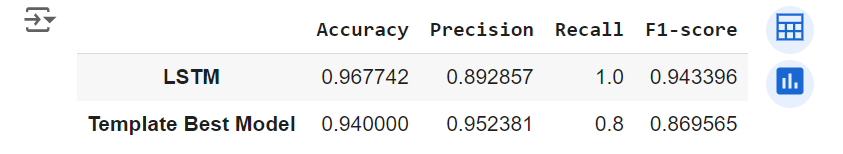
***Лучшая прогонка с 2000***



***# build the network***

nb\_features = seq\_array.shape[2] # Количество признаков (features) в последовательности

nb\_out = label\_array.shape[1] # Количество выходных классов (labels)

Sequential: Это класс, который используется для создания моделей, состоящих из последовательности слоев. Модель строится слой за слоем, где каждый последующий слой получает вход от предыдущего слоя.

model = Sequential()

LSTM (Long Short-Term Memory) – это рекуррентная нейронная сеть, которая способна обрабатывать временные ряды данных и сохранять информацию о предыдущих шагах времени. Она хорошо подходит для задач обработки последовательностей, таких как обработка текста, временных рядов и др.

# Добавление первого слоя LSTM

model.add(LSTM(

input\_shape=(sequence\_length, nb\_features), # Форма входных данных: длина последовательности и количество признаков

units=100, # Количество единиц (нейронов) в слое LSTM

return\_sequences=True # Возвращать ли полные последовательности (True) или только последний выход (False)

model.add(Dropout(0.2)) # Удаление 20% нейронов для предотвращения переобучения

# Добавление второго слоя LSTM

model.add(LSTM(

units=50, # Количество единиц (нейронов) в втором слое LSTM

return\_sequences=False)) # Возвращать только последний выход

model.add(Dropout(0.2)) # Удаление 20% нейронов

Dense: Это полносвязный слой, который соединяет все нейроны предыдущего слоя со всеми нейронами текущего слоя. В данном случае он служит выходом нашей модели.

model.add(Dense(units=nb\_out, activation='sigmoid')) # Полносвязный слой с активацией sigmoid для бинарной классификации

model.compile(loss='binary\_crossentropy', optimizer='adam', # Функция потерь для бинарной классификации # Оптимизатор Adam # Метрика для оценки точности

metrics=['accuracy'])

- loss: Функция потерь, которую оптимизирует модель. В данном случае используется 'binary\_crossentropy', что подходит для задачи бинарной классификации.

- optimizer: Алгоритм оптимизации, который используется для минимизации функции потерь. Здесь выбран Adam, эффективный алгоритм градиентного спуска, который автоматически настраивает скорость обучения для разных параметров.

В выводе model.summary() представлена информация о слоях модели, их выходных формах и количестве параметров. Давайте разберем, как считается количество параметров для каждого слоя:

1. **LSTM слой (lstm\_1)**:
   * **Форма выхода**: (None, 50, 100) - это означает, что LSTM будет обрабатывать последовательности длиной 50 и возвращать вектор размерности 100 для каждой временной точки.
   * **Количество параметров**: 50400.
   * Для LSTM слоя количество параметров рассчитывается по формуле:

Param=4×(units×(input\_dim+1))

где:

* + - units — количество нейронов в LSTM (в данном случае 100),
    - input\_dim — количество входных признаков (в данном случае не указано, но можно предположить, что это 1, если на вход подается одномерный сигнал).  
      Подставляя значения:

Param=4×(100×(input\_dim+1))=4×(100×(1+1))=4×200=800

Таким образом, для LSTM слоя с 100 нейронами и 1 входным признаком общее количество параметров будет 800, но так как мы имеем 50 временных шагов, это значение умножается на 50, что в итоге дает 50400.

1. **Dropout слой (dropout\_1)**:
   * **Количество параметров**: 0.
   * Dropout не имеет обучаемых параметров, он просто отключает определенный процент нейронов во время тренировки.
2. **LSTM слой (lstm\_2)**:
   * **Форма выхода**: (None, 50) - этот слой возвращает вектор размерности 50 для каждого временного шага.
   * **Количество параметров**: 30200.
   * Аналогично, для второго LSTM слоя:

Param=4×(50×(100+1))=4×(50×101)=4×5050=20200

Это значение также умножается на 1, так как у нас один временной шаг, и общее количество параметров будет 30200.

1. **Dropout слой (dropout\_2)**:
   * **Количество параметров**: 0.
   * Как и в первом случае, этот слой не имеет параметров.
2. **Dense слой (dense\_1)**:
   * **Форма выхода**: (None, 1) - этот слой возвращает одно значение.
   * **Количество параметров**: 51.
   * В полносвязном слое количество параметров рассчитывается по формуле:

Param=input\_dim×output\_dim+output\_dim

где input\_dim — количество входных нейронов (в данном случае 50 от предыдущего LSTM слоя), а output\_dim — количество выходных нейронов (в данном случае 1).  
Подставляя значения:

Param=50×1+1=50+1=51

Таким образом, общее количество параметров модели составляет 80,651, где параметры LSTM слоев и Dense слоя составляют основную часть.

Изучить и описать назначение используемых методов и параметров.

%%time

# fit the network

model.fit(seq\_array, label\_array, epochs=10, batch\_size=200,

validation\_split=0.05, verbose=1,

callbacks = [keras.callbacks.EarlyStopping(monitor='val\_loss',

min\_delta=0, patience=0, verbose=0, mode='auto')]

**Параметры метода fit:**

1. **seq\_array**:
   * Это массив входных данных, на которых будет обучаться модель. В контексте задач временных рядов или последовательностей, это может быть массив последовательностей, которые подаются на вход модели.
2. **label\_array**:
   * Это массив целевых меток (или выходных значений), соответствующих входным данным. Каждое значение в этом массиве соответствует выходу, который модель должна предсказать для соответствующего входа.
3. **epochs=10**:
   * Этот параметр указывает количество полных проходов (эпох) по всему обучающему набору данных. В данном случае модель будет обучаться 10 эпох.
4. **batch\_size=200**:
   * Этот параметр определяет количество образцов, обрабатываемых перед обновлением параметров модели. В данном случае модель будет обрабатывать 200 образцов за один раз. Использование батчей помогает оптимизировать процесс обучения и уменьшить потребление памяти.
5. **validation\_split=0.05**:
   * Этот параметр указывает, какую долю обучающих данных следует выделить для валидации. В данном случае 5% данных будет использовано для оценки производительности модели на валидационном наборе после каждой эпохи. Это помогает предотвратить переобучение.
6. **verbose=1**:
   * Этот параметр управляет выводом информации во время обучения. Значение 1 означает, что будет выводиться информация о прогрессе обучения после каждой эпохи. Это может включать значения потерь и точности.
7. **callbacks**:
   * Этот параметр принимает список обратных вызовов, которые будут вызываться в процессе обучения. В данном случае используется EarlyStopping, который позволяет остановить обучение, если валидационная потеря (val\_loss) не улучшается в течение заданного количества эпох.
   * **keras.callbacks.EarlyStopping**:
     + **monitor='val\_loss'**: Указывает, что нужно следить за валидационной потерей.
     + **min\_delta=0**: Минимальное изменение, которое будет считаться улучшением. В данном случае любое изменение не менее 0 будет считаться улучшением.
     + **patience=0**: Количество эпох, в течение которых нужно ждать улучшения. Если валидационная потеря не улучшается в течение 0 эпох, обучение будет остановлено немедленно.
     + **verbose=0**: Уровень подробности вывода. Значение 0 означает, что не будет выводиться информация о срабатывании обратного вызова.
     + **mode='auto'**: Режим, который автоматически определяет, следует ли минимизировать или максимизировать значение, в зависимости от того, что отслеживается (в данном случае val\_loss).