Telstra Network Disruptions

Тренировки ML
Дарья Васюкова
28 мая 2016

О задаче

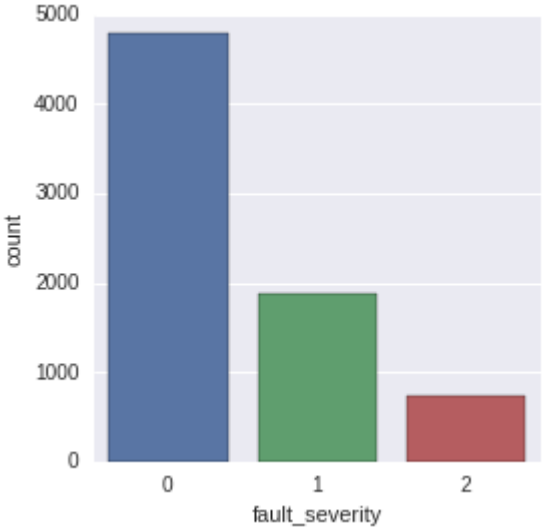
- Предсказать проблемы со связью в сети Telstra
- Классификация на 3 класса
  - 0 – нет проблем
  - 1 – небольшие проблемы
  - 2 – серьёзные проблемы
- Нужно предсказать вероятность каждого класса
- Оценка по multiclass logloss
- Каждый пример – это (время + место), но время в явном виде не дано

О данных

- train.csv: 7381 пример

- severity_type.csv (1:1)
- event_type.csv (1:n)
- resource_type.csv (1:n)
- log_feature.csv (1:n + volume)

	location	fault_severity
id		
14121	118	1
9320	91	0
14394	152	1
8218	931	1
14804	120	0



Базовые фичи

- One-hot кодированные переменные
  - значение volume вместо 1 для log_feature
  - только достаточно распространенные категории
- Label-кодированные переменные
- Подсчет количества для (1:n) категориальных переменных
- Различные агрегаты переменной volume по id (min, max, sum, count, std, mean)

Волшебная фича

- Соотношение location и severity_type
- Столбец location отсортирован
- Добавить переменную – номер примера внутри каждой location-группы

```
sev.head()
```

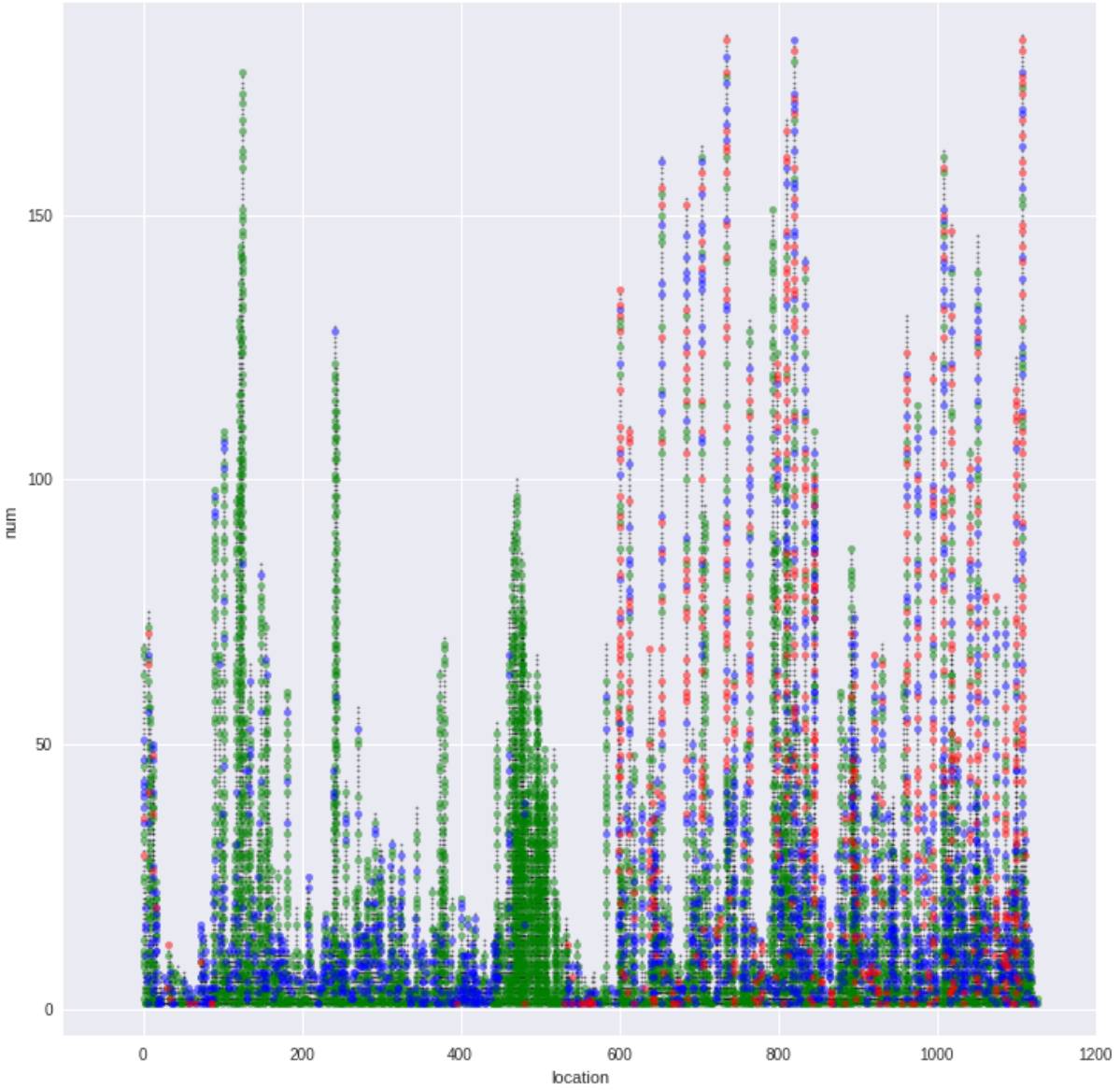
	severity_type
id	
6597	2
8011	2
2597	2
5022	1
6852	1

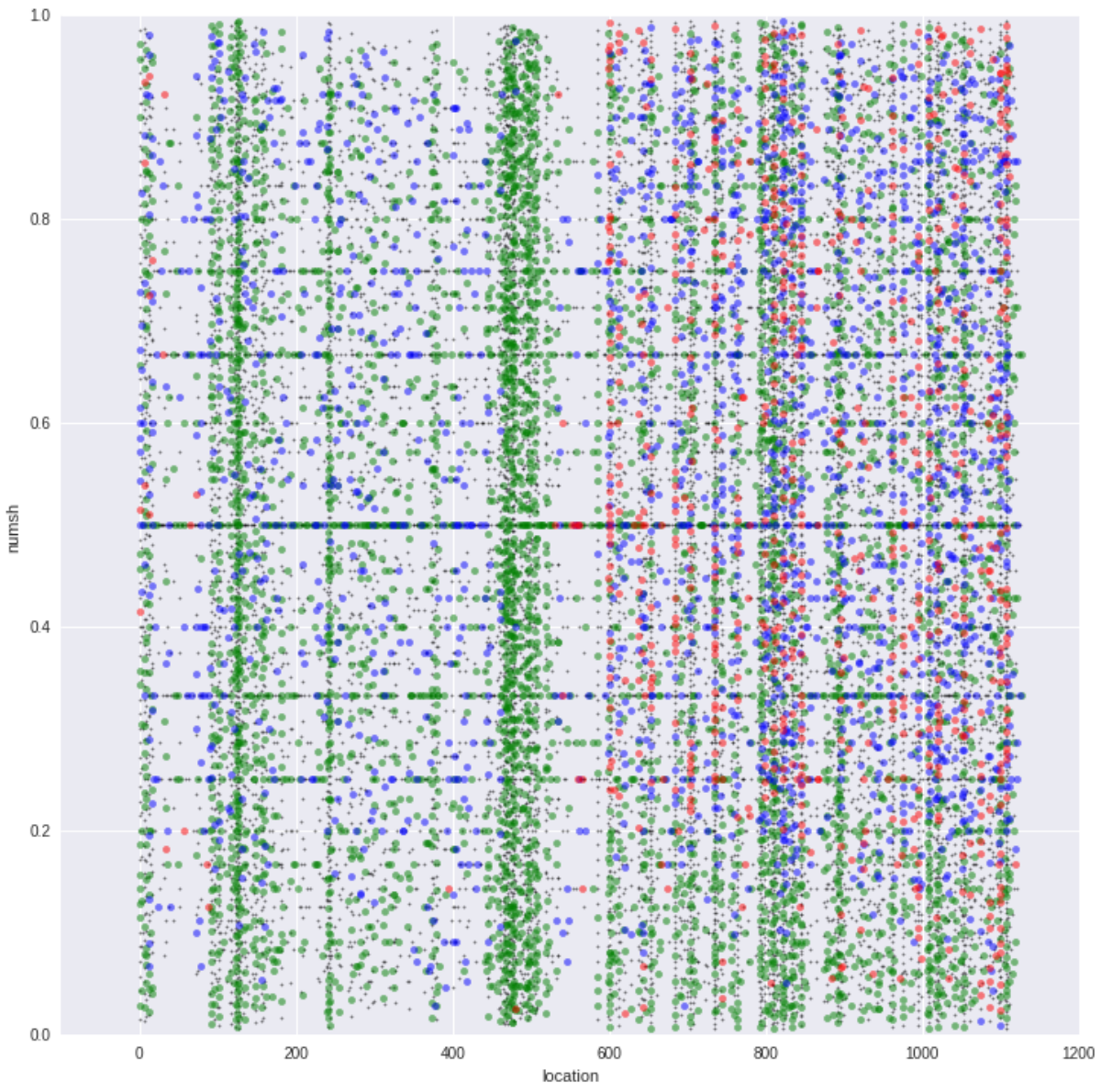
```
sev.head(3)
```

	severity_type	location	num
id			
6597	2	1	0
8011	2	1	1
2597	2	1	2

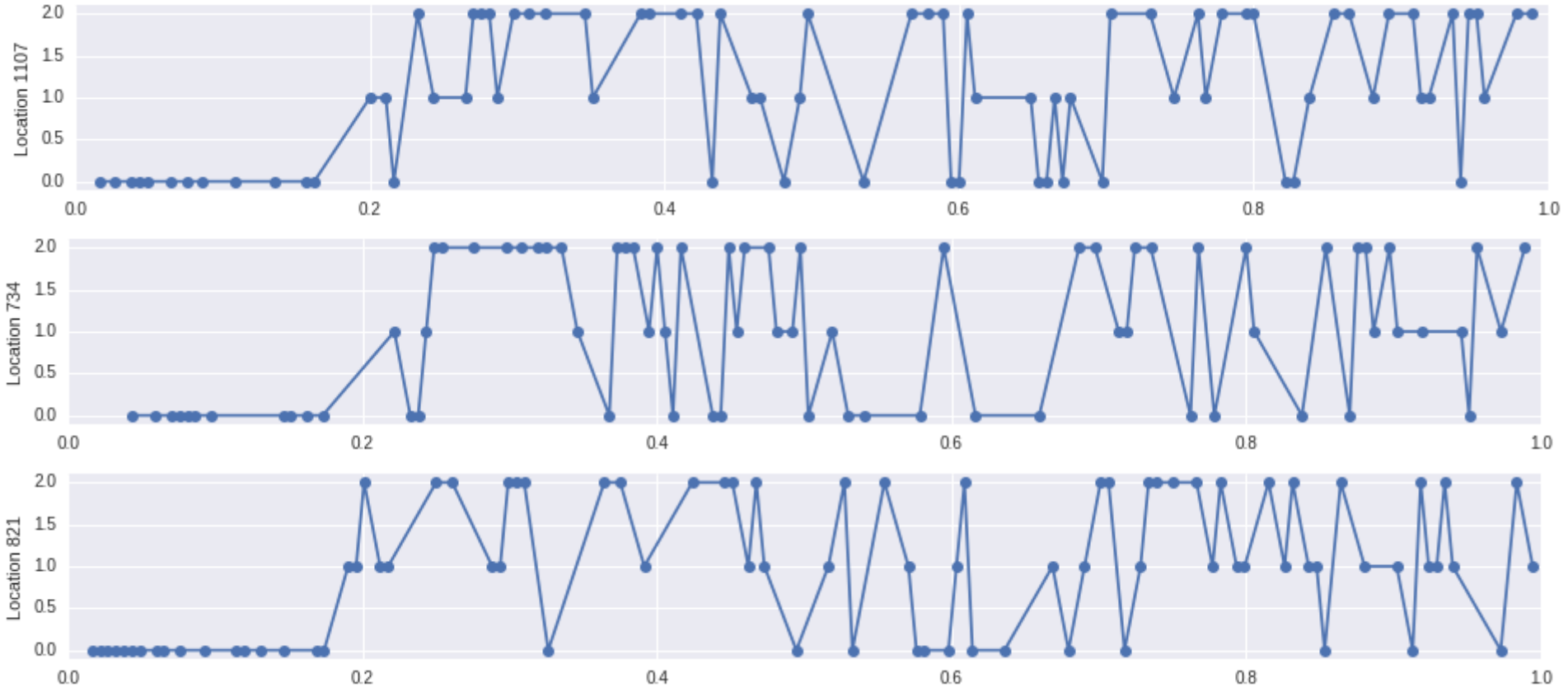
```
sev.tail(3)
```

	severity_type	location	num
id			
6488	2	999	13
878	2	999	14
4464	1	999	15


num
location
0
50
100
150
0
200
400
600
800
1000
1200

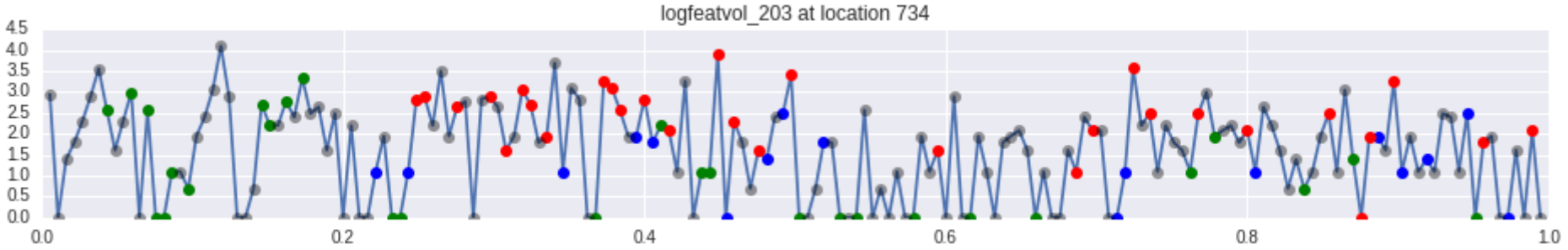

1.0
0.8
0.6
0.4
0.2
0.0
numsh
0
200
400
600
800
1000
1200
location

Целевая переменная от “времени”



Дополнительные фичи с использованием “времени”

- Можем смотреть на графики важных фич от “времени”



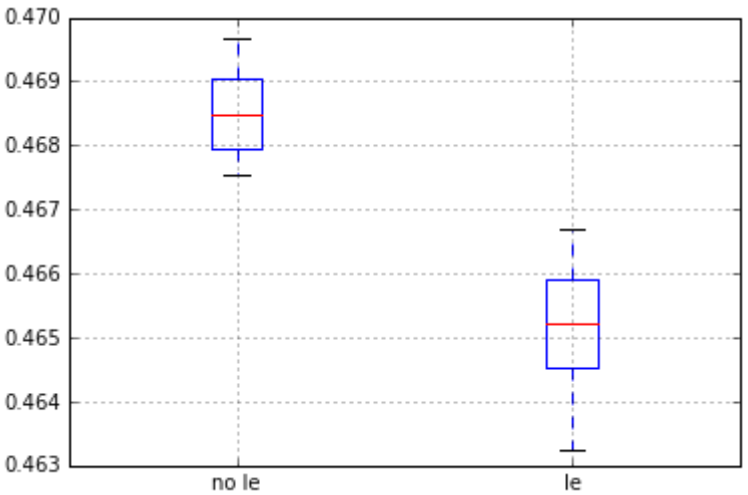
- Разница между текущим уровнем и скользящим средним

Фичи с использованием целевой переменной

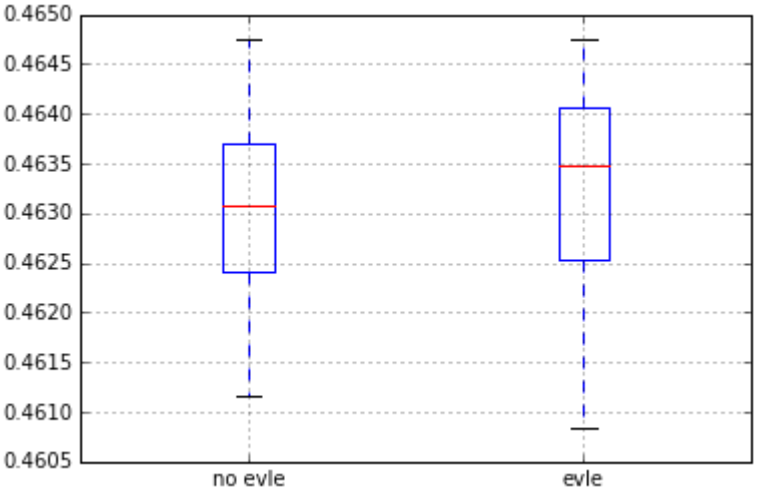
- Прошлое и следующее значение
  - по “времени” внутри каждой location
- Вероятности значений целевой переменной для каждой location
- Как не переобучиться:
  - Вычислять внутри цикла кросс-валидации
  - Leave-one-out encoding: для каждого примера подставить вероятности, полученные подсчетом **других** примеров в этой location.

Отбор фич

- Придумали новую фичу
- Проводим несколько baseline-экспериментов без нее
- Потом столько же, добавив новую
- Сравниваем результаты на графике



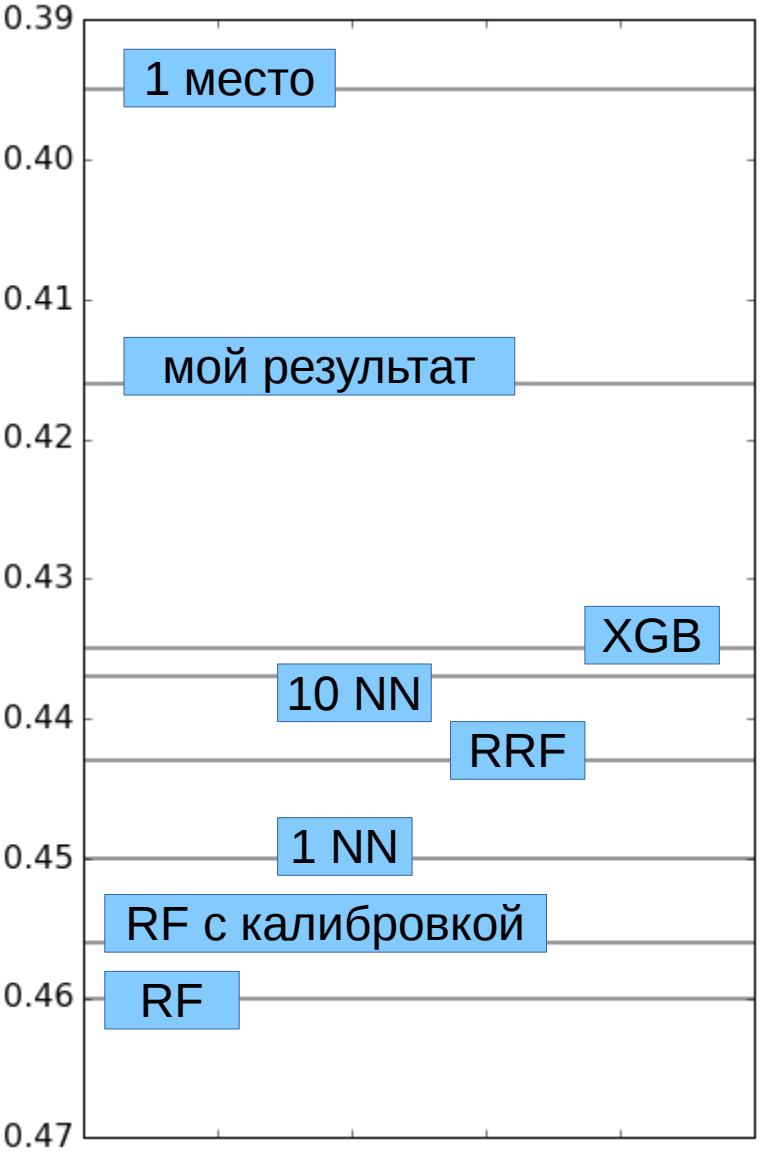
Эту оставим



А эта бесполезная

Модели

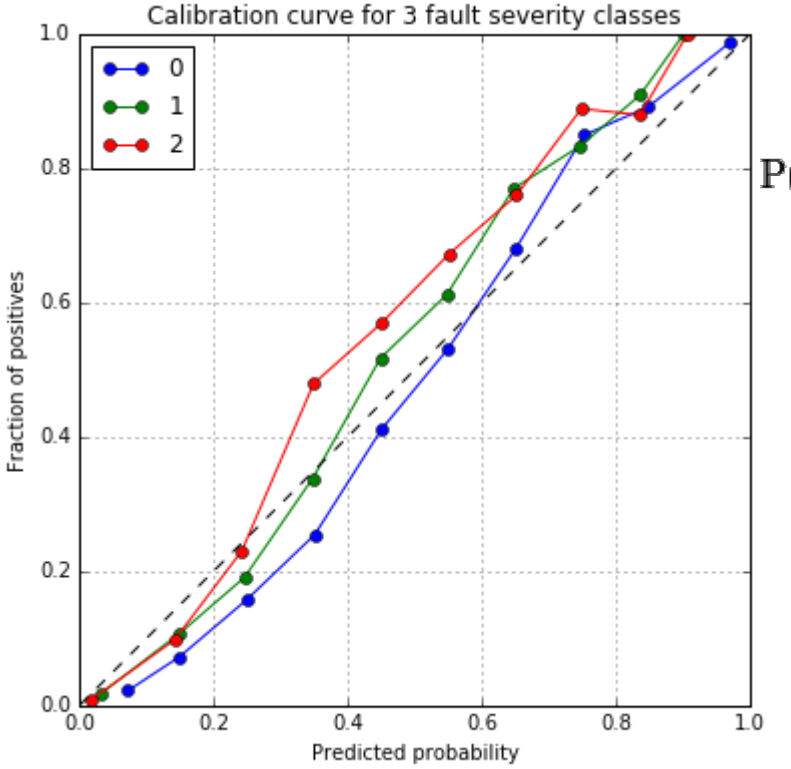
- Random Forest Classifier
- Extra Trees Classifier
- Refined RF and ET
- XGBoost
- Neural nets



Калибровка вероятностей: RF

Оценить качество вероятностей можно с помощью “reliability curve”:

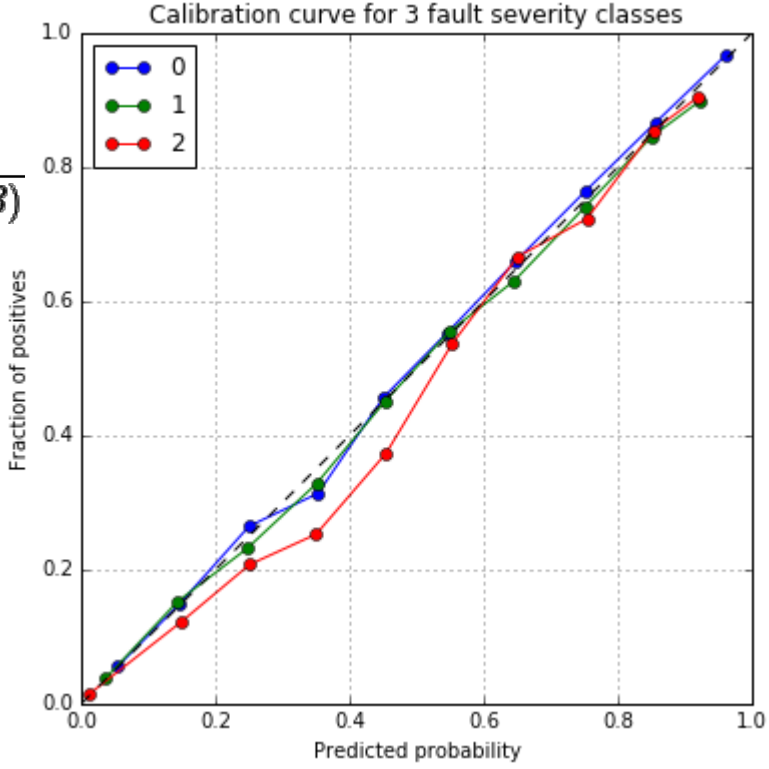
Среди примеров, для которых мы предсказали вероятность ≈ p – какая доля положительных?



Platt scaling

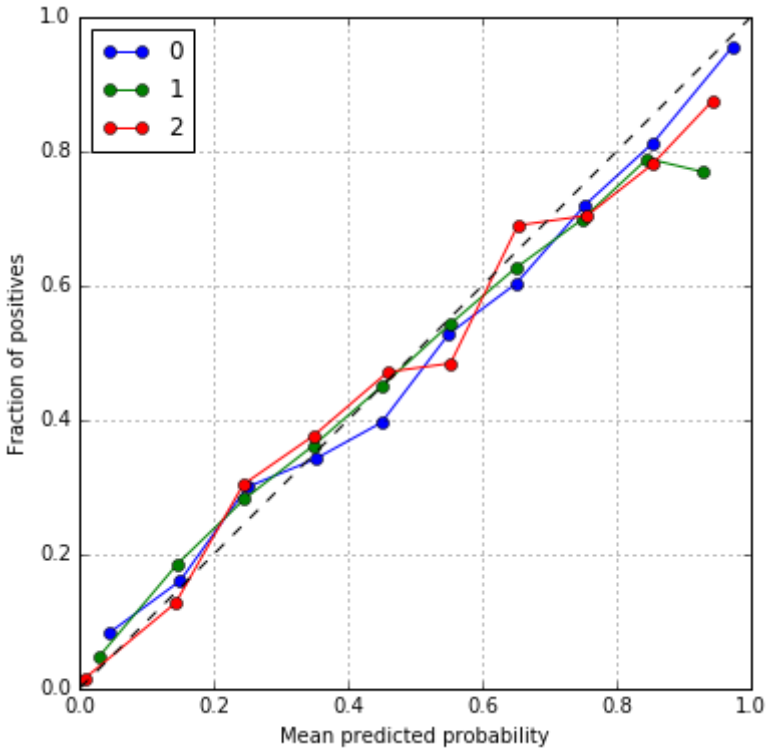
$$\mathrm{P}(y=1|x) = \frac{1}{1+\exp(Af(x)+B)}$$

LB score:
0.479 → 0.472

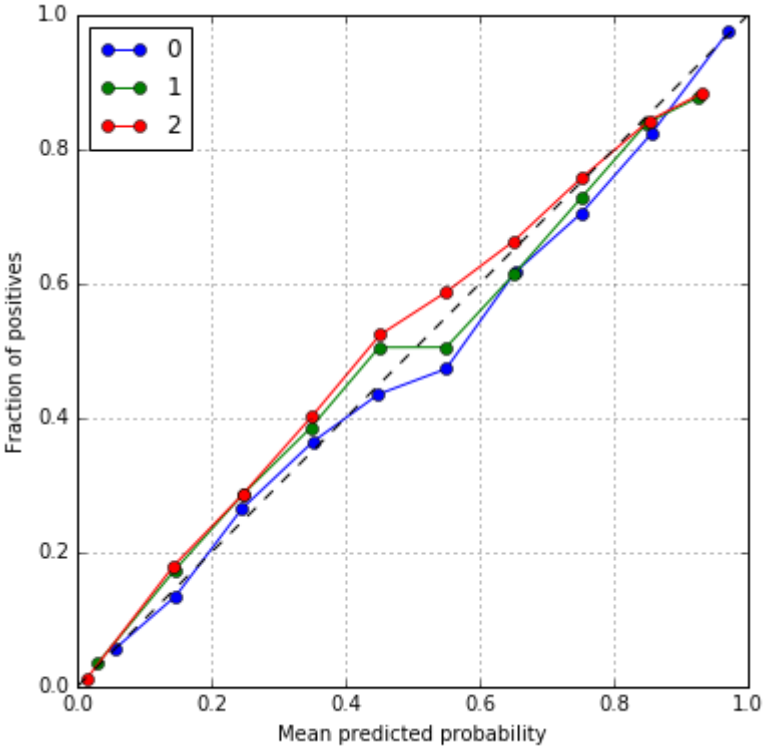


Калибровка вероятностей: NN и XGB

NN



XGBoost



Кривые ближе к диагонали, калибровка не дает улучшений

Global Refinement of Random Forest

Идея

- Все деревья RF строятся независимо друг от друга
- Можем получить улучше-ние, если выучим новые значения в листьях, оптимизируя все одновременно

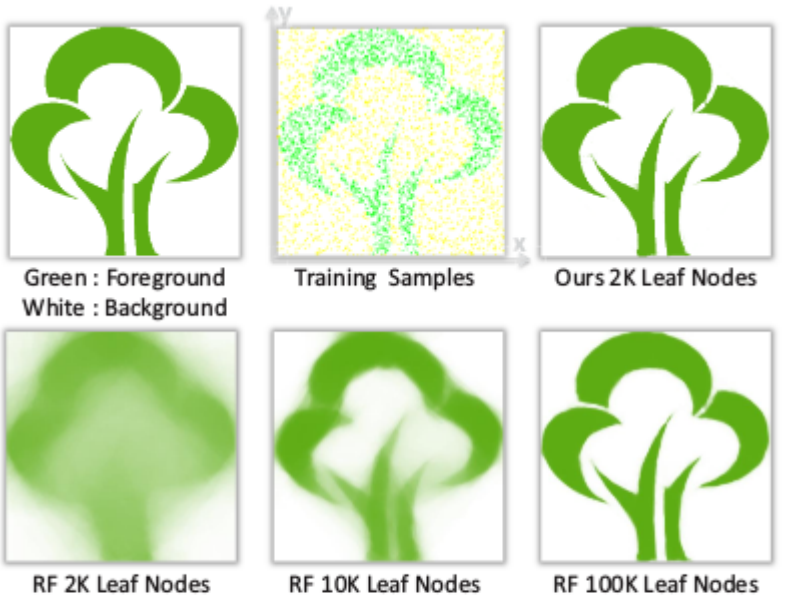


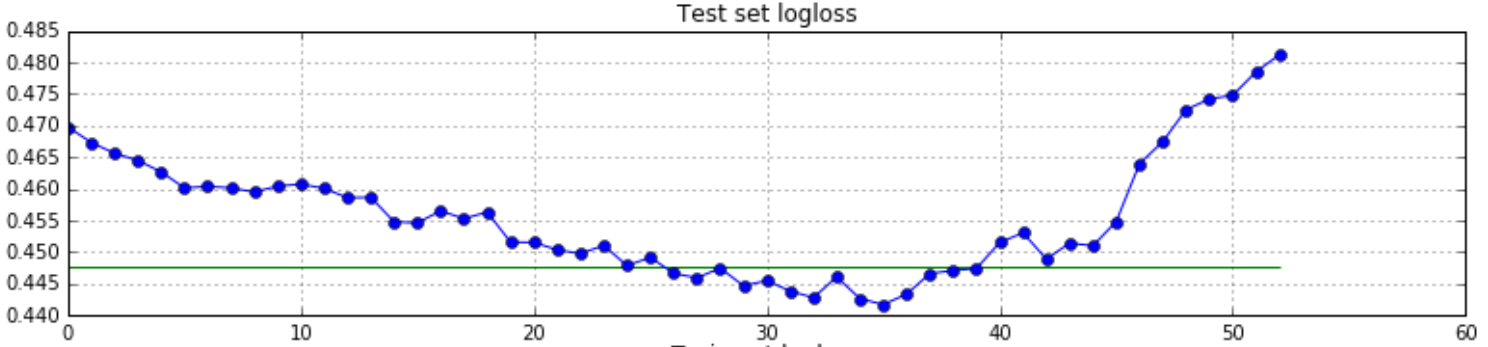
Figure 1. A toy classification task. From left to right, the first row shows the groundtruth label map, training data points, and the probability map predicted by our refined forest. The second row shows the probability map predicted by standard random forest (RF) trained with various depth ($D_{\max} = 5, 8, 12$). Our refined forest can clearly separate the two classes using a smaller number of leaf nodes. In all cases, we use 100 trees in the forest.

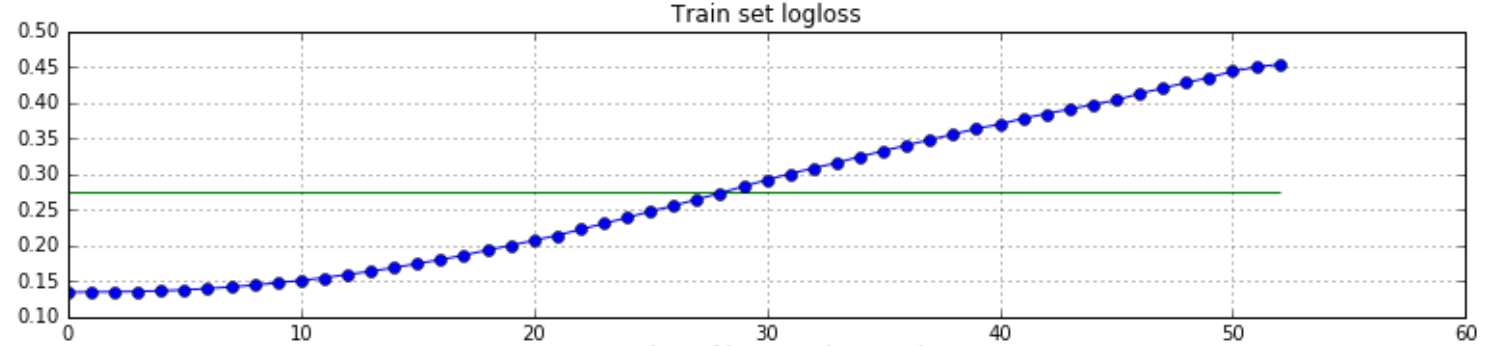
Global Refinement of Random Forest

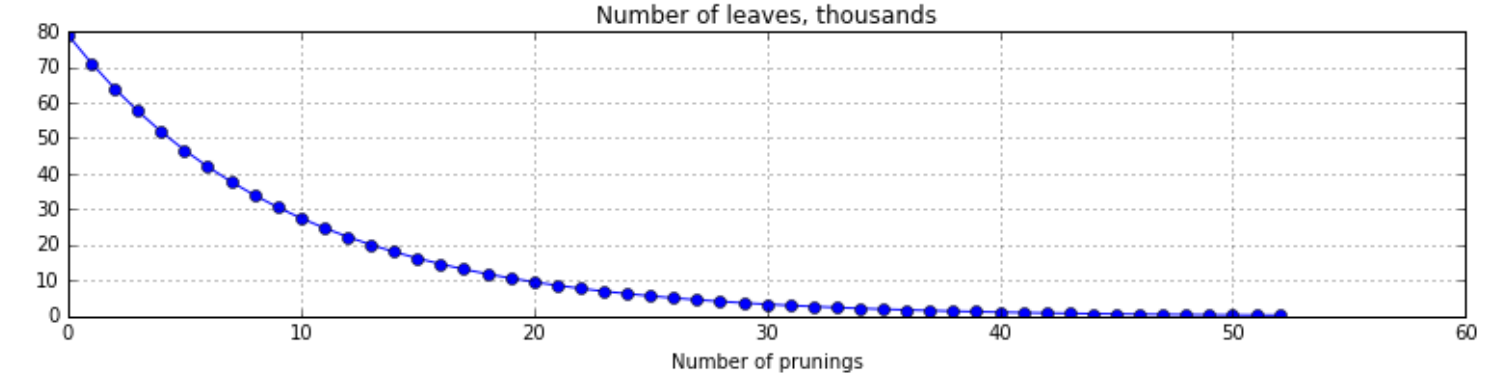
Алгоритм

- Строим лес на обучающей выборке
- Листья = наши новые фичи
- Строим линейную регрессию (с регуляризацией)
- Обрезка деревьев: убираем пары листьев, у которых самые маленькие коэффициенты
- Повторять п. 3-4, пока есть улучшение на валидирующей выборке

Пример refinement-процесса





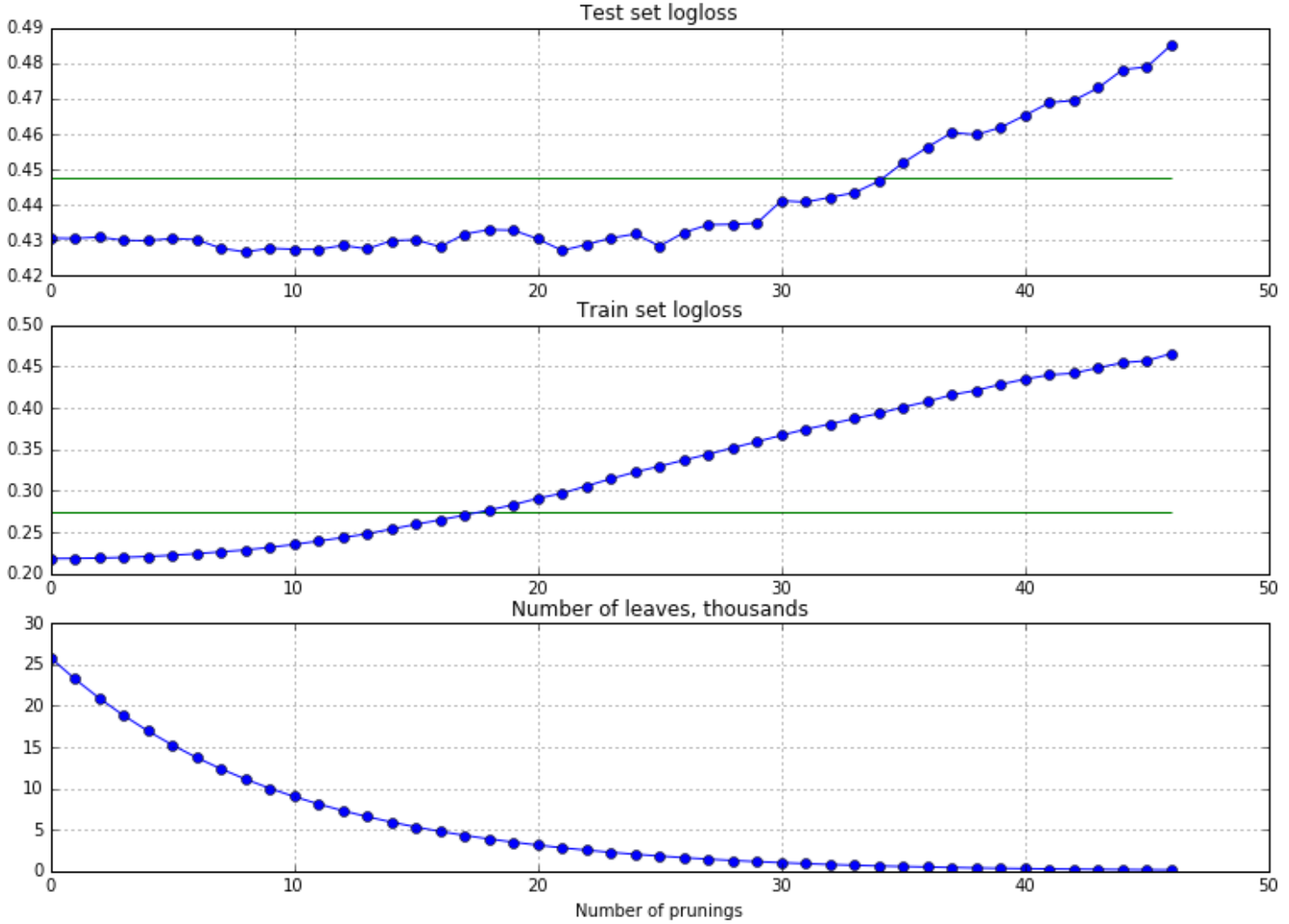


RF 200 деревьев

max_features = 20

min_samples_leaf = 5

Нужны другие настройки RF



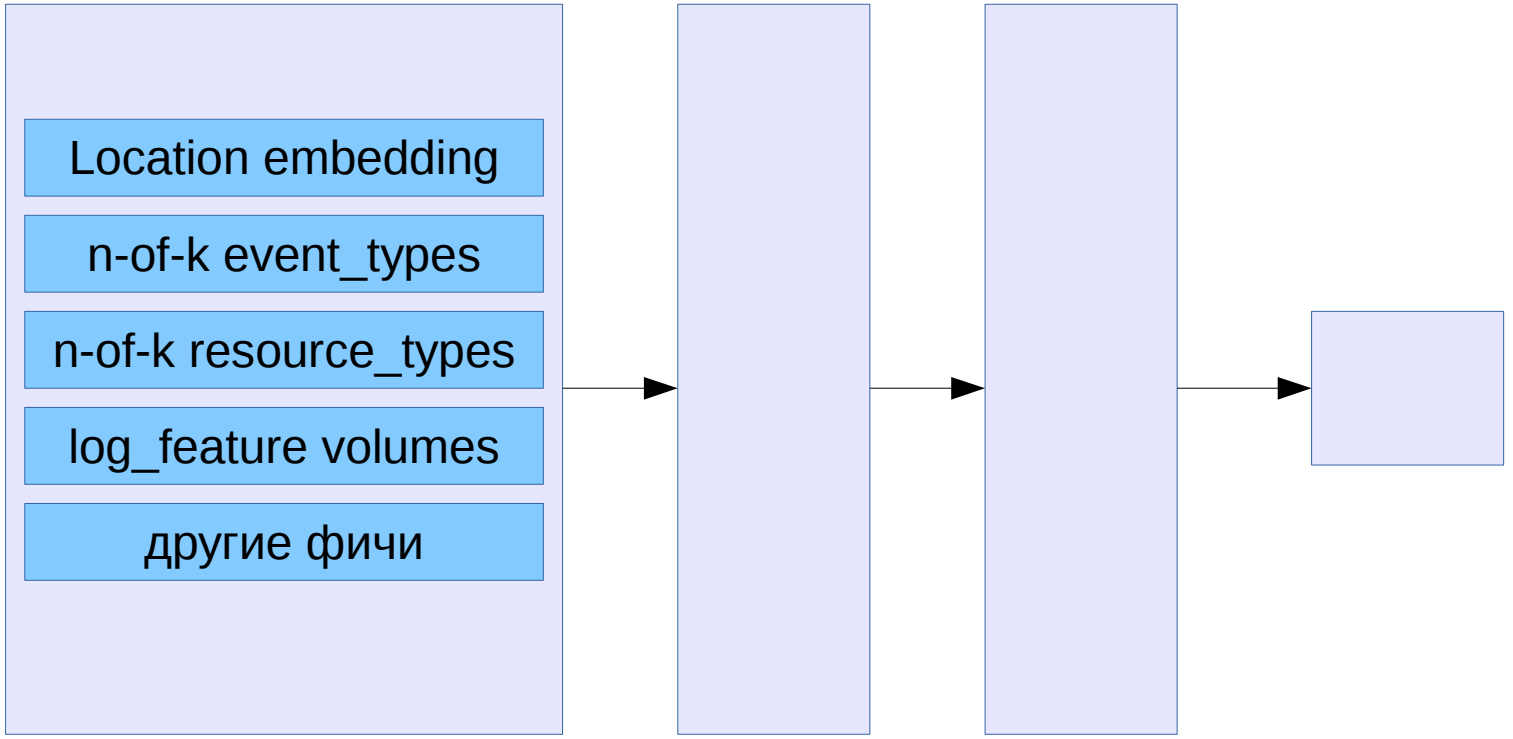
RF 300 деревьев

max_features = 5

max_depth = 7

Нейронные сети

Rossmann Store Sales 3rd place – neural nets with category embeddings



Входной слой
L1 и L2-регуляризация

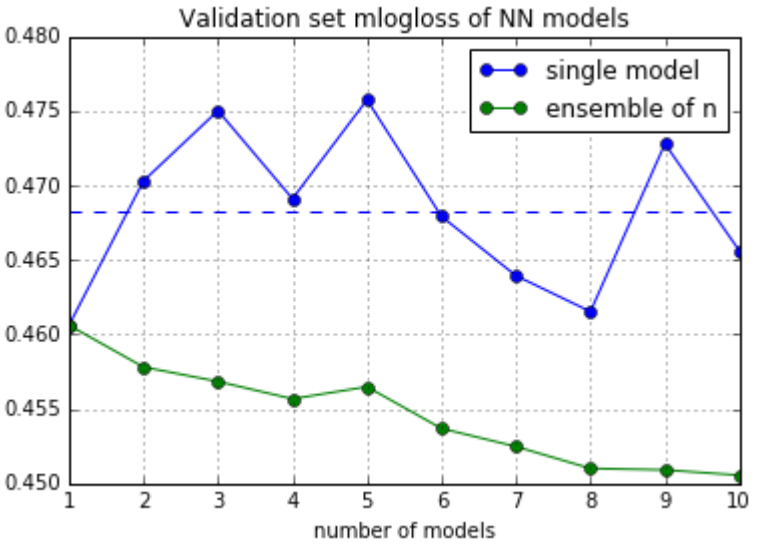
2 скрытых слоя
ReLU-активация
Dropout

Выходной слой:
вероятности 3 классов

Составление ансамбля

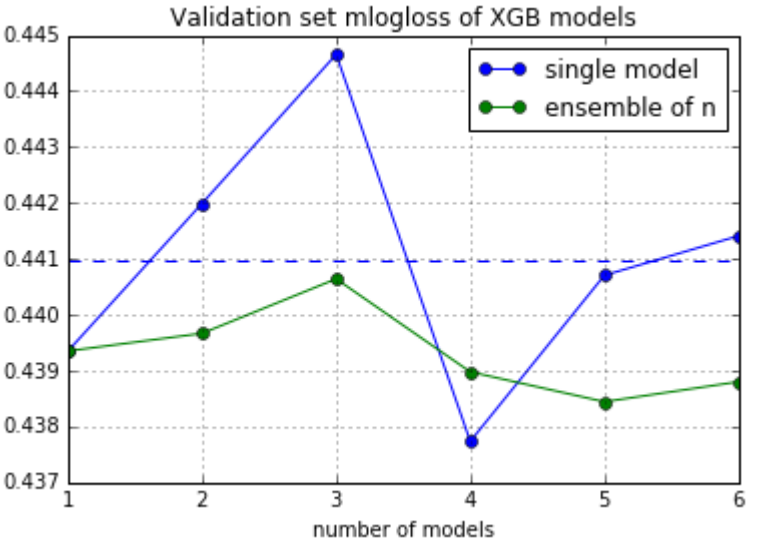
NN

- Одна структура сети
- Разные инициализации
- Усреднение результатов даёт хорошее улучшение



XGB

- Модели с разными настройками
- Каждая отдельная модель лучше
- Усреднение почти не даёт улучшения

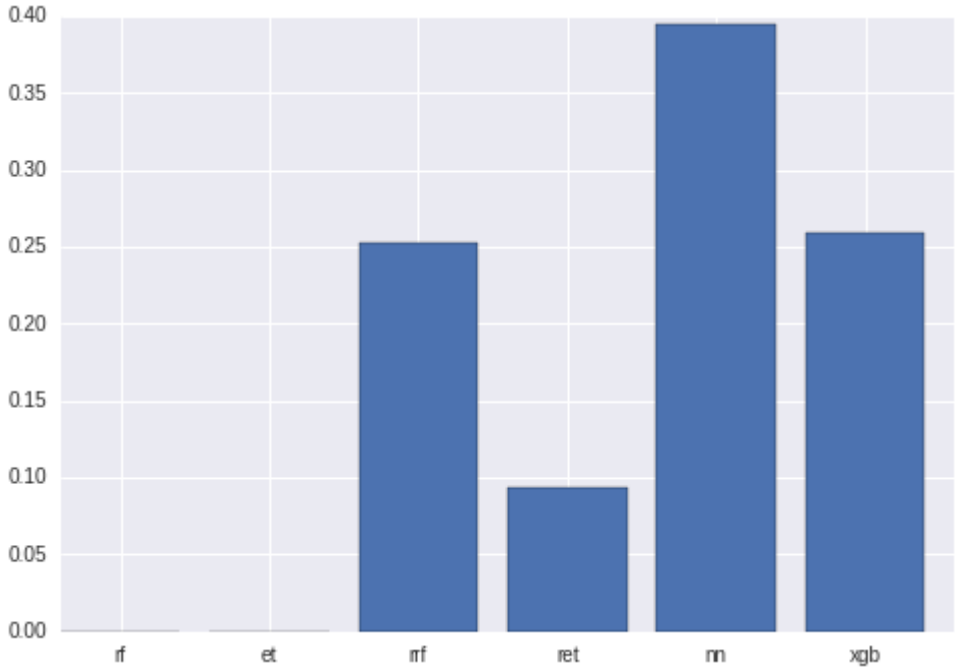


Составление ансамбля

- Линейная комбинация моделей 6 классов
- Коэффициенты (веса) моделей – неотрицательные, в сумме 1

Веса `w = softmax(x) = exp(x) / (sum(exp(x))`

Оптимальные x ищем с помощью `scipy.optimize.minimize`



Автонастройка моделей с помощью Sacred и hyperopt

Sacred

- Инструмент проведения вычислительных экспериментов
- Помогает
  - Следить за всеми параметрами эксперимента
  - Проводить эксперименты при разных настройках
  - Сохранять настройки и результаты в базу данных
  - Воспроизводить результаты

hyperopt

- Библиотека для оптимизации гиперпараметров
- Поддерживает optimizing over awkward search spaces with real-valued, discrete, and conditional dimensions.

Пример Sacred-эксперимента

Script to train an SVM on the iris dataset

```
from numpy.random import permutation
from sklearn import svm, datasets




C = 1.0
gamma = 0.7



iris = datasets.load_iris()
perm = permutation(iris.target.size)
iris.data = iris.data[perm]
iris.target = iris.target[perm]
clf = svm.SVC(C, 'rbf', gamma=gamma)
clf.fit(iris.data[:90],
        iris.target[:90])
print(clf.score(iris.data[90:],
                iris.target[90:]))
```

The same script as a Sacred experiment

```
from numpy.random import permutation
from sklearn import svm, datasets
from sacred import Experiment
ex = Experiment('iris_rbf_svm')

@ex.config
def cfg():
  C = 1.0
  gamma = 0.7

@ex.automain
def run(C, gamma):
  iris = datasets.load_iris()
  per = permutation(iris.target.size)
  iris.data = iris.data[per]
  iris.target = iris.target[per]
  clf = svm.SVC(C, 'rbf', gamma=gamma)
  clf.fit(iris.data[:90],
          iris.target[:90])
  return clf.score(iris.data[90:],
                   iris.target[90:])
```

Пример использования эксперимента

```
from src.experiments.RFexp import ex as rfex
from src.experiments.XGBexp import ex as xgbex
```

Провести эксперимент:

In [2]:
```
run = rfex.run()
```

```
INFO - random_forest - Running command 'rf'
INFO - random_forest - Started
INFO - random_forest - Result: 0.4609685816608816
INFO - random_forest - Completed after 0:00:32
```

Провести эксперимент с другими настройками:

In [34]:
```
run = rfex.run(config_updates = {"clfparams.max_features":1})
```

```
WARNING - root - Changed type of config entry "clfparams.max_features" from str to int
INFO - random_forest - Running command 'rf'
INFO - random_forest - Started
INFO - random_forest - Result: 0.5732386736378875
INFO - random_forest - Completed after 0:00:29
```

Sacred + hyperopt

Определяем пространство поиска

In [38]:
```
space = {'max_features':hp.quniform('max_features',1,20,1),
         'min_samples_leaf':hp.quniform('min_samples_leaf',1,10,1)}
```

В целевой функции запускаем эксперимент и возвращаем его результат

In [39]:
```
def rf_objective(params):
    config={'series':"RF01"}
    for (key,value) in params.items():
        config["clfparams.{}".format(key)] = int(value)
    run = rfex.run(config_updates=config)
    return run.result
```

Начинаем оптимизацию

In [40]:
```
best = fmin(fn=rf_objective,
    space=space,
    algo=tpe.suggest,
    max_evals=20)
print(best)
```

```
INFO - hyperopt.tpe - tpe_transform took 0.001224 seconds
INFO - hyperopt.tpe - TPE using 0 trials
WARNING - root - Changed type of config entry "clfparams.max_features" from str to int
INFO - random_forest - Running command 'rf'
INFO - random_forest - Started
INFO - random_forest - Result: 0.4618138426460098
```

Sacred + hyperopt

Просматриваем результаты всех проверенных конфигураций

In [44]:
```
client = MongoClient()
db = client.telstra
runs = db.default.runs
```

In [57]:
```
def find():
    return runs.find({"config.series":"RF01","status":"COMPLETED"},sort = [('result',1)])
df = pd.DataFrame(0, index = np.arange(find().count()), columns = [])
for name in space.keys():
    df[name] = [r['config']['clfparams'][name] for r in find()]
df['result'] = [r['result'] for r in find()]
df = df.sort_values(by='result')
df.head()
```

Out[57]:

	max_features	min_samples_leaf	result
0	19	3	0.453707
1	19	3	0.453742
2	16	3	0.455899
3	13	2	0.456388
4	8	2	0.456891

Спасибо за внимание!

Ссылки:

- Описание тут: http://gereleth.github.io/Telstra-Network-Disruptions-Writeup/
- Код здесь: https://github.com/gereleth/kaggle-telstra
- “Global Refinement of Random Forest” article
- Rossmann neural net solution sharing thread