Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра безопасности информационных систем (БИС)

Аутентификация по параметрам динамики простановки подписи на графическом планшете

Аналитический отчет по курсу повышения квалификации «Методы анализа и прогнозирования данных»

Слушатель

М.В. Гурова

13 декабря 2022

Принял

канд. техн. наук,

доцент кафедры КИБЭВС

Е.Ю. Костюченко

___ декабря 2022

Томск 2022

Содержание

Задание

Для анализа изначально используется датасет, представляющий собой набор данных подписей пользователей, по несколько десятков на каждого пользователя, представленные в виде 144 параметров подписи, дающих полную информацию о динамике написания подписи (файл sign_dump_part.csv). Однако, для выполнения данного задания предполагается их попарное сравнение. Для этого набор обрабатывается следующим образом:

Исходный набор
[Параметры записи 1] [Принадлежность записи 1] [Параметры записи 2] [Принадлежность записи 2] … [Параметры записи n] [Принадлежность записи n]

Итоговый набор
[Параметры записи 1] [Параметры записи 1][1] [Параметры записи 1] [Параметры записи 2][0, если Принадлежность записи 1!= Принадлежность записи 2, иначе 1] … [Параметры записи i] [Параметры записи j][0, если Принадлежность записи i!= Принадлежность записи j, иначе 1] [Параметры записи n] [Параметры записи n][1]

Таким образом, проводится попарное комбинирование параметров и если записи в комбинации принадлежат одному пользователю, то выход 1, иначе 0.

В данной работе проведем исследование данных тремя различными классификаторами в рамках анализа набора данных для решения задачи классификации на два класса: 0 – половины параметров от разных пользователей и 1 – половины параметров от одного и того же пользователя. Выполнение будет проводиться в интерактивной среде разработки Jupyter Notebook.

Набор данных

Для формирования итогового набора данных был выбран файл sign_dump_part.csv, после чего в сторонней среде Google Colab в результате выполнения кода, представленного в файле Create_Full.ipynb, был порожден набор данных размером более 9 Гб.

Размерность полученного набора данных составляет 2045*2045 записей из 144+144+1 полей, итого размер набора равен 4 182 025*289.

Учитывая большой размер набора данных, ограниченность ресурсов, используемых для его обработки, а также однородность по процедуре построения, работа по построению классификаторов проводилась не на всем наборе, а на его части – 100 000 записей, перемешанных для случайного порядка.

Процедура обрезки набора с перемешиванием и выбором размера приведена в файле Create_Small.ipynb, в результате которого мы получаем итоговый набор размера 100 000*289, для которого и будем в дальнейшем строить классификаторы. Последний столбец в этом наборе – зависимая переменная, остальные столбцы – независимые переменные (параметры подписи).

Стоит отметить сильный дисбаланс данных, полученный в наборе – количество примеров, принадлежащих разным пользователям, значительно превышает количество примеров, принадлежащих одинаковым пользователям (практически в 10 раз).

Итоговый набор данных также был разделен на тестовую и обучающую выборки, которые соответственно использовались в дальнейшем для непосредственного обучения модели и для оценки ее качества. Разделение проводили в соотношении 30% (тестовая выборка) на 70% (обучающая выборка) с сохранением пропорции классов (с помощью задания параметра stratify функции train_test_split).

Используемые классификаторы, их архитектура и параметры построения

В качестве методов для построения классификаторов были выбраны модели на основе нейронной сети, а также ансамблевые методы, которые довольно часто дают лучшие результаты для построения моделей машинного обучения, а также отличаются относительным быстродействием, что особенно важно при ограниченных вычислительных ресурсах. В качестве ансамблевых методов были использованы адаптивный и градиентный бустинги (AdaBoost и GradientBoosting). Разница между этими бустингами заключается в том, как алгоритмы идентифицируют слабые модели. AdaBoost выявляет их на основании высоких значений весов, а GradientBoosting – на основании градиентов функции потерь.

Пример кода выполнения представлен в файле Itog_Case.ipynb.

Классификатор на основе нейронной сети

Первоначально было запущено автомоделирование Deep Learning на платформе RapidMiner, однако построенная в результате модель обладала меньшей точностью, чем удалось достичь при подборе параметров вручную.

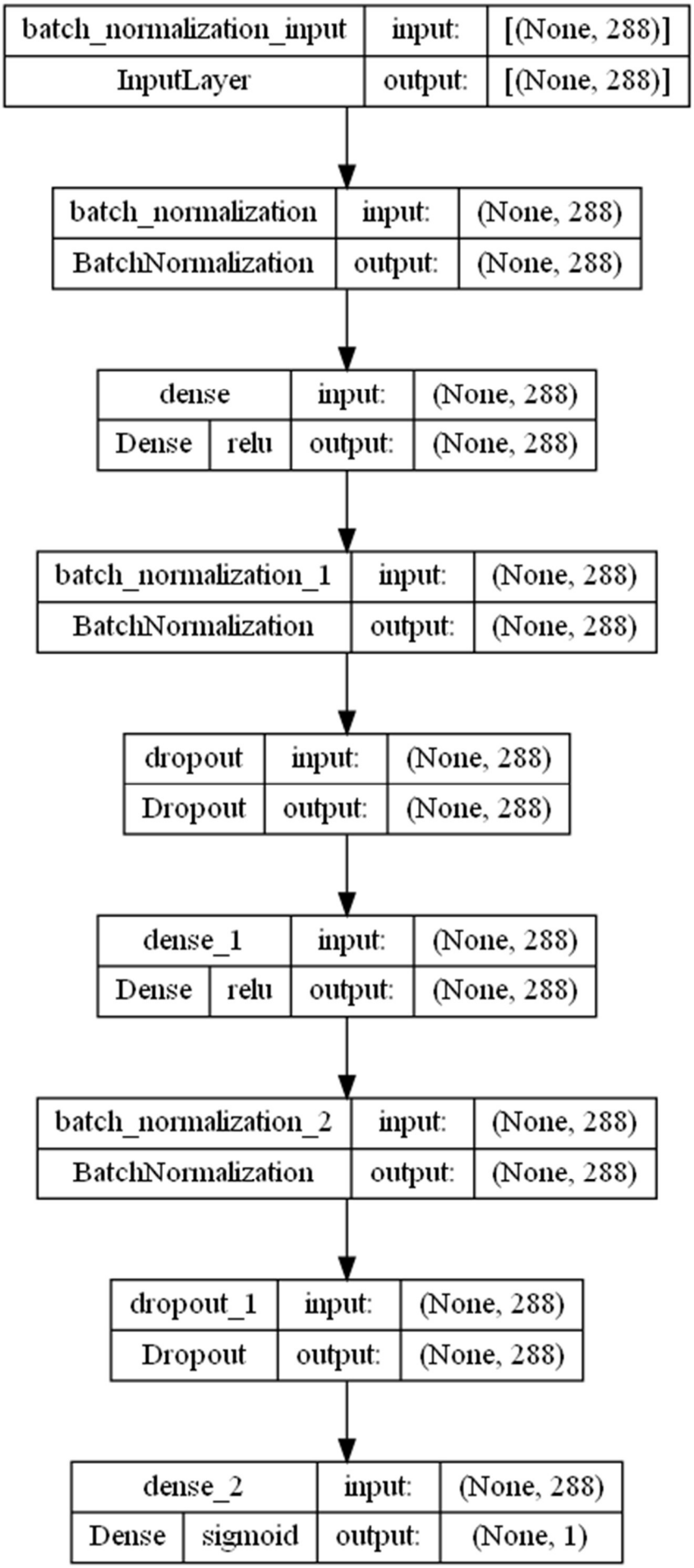
Построение классификатора на основе нейронной сети:

```
model = Sequential()
model.add(BatchNormalization(input_shape=(288,)))

model.add(Dense(288, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(288, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=[f1_m])
```

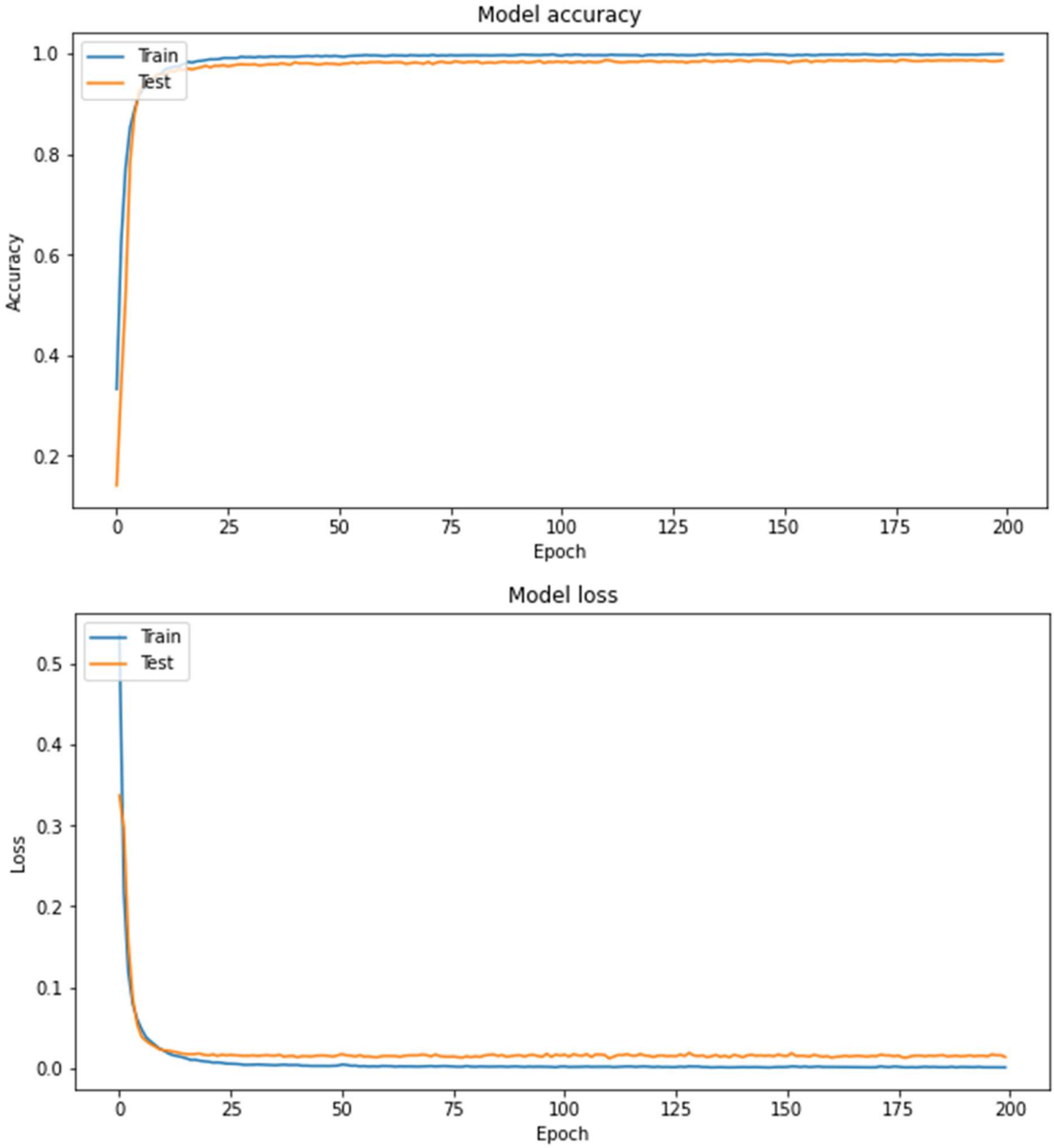
Обучение классификатора на основе нейронной сети:

```
results = model.fit(X_train,y_train,validation_data = (X_test,y_test),batch_size = 1000,epochs =200)
```

Рассмотрим наглядно архитектуру данной нейронной сети:


batch_normalization_input
input: [(None, 288)]
InputLayer
output: [(None, 288)]
batch_normalization
input: (None, 288)
BatchNormalization
output: (None, 288)
dense
input: (None, 288)
Dense
relu
output: (None, 288)
batch_normalization_1
input: (None, 288)
BatchNormalization
output: (None, 288)
dropout
input: (None, 288)
Dropout
output: (None, 288)
dense_1
input: (None, 288)
Dense
relu
output: (None, 288)
batch_normalization_2
input: (None, 288)
BatchNormalization
output: (None, 288)
dropout_1
input: (None, 288)
Dropout
output: (None, 288)
dense_2
input: (None, 288)
Dense
sigmoid
output: (None, 1)

Отрисуем показатели точности и потерь построенной нейронной сети:



По графикам видно, что построенная нейронная сеть достаточно точно выполняет поставленную задачу классификации, что подтверждается вычисленным значением F-меры, равным 0.987.

Значение F-меры для классификатора на основе нейронной сети:

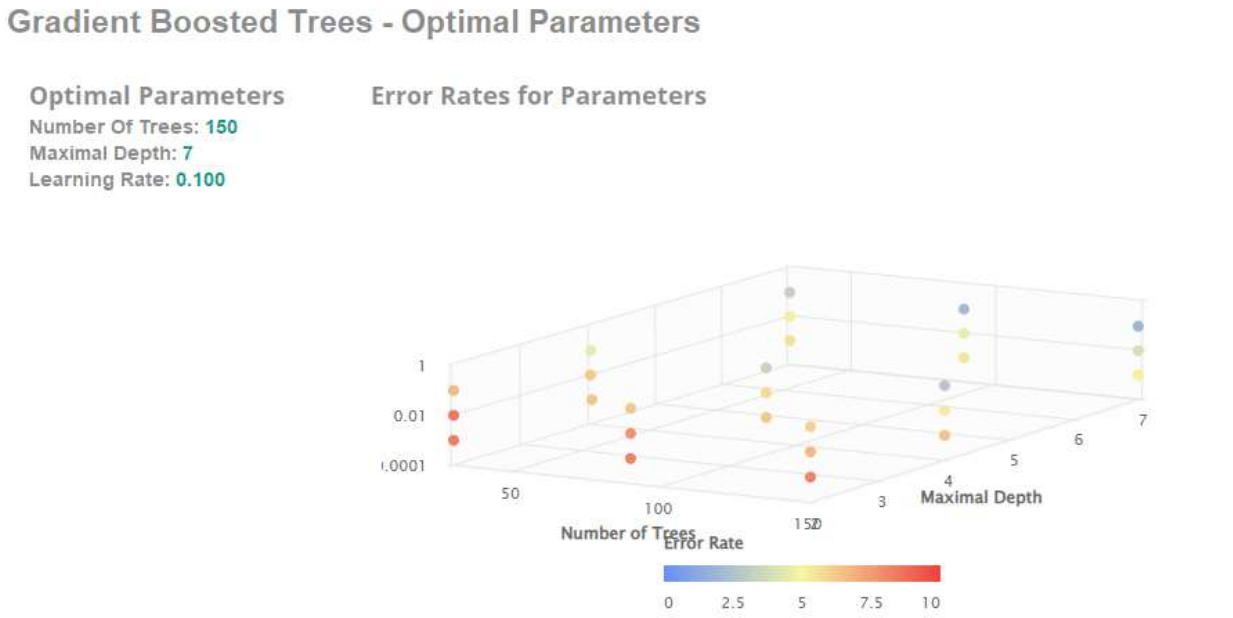
```
f1_nn = results.history['val_f1_m'][-1]
np.round(f1_nn, 3)
```

Классификатор на основе Gradient Tree Boosting (градиентный бустинг)

Оптимальные параметры для данного классификатора были получены с помощью платформы для анализа данных RapidMiner. Автомоделирование методом Gradient Tree Boosting на нашем наборе данных показало наилучший результат (значение F-меры 98.2%) при следующих параметрах:

- количество деревьев – 150;
- максимальная глубина – 7;
- скорость обучения – 0.1.

Результат работы автомоделирования RapidMiner представлен ниже:



Number of Trees	Maximal Depth	Learning Rate	Error Rate
150	4	0.100	2.8%
30	7	0.100	3.2%
90	7	0.100	2.1%
150	7	0.100	1.9%

Построение классификатора на основе Gradient Tree Boosting:

```
from sklearn.ensemble import GradientBoostingClassifier

model_gb = GradientBoostingClassifier(random_state = random_state, learning_rate = 0.1, n_
estimators = 150, max_depth = 7)
```

Обучение классификатора на основе Gradient Tree Boosting:

```
model_gb.fit(X_train, y_train)
```

Значение F-меры для классификатора на основе Gradient Tree Boosting:

```
f1_gb = f1_m(y_test, np.round(model_gb.predict_proba(X_test)[:, 1]))
```

```
f1_gb.numpy()
```

Построенный классификатор также хорошо выполняет поставленную задачу, значение F-меры = 0.933.

Классификатор на основе Adaptive Boosting (адаптивный бустинг)

Построение и обучение классификатора на основе Adaptive Boosting:

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold

parameters = {'base_estimator__max_depth':[5,6,7,8],
              'n_estimators':[150,200,250,300]
              }

adc = AdaBoostClassifier(base_estimator=DecisionTreeClassifier())

model_ad = RandomizedSearchCV(adc, parameters,n_iter=5, cv=StratifiedKFold(n_splits=3,
shuffle=True, random_state=random_state), random_state=random_state, scoring='f1',
n_jobs=-1)
model_ad.fit(X_train,y_train)

print('Best params', model_ad.best_params_)
print('Best score', model_ad.best_score_)
```

Значение F-меры для классификатора на основе Adaptive Boosting:

```
f1_ad = f1_m(y_test, np.round(model_ad.predict_proba(X_test)[:, 1]))
f1_ad.numpy()
```

С помощью рандомизированного поиска были подобраны следующие параметры для модели адаптивного бустинга, при которых значение F-меры получилось равным 0.907:

- количество деревьев – 200;
- максимальная глубина – 5.

В качестве базового алгоритма были выбраны деревья решений.

Классификатор дает неплохой результат, что подтверждается значением F-меры и матрицей неточности, однако возможно будет достигнут лучший результат при применении поиска по сетке, который требует значительно больших вычислительных ресурсов и затрат времени (особенно при расширении пространства параметров, в том числе использование другого базового алгоритма вместо деревьев решений).

Результаты и их анализ

В результате проведенного эксперимента по подбору параметров были получены три значения F-меры по каждому из классификаторов, построенных на основе нейронной сети и ансамблевых методов AdaBoost и GradientBoosting:

Классификатор	Значение F-меры
Классификатор на основе нейронной сети	0.987
Классификатор на основе Gradient Tree Boosting	0.933
Классификатор на основе Adaptive Boosting	0.907

Видно, что исходя из перебранного множества значений можно рекомендовать выбор классификатора на основе нейронной сети, обеспечивающего значение F-меры, равном 0.987.