog: 120896

And the total number of images is 648473 left from the starting 800000.

After all transformations here are the features we used to train algorithms on:

- **Target**: 1 for signal and 0 for background.
- **Polarisation**: number from 1 to 8 describing the polarisation angle.
- **Amplitude**: parameter of the Gauss fit, describing the brightness of the grain.
- x_{cent}: X coordinate of the grain centre.
- y_{cent}: Y coordinate of the grain centre.
- Σ_{xy}: off-diagonal parameter of the covariance matrix, describing the rotation of the Gaussian.
- **Area**: πab, area of the elliptic fit.
- ε: b/a, eccentricity of the elliptic fit.
- **Minor axis**: a, the smaller axis of the elliptic fit.

2 Training the algorithms

We used 3 amounts of data to train the algorithms: 1000 samples of each class, 5000 samples and 20000 samples. Each sample means 8 images with different polarisations.

The first try was fast to fit and allowed me to test different algorithms. The second one showed more realistic performance and the last one was to check how the performance changes with increasing the amount of data.

We used **precision** as a scoring metrics, since we are more interested in not detecting background as a WIMP, than in detecting all the WIMPs. But we also calculated accuracy to compare with Sergey's results.

2.1 Logistic Regression

The easiest and the fastest method, which was obviously not going to work well, but was worth trying, is Logistic Regression. So the scores for different amounts of data are:

- 0.57 for 1000 samples of each class
- 0.561 for 5000 samples of each class
- 0.589 for 20000 samples of each class

Different regularisations change only the fourth decimal of the result.
This is much worse, than CNN result of Sergey, as expected.

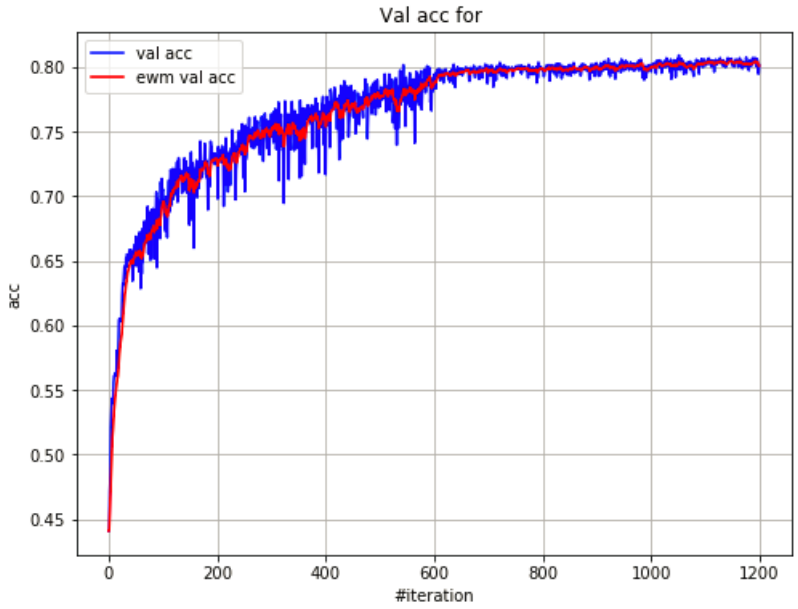
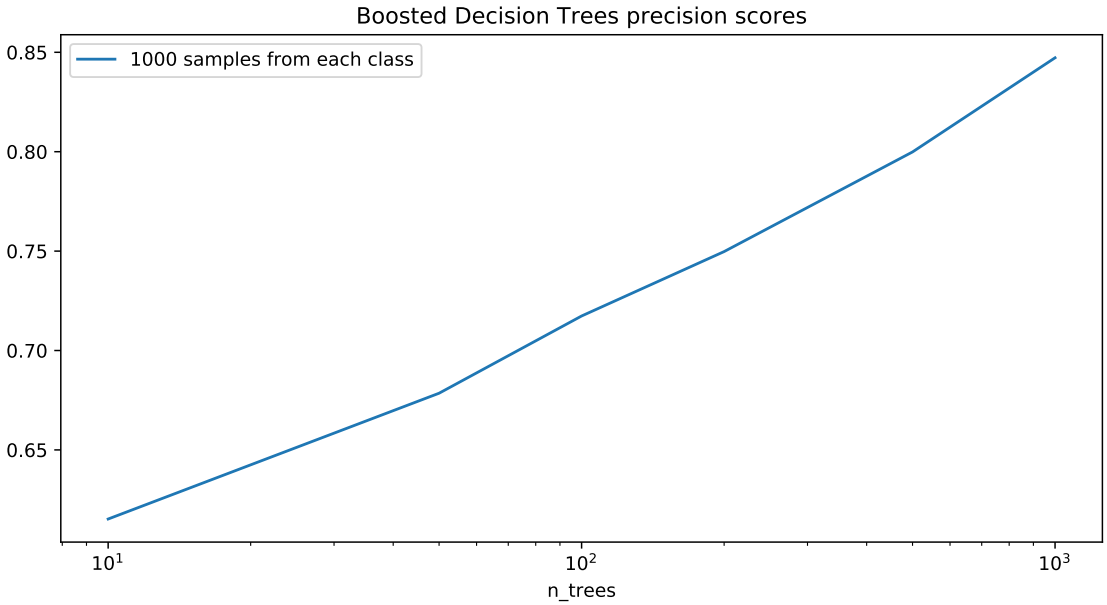


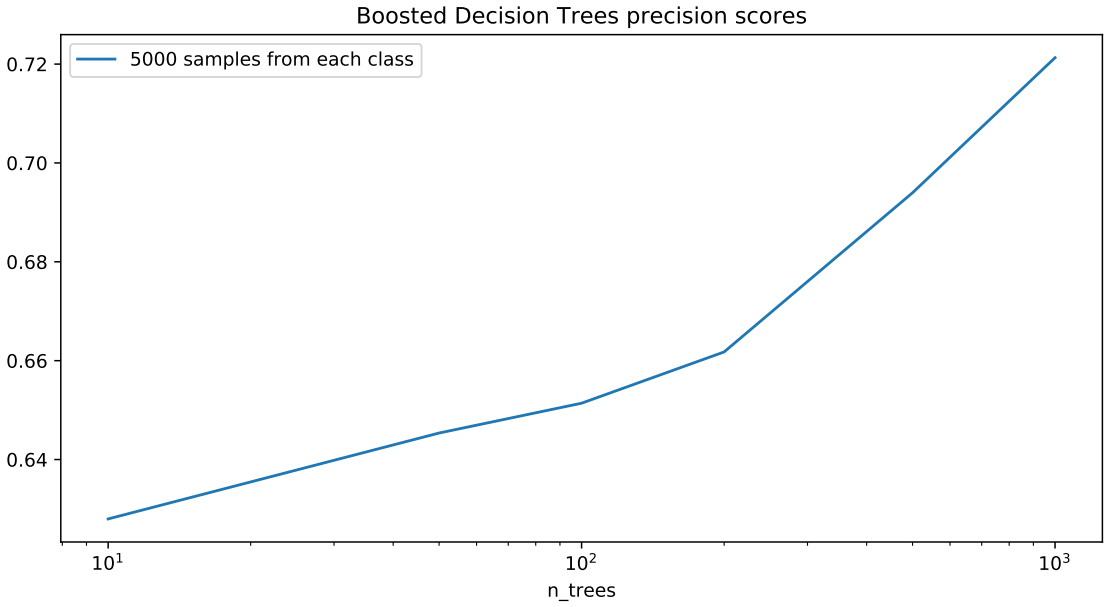
Figure 5: The accuracy score of Sergey's Convolutional Neural Network

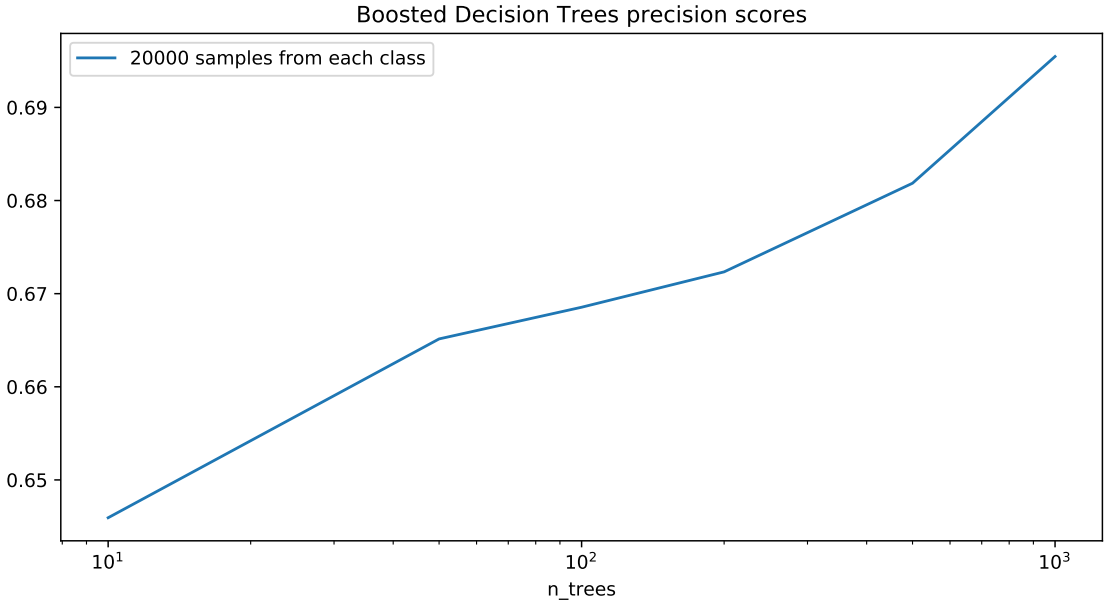
2.2 Boosted Decision Trees

More promising algorithm was BDT. To check it's performance we trained it with all default Scikit-learn parameters (maximum tree depth, splitting criterion etc.), but with different number of estimators.

The most interesting result is that the scores of algorithm are getting **lower** for bigger amount of training data, but continue to grow with number of trees.







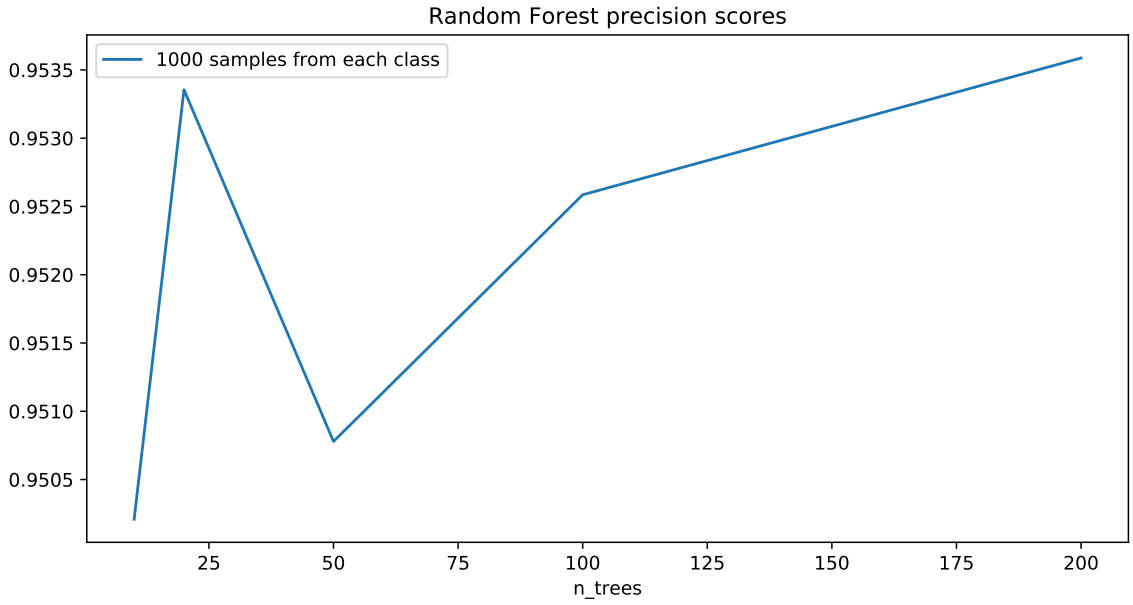
Still, it seems possible to outscore Sergey's CNN (Fig. 5), since the score continues growing logarithmically with number of trees, but the computational cost grows linearly and there are the risk of overfitting the data for huge amounts of estimators.

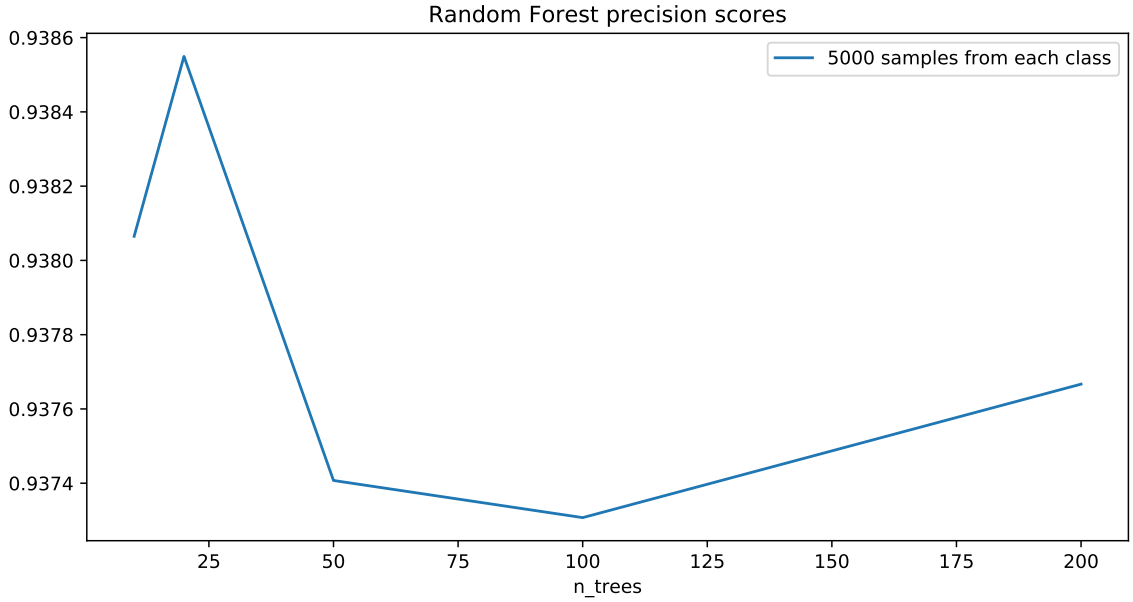
2.3 Random Forests

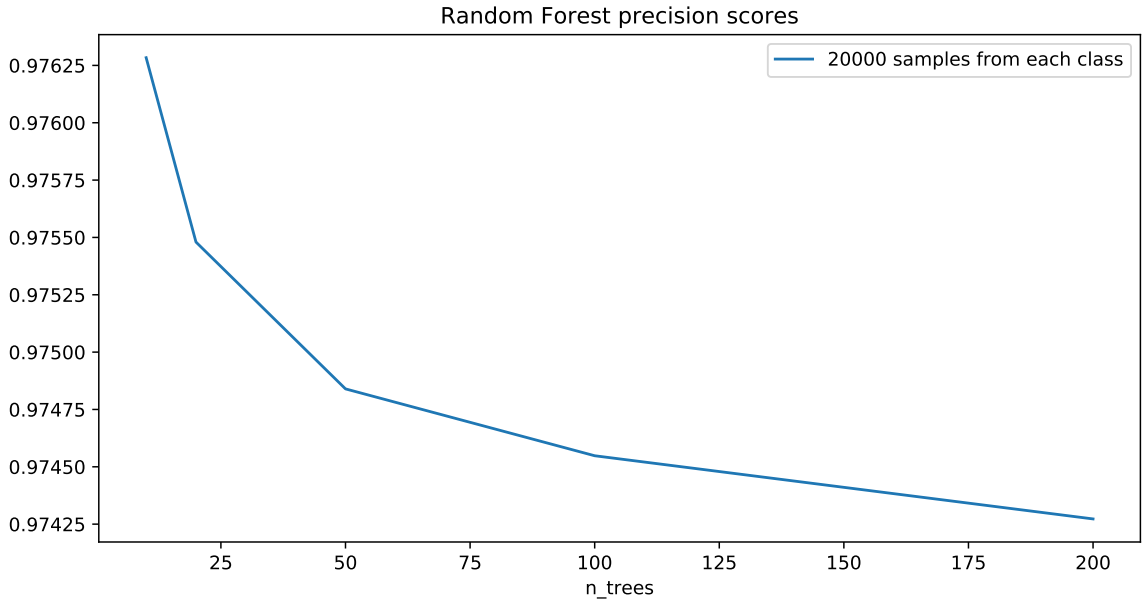
Random Forests turned out to be much better, than Boosted Decision Trees. First of all, it trains faster. Secondly, it does not tend to overfit the data with increasing the number of estimators.

Finally, it shows great performance even for small number of estimators for any amount of data.

Here again we used default Scikit-learn parameters for the algorithm, that means performance still can be improved by tuning hyperparameters.







For the fair comparison of the performance with Sergey's results (Fig. 5) , the **accuracy** for 100 estimator Random Forest is 0.977, which is still much higher.

3 Conclusions

The Random Forest approach gave great results and seems very promising. There are different possible ways to improve it's performance. The first thing to try is tuning different hyperparameters and training on the whole available data set. More important thing is the Gaussian fit itself. It is possible to find another better set of features to represent the images, and it is definitely important to handle divergent images in some way. We are loosing about 1/5 of total data, but even bigger part of signal data.

CNN's are good because they process all the data without missing any information, but still give poorer performance than RF. What's more, they need much more computational power to train even comparing to Random Forests with Gaussian fits together.

So it seems a good idea to play around with Random Forests before we get much more computational power.