Доклад №1

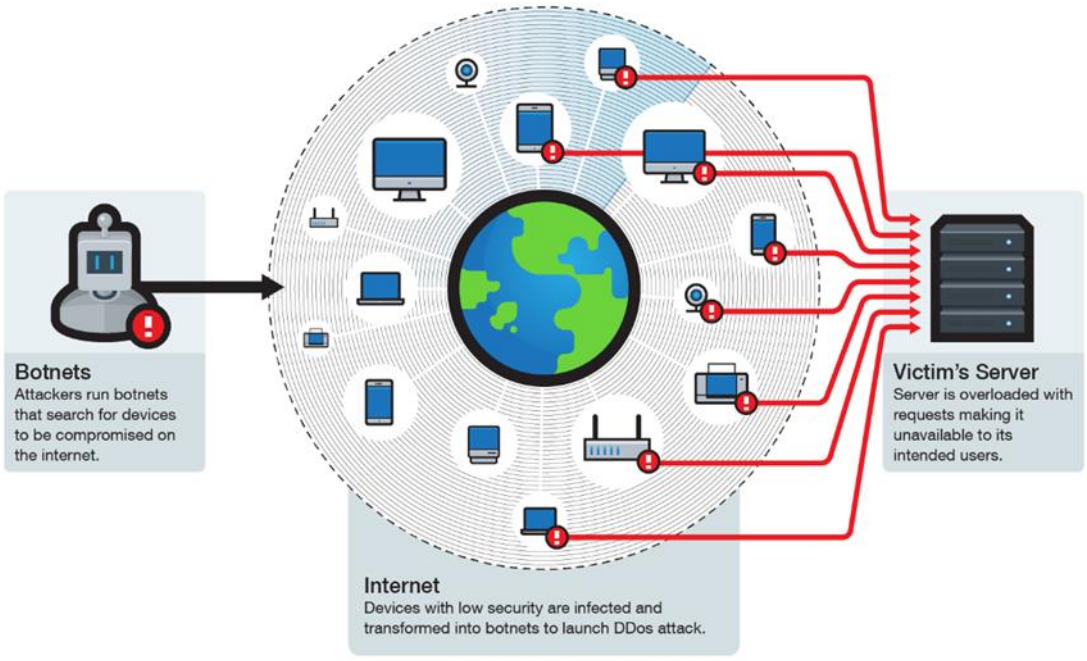
1. **Общая информация по данным**

Источник:
http://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT
Создатель: Yair Meidan
Дата загрузки: 2018-03-19
Задачи: Классификация, Кластеризация

Набор данных N-BaIoT был собран из реального сетевого трафика девяти устройств Интернета вещей.

Набор данных N-BaIoT содержит 115 атрибутов, все эти атрибуты представляют собой статистический анализ, который извлекается из пакетного трафика за различные периоды

Что такое ботнет?
Злоумышленники запускают бот-сети, которые выполняют поиск устройств со слабой защитой в сети Интернет, чтобы те могли быть заражены и трансформированы в ботов для осуществления DDos атак на сервер жертвы. Сервер переполняется запросами из-за чего становится недоступным для пользователей.



2. **Описание целевой задачи анализа**

Исходя из данных, было решено применить классификацию для того, что бы различить данные о доброкачественном и вредоносном трафике.
Вредоносные данные можно разделить на 2 вида атак, осуществляемых 2 сетями-ботов, а так же 1 класс «доброкачественных».

3. **Метаинформация**
   Формат: Многомерный, Последовательный
   Количество атрибутов: 115
   Типы атрибутов: числовые
   Количество строк: 7062606

Data Set Characteristics:	Multivariate, Sequential	**Number of Instances:**	7062606	**Area:**	Computer
Attribute Characteristics:	Real	**Number of Attributes:**	115	**Date Donated**	2018-03-19
Associated Tasks:	Classification, Clustering	**Missing Values?**	N/A	**Number of Web Hits:**	114042

4. **Ограничения данных**
   Пропущенных значений нет
5. **Предлагаемый ML алгоритм**

   Random forest
   Алгоритм машинного обучения, заключающийся в использовании ансамбля решающих деревьев.

   Каждое дерево в случайном лесу возвращает прогноз класса, и класс с наибольшим количеством голосов становится прогнозом леса.

   Имеет высокую точность предсказания

   Способен эффективно обрабатывать данные с большим числом признаков и классов.

   Один из немногих алгоритмов, который можно использовать в абсолютном большинстве задач.
6. **Необходимые настройки для алгоритма**
   Настройки алгоритма RandomForestClassifier будем использовать по умолчанию.
7. **Ожидаемая модель знаний**
   Модель будет предназначена для того, чтобы различать данные о трафике: доброкачественном, вредоносном. Способность предложенного нами метода заключается в обнаружении атак, когда они запускались с заражённых устройств IoT, которые были частью ботнета.
8. **Предлагаемые методы и критерии оценки построенных моделей**
   Accuracy-доля правильных ответов алгоритма
   Precision-доля объектов, определенных алгоритмом в конкретный класс, действительно относящихся к этому классу
   Recall-доля объектов конкретного класса, найденных алгоритмом, по отношению к общему количеству объектов этого класса

Доклад №2

1. **С чем работали и что хотели получить**
   Набор данных N-BaIoT был собран из реального сетевого трафика девяти устройств Интернета вещей.

   Набор данных N-BaIoT содержит 115 атрибутов, все эти атрибуты представляют собой статистический анализ, который извлекается из пакетного трафика за различные периоды.
   Random forest
   Исходя из данных, было решено применить классификацию для того, что бы различить данные о доброкачественном и вредоносном трафике.
   Вредоносные данные можно разделить на 2 вида атак, осуществляемых 2 сетями-ботов, а так же 1 класс «доброкачественных».
2. **Процесс анализа**
   - Чтение данных из файлов и запись в датафрейм
   - Изменение названий столбцов, содержащих точку
   - Добавление меток класса по названиям файла
   - Изменение типов данных на вещественный
   - Построение круговой диаграммы классов для определение несбалансированности данных
   - Устранение несбалансированности
   - Случайная перетасовка записей
   - Разделение на обучающие и тестовые данные
   - Обучение модели
   - Тестирование модели
   - Оценка модели
3. **Настройки/преобразования данных**
   Функция загрузки данных (на примере доброкачественных)

```
def load_data_class(PATH, class_name):
    if class_name=='benign':
        for i in range(9):
            directory=PATH+str(i+1)+".benign.csv"
            data = sc.textFile(directory)
            numbers_rdd = data.map(lambda s : list(map(str, s.split(","))))
            #выбираем названия столбцов из рдд
            columns=numbers_rdd.take(115)
            df = numbers_rdd.toDF(columns[0])
            #убираем первую строку из датафрейма (дублируются названия столбцов)
            df=df.filter(df.MI_dir_L5_weight!="MI_dir_L5_weight")
            #убираем "." из названий столбцов, спарк не распознает их
            df=remove_decimal_points_colname(df)
            #записываем метку класса
            df=df.withColumn('label', lit('1'))
            #переводим данные во float
            df = df.select(*(col(c).cast("float").alias(c) for c in df.columns))
            if i==0:
                dff=df
            else:
                dff=dff.union(df)
        return dff
```

Функция удаления точек

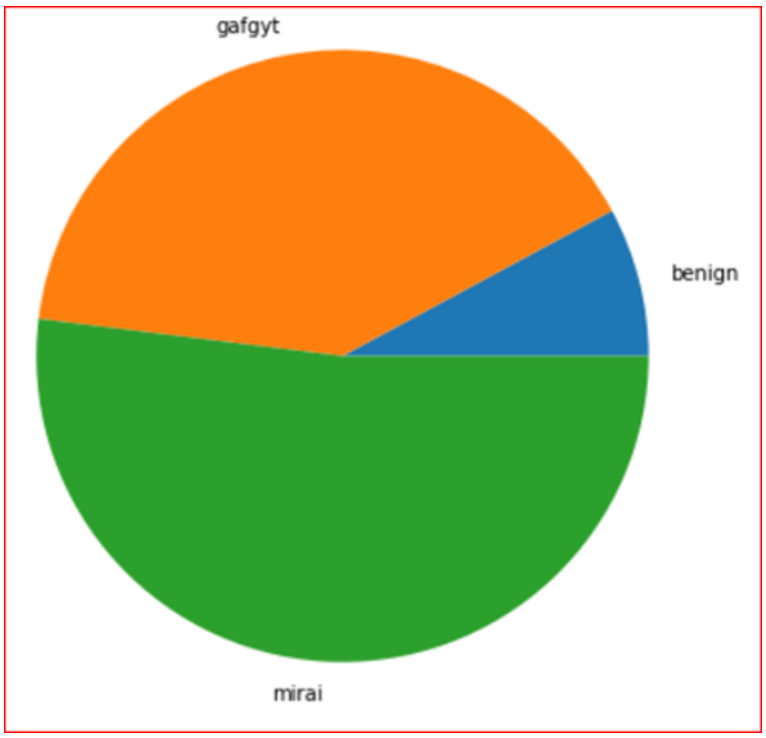
```
def remove_decimal_points_colname(dataset_df):
    #создаём список
    input_feature_list = []
    #возвращаем имена всех столбцов в виде списка
    columns=dataset_df.columns
    for col in columns:
        #убираем точки
        new_name = col.replace('.','')
        #добавляем строки
        input_feature_list.append(new_name)
        #возвращает новый объект DataFrame
    dataset_df = dataset_df.toDF(*input_feature_list)
    return dataset_df
```

4. Настройки функций

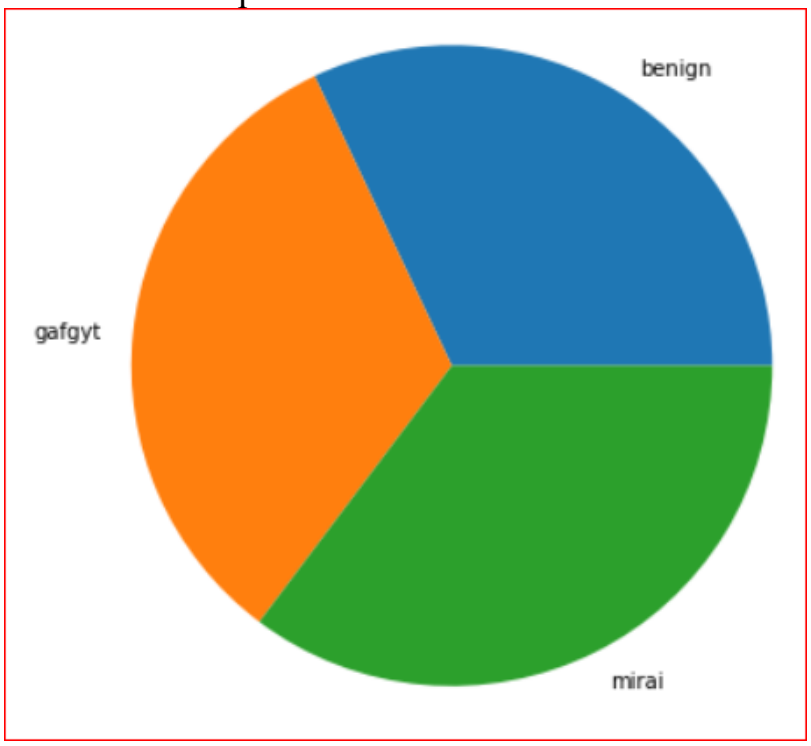
```
#!spark --session gr0371-sk6
def under_sample(major_df,major_df_count,major_df2,major_df_count2,minor_df,minor_df_count):
    ratio=int(major_df_count/minor_df_count)
    sampled_maj_df = major_df.sample(False, 1/ratio)
    ratio=int(major_df_count2/minor_df_count)
    sampled_maj_df2 = major_df2.sample(False, 1/ratio)
    return sampled_maj_df,sampled_maj_df2, minor_df

def remove_class_imbalance(benign_df, benign_count, gafgyt_df, gafgyt_count, mirai_df, mirai_count):
    if gafgyt_count < benign_count and   gafgyt_count < mirai_count:
        benign_df, mirai_df, gafgyt_df = under_sample(benign_df, benign_count, mirai_df, mirai_count, gafgyt_df, gafgyt_count)
        return benign_df, mirai_df, gafgyt_df
    elif benign_count< gafgyt_count and  benign_count < mirai_count:
        gafgyt_df, mirai_df, benign_df=under_sample(gafgyt_df, gafgyt_count, mirai_df, mirai_count,benign_df, benign_count)
        return benign_df, mirai_df, gafgyt_df
    elif mirai_count < benign_count and mirai_count < gafgyt_count:
        gafgyt_df, benign_df, mirai_df=under_sample(gafgyt_df, gafgyt_count, benign_df, benign_count, mirai_df,mirai_count)
        return benign_df, mirai_df, gafgyt_df
```

Функция балансировки классов с помощью undersample

До балансировки:



После балансировки:




```
#!spark --session gr0371-sk6
from pyspark.sql.functions import rand
def random_shuffle(df):
    return df.orderBy(rand())
```

Случайная перетасовка датафрейма, чтобы сделать данные более репрезентативными для каждого класса

```
cols = df.columns
cols.remove('label')
va = VectorAssembler(
    inputCols=cols,
    outputCol="features"
).transform(df)
train, test = va.randomSplit([0.7, 0.3])
```

Запись датафрейма в Vector Assembler и разделение на обучающие и тестовые данные в соотношении 70 на 30

5. **Настройки алгоритма**

```
rf = RandomForestClassifier(
    labelCol="label",
    featuresCol="features",
    maxDepth=4,
    numTrees=5)
model = rf.fit(train)
predictions=model.transform(test)
```

6. **Полученные результаты**

```
#!spark --session gr0371-sk6
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction")
accuracy = evaluator.evaluate(predictions, {evaluator.metricName: "accuracy"})
print(('Accuracy is: %s') % accuracy)
```

```
Accuracy is: 0.9914646879534444
```

```
#!spark --session gr0371-sk6
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction")
precision = evaluator.evaluate(predictions, {evaluator.metricName: "weightedPrecision"})
print(('Precision is: %s') % precision)
```

```
Precision is: 0.9913015549889204
```

```
#!spark --session gr0371-sk6
recall = evaluator.evaluate(predictions, {evaluator.metricName: "weightedRecall"})
print(('Recall is: %s') % recall)
```

```
Recall is: 0.9910830641554916
```

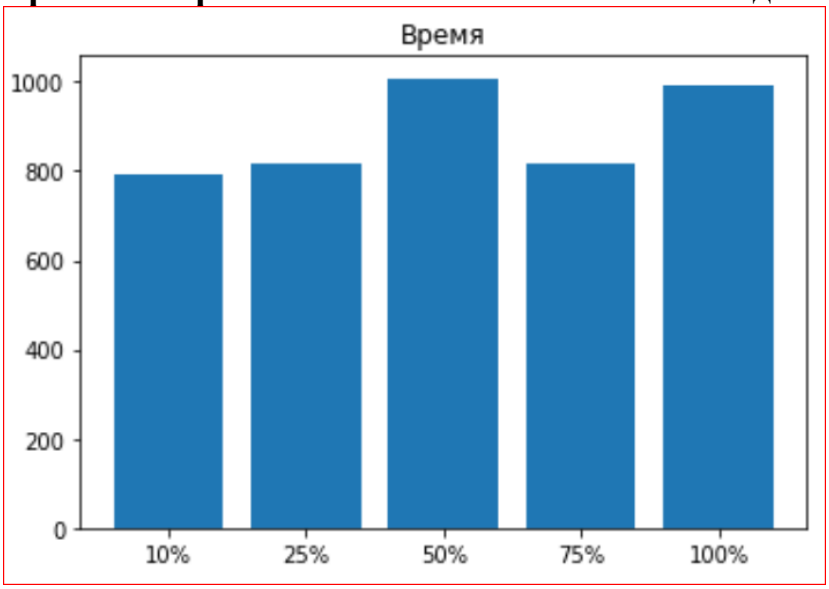
7. **Вывод в соответствии с поставленной задачей**

Модель различает данные о трафике: доброкачественном, вредоносном. Способность разработанного нами метода заключается

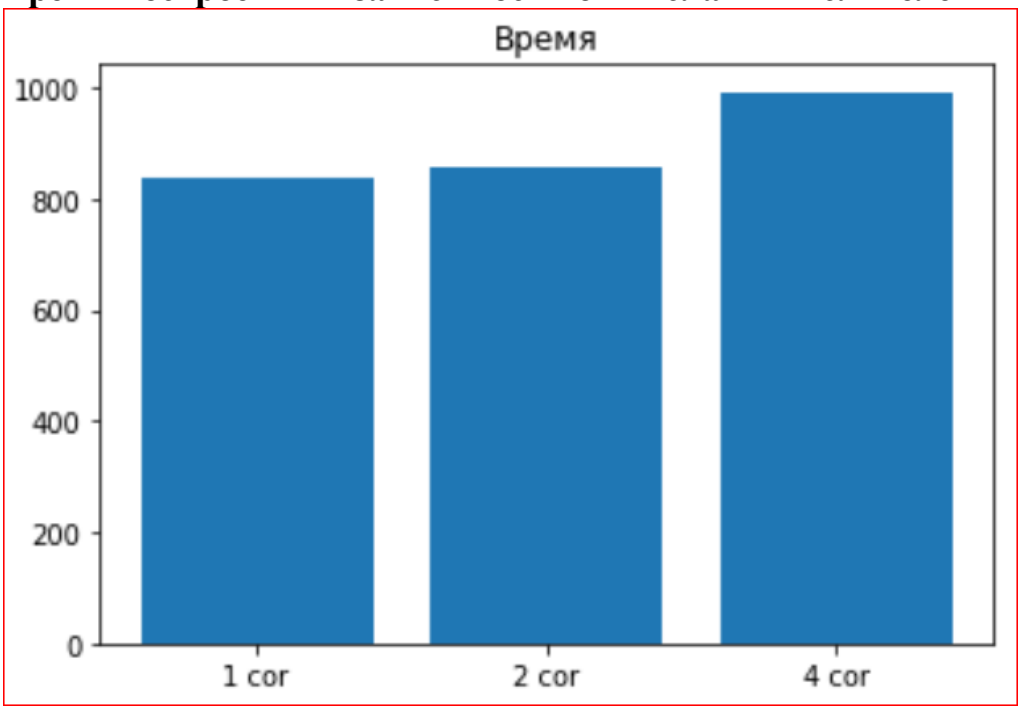
в обнаружении атак, когда они запускались с заражённых устройств IoT, которые были частью ботнета.

Доклад №3

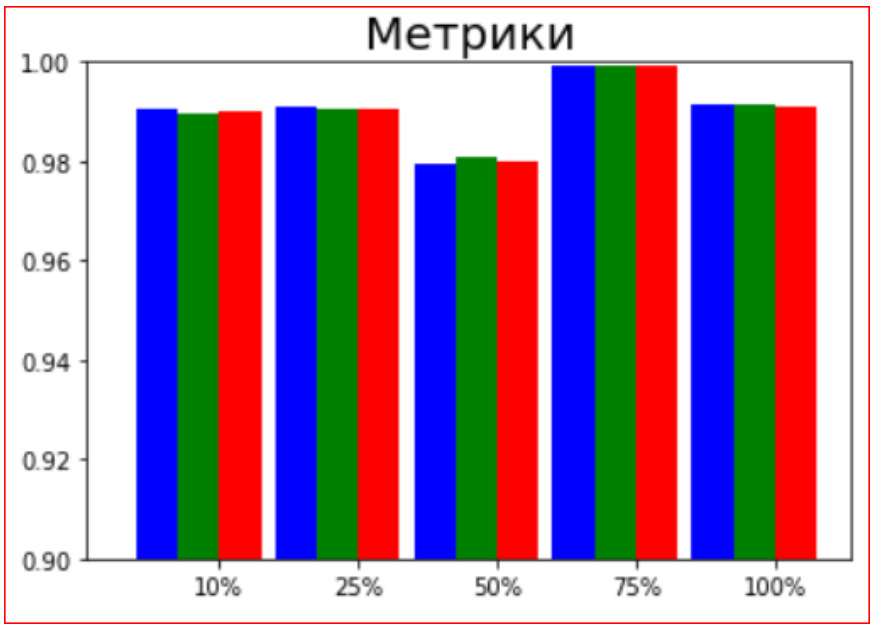
1. **Время построения в зависимости от объема данных**



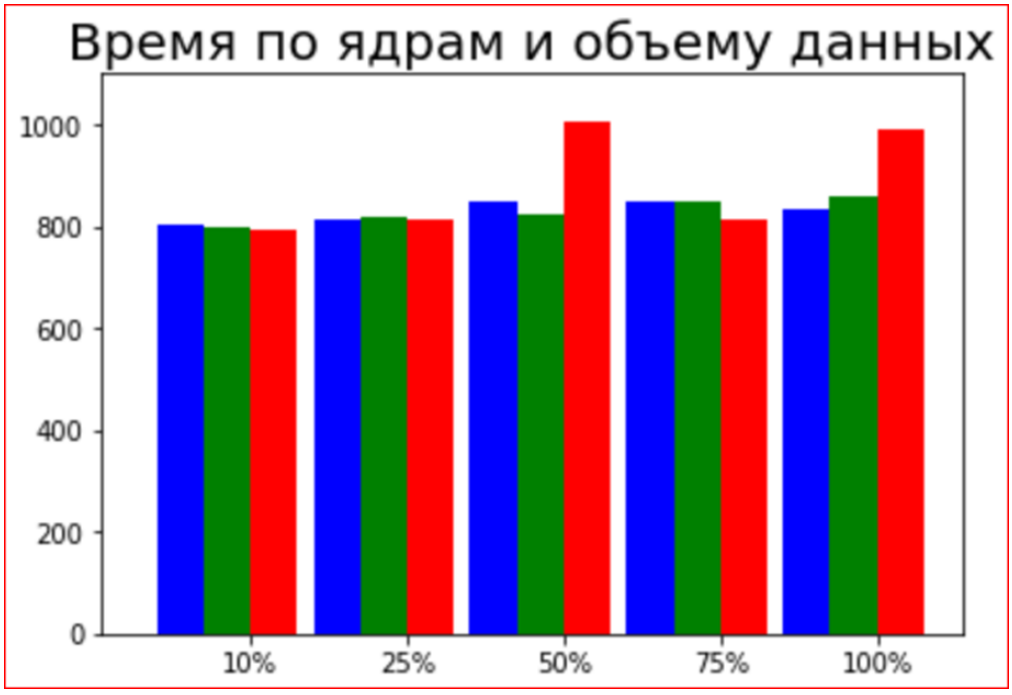
2. **Время построения в зависимости от числа вычислителей**



3. **Построенные модели и их оценки**
   Оценки модели в зависимости от объема данных (в % от исходного датасета)



Accuracy
Precision
Recall



1 ядро
2 ядра
4 ядра

4. Вывод:

Наиболее точно алгоритм работает на 75% данных. При этом такая выборка обеспечивает наибольшую скорость обучения.