Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

|  |
| --- |
|  |

**ЛабоРАТОРНАЯ РАБОТА №11**

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

|  |  |  |
| --- | --- | --- |
| Студент |  | К. Д. Зюсько |
| Преподаватель |  | М. В. Стержанов |

Минск 2019

ХОД РАБОТЫ

**Задание.**

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.

**Результат выполнения:**

1. Данная задача является задачей бинарной классификации, поскольку ответы представляют собой бинарное значение: 0 или 1. Входной вектор (challenge) представляет из себя битовый вектор заранее определённой последовательности.

1. Код реализации:

X, y = load\_data(32)

X\_der = der\_challenge(X)

np.unique(y, return\_counts=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_der, y)

lr\_model = LogisticRegression()

lr\_model.fit(X\_train, y\_train)

y\_test\_predicted = lr\_model.predict(X\_test)

print('Accuracy: ', accuracy\_score(y\_test, y\_test\_predicted))

print('F1: ', f1\_score(y\_test, y\_test\_predicted))

Результат выполнения:

Accuracy: 0.9922666666666666

F1: 0.992844806316309

3. Код реализации:

def get\_model\_errors(X\_train, y\_train, X\_test, y\_test):

bit\_depth = X\_train.shape[1]

lr\_model = LogisticRegression()

svm\_model = SVC()

nn\_model = MLPClassifier(

solver='lbfgs',

alpha=1e-5,

hidden\_layer\_sizes=(int(bit\_depth \* 2), int(bit\_depth \* 1.5)),

random\_state=1

)

lr\_model.fit(X\_train, y\_train)

svm\_model.fit(X\_train, y\_train)

nn\_model.fit(X\_train, y\_train)

errors = [

accuracy\_score(y\_test, lr\_model.predict(X\_test)),

accuracy\_score(y\_test, svm\_model.predict(X\_test)),

accuracy\_score(y\_test, nn\_model.predict(X\_test))

]

return np.array(errors)

4. Ответ:

В качестве метрик можно использовать как accuracy, так и F1-score. Но, поскольку, во следующих пунктах используется accuracy, то в дальнейшем будет использоваться именно эта метрика.

5. Ответ:

Accuracy зависит от разрядности (количества бит) в challenge. Так можно заметить, что при разрядности в 8 бит все алгоритмы выдают accuracy ~1.0-0.99. Но с ростом разрядности challenge этот процент проседает (для разных алгоритмов он также становиться различным), и лучший коэффициент был получен с логистической регрессией и составил 0.97.

6. Графики приведены ниже:

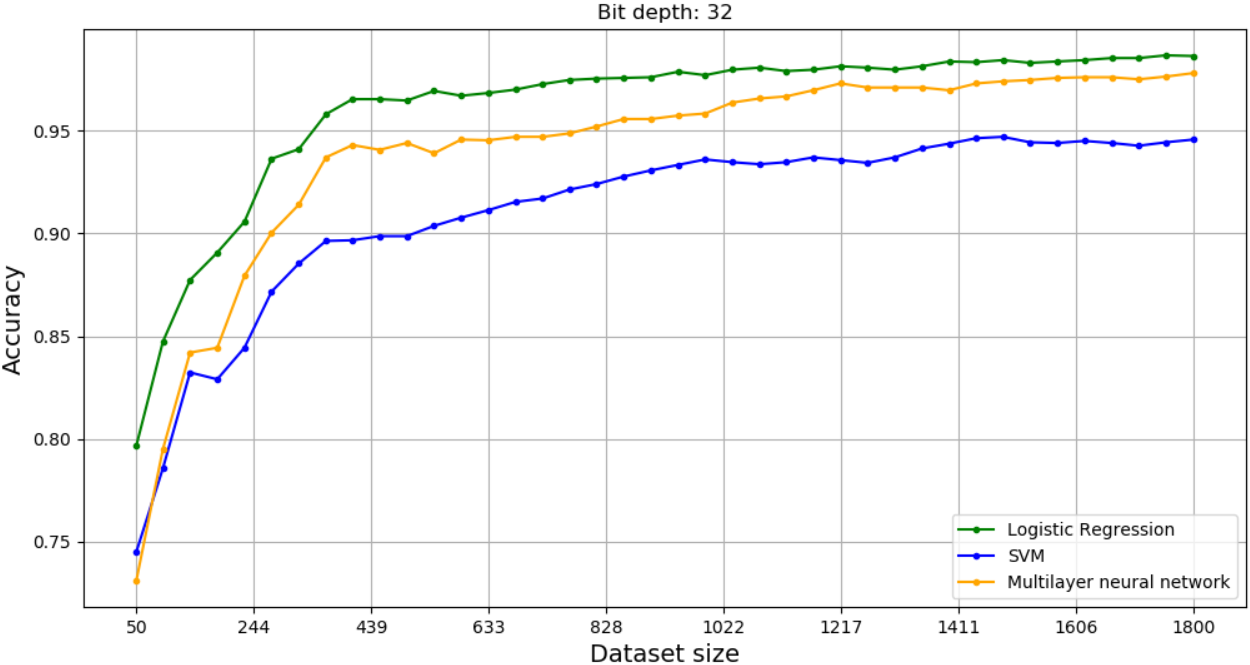


Рисунок 1 – график зависимости точности от размера обучаемой выборки при N=32

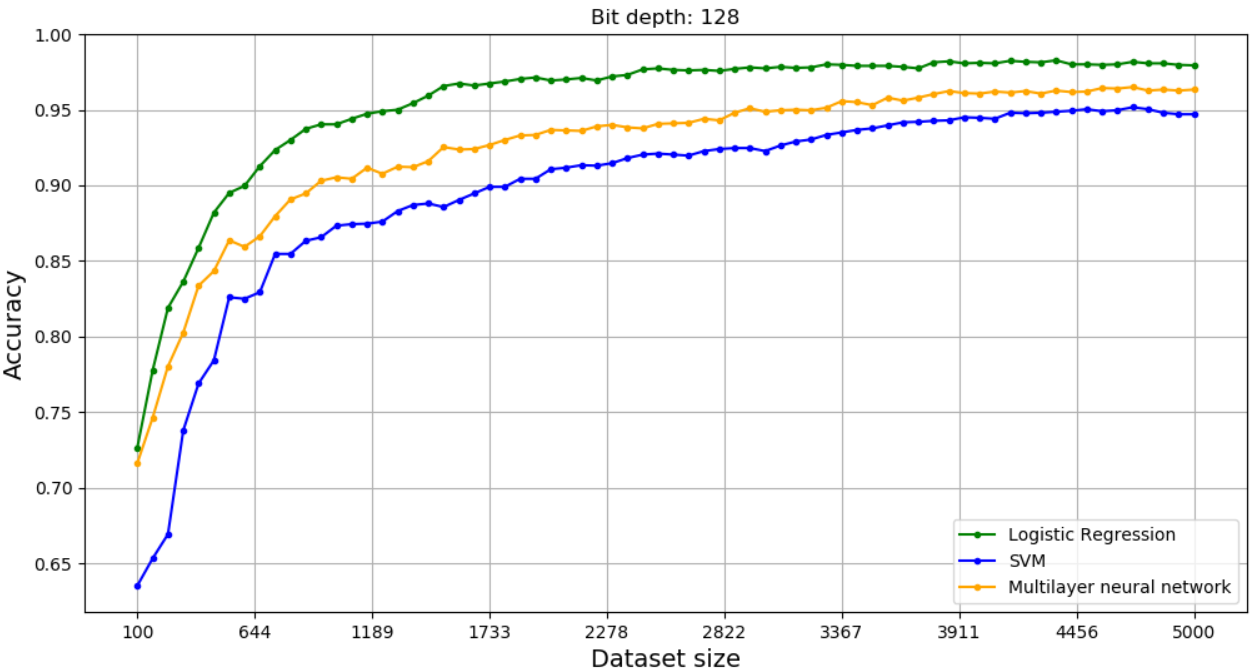


Рисунок 2 – график зависимости точности от размера обучаемой выборки при N=128

Данные, представленные в табличном виде:

Таблица 1 – размер выборок в зависимости от N для каждого алгоритма, для достижения точности > 95%

|  |  |  |
| --- | --- | --- |
| Алгоритм | N | Размер выборки |
| Логистическая регрессия | 32 | ~300 |
| 128 | ~1200 |
| Метод опорных векторов | 32 | ~1930 |
| 128 | ~4750 |
| Многослойная нейронная сеть | 32 | ~800 |
| 128 | ~2900 |

7-8. График приведён ниже:

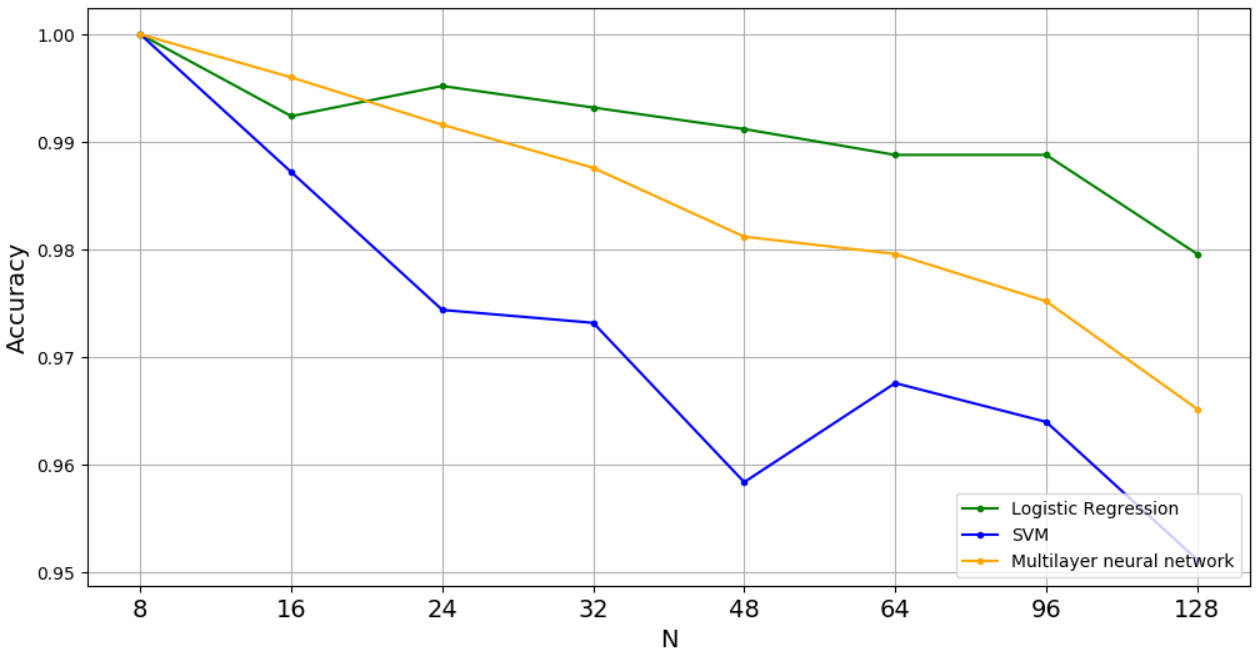
****

Рисунок 3 – зависимость точности от N**Программный код:**

from sklearn.neural\_network import MLPClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import f1\_score, accuracy\_score

from sklearn.svm import SVC

import matplotlib.pyplot as plt

import os

import numpy as np

import warnings

warnings.filterwarnings("ignore", category=FutureWarning)

def load\_data(bit\_depth, row\_limit = 15000):

ch = []

resp = []

with open(os.path.join(os.path.dirname(\_\_file\_\_), 'data', f'Base{bit\_depth}.txt'), 'r') as fp:

for idx, line in enumerate(fp):

if idx >= row\_limit:

break

data = line.strip().split(' ')

ch.append(np.asarray(list(data[0]), dtype=np.int8))

resp.append(np.asarray(data[1], dtype=np.int8))

X = np.asarray(ch)

y = np.array(resp)

return X, y

def der\_challenge(challenges):

challenges\_der = np.zeros(challenges.shape)

challenges = 1 - 2 \* challenges

for i in range(len(challenges)):

challenge = challenges[i]

challenges\_der[i][0] = challenge[0]

for j in range(1, len(challenge)):

challenges\_der[i][j] = challenges\_der[i][j-1] \* challenge[j]

return challenges\_der

X, y = load\_data(32)

X\_der = der\_challenge(X)

np.unique(y, return\_counts=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_der, y)

lr\_model = LogisticRegression()

lr\_model.fit(X\_train, y\_train)

y\_test\_predicted = lr\_model.predict(X\_test)

print('Accuracy: ', accuracy\_score(y\_test, y\_test\_predicted))

print('F1: ', f1\_score(y\_test, y\_test\_predicted))

def get\_model\_errors(X\_train, y\_train, X\_test, y\_test):

bit\_depth = X\_train.shape[1]

lr\_model = LogisticRegression()

svm\_model = SVC()

nn\_model = MLPClassifier(

solver='lbfgs',

alpha=1e-5,

hidden\_layer\_sizes=(int(bit\_depth \* 2), int(bit\_depth \* 1.5)),

random\_state=1

)

lr\_model.fit(X\_train, y\_train)

svm\_model.fit(X\_train, y\_train)

nn\_model.fit(X\_train, y\_train)

errors = [

accuracy\_score(y\_test, lr\_model.predict(X\_test)),

accuracy\_score(y\_test, svm\_model.predict(X\_test)),

accuracy\_score(y\_test, nn\_model.predict(X\_test))

]

return np.array(errors)

def show\_data\_count\_dependency(bit\_depth, data\_sizes):

X, y = load\_data(bit\_depth)

X\_der = der\_challenge(X)

accuracy\_data = []

for training\_size in data\_sizes:

training\_size = int(training\_size)

X\_train = X\_der[:training\_size]

y\_train = y[:training\_size]

X\_test = X\_der[training\_size:training\_size + 3000]

y\_test = y[training\_size:training\_size + 3000]

errors = get\_model\_errors(X\_train, y\_train, X\_test, y\_test)

accuracy\_data.append(np.concatenate([np.array([training\_size]), errors], axis=None))

accuracy\_data = np.array(accuracy\_data)

plt.figure(figsize=(12,6))

plt.plot(accuracy\_data[:,0], accuracy\_data[:,1], marker='.', color='green', label='Logistic Regression')

plt.plot(accuracy\_data[:,0], accuracy\_data[:,2], marker='.', color='blue', label='SVM')

plt.plot(accuracy\_data[:,0], accuracy\_data[:,3], marker='.', color='orange', label='Multilayer neural network')

plt.title(f'Bit depth: {bit\_depth}')

plt.xticks(np.linspace(np.min(data\_sizes), np.max(data\_sizes), 10), fontsize=10)

plt.xlabel('Dataset size', fontsize=14)

plt.ylabel('Accuracy', fontsize=14)

plt.legend(loc='lower right')

plt.grid(True)

plt.show()

show\_data\_count\_dependency(32, np.linspace(50, 1800, 40))

show\_data\_count\_dependency(128, np.linspace(100, 5000, 70))

def show\_N\_count\_dependency(files):

accuracy\_data = []

for file in files:

X, y = load\_data(file)

X\_der = der\_challenge(X)

training\_size = 10000

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_der[:training\_size], y[:training\_size])

errors = get\_model\_errors(X\_train, y\_train, X\_test, y\_test)

accuracy\_data.append(errors)

accuracy\_data = np.array(accuracy\_data)

plt.figure(figsize=(12,6))

x = np.linspace(1, len(files), len(files))

plt.plot(x, accuracy\_data[:,0], marker='.', color='green', label='Logistic Regression')

plt.plot(x, accuracy\_data[:,1], marker='.', color='blue', label='SVM')

plt.plot(x, accuracy\_data[:,2], marker='.', color='orange', label='Multilayer neural network')

plt.xticks(x,files, fontsize=14)

plt.xlabel('N', fontsize=14)

plt.ylabel('Accuracy', fontsize=14)

plt.legend(loc='lower right')

plt.grid(True)

plt.show()

files = [8, 16, 24, 32, 48, 64, 96, 128]

show\_N\_count\_dependency(files)