

Data&Science: HEP triggers

Мельник Богдан



О себе

- Работаю в Яндексе
- Разрабатываю поиск в Яндекс.Маркете
- Играю в Kaggle
- Семинарист в ШАДе

Конкурс

- Основан на данных из БАКа
- Был анонс на "Data&Science: Большой адронный коллайдер"
- Проходил на Kaggle in Class с 15 сентября по 2 октября

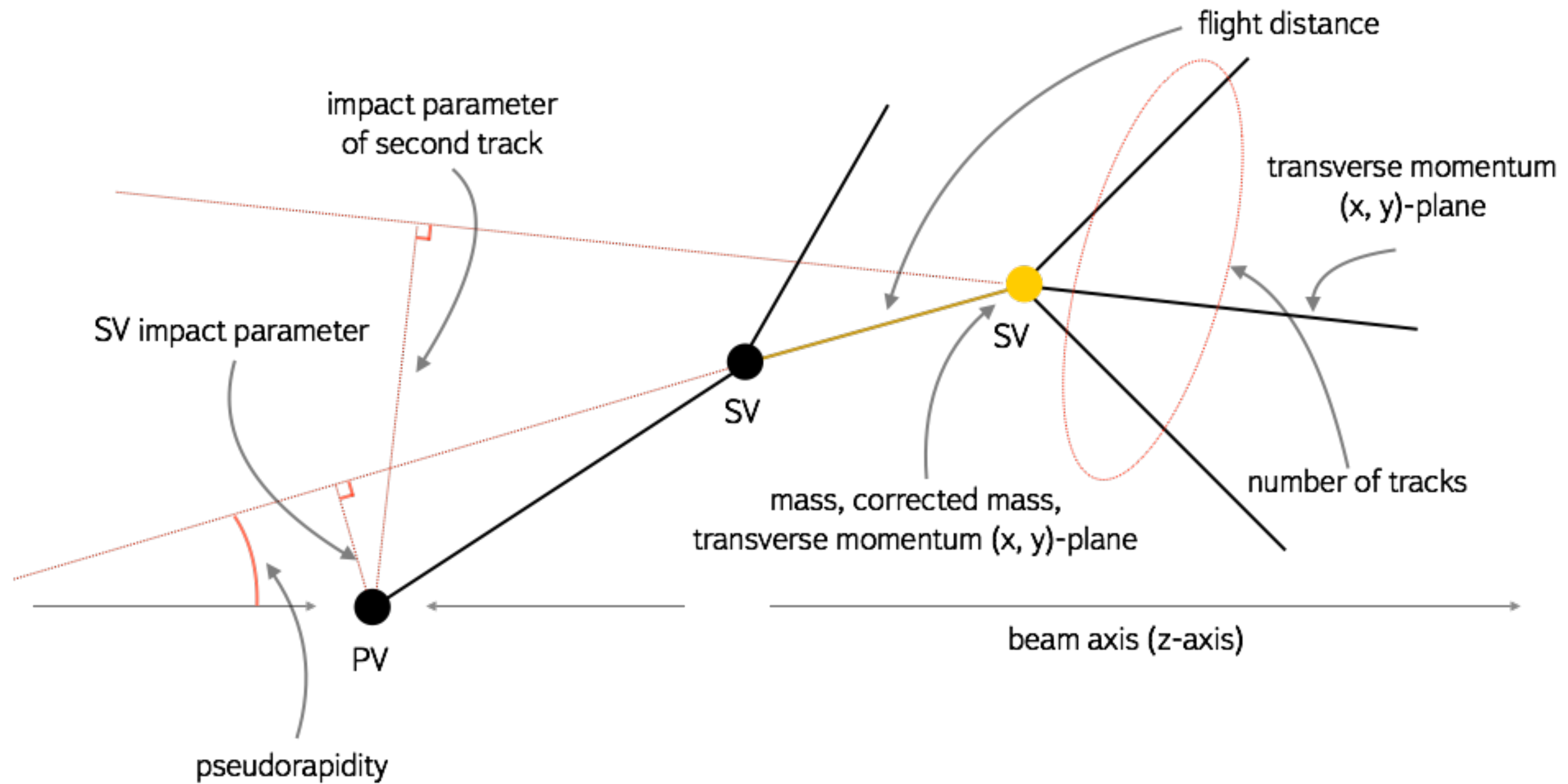
Конкурс

#	Δrank	Team Name	Score 	Entries	Last Submission UTC (Best – Last Submission)
1	—	Bogdan Melnik	0.97825	37	Sun, 02 Oct 2016 12:08:26 (-3.1h)
2	—	KonstantinOstrovsky	0.97565	18	Sat, 01 Oct 2016 17:00:40
3	—	quants 	0.97548	44	Sat, 01 Oct 2016 23:22:29
4	↑1	Aleksey Zubakov	0.96688	10	Thu, 29 Sep 2016 22:07:10
5	↓1	Igor.Slinko	0.96668	10	Sun, 02 Oct 2016 18:48:02 (-0.3h)
6	—	DmitriyG	0.96632	26	Sat, 01 Oct 2016 22:05:00 (-2.5d)
7	—	Ivan Potapov	0.96540	14	Thu, 29 Sep 2016 11:30:34 (-2h)
8	—	Marty Lee	0.96523	3	Tue, 20 Sep 2016 21:07:34 (-0h)
9	—	CrocoPie	0.95374	6	Mon, 19 Sep 2016 21:01:14 (-23.7h)
10	—	protan	0.95359	1	Sat, 17 Sep 2016 21:19:13

Задача

- Даны столкновения двух протонов
- При столкновении генерируется несколько неустойчивых частиц
- Через некоторое время эти частицы распадаются

Задача



● decay vertex of studied particle

Задача

- БАК генерирует много данных (~2 млн. событий в секунду)
- Большинство столкновений не несут никакой пользы для науки
- Чтобы не хранить все события необходимо иметь систему, позволяющую оставлять только важные события
- Система должна работать онлайн во время эксперимента на БАК

Задача

- БАК генерирует много данных (~2 млн. событий в секунду)
- Большинство столкновений не несут никакой пользы для науки
- Чтобы не хранить все события необходимо иметь систему, позволяющую оставлять только важные события
- Система должна работать онлайн во время эксперимента на БАК

Данные

- 498412 строчек в train, 460653 строчек в test
- Каждая строчка соответствует частице, полученной при столкновении
- 13 величин, описывающих частицы (масса, импульс и т.п.)

Данные

- 81068 событий в train, 84000 событий в test
- Одному событию соответствует несколько частиц
- Каждое событие может быть либо интересным, либо неинтересным

Данные

	EventID	Label	Mass	Corrected_mass	Pt
0	0	1	3440.680014	5202.580014	15583.900014
1	0	1	1319.829991	2465.479991	3477.259991
2	0	1	2732.810016	5804.080016	9356.570016
3	0	1	1674.579998	4423.859998	6889.459998
4	0	1	1844.839993	5744.339993	12385.899993
5	0	1	702.708994	2990.169994	9823.039994
6	1	0	3394.169986	5829.519986	7865.049986
7	2	0	1637.959993	5097.399993	2364.759993

Метрика

- Бинарная классификация для событий
- ROC AUC

Наивное решение

- Предсказывать важность каждой частицы отдельно
- Нужно как-то агрегировать результаты по частицам для каждого события

Наивное решение

- `XGBClassifier(n_estimators=1000, max_depth=5, learning_rate=0.05)`
 - `max()` — 0.9673
 - `median()` — 0.9652
- `+objective='rank:pairwise'`
 - `max()` — 0.9668
 - `median()` — 0.9670

Стекинг!

- Будем использовать результаты, полученные для каждой частицы, для создания предсказания целого события
- Кроме предсказаний будем использовать агрегированные исходные данные
- Для стекинга я разбил данные по событиям на 5 фолдов

Первый слой

- Пять классификаторов, использующие исходные данные, как в наивном решении
 - `XGBClassifier(n_estimators=1000, max_depth=5, learning_rate=0.05)`
 - `XGBClassifier(n_estimators=100, max_depth=100, learning_rate=0.1)`
 - `XGBClassifier(n_estimators=100, max_depth=15, learning_rate=0.4)`
 - `RandomForestClassifier(n_estimators=1000)`
 - `ExtraTreesClassifier(n_estimators=1000)`

Обработка результатов первого слоя

- Результаты работы классификаторов были агрегированы по событию
 - 5 квантилей (0., .25, .5, .75, 1.)
 - Отношение минимума к максимуму
 - Отношение максимума к минимуму
 - Разница максимума и минимума

Агрегация данных

- Чтобы использовать исходные данные во втором слое их необходимо агрегировать по событию
- Для каждого события я посчитал:
 - Количество частиц
 - Для каждого физического измерения
 - Среднее
 - Стандартное отклонение
 - 5 квантилей (0., .25, .5, .75, 1.)
 - Отношение максимального значения к минимальному и наоборот
 - Разница между максимальным и минимальным значением

Второй слой

- В итоге я получил матрицу размером 81068 на 171
- Будем её использовать для предсказания важности каждого события
- `XGBRegressor(objective='rank:pairwise', n_estimators=1000, learning_rate=0.05, max_depth=5, colsample_bytree=0.4, min_child_weight=3.83)`
- AUC — 0.980791

Больше классификаторов!

- `XGBRegressor(objective='binary:logistic', n_estimators=1000, learning_rate=0.05, max_depth=5, colsample_bytree=0.4)`
- `XGBRegressor(objective='rank:pairwise', n_estimators=1000, learning_rate=0.01, max_depth=5, subsample=0.8)`
- `XGBRegressor(objective='rank:pairwise', n_estimators=1000, learning_rate=0.01, max_depth=6, colsample_bytree=0.3)`
- `RandomForestRegressor(n_estimators=1000)`
- `ExtraTreesRegressor(n_estimators=1000)`

Усредняем ранги

- Если усреднить ранги между всеми полученными классификаторами, то можно ещё немного улучшить результат
- AUC — 0.98186
- Что делать дальше?

Третий слой!

- На вход третьему слою подадим ту же информацию, что и второму плюс результаты работы второго слоя
- `XGBRegressor(objective='rank:pairwise', n_estimators=1000, learning_rate=0.01, max_depth=5, colsample_bytree=0.4)`
 - AUC — 0.98190
- `XGBRegressor(objective='binary:logistic', n_estimators=1000, learning_rate=0.01, max_depth=5, colsample_bytree=0.4)`
 - AUC — 98168

Финальное объединение

- Для получения финального результата я оптимизировал AUC с помощью `scipy.optimize.minimize`
- $f(x) = x[0] * (clf_0 \wedge x[1]) + \dots + x[n-1] * (clf_i \wedge x[n])$
- x — вектор коэффициентов
- clf_i — вектор предсказаний i -го классификатора
- Запускал `minimize` около 10000 раз для случайного x

Результат

- AUC — 0.98261
- Public: 0.97765
- Private: 0.97825
- Обучение занимало около 4-х часов на сервере с 32-мя ядрами

Вопросы?