Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Кафедра информатики

Лабораторная работа №10

«Реализация криптографических атак с помощью машинного обучения на физически неклонируемые функции»

Выполнил: Чёрный Родион Павлович
магистрант кафедры информатики
группа №858642

Проверил: доцент кафедры информатики
Стержанов Максим Валерьевич

Минск 2019

Постановка задачи

1. Сформулируйте задачу в терминах машинного обучения.
2. Обучите модель, которая могла бы предсказывать ответы по запросам, которых нет в обучающей выборке.
3. Применить как минимум 3 различных алгоритма (например, метод опорных векторов, логистическая регрессия и градиентный бустинг).
4. Какая метрика наиболее подходит для оценки качества алгоритма?
5. Какой наибольшей доли правильных ответов (Accuracy) удалось достичь?
6. Какой размер обучающей выборки необходим, чтобы достигнуть доли правильных ответов минимум 0.95?
7. Как зависит доля правильных ответов от N?
8. Ответы на вопросы представьте в виде графиков.

Описание реализации

1. На вход модели подается вектор длины N, состоящий из случайных значений 0 и 1. На выходе - такое же бинарное значение: 0 или 1. Необходимо, на основании обучающего набора данных, а также знаний внутренней структуры ФНФ типа арбитр, научиться предсказывать выходное значение ФНФ.

2. Реализация симуляции ФНФ типа арбитр:

```
class Stage:
    delay_out_a = 0.
    delay_out_b = 0.
    selector = 0
    def __init__(self, delay_a, delay_b):
        self.delay_out_a = delay_a
        self.delay_out_b = delay_b
    def set_selector(self,s):
        self.selector = s
    def get_output(self, delay_in_a, delay_in_b):
        if self.selector == 0:
            return (delay_in_a + self.delay_out_a,
                    delay_in_b + self.delay_out_b)
        else:
            return (delay_in_b + self.delay_out_a,
                    delay_in_a + self.delay_out_b)
class ArbiterPUF:
    def __init__(self, n):
        self.stages = []
        for _ in range(n):
            self.stages.append(Stage(random.random(), random.random()))
    def get_output(self, chall):
        self._set_challenge(chall)
        delay = self._compute_output()
        if delay[0] < delay[1]:
            return 0
        else:
            return 1
    def _set_challenge(self, chall):
        for stage, bit in zip(self.stages, chall):
            stage.set_selector(bit)
    def _compute_output(self):
        delay = (0, 0)
        for s in self.stages:
            delay = s.get_output(delay[0], delay[1])
        return delay
```

Сгенерируем обучающую и контрольную выборку:

```
learningX = [[random.choice([0, 1]) for _ in range(N)] for _ in range(LS)]
learningY = [apuf.get_output(chall) for chall in learningX]

testingX = [[random.choice([0, 1]) for _ in range(N)] for _ in range(TS)]
```

```
testingY = [apuf.get_output(chall) for chall in testingX]

learningX = [convert_into_features_vector(c) for c in learningX]
testingX = [convert_into_features_vector(c) for c in testingX]
```

3. Обучим алгоритм логистической регрессии, метод опорных векторов и алгоритм градиентного бустинга на исходных данных. Размер обучающей выборки возьмем равным 600.

```
lr = LogisticRegression()
lr.fit(learningX, learningY)
svc = SVC()
svc.fit(learningX, learningY)
gb = GradientBoostingClassifier()
gb.fit(learningX, learningY)
```

4, 5. Проверим каждую из моделей на контрольной выборке. Результаты сравнения приведены в таблице 1.

Алгоритм	**Оценка качества модели**
Линейная регрессия	0.968400
Метод опорных векторов	0.923500
Градиентный бустинг	0.868600

Как видим, лучший результат показывает линейная регрессия. При размере обучающей выборки в 600 элементов алгоритм показывает точность предсказаний > 0.95

6,7. Проверим, как меняется точность предсказаний от размера обучающей выборки. График зависимости приведен на рисунке 1.

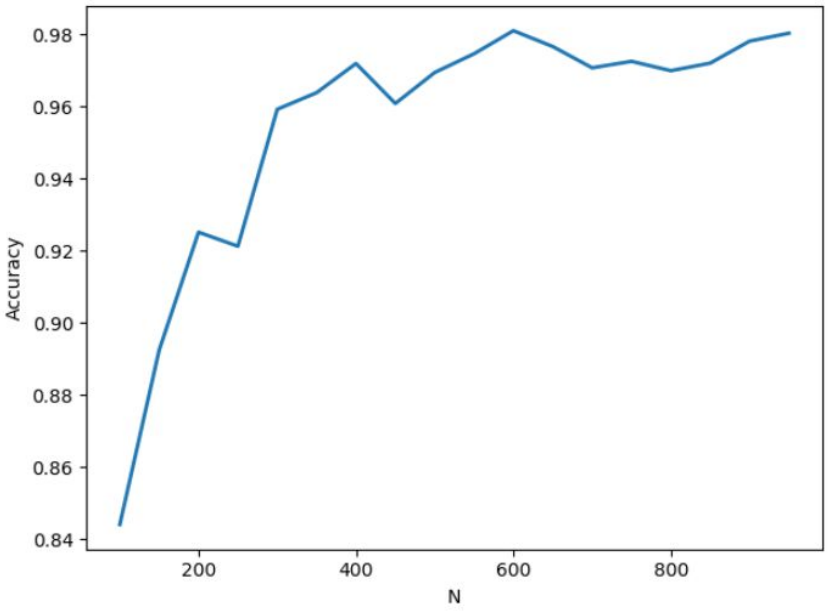

Рисунок 1 - График зависимости качества модели от размера обучающей выборки

Из графика видно, что минимальный размер обучающей выборки для достижения доли правильных ответов в 0.95 это примерно 300 элементов.