Министерство науки и высшего образования РФ

Томский государственный университет
систем управления и радиоэлектроники

Новохрестова Д.И. Лаптев П.Ю.

Безопасность информационно аналитических систем: Указания к лабораторным работам

Учебно-методическое пособие к лабораторным работам
для студентов направлений подготовки входящих в укрупненные группы
специальностей и направлений 09.00.00 и 10.00.00 всех уровней образования
(бакалавриат, магистратура, специалитет)

УДК 004.8

ББК 32.813.5

Н 50

Новохрестова, Д. И. Безопасность информационно аналитических систем: Учебно-методическое пособие к лабораторным работам [Электронный ресурс] / Д. И. Новохрестова, П. Ю. Лаптев. — Томск: ТУСУР, 2025. — 50 с.

Настоящие методические указания содержит описания лабораторных работ по дисциплине «Безопасность информационно аналитических систем» для направлений подготовки входящих в укрупненные группы специальностей и направлений 09.00.00 и 10.00.00 всех уровней образования (бакалавриат, магистратура, специалитет).

Одобрено на заседании кафедры КИБЭВС протокол №7 от 30.08.2024 года

УДК 004.8
ББК 32.813.5

Введение

В данном методическом пособии представлены лабораторные работы, которые направлены на закрепление полученных знаний в области безопасности информационно аналитических систем, а именно в задаче защиты наборов данных от атак на них.

Целью является развитие у студентов практических навыков в реализации атак отравления и состязательных атак на наборы данных, представленных в формате изображений, аудио записей и числовых рядов, а также в получении навыков и знаний по защите от использованных методов атак.

ЛАБОРАТОРНАЯ РАБОТА № 1
ОТРАВЛЕНИЕ НАБОРОВ ДАННЫХ

Целью работы является ознакомление с методологией отравления наборов данных, используя набор данных MNIST.

Задачи

1. Обучить модель нейронной сети на чистых данных MNIST.
2. Провести отравление набора данных для тестовой и обучающей выборок.
3. Протестировать исходную модель на отравленной тестовой выборке.
4. Обучить новую модель на отравленной обучающей выборке.
5. Оформить отчет о проделанной работе согласно ОС ТУСУР 01-2021.

Критерии Оценки

3 балла: Атака на основе пары цифр.

4 балла: Атака на основе трех атакуемых цифр и одной атакующей.

5 баллов: Атака на основе трех атакуемых цифр, похожих на атакующую.

Краткие теоретические сведения

Отравление данных — это один из видов атак на системы машинного обучения, при котором злоумышленник намеренно вводит ложные или искаженные данные в обучающую выборку с целью ухудшить качество модели или заставить её принимать неверные решения.

Виды отравления данных:

– Отравление бэкдором – бэкдор-атаки предполагают внедрение уязвимости, которая будет служить «бэкдором» для злоумышленника. Затем точка доступа используется для манипулирования производительностью и выходными данными модели.

– Отравление бэкдором может быть как целевой, так и нецелевой атакой в зависимости от конкретных целей злоумышленника.

– Атака на доступность и снижение модели – это тип кибератаки, которая пытается нарушить работу системы или сервиса путем заражения его данных. Например, злоумышленники с помощью «отравления данными» заставляют систему выдавать ложноположительные или отрицательные результаты.

В качестве примера атаки на качество модели Сафиуллин приводит случай с чат-ботом Tay от Microsoft. Tay, разработанный для взаимодействия с пользователями Twitter с помощью машинного обучения, попал под скоординированную атаку. Злоумышленники публиковали оскорбительные сообщения, на которые Tay начал генерировать схожий подстрекательский контент. Через 24 часа Microsoft отключила бота и извинилась перед пользователями.

– Атака с инверсией модели – атака с инверсией модели использует ответы модели (ее выходные данные) для воссоздания набора данных или генерации предположений (ее входных данных). При этом типе атаки злоумышленником чаще всего является сотрудник или другой одобренный пользователь системы, поскольку для отравления данных нужен доступ к данным модели.

– Скрытая атака — это особенно тонкая форма отравления данными, при которой злоумышленник медленно редактирует набор данных или внедряет компрометирующую информацию.

Со временем совокупный эффект от этой деятельности может привести к искажениям в модели, которые повлияют на ее общую точность. Поскольку эти атаки проводятся «незаметно», отследить или исправить проблему довольно трудно.

Методы атаки:

– Токсичные данные: вводятся данные, которые, как известно, вводят в заблуждение модель (например, непривычные паттерны).

– Атакующие экземпляры: вводятся точки данных, которые близки к границе классификации, чтобы изменить решающие границы.

Эффекты отравления данных:

– Потеря точности модели.

– Увеличение числа ложных срабатываний или пропусков критически важных событий.

– Ухудшение модели на определённых поднаборах данных.

Требования к выполнению лабораторно работы

– Ознакомиться с методическими указаниями, а также представленными примерами в блокноте Google Colab.

– Выполнить индивидуальное задание, проведя отравления набора данных для тестовой и обучающей выборок, протестировав исходной модели на отравленной тестовой выборке и проведя обучение новой модели на отравленной обучающей выборке.

– Ответить на контрольные вопросы.

1 ВЫБОР ВАРИАНТА

На основе ФИО выдается случайный вариант с помощью кода представленного на рисунке 1.

```
# Выбор варианта
fio = 'Иванов Иван Иванович'  # ФИО - основа генератора
fio_bytes = fio.encode('utf-8')  # Привели в байты
fvar = int.from_bytes(fio_bytes, 'big')  # Привели в число
varseed = fvar % 1000000000  # % - остаток от деления - ядро генератора

# Генерация номера варианта с увеличенным разбросом
number = fvar % 10  # Получили номер варианта
random.seed(varseed)  # Установили сид для генерации случайных чисел
target = random.randint(0, 9)  # Генерируем случайное число от 0 до 9

# Убедимся, что target не равен number
while target == number:
    target = random.randint(0, 9)

print("Провести отравление - данные по цифре", number, "попытаться выдать за данные по цифре", target)
```

Рисунок 1 – Выбор варианта20

2 ОБУЧЕНИЕ НА ЧИСТЫХ ДАННЫХ

В качестве набора данных будет использован набор MNIST, содержащий изображения рукописных цифр. Загрузить MNIST в формате обучающей и тестовой выборки можно с помощью библиотеки Tensorflow. Также после загрузки набора данных, необходимо провести нормализацию данных, приведя все значения пикселей к диапазону от 0 до 1. Код загрузки и нормализации набора данных представлен на рисунке 2.

```
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
```

Рисунок 2 – Загрузка данных MNIST

Для решения задачи классификации рукописных цифр будет использоваться простая полносвязная нейронная сеть, состоящая из трёх слоёв: входной слой, скрытый слой со 128 нейронами, выходной слой с 10 нейронами. В качестве входного слоя используется слой Flatten, преобразующий многомерные массивы в вектор значений. В качестве оптимизатора сети используется Adam, а в качестве функции потерь – категориальная кросс-энтропия. Запуск обучения выполняется с помощью функции *fit* объекта построенной модели, с указанием обучающих данных, валидационных данных, а также параметров обучения – в данном случае только количество эпох, равное 10. Код создания и обучения модели нейронной сети представлен на рисунке 3.

```
model_mnist = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),  # Преобразование 2D в 1D
  tf.keras.layers.Dense(128, activation='relu'),  # Скрытый слой
  tf.keras.layers.Dense(10, activation='softmax') # Выходной слой
])

model_mnist.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'],
)

history_mnist = model_mnist.fit(train_images, train_labels, epochs=10, validation_data=
(test_images, test_labels))
```

Рисунок 3 – Построение и обучение модели

3 ОТРАВЛЕНИЕ ДАННЫХ

В зависимости от варианта необходимо повести отравление данных по определенной цифре или цифрам и попытаться выдать её/их за другую. Атака заключается в замене метки цифр, чтобы обмануть модель нейронной сети и заставить её делать ошибки при классификации. Для проведения атаки будет использоваться собственная функция poison_attack, которую необходимо реализовать таким образом, чтобы на вход функция получала массив меток, атакуемые цифры, атакующую цифру и объём замены, а на выходе с функции возвращался новый массив отравленных данных (рисунок 4). При необходимости можно изменить структуру подаваемых или возвращаемых данных. Выбор атакующей и атакуемой цифр осуществляется автоматически при генерации варианта. В случае выполнении атак на несколько цифр необходимо помимо выданной атакуемой цифры сгенерировать ещё две атакуемых цифры случайным образом (оценка 4), либо же выбрать самостоятельно две дополнительные атакуемые цифры, которые имеют схожее написание с исходной атакуемой цифрой.

```
def poison_attack(labels, preys, attacker, volume_of_poison=50):
  # Реализуйте функцию отравления данных
  # Функция должна возвращать новый массив numpy, где атакуемые цифры заменины
  # на атакующие в соответствии с указанным объёмом замены volume_of_poison

  # При необходимости можно изменять подаваемые переменные, или формат возвращения данных
  return poisoned
```

Рисунок 4 – Реализация отравления данных

Пример использования функции отравления с предложенной структурой входных и выходных данных представлен на рисунке 5, а на рисунке 6 представлен результат проведения отравления с атакуемой цифрой 7 и атакующей цифрой 8.

```
# Объём отравления в диапазоне 0-1, где 0 - 0%, 1 - 100%
vol_pois=1

# Реализуйтие перевод объёма отравления из процентного соотношения к количеству записей
vol_pois=...
print('volume of poisoned data',vol_pois)

# Отравление тестовой выборки
poisoned_test=poison_attack(test_labels, preys, attacker, vol_pois)
print(poisoned_test.shape)
```

Рисунок 5 – Отравление тестовой выборки

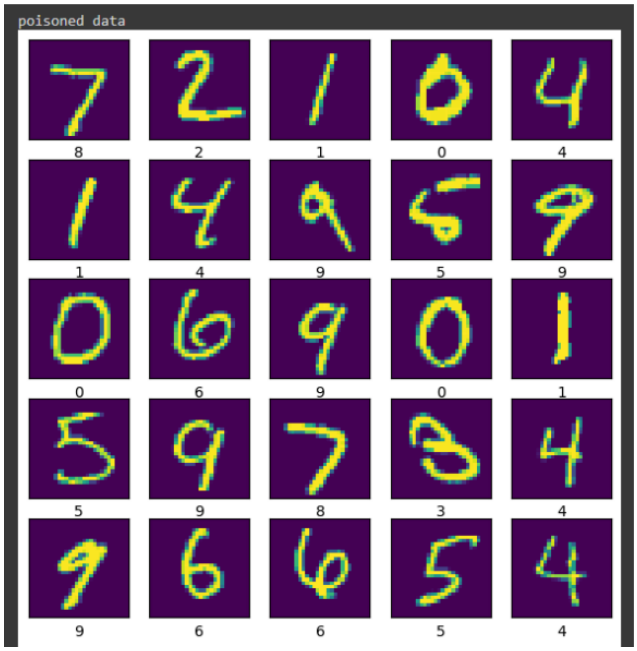

Рисунок 6 – Отравленные данные

4 ОЦЕНКА КАЧЕСТВА РАБОТЫ МОДЕЛИ

Для оценки качества работы модели можно воспользоваться разным функционалом. Например, можно провести проверку модели с помощью функции *evaluate*, подав на вход функции тестовые данные и метки тестовых данных. В данном случае будет выведена метрика качества, заданная при построении модели.

Также можно провести оценку качества модели используя подробный отчёт классификации библиотеки SciKit-learn. Для этого необходимо воспользоваться функцией classification_report, подав на неё метки тестовых данных и полученные предсказания на основе тестовых данных. Преимущества данного метода проверки заключается в большем количестве анализируемых метрик, что позволяет более глубоко оценить качество работы модели.

Помимо данных вариантов оценки, можно воспользоваться матрицей запутанности, показывающей наглядное представление распределения предсказаний.

Пример кода для оценки качества модели на основе отчёта классификации и матрицы запутанности представлен на рисунке 7, а на

рисунке 8 представлен сам отчёт о классификации (левая часть рисунка 8) и матрица запутанности (правая часть рисунка 8).

```python
# Проверка модели встроенными методами keras
test_loss, test_acc = model_mnist.evaluate(test_images, poisoned_test, verbose=2)

from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Предсказание меток на отравленной тестовой выборке
preds=model_mnist.predict(test_images)
preds_classes=np.argmax(preds,axis=1)

# Вывод отчёта по результатам классификации для каждого класса
print(classification_report(poisoned_test, preds_classes))

# Построение и вывод матрицы запутанности результатов
cm = confusion_matrix(poisoned_test, preds_classes)

plt.figure(figsize = (10,8))
sns.heatmap(cm, annot = True, fmt = 'd')
plt.title('Confusion Matrix')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```

Рисунок 7 – Оценка модели

```
313/313 - 1s - 3ms/step - accuracy: 0.8896 - loss: 1.2596
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.97      0.99      0.98      1032
           3       0.98      0.52      0.68      1983
           4       0.99      0.97      0.98       982
           5       0.97      0.98      0.97       892
           6       0.97      0.99      0.98       958
           7       0.98      0.98      0.98      1028
           8       0.00      1.00      0.00         1
           9       0.98      0.97      0.98      1009

    accuracy                           0.89     10000
   macro avg       0.88      0.94      0.85     10000
weighted avg       0.98      0.89      0.92     10000
```

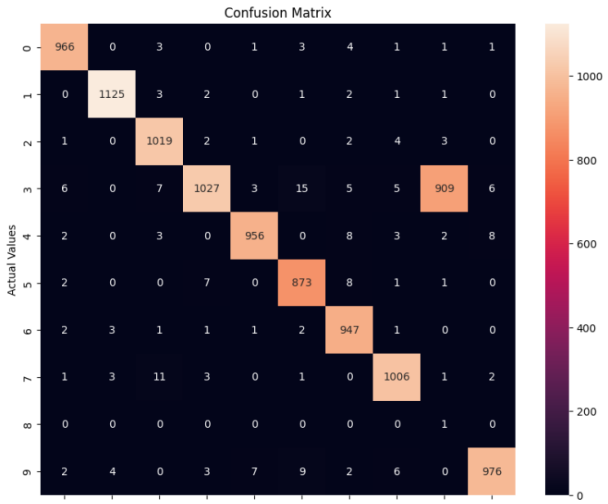

Рисунок 8 – Оценка качества работы модели

Контрольные вопросы

1. Что такое отравление данных?
2. Какие существуют виды отравления данных?
3. Какие признаки могут указывать на возможное отравление данных?
4. Как отравление данных может повлиять на процесс обучения?
5. Можно ли использовать отравление данных во благо?

ЛАБОРАТОРНАЯ РАБОТА №2
ОТРАВЛЕНИЕ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕК CLEVERHANS И ADVERSAL ROBUSTNESS TOOLBOX

Цель работы – ознакомление с методологией состязательных атак на наборы данных с изображениями на примере набора данных MNIST и библиотек CleverHans и Adversal Robustness Toolbox.

Краткие теоретические сведения

Состязательные атаки — это методы, направленные на обман моделей машинного обучения, особенно глубоких нейронных сетей, путём внесения небольших, зачастую незаметных изменений в входные данные. Эти изменения могут привести к тому, что модель будет выдавать неправильные или неожиданные результаты.

Состязательные атаки могут быть использованы для различных целей, включая:

1) Тестирование устойчивости моделей: Исследователи используют состязательные атаки для оценки надежности и устойчивости моделей к потенциальным угрозам;

2) Безопасность: В контексте безопасности такие атаки могут быть использованы злоумышленниками для обхода систем распознавания лиц, систем автоматического вождения и других приложений;

3) Улучшение моделей: Понимание того, как модели реагируют на состязательные примеры, может помочь в их улучшении и повышении устойчивости к подобным атакам.

CleverHans — это библиотека на Python, предназначенная для разработки и оценки алгоритмов защиты от атак на системы машинного обучения, особенно в области глубокого обучения. Она предоставляет инструменты для создания и анализа различных типов атак на модели, включая атаки с добавлением шумов, атаки с перенастройкой (adversarial

examples) и другие подходы, которые могут пытаться обмануть модели, обученные на данных.

Основные функции CleverHans включают:

1) Генерация атак: Библиотека предлагает реализации различных методов создания адверсариальных примеров, которые могут использоваться для тестирования устойчивости моделей.

2) Оценка модели: CleverHans позволяет исследователям и разработчикам оценивать, насколько их модели устойчивы к атакам, что является важным аспектом в разработке безопасных систем машинного обучения.

3) Интеграция с другими библиотеками: CleverHans может работать в сочетании с популярными фреймворками глубокого обучения, такими как TensorFlow и PyTorch, что делает ее удобной для использования в существующих проектах.

Adversarial Robustness Toolbox (ART) — это библиотека, разработанная для работы с системами машинного обучения и искусственного интеллекта с акцентом на устойчивость к атакам с использованием adversarial attacks. Эта библиотека предоставляет инструменты для создания, обучения и оценки моделей, способных противостоять атакам, которые могут быть направлены на искажение их выводов.

Основные функции ART включают:

1) Крепление моделей: Инструменты для защиты моделей, включая методы, такие как adversarial training (обучение с использованием атак), и другие техники улучшения устойчивости.

2) Генерация атак: Различные методы создания adversarial примеров, которые можно использовать для тестирования уязвимостей модели.

3) Оценка устойчивости: Методы оценки того, насколько хорошо модель справляется с adversarial атаками.

4) Совместимость с различными библиотеками: ART поддерживает широкий спектр фреймворков машинного обучения, таких как TensorFlow, PyTorch и другие.

Задание на лабораторную работу

1. Обучить модель нейронной сети на чистом наборе данных MNIST;
2. Провести направленную и не направленную атаки на обучающую и тестовую выборки с помощью библиотеки CleverHans;
3. Провести атаку на обучающую и тестовую выборки с помощью библиотеки Adversal Robustness Toolbox;
4. Проверить ранее обученную модель на атакованных тестовых выборках и провести обучение новых моделей на атакованных обучающих выборках;
5. Сравнить полученные результаты;
6. Оформить отчет о проделанной работе согласно ОС ТУСУР 01-2021.

Критерии оценок:

1. Для оценки «3» необходимо обучить нейронную сеть на данных и протестировать. Также провести направленные и не направленные атаки используя библиотеку CleverHans в соответствии с полученным вариантом;
2. Для оценки «4» необходимо провести атаки на тестовую и обучающую выборки используя библиотеку Adversal Robustness Toolbox. Также выполнить требования для оценки «3»;
3. Для оценки «5» необходимо провести сравнение всех полученных результатов зафиксировав в отчет метрики и матрицы запутанности. Провести одну любую атаку с измененным параметром eps. Также выполнить требования для оценки «3,4».

1 ПОДГОТОВИТЕЛЬНЫЙ ЭТАП

Перед началом работы необходимо установить библиотеку tensorflow версией 2.15.0 (рисунок 1), поскольку на более новых версиях библиотека ART не будет работать.

В случае если библиотека устанавливается после выполнения какого-либо этапа, тогда потребуется перезапустить ядро сеанса. Это сбросит все переменные и определённые библиотеки.

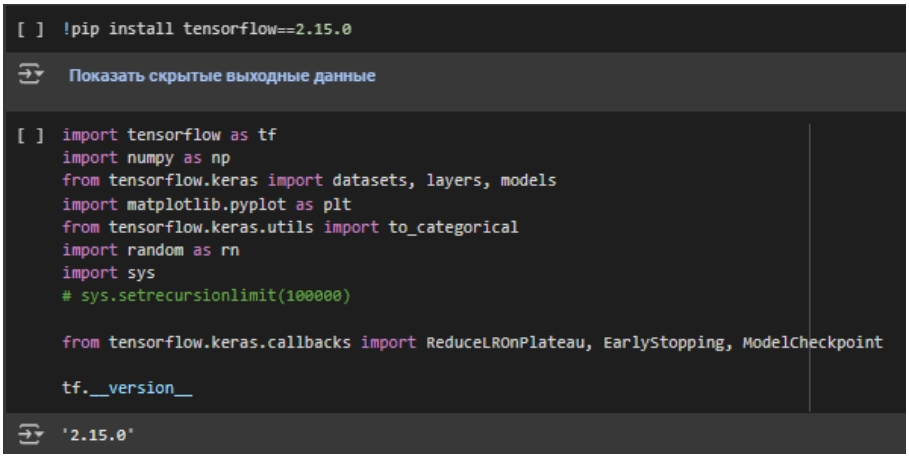

Рисунок 1 – Загрузка библиотек

Введите свои ФИО (полностью) в именительном падеже, как на рисунке 2.

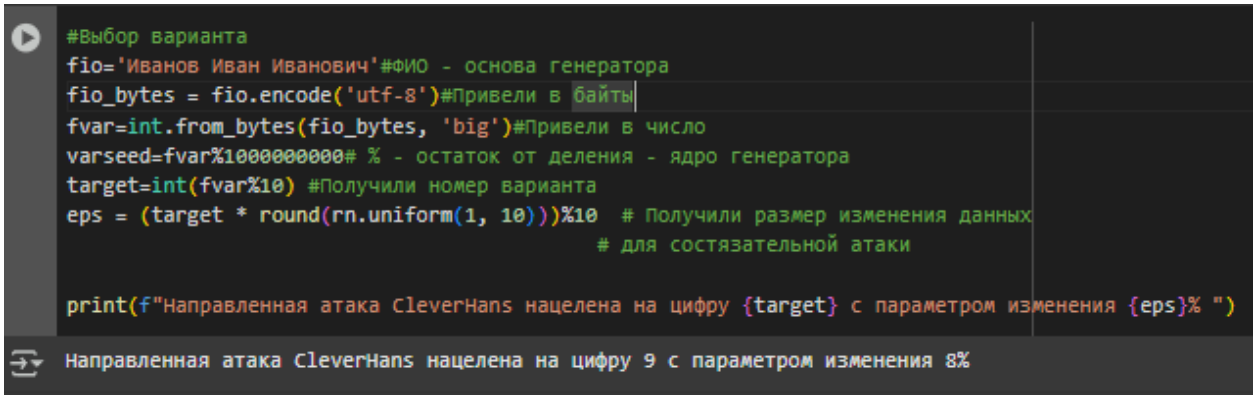

Рисунок 2 – Получение варианта задания

2 ОБУЧЕНИЕ НА ЧИСТЫХ ДАННЫХ

Для выполнения данной работы скачиваем набор данных MNIST и проверяем полученные данные (рисунок 3).

```
[ ] (train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()
    train_images, test_images = train_images / 255.0, test_images / 255.0

    print(train_images.shape, train_labels.shape)
    print(test_images.shape, test_labels.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
```

```
plt.figure(figsize=(7,7))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.xlabel(train_labels[i])
    plt.imshow(train_images[i])
plt.show()
```

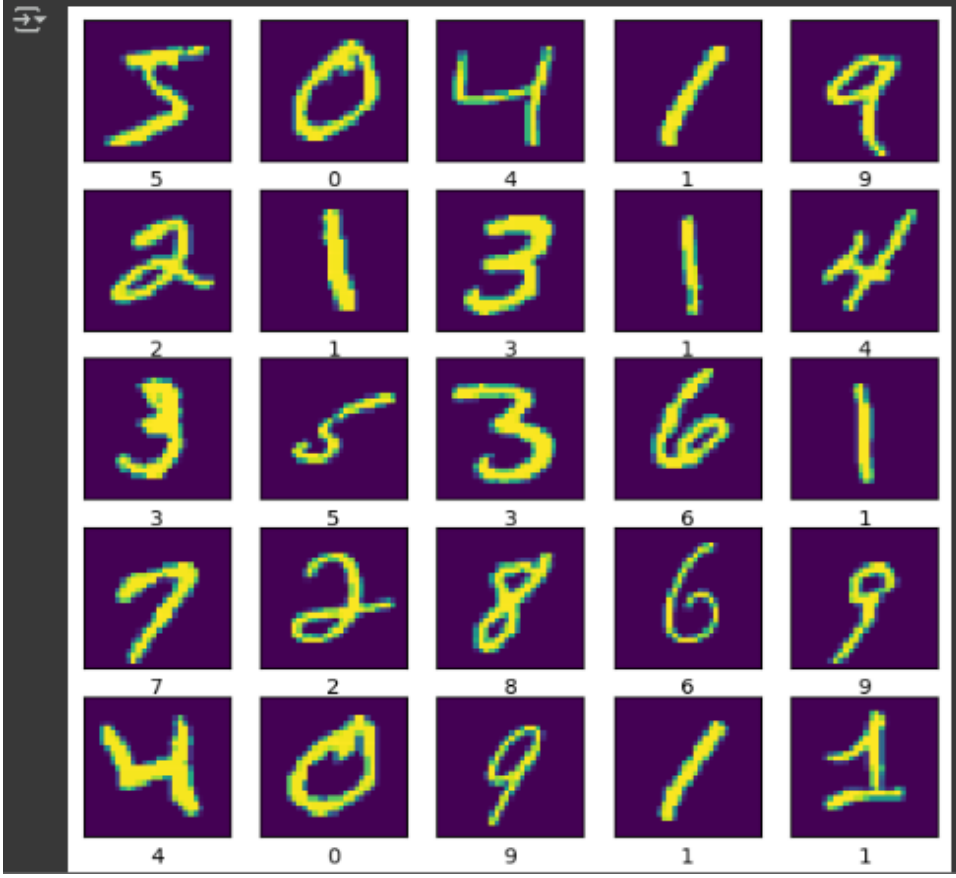

Рисунок 3 – Набор данных

Следующим этапом строим модель, состоящую из 3-х слоев (рисунок 4).

Слой **Flatten** в нейросетевых моделях используется для преобразования многомерного тензора (например, выходных данных из свёрточного слоя) в одномерный вектор. Обычно это делается перед подачей этих данных в полносвязный слой (Dense Layer). Когда вы работаете с изображениями

(например, размером 28x28 пикселей), выход свёрточного слоя может быть представлен как трехмерный тензор (высота, ширина, количество каналов). Слой Flatten преобразует этот тензор в одномерный вектор, который затем можно использовать в полносвязном слое для классификации или регрессии.

Слой **Dense** (также известный как полносвязный слой или линейный слой) в нейронных сетях отвечает за выполнение линейного преобразования данных, получаемых на входе, с последующим применением активационной функции. В данном случае используются такие функции активации как **ReLU** и **softmax**.

Суть работы ReLU заключается в том, что если входное значение больше нуля, функция возвращает это значение, а если меньше или равно нулю — возвращает ноль. Таким образом, ReLU пропускает положительные значения, а отрицательные значения обнуляет.

Функция активации softmax — это функция, которая преобразует вектор из произвольных действительных чисел в вектор вероятностей, суммирующийся в 1.

Также стоит отметить использование оптимизатора. Оптимизатор Adam (Adaptive Moment Estimation) используется в машинном обучении и глубоком обучении для эффективного обновления параметров моделей, таких как нейронные сети.

Основные характеристики и преимущества Adam:

1) Адаптивная скорость обучения: Adam автоматически настраивает скорость обучения для каждого параметра, что позволяет ему более эффективно обучаться на данных с разной дисперсией.

2) Использование моментов: Adam использует скользящие средние градиентов (первого момента) и квадратов градиентов (второго момента), что помогает избежать проблем с шумом и колебаниями градиентов.

3) Комбинация адаптивного градиента и моментов: Это позволяет Adam быстро сходиться в начале обучения и более точно на финишных этапах.

4) Относительная простота настройки: Adam имеет всего несколько гиперпараметров (например, скорость обучения, β1 и β2), что позволяет легко использовать его в большинстве задач.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.summary()

model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 64)                50240

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 50890 (198.79 KB)
Trainable params: 50890 (198.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Рисунок 4 – Построение модели

Следующим шагом, необходимо начать обучение нашей построенной модели (рисунок 5).

```
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1,  mode='min', restore_best_weights= True)

history = model.fit(train_images, train_labels, epochs=20,
                    validation_data=(test_images, test_labels),
                    callbacks=[early_stopping])

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='lower right')
plt.show()

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

Рисунок 5 – Параметры обучения модели

Параметр *early_stopping* используется для предотвращения переобучения модели при обучении нейронной сети. Он позволяет остановить обучение, когда модель перестает улучшаться на валидационном наборе данных.

В параметре *history* указываются основная информация процессе обучения модели. В большинстве библиотек для глубокого обучения, таких как Keras, *history* возвращается после вызова метода fit() и включает в себя данные о метриках и потерях (loss) на каждой эпохе обучения.

После успешного обучения можно увидеть график потерь и график точности на каждой эпохе, а также итоговую достигнутую точность (рисунок 6).

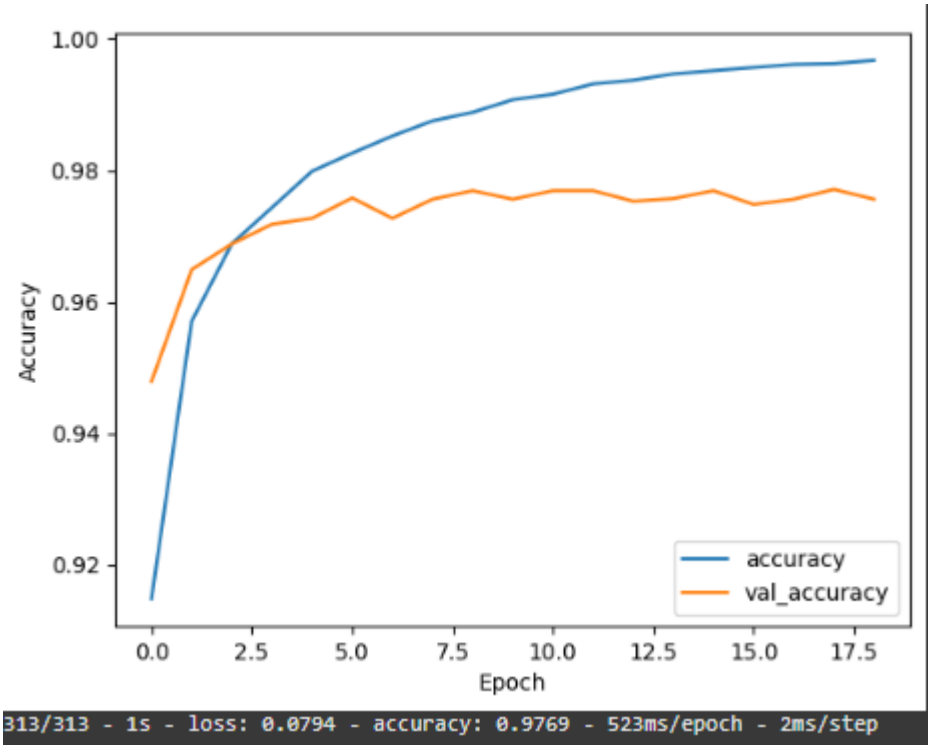

Рисунок 6 – График точности после обучения модели

Далее необходимо построить матрицу запутанности по результатам нашего обучения (рисунок 7). Также помимо матрицы после выполнения последнего блока кода на данном этапе, можно увидеть итоговые результаты (рисунок 8), основанные на таких метриках как precision, recall, f1-мера, точность, усредненная точность и взвешенная точность.

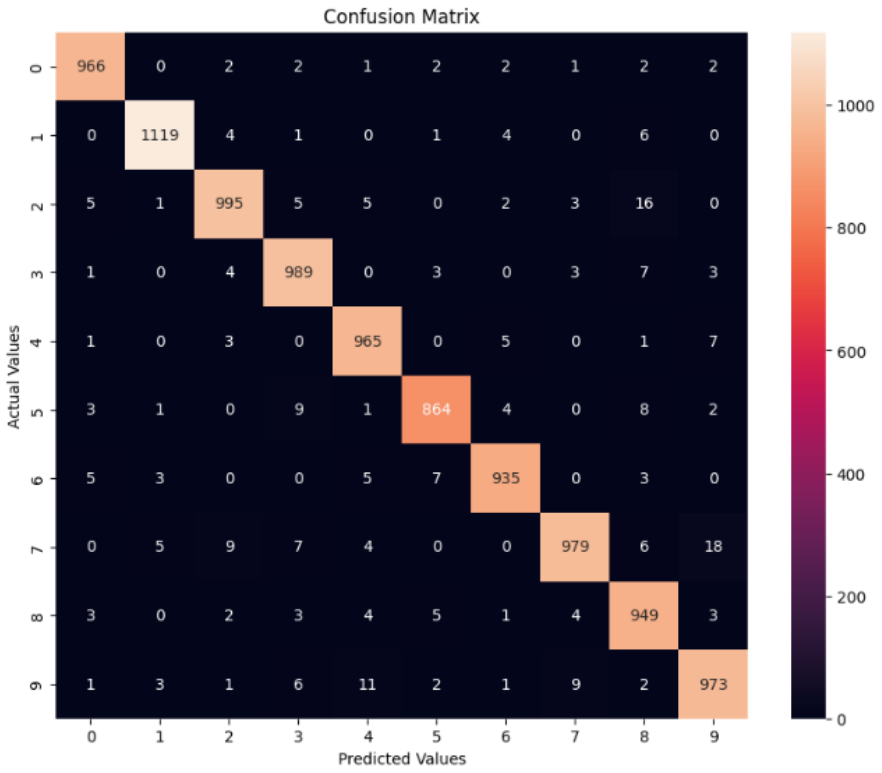

Рисунок 7 – Матрица запутанности

	precision	recall	f1-score
0	0.98	0.99	0.98
1	0.99	0.99	0.99
2	0.98	0.96	0.97
3	0.97	0.98	0.97
4	0.97	0.98	0.98
5	0.98	0.97	0.97
6	0.98	0.98	0.98
7	0.98	0.95	0.97
8	0.95	0.97	0.96
9	0.97	0.96	0.96
accuracy			0.97
macro avg	0.97	0.97	0.97
weighted avg	0.97	0.97	0.97

Рисунок 8 – Значения метрик

3 CLEVERHANS

Для начала необходимо установить CleverHans запустив первую ячейку кода в данном этапе. Также в начале раздела вы можете увидеть пример атаки на одно изображение.

Этот этап включает в себя три шага:

1) Проведение не направленной атаки. Не направленная атака на отравление данных подразумевает, что злоумышленник пытается внести

изменения в обучающую выборку таким образом, чтобы ухудшить производительность модели, но не имеет конкретной цели или направления. Это может включать добавление случайных или неправильно размеченных данных в обучающую выборку, что может привести к снижению точности модели;

2) Проведение не направленной атаки. Это атаки, где злоумышленник имеет четкую цель, например заставить модель классифицировать определенный класс неправильно (в данном случае – конкретные цифры), не направленные атаки более хаотичны и могут быть сложнее для обнаружения и предотвращения. Они могут использоваться для создания "шума" в данных, что затрудняет обучение модели.

3) Финальный этап этого блока заключается в обучении модели на атакованной обучающей выборке.

На рисунке 9, в показанном блоке кода вам необходимо написать функцию для реализации атаки на тестовую выборку и провести тестирование.

```
[ ]  def get_adv_set(testset,model):
       # Напишите функция для реализации атаки
       # На вход функция принимает тестовую выборку,
       # а на выходе выдаёт изменённые данные в формате массива numpy
       return advset

     adv_test=get_adv_set(test_images,model)
     adv_test=np.reshape(adv_test, test_images.shape)
     print(adv_test.shape)
```

Рисунок 9 – Написание функции для не направленной атаки

Используя следующую ячейку с кодом (рисунок 10), вы можете провести необходимое тестирование и сравнение результатов с изначальными, которые были получены на обучении на чистых данных. Различия будут видны как в значениях метрик точности, так и в матрице запутанности.

```
preds=model.predict(adv_test)
preds_classes=np.argmax(preds,axis=1)

print(classification_report(test_labels, preds_classes))

cm = confusion_matrix(test_labels, preds_classes)

plt.figure(figsize = (10,8))
sns.heatmap(cm, annot = True, fmt = 'd')
plt.title('Confusion Matrix')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```

Рисунок 10 – Блок кода для тестирования

Далее, нужно реализовать направленную атаку для всей тестовой выборки написав функции для ее реализации (рисунок 11). Аналогичным способом, как и при проведении не направленной атаки нужно провести тестирование данного метода.

```
def get_adv_targeted_set(testset, model, target):
  # Напишите функцию для реализации направленной атаки на тестовую выборку
  # На вход функция принимает тестовую выборку и цель атаки,
  # а на выходе выдаёт изменённые данные в формате массива numpy
  return advset

adv_targeted_test=get_adv_targeted_set(test_images,model,1)
adv_targeted_test=np.reshape(adv_targeted_test, test_images.shape)
print(adv_test.shape)
```

Рисунок 11 – Написание функции для направленной атаки

Заключительным этапом данного раздела необходимо провести обучение на атакованной обучающей выборке с помощью не направленной и направленной атак. Используя ранее написанные функции, необходимо изменить данные на вход функции с тестовой на обучающую выборку. На рисунке 12 представлен пример отравления нескольких изображений с помощью не направленной атаки.

Рисунок 12 – Пример отравления нескольких изображений

Следующим шагом строим модель и проведем ее обучение. После чего проводим тестирование как и ранее на тестовой выборке, сравнивая полученные результаты.

Такой же этап проводим и для направленной атаки.

4 ADVERSARIAL ROBUSTNESS TOOLBOX

Для начала необходимо установить Adversarial Robustness Toolbox запустив первую ячейку кода в данном этапе.

Необходимо провести атаки на тестовую и обучающую выборки.

Начнем с атаки на тестовую выборку. Для проведения атаки необходима необученная модель, которая будет преобразована к нужному формату (рисунок 13). Для этого построим новую модель и обучим её после перевода к необходимому формату.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.summary()

model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

Показать скрытые выходные данные

```
[ ] # Перевод новой модели к необходимому формату и её обучение
classifier = KerasClassifier(model=model, clip_values=(0,1), use_logits=False)
classifier.fit(train_images, train_labels, nb_epochs=20)
```

Рисунок 13 – Преобразование модели

Далее, на рисунке 14 показан пример атаки для одного изображения. Необходимо реализовать атаку для всей тестовой выборки дописав функции в нижнем блоке.

```
# Выбираем метод атаки и степень изменения eps
attack = FastGradientMethod(estimator=classifier, eps=0.2)
# выбираем первое изображение тестовой выборки и проводим на него атаку
image = test_images[0]
example = attack.generate(image.reshape(1,28,28))

plt.imshow(example.reshape(28,28))
plt.show()
```

Показать скрытые выходные данные

```
[ ] # Реализуйте атаку для всей тестовой выборки
x_test_adv=...
```

Рисунок 14 – Пример атаки для одного изображения и блок задания

После чего просмотрите изменения на матрице запутанности и метриках точности используя последний блок кода.

Аналогичным методом нужно провести атаку на обучающую выборку.

Контрольные вопросы

1. Что такое состязательные атаки?
2. Для чего используются состязательные атаки?
3. Что из себя представляет библиотека CleverHans?
4. Что из себя представляет библиотека Adversal Robustness Toolbox?
5. Для чего используется матрица запутанности?

ЛАБОРАТОРНАЯ РАБОТА № 3
МЕТОДОЛОГИЯ ОТРАВЛЕНИЯ РЕЧЕВЫХ ДАННЫХ

Целью работы является ознакомление с методологией отравления речевых данных.

Краткие теоретические сведения

Отравление данных (Data Poisoning) — это тип атаки на модели машинного обучения, при котором злоумышленник внедряет в обучающие данные искажённую или вредоносную информацию. Цель подобных атак — ухудшение качества работы модели или создание предсказуемых уязвимостей.

Основные виды атак:

Атаки на доступность (Availability Attacks). Цель — ухудшение общей производительности модели за счёт включения большого количества шумных или некорректных данных. Пример: скрытные атаки. Суть скрытных атак состоит в том, что злоумышленник изменяет входные данные таким образом, чтобы обмануть модель, не вызывая подозрений у пользователя. Модификации часто минимальны и едва заметны, но приводят к ошибочным предсказаниям модели.

1) Атаки на целостность (Integrity Attacks). Злоумышленник добавляет данные таким образом, чтобы модель давала предсказуемые неправильные ответы для определённых входов. Пример: атака бэкдором. В таких атаках злоумышленник внедряет в обучающие данные скрытые триггеры (например, определённые шумы, слова или аудиометки), которые не влияют на обычное поведение модели, но вызывают заранее запланированное искажение её ответа при активации.

2) Атаки на конфиденциальность (Privacy Attacks). Попытки извлечь конфиденциальные данные из модели на основе особенностей её поведения. Пример: атака с инверсией модели. Злоумышленник пытается восстановить

конфиденциальные или приватные данные, использованные для обучения модели, на основе её предсказаний. Это делается путём анализа градиентов, выходных вероятностей или закономерностей модели при различных входных данных.

Требования к выполнению лабораторной работы

1. Ознакомиться с методическими указаниями, а также представленным примерамом в блокноте Google.Collab.

2. Выполнить индивидуальные задания, описанные в блокноте Google Collab.

3. Ответить на контрольные вопросы.

1 ХОД РАБОТЫ

Перед началом работы необходимо скачать архив с набором данных для работы. Набор данных представляет из себя 200 аудиозаписей формата wav, из которых 100 записей – реальные, а оставшиеся сгенерированные. После того как набор данных будет скачан, необходимо загрузить его в файловое пространство Collab с помощью специального функционала по выбору и загрузке файлов, либо же с помощью функционала drag-and-drop. Также это можно выполнить через функционал, показанный на рисунке 1.1. Кроме того были указаны пути для реальных и фейковых данных.

```
# Подключаем гугл диск к блокноту
from google.colab import drive
drive.mount('/content/drive')

# Разархивируйте архив с данными, указав правильный путь к нему
!unzip /content/drive/MyDrive/Audio.zip

# Путь к данным
REAL_PATH = '/content/Audio/Real/'
FAKE_PATH = '/content/Audio/Fake/'
```

Рисунок 1.1 – Подключение Google диска

Для того, чтобы классификатор мог распознавать подлинность речи, необходимо извлечь признаки из аудиозаписей. На рисунке 1.2 изображена

функция, с помощью которой это можно сделать. Функция load_audio_files() выполняет следующие действия:

- Принимает путь к папке с аудиофайлами и метку (label) в качестве входных параметров;
- Считывает все файлы формата .wav из указанного каталога;
- Загружает аудиофайлы с помощью librosa.load() с частотой дискретизации 16 кГц;
- Извлекает признаки MFCC (Mel-frequency cepstral coefficients) с помощью librosa.feature.mfcc(), используя 13 коэффициентов;
- Рассчитывает среднее значение MFCC по временной оси и добавляет его вместе с меткой в список data.

```python
def load_audio_files(path, label):
    """
    Загружает аудиофайлы из каталога и извлекает MFCC-признаки.
    """
    data = []
    for file_name in os.listdir(path):
        if file_name.endswith(".wav"):  # Убедимся, что файл имеет формат .wav
            file_path = os.path.join(path, file_name)
            audio, sr = librosa.load(file_path, sr=16000)  # Загрузка аудиофайла с частотой 16 kHz
            mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)  # Исправлено: аргументы передаются корректно
            data.append((mfcc.mean(axis=1), label))  # Среднее значение MFCC по времени
    return data
```

Рисунок 1.2 – Функция извлечения признаков

Затем разделите исходный набор данных на тренировочную и обучающие выборки с разделением признаков и меток (рисунок 1.3).

```python
# Загрузка данных
real_data = load_audio_files(REAL_PATH, 1)  # Реальные записи (метка 1)
fake_data = load_audio_files(FAKE_PATH, 0)  # Фейковые записи (метка 0)

# Объединение данных
data = real_data + fake_data
np.random.shuffle(data)

# Разделение признаков и меток
X, y = zip(*data)
X = np.array(X)
y = np.array(y)

# Разделение на тренировочную и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Рисунок 1.3 – Разделение на выборки

Постройте модель на основе случайного дерева и проведите обучение с вычислением метрик точности (рисунок 1.4).

```python
# Обучение модели на чистых данных
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Оценка качества модели
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
```

Рисунок 1.4 – Построение и обучение модели

Пример результата обучения на чистых данных представлен на рисунке 1.5.

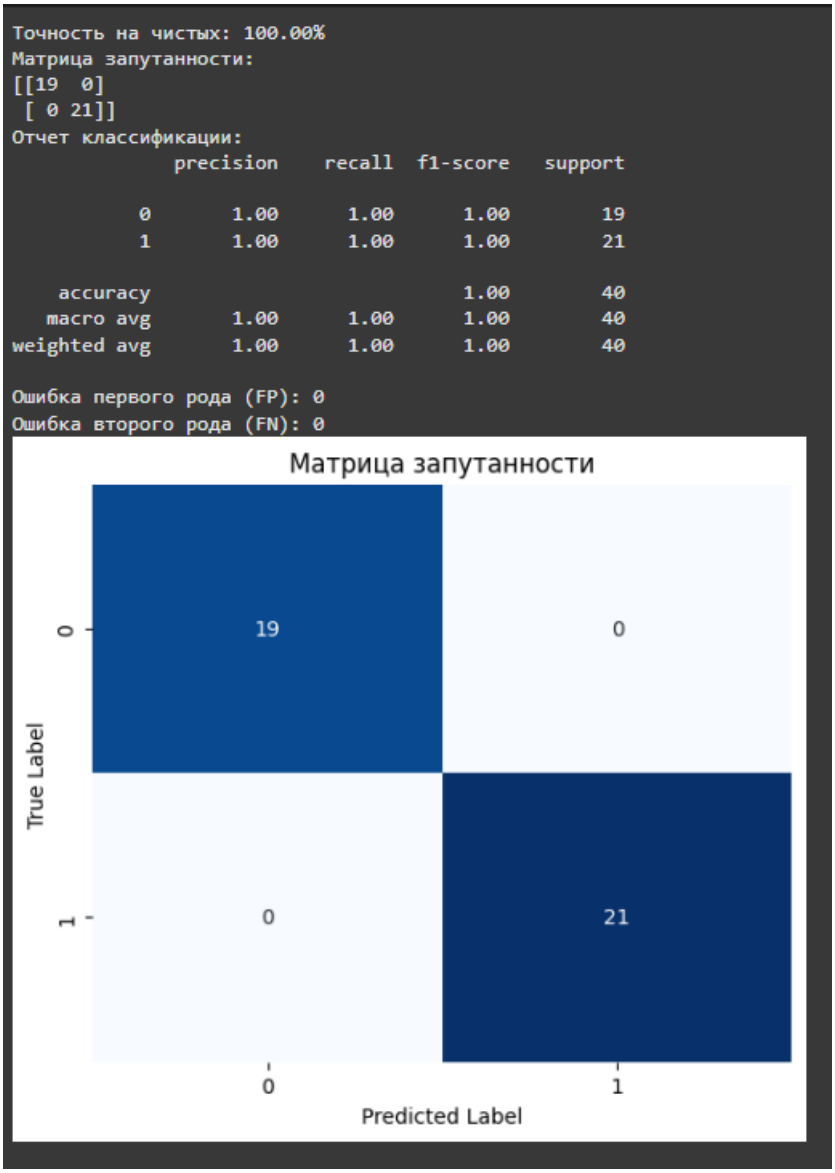

Рисунок 1.5 – Результат обучения на чистых данных

Напишите функцию отравления данных. Она должна менять метки записей на противоположные. Записи для отравления должны браться

случайно (рисунок 1.6).

Функция отравления

```
[ ]  def poison_data(X, y, poison_rate):
         # Реализуйте функцию отравления данных
         # изменяя метки на противоположные (реальные на фейковые и наоборот)
         # с добавлением рандомизации выбора данных для отравления.

         return poisoned_X, poisoned_y
```

Рисунок 1.6 – Функция отравления данных

Пример результата обучения на отравленных данных представлен на рисунке 1.7.

```
Точность на отравленных данных: 2.50%
Матрица запутанности:
[[ 0 19]
 [20  1]]
Отчет классификации:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        19
           1       0.05      0.05      0.05        21

    accuracy                           0.03        40
   macro avg       0.03      0.02      0.02        40
weighted avg       0.03      0.03      0.03        40

Ошибка первого рода (FP): 19
Ошибка второго рода (FN): 20
```

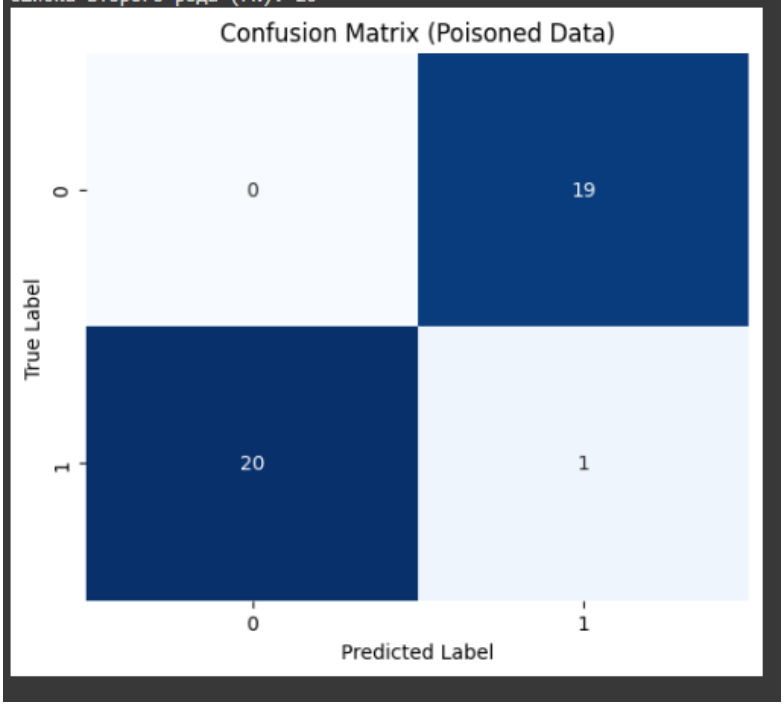

Рисунок 1.7 – Результат обучения на отравленных данных

Контрольные вопросы

1. Что подразумевается под отравлением речевых данных и как эта методология может повлиять на работу моделей машинного обучения?

2. Какие виды атак на речевые данные можно выделить и как они различаются по своей цели и способу реализации?

3. Какими способами можно обнаружить и предотвратить отравление речевых данных на этапе подготовки набора данных?

4. Какой тип метрик используется для оценки устойчивости модели к отравленным данным? Приведите примеры.

ЛАБОРАТОРНАЯ РАБОТА № 4
МЕТОДОЛОГИЯ СОСТЯЗАТЕЛЬНЫХ АТАК НА РЕЧЕВОЙ СИГНАЛ

Целью работы является ознакомление с методологией состязательных атак на речевой сигнал.

Краткие теоретические сведения

Состязательные атаки (adversarial attacks) представляют собой искусственно созданные изменения данных, которые приводят к ошибкам работы моделей машинного обучения. В контексте обработки речевых сигналов такие атаки направлены на изменение аудиосигнала таким образом, чтобы модель голосового распознавания либо ошиблась, либо выдала предсказание, выгодное атакующему.

Методы создания состязательных атак:

1) FGSM — это один из самых простых и эффективных методов создания состязательных атак на модели машинного обучения. Его основная идея заключается в добавлении направленного шума к входным данным на основе градиента функции ошибки модели по входу.

2) Универсальные состязательные возмущения (Universal Adversarial Perturbations). UAP представляет собой метод создания универсальных возмущений, которые могут успешно атаковать широкий набор входных данных, а не только конкретный сигнал. Это делает метод особенно опасным, так как одно и то же возмущение можно применять ко множеству различных аудиосигналов, вызывая ошибки модели.

3) PGD является расширенной версией FGSM (Fast Gradient Sign Method) и представляет собой итеративный метод создания состязательных примеров. Основная идея заключается в том, чтобы последовательно добавлять небольшие изменения к исходному сигналу, двигаясь в направлении градиента функции ошибки, и при этом проецировать результат в допустимую область

ограничений (например, по допустимому уровню шума).

4) Метод CW является одной из наиболее мощных и точных атак на нейронные сети. Он был предложен Николаусом Карлини и Дэвидом Вагнером и представляет собой оптимизационную задачу, где целью является минимизация возмущений сигнала при достижении ошибки предсказания модели.

Требования к выполнению лабораторной работы

1. Ознакомиться с методическими указаниями, а также представленным примерамом в блокноте Google.Collab.

2. Выполнить индивидуальные задания, описанные в блокноте Google Collab.

3. Ответить на контрольные вопросы.

1 ХОД РАБОТЫ

В данной работе используется тот же набор данных, что и в лабораторной работе №3. Загрузите набор данных в сессионное хранилище, с помощью функционала, показанного на рисунке 1.1.

```
# Подключаем гугл диск к блокноту
from google.colab import drive
drive.mount('/content/drive')

# Разархивируйте архив с данными, указав правильный путь к нему
!unzip /content/drive/MyDrive/Audio.zip

# Путь к данным
REAL_PATH = '/content/Audio/Real/'
FAKE_PATH = '/content/Audio/Fake/'
```

Рисунок 1.1 – Подключение Google диска

Для извлечения признаков используется функция load_audio_files (рисунок 1.2). Функция load_audio_files() выполняет следующие действия:

- Принимает путь к папке с аудиофайлами и метку (label) в качестве входных параметров;
- Считывает все файлы формата .wav из указанного каталога;
- Загружает аудиофайлы с помощью librosa.load() с частотой дискретизации 16 кГц;
- Извлекает признаки MFCC (Mel-frequency cepstral coefficients) с помощью librosa.feature.mfcc(), используя 13 коэффициентов;
- Рассчитывает среднее значение MFCC по временной оси и добавляет его вместе с меткой в список data.

```python
def load_audio_files(path, label):
    """
    Загружает аудиофайлы и извлекает MFCC-признаки.
    """
    data = []
    for file_name in os.listdir(path):
        if file_name.endswith(".wav"):
            file_path = os.path.join(path, file_name)
            try:
                audio, sr = librosa.load(file_path, sr=16000)
                mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
                data.append((mfcc.mean(axis=1), label))
            except Exception as e:
                print(f"Ошибка при обработке {file_name}: {e}")
    return data
```

Рисунок 1.2 – Функция извлечения признаков

Затем разделите исходный набор данных на тренировочную и обучающие выборки с разделением признаков и меток (рисунок 1.3).

```python
# Загрузка и объединение данных
real_data = load_audio_files(REAL_PATH, 1)
fake_data = load_audio_files(FAKE_PATH, 0)
data = real_data + fake_data
np.random.shuffle(data)

# Разделение признаков и меток
X, y = zip(*data)
X = np.array(X)[..., np.newaxis]  # Добавляем канал для работы с CNN
y = np.array(y)

# Разделение данных
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Рисунок 1.3 – Разделение на выборки

Для дальнейшей работы из исходного набора данных необходимо составить пары случайных аудиозаписей. Сделать вы это сможете с помощью

функции показанной на рисунке 1.4. Функция make_pairs() создаёт пары аудиозаписей для задач обучения сиамских нейронных сетей.

```
def make_pairs(images, labels):
    pairImages = []
    leftInput = []
    rightInput = []
    pairLabels = []
    numClasses = len(np.unique(labels))
    idx = [np.where(labels == i)[0] for i in range(0, numClasses)]

    for idxA in range(len(images)):
        currentImage = images[idxA]
        label = labels[idxA]

        idxB = np.random.choice(idx[label])
        posImage = images[idxB]
        pairImages.append([currentImage, posImage])
        leftInput.append(currentImage)
        rightInput.append(posImage)
        pairLabels.append(1)

        negIdx = np.where(labels != label)[0]
        negImage = images[np.random.choice(negIdx)]
        pairImages.append([currentImage, negImage])
        leftInput.append(currentImage)
        rightInput.append(negImage)
        pairLabels.append(0)

    return (np.array(leftInput), np.array(rightInput), np.array(pairLabels))

X_left, X_right, y_pairs = make_pairs(X, y)
```

Рисунок 1.4 – Функция создания пар аудиозаписей

Создайте и обучите простую сиамскую нейронную сеть на чистых данных (рисунок 1.5).

```
def build_model():
    inputA = keras.layers.Input(shape=X.shape[1:])
    inputB = keras.layers.Input(shape=X.shape[1:])

    base_network = keras.Sequential([
    keras.layers.Conv1D(32, 3, activation='relu', input_shape=(13, 1)),
    keras.layers.MaxPooling1D(2),
    keras.layers.Conv1D(64, 3, activation='relu'),
    keras.layers.MaxPooling1D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(64, activation='relu')
    ])


    encodedA = base_network(inputA)
    encodedB = base_network(inputB)

    L1_layer = keras.layers.Lambda(lambda tensors: 1 - tf.abs(tensors[0] - tensors[1]))
    L1_distance = L1_layer([encodedA, encodedB])

    output = keras.layers.Dense(1, activation='sigmoid')(L1_distance)
    model = keras.models.Model(inputs=[inputA, inputB], outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')])

    return model
```

Рисунок 1.5 – Создание модели

Пример результата обучения на чистых данных представлен на рисунке 1.6.

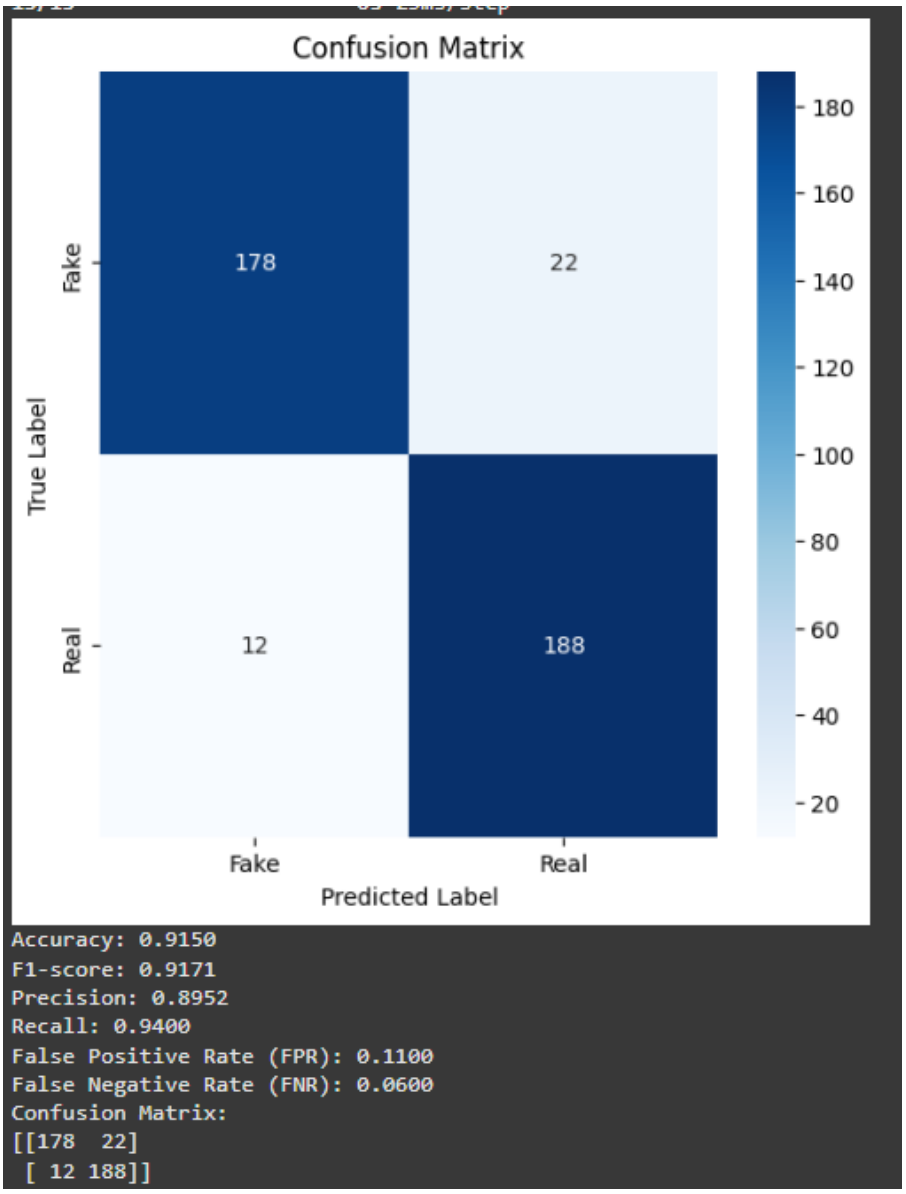

Рисунок 1.6 – Результат обучения на чистых данных

Напишите функцию состязательной атаки и обучите нейронную сеть на отравленных данных с вычислением метрик точности (рисунок 1.7).

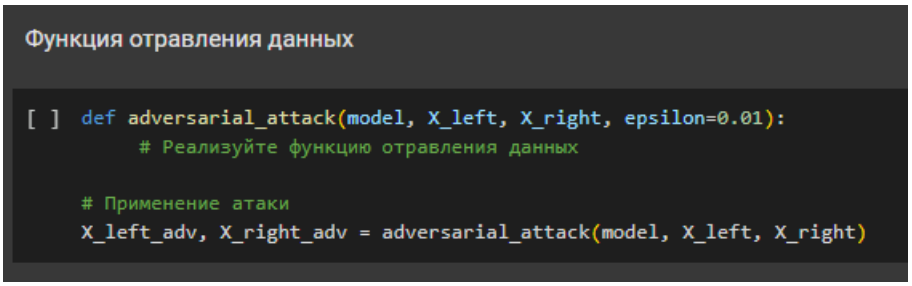

Рисунок 1.7 – Функция состязательной атаки

Пример результата обучения на отравленных данных представлен на рисунке 1.8.

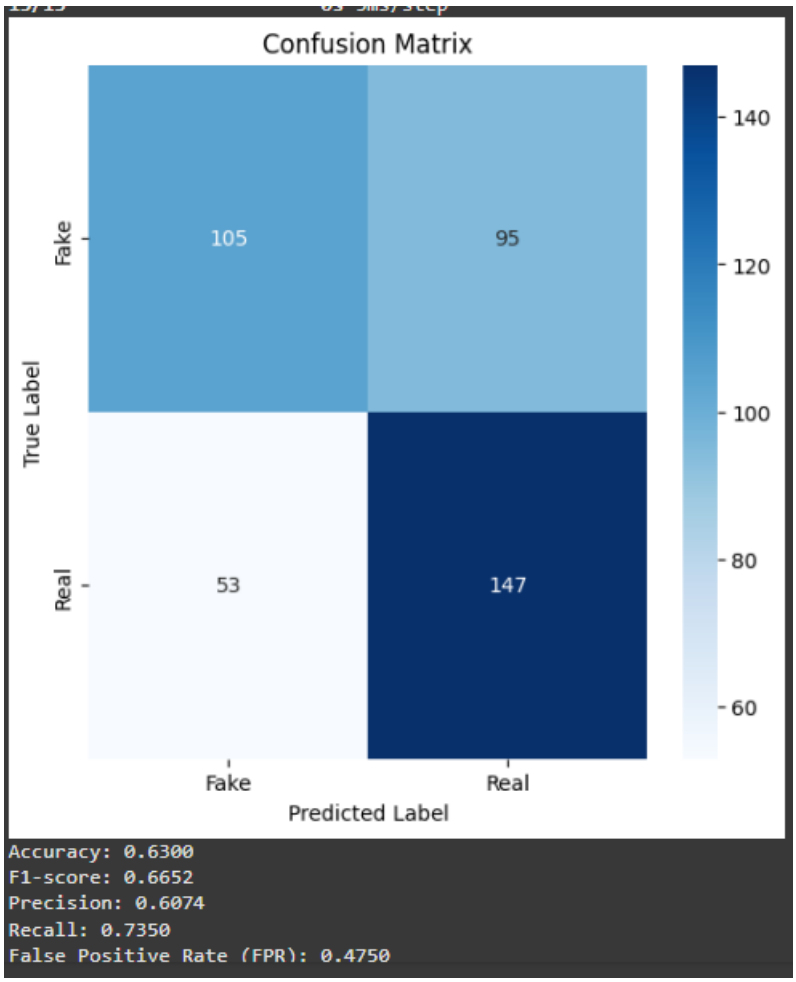

Рисунок 1.8 – Результат обучения на отравленных данных

Контрольные вопросы

1. Каковы основные принципы состязательных атак на речевые сигналы и их отличие от атак на текстовые или визуальные данные?

2. Какие методы и алгоритмы используются для генерации состязательных примеров в аудиосигналах? Приведите примеры.

3. Каковы основные метрики для оценки эффективности состязательных атак на речевые сигналы и их влияние на качество распознавания речи?

4. Какие подходы существуют для защиты систем автоматического распознавания речи от состязательных атак? Оцените их преимущества и недостатки.

ЛАБОРАТОРНАЯ РАБОТА №5
АТАКИ НА НАБОРЫ ДАННЫХ ВРЕМЕННЫХ РЯДОВ

Целью работы является ознакомление с методологией атак на наборы данных временных рядов и отравлением весов модели на примере набора данных Tennessee Eastman Process.

Краткие теоретические сведения

Временные ряды – это собранный в разные моменты времени статистический материал о значении каких-либо параметров исследуемого процесса.

Временные ряды включают два обязательных элемента: время и конкретное значение показателя (уровень ряда). В качестве показателя времени могут указываться либо определённые моменты времени (даты), либо отдельные периоды (сутки, месяцы, кварталы, полугодия, годы и т. д.).

Анализ временных рядов позволяет выявить закономерности и использовать их для оценки характеристик процесса в будущем. Они применяются в таких областях, как экономика, финансы, маркетинг и метеорология.

Tennessee Eastman Process (TEP) – это реалистичная модель химического индустриального процесса. TEP используется в качестве примера для управления промышленным процессом в масштабах всего предприятия, статистического мониторинга процессов, обнаружения неисправностей датчиков и идентификации сетевых моделей на основе данных.

Атаки отравлением. Сценарий атаки – так называемое «отравление информации». Данные дополняются необходимыми злоумышленнику материалами, что обеспечивает неверное обучение нейросети для определенных ситуаций. Суть данной атаки на нейросеть заключается в том, чтобы принудить ее делать что-то, не совпадающее с изначальной моделью на конкретном примере.

Отравление весов модели – это атака на системы машинного обучения, при которой злоумышленник изменяет веса модели.

Задание на лабораторную работу

1. Обучить модель нейронной сети на чистых данных.
2. Провести состязательную атаку, изменяя значения сигналов на заданную по варианту величину, для обучающей и тестовой выборок.
3. Провести обучение новой модели на атакованной обучающей выборке.
4. Провести тестирование исходной модели на атакованной тестово й выборке.
5. Провести отравление весов модели, обученной на чистых данных.
6. Провести тестирование модели с отравленными весами.
7. Оформить отчет о проделанной работе согласно ОС ТУСУР 01-2021.

Критерии оценок:

1. Для оценки «**3**» необходимо реализовать состязательную атаку, обучить нейронную сеть на атакованных обучающих данных и протестировать. Также протестировать чистую модель на атакованных данных;

2. Для оценки «**4**» необходимо провести отравление последнего слоя модели и протестировать работу модели. Также выполнить требования для оценки «3»;

3. Для оценки «**5**» необходимо провести отравление всех весов модели и протестировать работу модели. Также выполнить требования для оценки «3».

1 ПОДГОТОВИТЕЛЬНЫЙ ЭТАП

Введите свои ФИО (полностью) в именительном падеже, как на рисунке 1.

```python
import random as rn

#Выбор варианта
fio='Иванов Иван Иванович'#ФИО - основа генератора

temp_fio = 0
for i in fio:
  if i.isupper():
    temp_fio += ord(i)

fio_bytes = fio.encode('utf-8')#Привели в байты
fvar=int.from_bytes(fio_bytes, 'big') + temp_fio + len (fio)  #Привели в число
varseed= fvar%1000000000# % - остаток от деления - ядро генератора
step= abs(fvar%10 - len(fio)%10 + temp_fio%10) # Получили частоту
                                               # изменений данных
eps = (step * round(rn.uniform(1, 10)))%10  # Получили размер изменения данных
                                            # для состязательной атаки

print(f"Провести состязательную атаку, изменив каждое {step}-е число в элементе числового ряда на {eps}% ")
```

Рисунок 1 – Получение варианта задания

Параметр $step$ отвечает за частоту изменений элемента в наборе данных при проведении состязательной атаки. Например, если $step = 10$, то внутри одного конкретного элемента набора данных необходимо будет менять каждое десятое число в элементе.

Параметр eps отвечает за объем атакуемых данных. К примеру, если $eps = 10$, то необходимо изменить элемент массива на 10 процентов от исходного. Таким образом, описанные ранее параметры являются индивидуальным вариантом к данной лабораторной работе.

После получения варианта загрузите набор данных в сессионное хранилище Google Colab. Сделать это можно двумя способами:

1. Напрямую через Kaggle. Необходимо указать данные вашего профиля Kaggle в параметрах «Секреты» блокнота (пример на рисунке 2). Для того, чтобы получить необходимые данные, перейдите в настройки профиля Kaggle и в разделе «API» нажмите кнопку «Create New Token». Будет создан новый токен и скачан json-файл, содержащий информацию о ключах доступа к нему.

Укажите параметры соответственно:

1. KAGGLE_KEY = параметр key из json-файла;

2. KAGGLE_USERNAME = параметр username из json-файла.

Затем выполните код, представленный на рисунке 3.

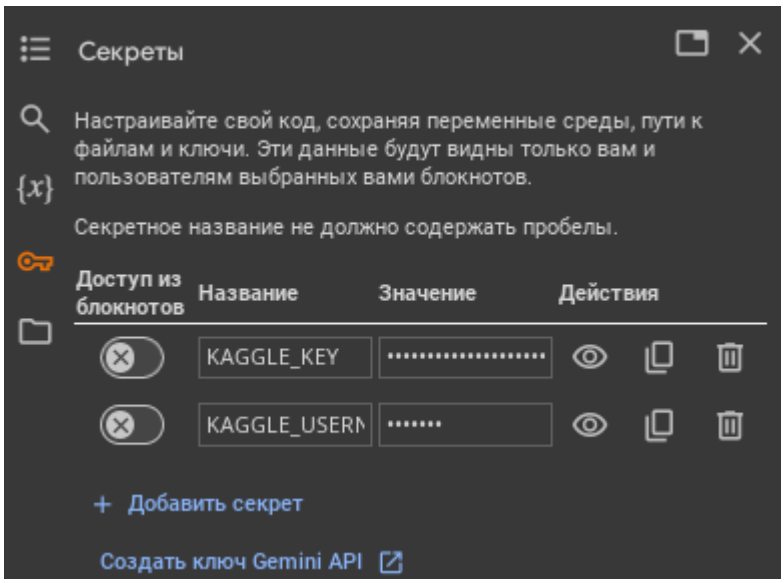

Рисунок 2 – Указание данных профиля Kaggle

```
from google.colab import userdata
import os

# Считываем токены указанные в параметрах блокнота
os.environ["KAGGLE_KEY"] = userdata.get('KAGGLE_KEY')
os.environ["KAGGLE_USERNAME"] = userdata.get('KAGGLE_USERNAME')

# Устанавливаем библиотеку для работы с Kaggle
! pip install -q kaggle

# Скачиваем и разархивируем датасет
!kaggle datasets download -d averkij/tennessee-eastman-process-simulation-datase
! unzip "tennessee-eastman-process-simulation-dataset.zip"
```

Рисунок 3 – Скачивание и разархивирование набора данных

2. Скачивание набора данных через Google Drive. Выполните код с рисунка 4. После выполнения кода выберите опцию «Подключиться к Google-диску», потребуется аккаунт Google, после выбора аккаунта и входа соглашайтесь на все разрешения для Google, нажмите «продолжить» во всех случаях (рисунок 5).

```
from google.colab import drive

# Подключаем Google Drive
drive.mount('/content/gdrive')
root = '/gdrive/My Drive/'

# Разархивируем датасет с Google Drive, указав корректный путь к нему
!unzip '/content/gdrive/MyDrive/work/TEP.zip' -d /content/
```

Рисунок 4 – Скачивание набора данных через Google Drive

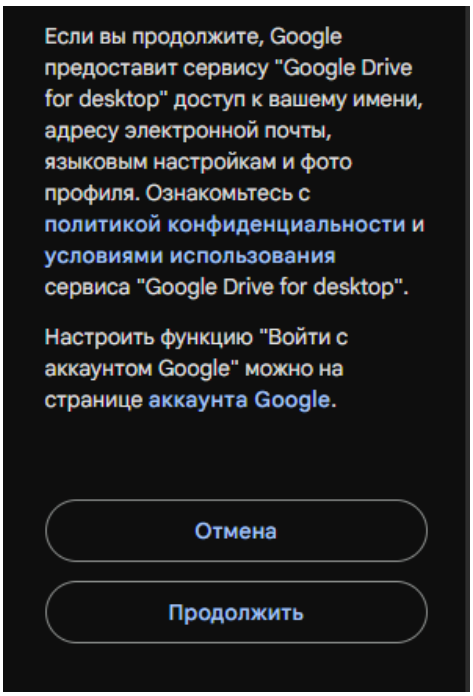

Рисунок 5 – Окно подключения к Google Drive

Стоит отметить, что подключение к Google Drive следует выполнять как можно скорее, поскольку по истечении небольшого количества времени, если подключение произведено не было, код для подключения к Google Drive выдает ошибку подключения. После успешного импорта набора данных в сессионное хранилище Google Colab необходима предварительная обработка набора данных для дальнейшей работы.

Установите библиотеку *pyreadr*. Данная библиотека нужна для первичного получения данных из набора данных, поскольку он состоит из файлов с расширением «.RData». Далее произведите импорт всех необходимых библиотек (рисунок 6).

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

import pyreadr
from random import *
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score, KFold, train_test_split

import sklearn
```

Рисунок 6 – Импорт необходимых пакетов

Выполните считывание набора данных в переменные и создайте соответствующие датафреймы (рисунок 7).

```
result = pyreadr.read_r("/content/TEP_FaultFree_Training.RData")
result1 = pyreadr.read_r("/content/TEP_Faulty_Training.RData")

df_train = result1['faulty_training']
df_ff = result['fault_free_training']

df = pd.concat([df_train,df_ff])

# Выводим размерности нашего набора данных
df.shape
```

Рисунок 7 – Создание датафрейма из набора данных

После этого требуется снизить размер полученного датафрейма, чтобы избежать переполнения ОЗУ (рисунок 8). Размер датафрейма снижается за счет фильтрации по значению столбца «simulationRun», равному 45. Таким образом, в датафрейме остаются только те строки, где данный параметр равен 45.

```
reduced_data = df[df['simulationRun']==45]
reduced_data.shape
```

Рисунок 8 – Снижение размера датафрейма

Затем рассмотрите содержимое полученного датафрейма с использованием вывода первых 5 записей (рисунок 9).

	faultNumber	simulationRun	sample	xmeas_1	xmeas_2	xmeas_3	xmeas_4	xmeas_5	xmeas_6	xmeas_7	...
440000	1.0	45.0	1	0.24930	3680.9	4542.4	9.3830	27.074	42.551	2703.2	...
440001	1.0	45.0	2	0.24940	3659.3	4507.5	9.4889	26.396	42.000	2705.1	...
440002	1.0	45.0	3	0.25103	3668.2	4492.4	9.3512	27.160	42.014	2705.9	...
440003	1.0	45.0	4	0.25400	3656.8	4521.1	9.2987	26.992	42.221	2706.9	...
440004	1.0	45.0	5	0.24469	3600.2	4581.0	9.3536	26.845	42.100	2707.4	...

Рисунок 9 – Вывод первых 5 записей датафрейма

После этого необходимо произвести нормализацию полученных данных и определить выборки для обучения и тестирования. Соответственно, переменных получится 4. Для этого выполните код, представленный на рисунке 10.

Теперь нужно определить архитектуру модели, которая будет использоваться в лабораторной работе. Проинициализируйте архитектуру

модели (рисунок 11). Архитектура нейронной сети представляет собой три полносвязных слоя. На вход нейронной сети подается переменная, представляющая собой размерность массива тренировочных данных. В рамках первых двух слоев задается 500 нейронов с функцией активации «selu». Третий слой является выходным. С использованием «np.unique» определяется необходимое количество нейронов для классификации, равное количеству классов. Функцией активации на выходе нейронной сети является «softmax». Функцией потерь является «SparseCategoricalCrossentropy».

Оптимизатором выбран «Adam» со скорость обучения, равной 0.001. Метрикой точности выбрана «accuracy».

```
# Установка и инициализация метода нормализации по стандартным отклонениям
from sklearn.preprocessing import StandardScaler, MinMaxScaler
sc = StandardScaler()
sc.fit(reduced_data.iloc[:,3:])

# Записываем признаки и метки в соответствующие переменные X и Y
X = reduced_data[reduced_data['sample']>20].iloc[:,3:]
Y = reduced_data[reduced_data['sample']>20]['faultNumber'].values

# Выводим информацию о размерностях считанных данных и количестве классов
print(X.shape, Y.shape)
print(np.unique(Y), len(np.unique(Y)))

# Разделяем считанный набор данных на обучающую и тестовую выборки
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Производим нормализацию признаков для обоих выборок
x_train_scaled = sc.transform(x_train)
x_test_scaled = sc.transform(x_test)

# Выводим информацию для каждой из выборок
print("Shape of x_train:", x_train_scaled.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of x_test:", x_test_scaled.shape)
print("Shape of y_test:", y_test.shape)
```

Рисунок 10 – Нормализация данных

```
import tensorflow as tf

# Определеяем слой считывания данных
inputs = tf.keras.layers.Input(shape=(x_train.shape[1],))

# Определяем скрытые слои с 500 нейроннами на каждом из слоёв
hidden_layer = tf.keras.layers.Dense(500, activation='selu')(inputs)
hidden_layer = tf.keras.layers.Dense(500, activation='selu')(hidden_layer)
# Определяем выходной слой, количество нейроннов на котором равно количеству классов
# А так же, указав в качестве функции активации softmax
outputs = tf.keras.layers.Dense(len(np.unique(Y)), activation='softmax')(hidden_layer)

# Определяем и компилируем итоговую модель НС
model = tf.keras.models.Model(inputs=inputs, outputs=outputs)
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), metrics=['accuracy'])

model.summary()
```

Рисунок 11 – Архитектура модели

После этого проведите обучение модели на чистых данных (рисунок 12).

```
from keras.callbacks import EarlyStopping

# Задаём параметры ранней остановки, в случае если модель не улучшается на протяжении 5 эпох
# В качестве отслеживаемого параметра выберем значение функции потерь на валидации
early_stop = EarlyStopping(monitor='val_loss', patience=5)

# Запустим обучение модели
history = model.fit(x_train_scaled, y_train, epochs=100, batch_size=256,
                    validation_data=(x_test_scaled, y_test), callbacks = [early_stop])

# Вывод графиков обучения
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Value')
plt.legend()
plt.show()

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Value')
plt.legend()
plt.show()
```

Рисунок 12 – Обучение модели на чистых данных

В рамках обучения выводятся графики, как на рисунках 13, 14.

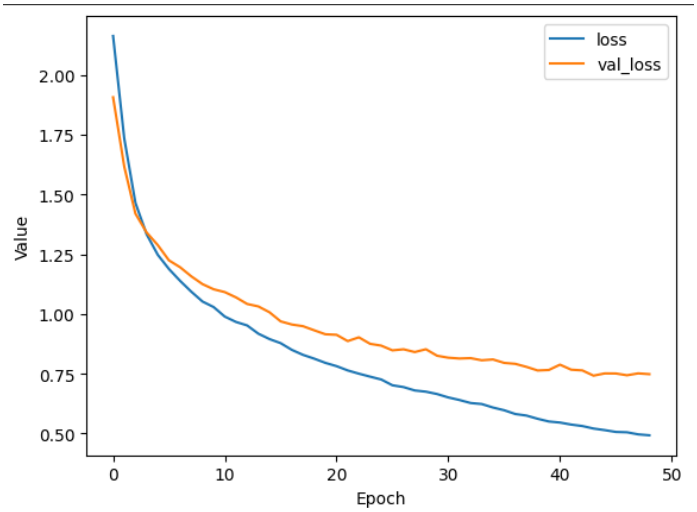

Рисунок 13 – График изменения потерь в рамках обучения нейронной сети

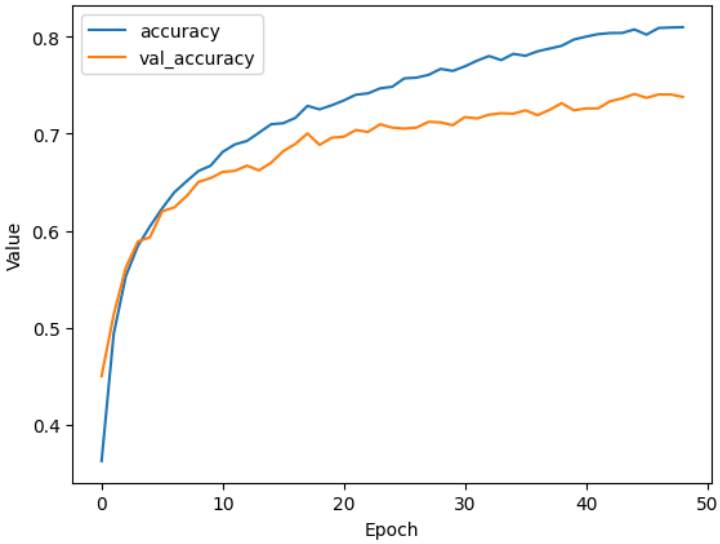

Рисунок 14 – График изменения точности в рамках обучения нейронной сети

После обучения протестируйте работу модели (рисунок 15).

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

preds=model.predict(x_test_scaled)
preds_classes=np.argmax(preds,axis=1)

print(classification_report(y_test, preds_classes))

cm = confusion_matrix(y_test, preds_classes)

plt.figure(figsize = (10,8))
sns.heatmap(cm, annot = True, fmt = 'd')
plt.title('Confusion Matrix')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```

Рисунок 15 – Тестирование модели на чистых данных

Получите метрики точности (рисунок 16) и матрицу запутанности (рисунок 17).

	precision	recall	f1-score	support
0.0	0.05	0.06	0.05	90
1.0	1.00	0.99	1.00	105
2.0	0.96	0.99	0.97	97
3.0	0.11	0.13	0.12	94
4.0	0.96	0.89	0.92	103
5.0	1.00	0.99	0.99	89
6.0	1.00	1.00	1.00	74
7.0	1.00	1.00	1.00	110
8.0	0.90	0.93	0.91	81
9.0	0.04	0.04	0.04	91
10.0	0.97	0.88	0.92	96
11.0	0.72	0.76	0.74	89
12.0	0.94	0.95	0.95	101
13.0	0.95	0.92	0.94	90
14.0	0.97	0.92	0.94	99
15.0	0.06	0.09	0.07	94
16.0	0.90	0.83	0.86	94
17.0	0.91	0.85	0.88	114
18.0	0.88	0.89	0.88	98
19.0	0.71	0.54	0.62	105
20.0	0.90	0.74	0.81	102
accuracy			0.74	2016
macro avg	0.76	0.73	0.74	2016
weighted avg	0.76	0.74	0.75	2016

Рисунок 16 – Метрики точности

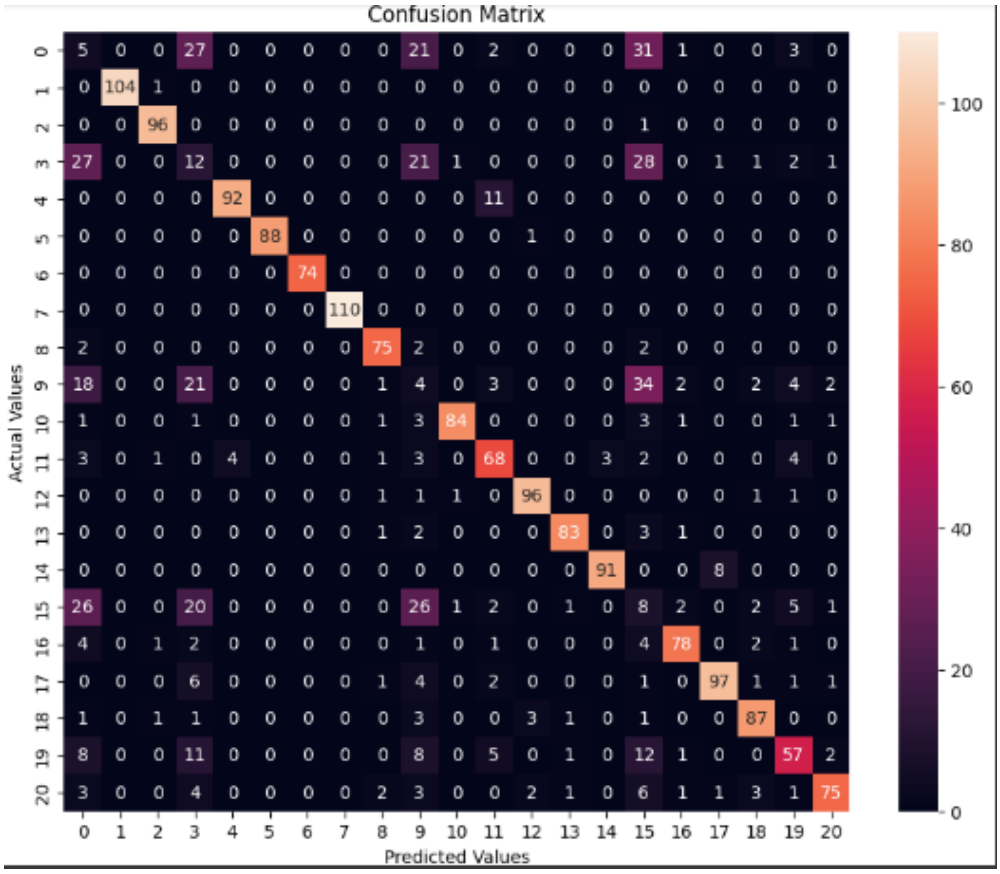

Рисунок 17 – Матрица запутанности

2 СОСТЯЗАТЕЛЬНЫЕ АТАКИ

Необходимо реализовать состязательную атаку, направленную на набор данных. Для этого требуется изменить каждый элемент массива на eps в процентном соотношении. Также нужно учесть параметр $step$, когда нужно будет менять только определенные значения внутри элемента массива. Также следует отметить, что изменение на eps должно быть положительным или отрицательным, т.е. это должно происходить с использованием случайного выбора. Таким образом, в массиве будут и уменьшенные значения, и увеличенные. Начальный код представлен на рисунке 18. Реализуйте функцию состязательной атаки под названием *attack*. Функция принимает на вход массив данных и для каждого из значений массива изменяет его на заданное значение eps в процентном соотношении в положительную или отрицательную сторону.

```
def attack(arr, eps, step):
  # Функция принимает на вход массив данных и для каждого из значений массива
  # Изменяет его на заданное значение eps в процентном соотношении
  # в положительную или отрицательную сторону
```

Рисунок 18 – Функция для реализации атаки

После реализации кода для проведения состязательной атаки проведите нормализацию полученных массивов (рисунок 19).

```
# Проводим нормализацию атакованных данных
x_train_attack_scaled = sc.transform(x_train_attack)
x_test_attack_scaled = sc.transform(x_test_attack)

print("Shape of x_train:", x_train_attack_scaled.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of x_test:", x_test_attack_scaled.shape)
print("Shape of y_test:", y_test.shape)
```

Рисунок 19 – Нормализация атакованных данных

Проведите инициализацию функцию для отрисовки сигнала. Получите сигналы для чистых данных и атакованных данных, как на рисунках 20, 21.

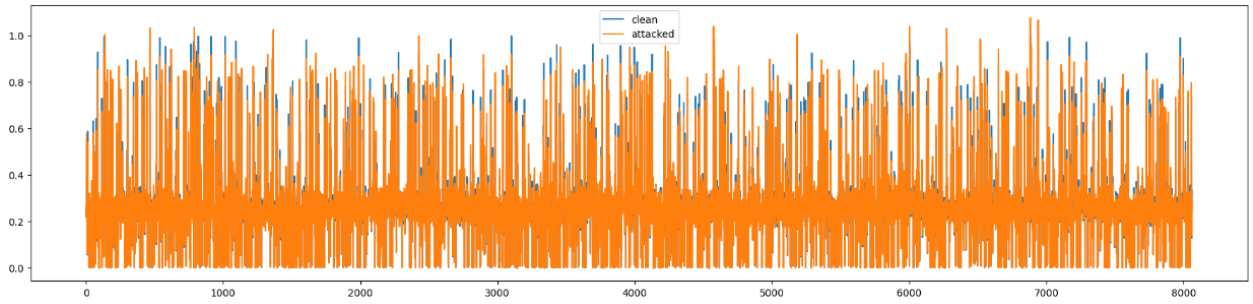

Рисунок 20 – Сигнал в рамках всего тренировочного набора данных

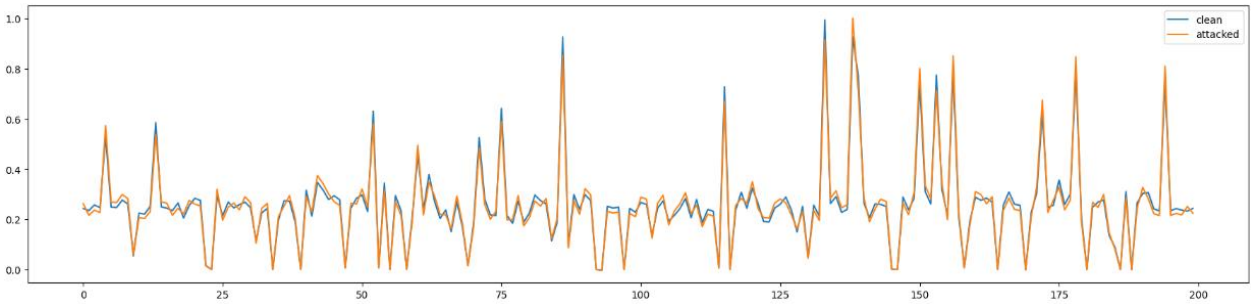

Рисунок 21 – Сигнал на первых 200 записях в тренировочном наборе данных

Сравните чистый сигнал и атакованный, рассмотрите отличия между ними.

После этого проинициализируйте новую модель для проведения обучения на атакованных данных. Архитектура и обучение модели остаются теми же. Затем проведите обучение на атакованных данных. Протестируйте работу модели, которая обучена на атакованных данных.

Сравните метрики точности с метриками, полученными на чистых данных. После этого проверьте работу модели на атакованных тестовых данных. Сравните результаты работы модели на чистых данных, на атакованных обучающих данных и на атакованных тестовых данных.

3 ОТРАВЛЕНИЕ ВЕСОВ МОДЕЛИ

Необходимо взять модель, которая ранее была обучена на чистых данных, и изменить веса на eps, затем проверить работу модели на тестовых данных. Для этого нужно получить веса, полученные при обучении модели. На рисунке 22 представлен блок, где необходимо реализовать отравление весов модели. С помощью *Keras* можно напрямую взять веса слоев модели или всей модели, а также изменить веса слоя или модели. Для получения или

изменения весов существуют готовые функции, встроенные в данной библиотеке.

```
# Реализуйте атаку путем изменения весов обученной ранее модели с учетом варианта
# Необходимо изменять веса в положительную или отрицательную сторону в
# Абсолютной величине eps

list_weights =
```

Рисунок 22 – Отравление весов

После реализации отравления весов протестируйте работу модели.

Сравните полученные метрики точности с метриками точности после проведения других атак и на чистом наборе данных. По итогам работы должно быть выполнено следующее:

1. Сравнение метрик точности работы модели на чистых данных и на атакованных обучающих данных;

2. Сравнение метрик точности работы модели на чистых данных и на атакованных тестовых данных;

3. Сравнение метрик точности работы модели с чистыми весами и с отравленными весами модели.

Все результаты представьте в отчете с выводами и описанием анализа полученных результатов.

Контрольные вопросы

1. Что такое временной ряд?

2. Из каких элементов состоят временные ряды? В каких областях используются временные ряды?

3. Суть атак отравлением.

4. Каким образом снижается размер датафрейма? Для чего это необходимо выполнять?

5. Какие функции библиотеки *Keras* позволяют получить список весов модели и поменять их?

Указания для организации самостоятельной работы

Целями самостоятельной работы являются систематизация, расширение и закрепление теоретических знаний.

Самостоятельная работа студента по дисциплине «Безопасность информационно аналитических систем» включает следующие виды активности:

1. Изучение тем теоретической части дисциплины, вынесенных для самостоятельной проработки.

2. Подготовка к лабораторным работам.

3. Подготовка реферата.

4. Выполнение индивидуальных заданий.

Изучение тем теоретической части дисциплины осуществляется на основе материала лекционных занятий.

В рамках выполнения подготовки к лабораторным работам рекомендуется детально познакомиться с теоретическим материалом по темам лабораторных работ, а также с последовательностью действий выполнения лабораторных работ, указанных в методических указаниях.