Flavours of Physics: Finding $\tau \to \mu\mu\mu$. First place solution

October 17, 2015

Team: Go Polar Bears
Names: Vladislav Mironov (littus), Alexander Guschin
Location: Russia, Moscow
Emails: hello@vladmironov.com, 1aguschin@gmail.com
Competition: Flavours of Physics: Finding $\tau \to \mu\mu\mu$

1 Summary

This competition depends on the three main parts: feature engineering (read as improvement of AUC score), agreement test and mass correlation. So there should be no surprise that each of our models solves exactly one problem at a time and like the famous Borromean rings supports each other (see 2.2–2.5 and 2.7, 2.8).

The significant improvment was made by discovering the way of mass calculation (see 2.1). Using summary momentum and velocity we created new feature: 'new_mass'. Unfortunately this feature wasn't perfect and had some correlation inaccuracy with original mass. That was the reason for our special XGBoost model for mass error only (see 2.6).

In the final submit there are two types of classification: by XGBoost and by Neural Networks. NN in our case play supportive role. Quality of it could be improved and the model could be used for a main prediction at the future. At last our models are combined by arithmetic and geometrical mean (see 2.9).

2 Model explanation

2.1 New physical features

$$\mathrm{p0\_pz} = \sqrt{\mathrm{p0\_p}^2 - \mathrm{p0\_pt}^2}$$
$$\mathrm{p1\_pz} = \sqrt{\mathrm{p1\_p}^2 - \mathrm{p1\_pt}^2}$$
$$\mathrm{p2\_pz} = \sqrt{\mathrm{p2\_p}^2 - \mathrm{p2\_pt}^2}$$
$$\mathrm{pz} = \mathrm{p0\_pz} + \mathrm{p1\_pz} + \mathrm{p2\_pz}$$
$$\mathrm{p} = \sqrt{\mathrm{pt}^2 + \mathrm{pz}^2}$$
$$\mathrm{speed} = \frac{\mathrm{FlightDistance}}{\mathrm{LifeTime}}$$
$$\mathrm{new\_mass} = \frac{\mathrm{p}}{\mathrm{speed}}$$

First of all we should find projection of the momentum to the z-axis for each small particle. Then summarise all of them for our big particle and get pz. After that we could find full momentum p.

From equation $time * Velocity = Distance$ we could find Velocity (in our case it's the 'speed' feature). Considering that FlightDistance calculated untruly and knowing it's error, theoretically, we could improve our prediction. But after few different implementations with no improvment we leave 'FlightDistance' as it is.

The same thing for 'new_mass'. From the scholar equation $p = mV$ we get the rest.

2.2 The Good XGB (XGB1)

Very small binary-logistic xgboost (only five trees!) which gives us excellent (but still not perfect) AUC, medium KS error and not-so-bad Cramer-von Mises error.

2.3 The Bad XGB (XGB2)

Satisfactory AUC, medium KS error and very low Cramer-von Mises error. The 'SPDhits' feature is cutted. The main trick is to use geometrical mean of the models. It radically decrease Cramer-von Mises error and improve AUC.

2.4 ~~and The Ugly XGB~~ and another XGB (XGB3)

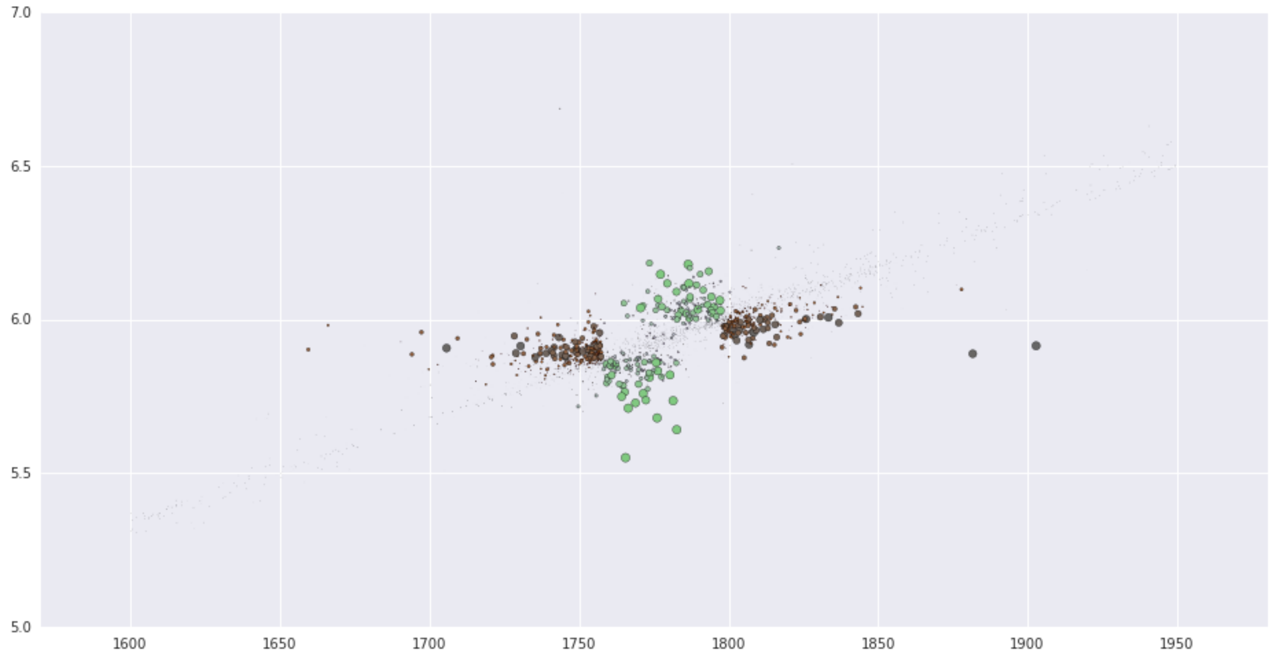
Good AUC, high KS error and very low Cramer-von Mises error.
Simple XGBoost with 700 trees and bagging.

2.5 Another Good XGB (XGB4)

And again very small forest with only three decision trees. Bagging and cut-edging parameters.

2.6 Corrected mass and new features

The main problem with 'new_mass' that it poory correlates with real mass and generates both false-positive and false-negative errors near signal/background border:



So we implement model which corrects this behavior. And it's the heaviest part of the solution. XGBoost with almost three thousand trees and with all features predicts new_mass error. Jointly we use multilevel kfold with bagging. In addition at this stage we calculate new_mass_delta and new_mass_ratio. This features we'll use for next standing XBG and Neural Networks:

$$\text{new\_mass\_delta} = \text{new\_mass} - \text{new\_mass2}$$
$$\text{new\_mass\_ratio} = \frac{\text{new\_mass}}{\text{new\_mass2}}$$

2.7 XGB with new mass (XGB5)

Heavy XGB with 1500 trees and all new features: new_mass2, new_mass_delta, new_mass_ratio. Very High AUC, high KS error and high Cramer-von Mises error. And ones again: bagging.

2.8 NN with new mass

Neural Network with all new features, bagging. One DenseLayer with 8 neurons. Any other configurations as: more layers, more neurons shows the same or less AUC. Also Dropout leads to poorer results.

All of this could be explained by physical nature of the contest. Every feature has clear and one-way dependence between each other.

2.9 Combining predictions

Final combination:

$$\text{first} = (\text{XGB1}^{0.5} * \text{XGB2}^{2}) * 0.5$$
$$\text{second} = (\text{XGB1} * \text{XGB2}^{0.85} * \text{XGB3}^{0.01} + \text{XGB2} * \text{XGB4}^{900} * 0.85 + \text{XGB3}^{1000} * 2)/3.85$$
$$\text{final} = 0.5 * \text{second}^{3.9} + 0.2 * \text{first}^{0.6} + 0.0001 * \text{second}^{0.2} * \text{first}^{0.01}$$

3 Dependencies

Python	2.7.6
Lasagne	0.2.dev1
Theano	0.7.0
ipython	4.0.0
nolearn	0.6a0.dev0
jupyter	1.0.0
pandas	0.16.2
scikit-learn	0.15.2
scipy	0.15.1
numpy	1.11.0.dev0+941a4e0
xgboost	0.4

4 Hardware

AmazonEC2 Cluster m4.4xlarge.
Our code runs on 16 cores and about 4GB RAM (sic!).
Time of execution is about 2 hours.