ТУСУР
TUSUR UNIVERSITY

Аутентификация по параметрам динамики простановки подписи на графическом планшете.

Выполнила:
Михалькевич А.Э.

Декабрь 2022

Введение

Объект исследования: Набор данных с параметрами подписей пользователя.
Предмет исследования: Попарное сравнение параметров и их бинарная классификация по принадлежности одному пользователю.

Цель работы: Необходимо провести исследование минимум 3 различными классификаторами в рамках анализа набора данных для решения задачи классификации на 2 класса – 0 – половины параметров от разных пользователей и 1 – половины параметров от одного и того же пользователя.

Задачи работы:

1. Сформировать и описать набор данных.
2. Создать и обучить модель нейронной сети, выбрать параметры.
3. Выбор классификаторов.
4. Балансировка данных.
5. Сравнить результаты классификации.
6. Заключение.

Формирование и описание набора данных

Размерность исходного набора данных составляет 2045*2045 запись из 144+144+1 полей, итого размер набора равен 4 182 025*289. Т.к. набор данных достаточно большой и в памяти помещается с трудом итоговую оценку буду проводить на 100000 записей. Процедура обрезки набора с перемешиванием и выбором размера приведена в файле 2_Create_Small.ipynb.

В результате обрезки получаем набор данных Small.csv на 100000 записей на основании которого проводим анализ данных представленный в приложенном блокноте.

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]

Создание нейронной сети

Создадим модель нейронной сети, состоящую из нескольких внутренних слоев и одного выходного слоя.

```
model = Sequential(
    [
        Dense(units=512, activation="relu", input_shape=(X_train.shape[-1],) ),
        BatchNormalization(),
        Dropout(0.3),
        Dense(units=256, activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(units=128, activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(units=32, activation="relu" ),
        BatchNormalization(),
        Dropout(0.3),
        Dense(units=1, activation="sigmoid"),
    ]
)
```

Параметры сети

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               147968

 batch_normalization (BatchN  (None, 512)              2048
 ormalization)

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 128)               32896

 batch_normalization_2 (Batc  (None, 128)              512
 hNormalization)

 dropout_2 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 32)                4128

 batch_normalization_3 (Batc  (None, 32)               128
 hNormalization)

 dropout_3 (Dropout)         (None, 32)                0

 dense_4 (Dense)             (None, 1)                 33

=================================================================
Total params: 320,065
Trainable params: 318,209
Non-trainable params: 1,856
_________________________________________________________________
```

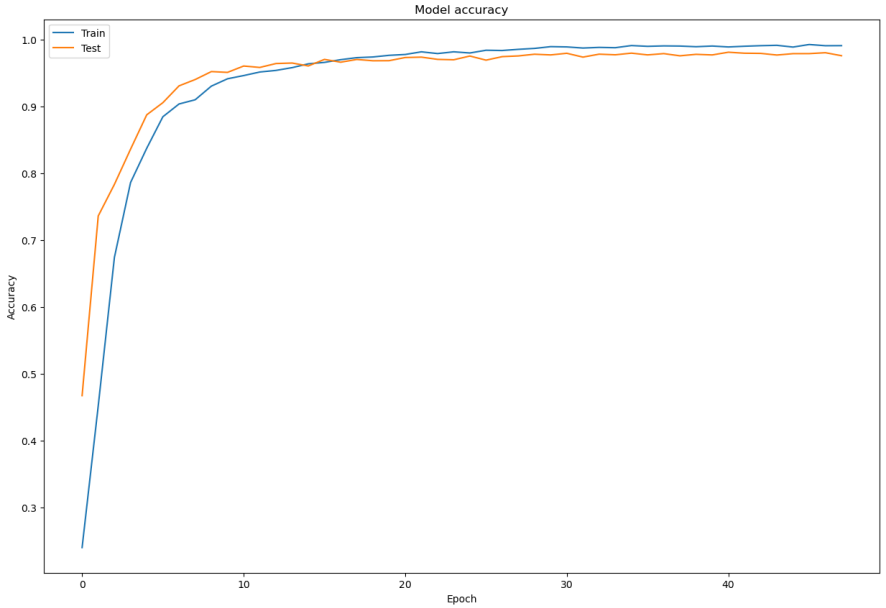
Результаты обучения модели

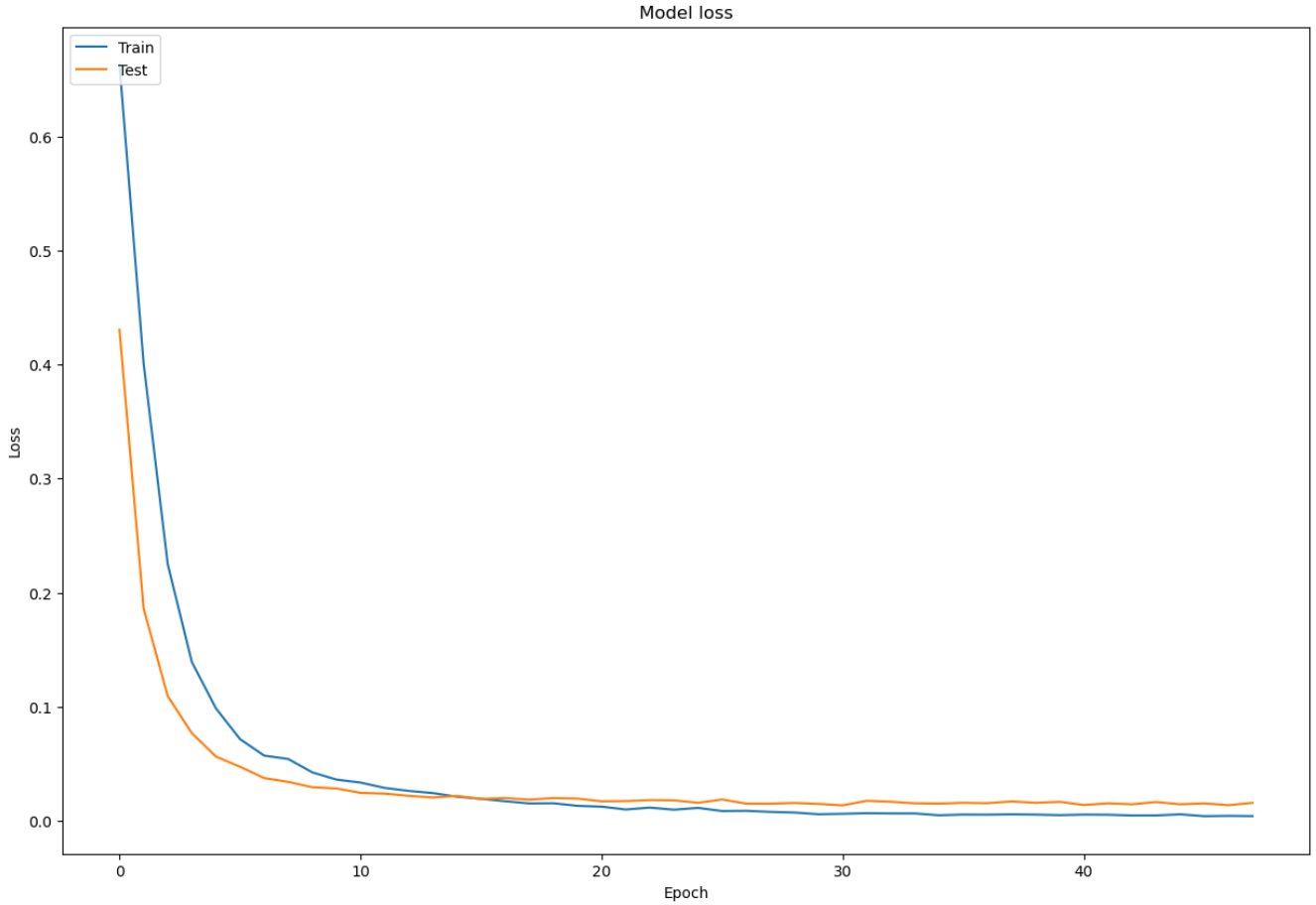
Обучаю модель используя следующие параметры:
batch_size = 1024, epochs=300, callbacks=[early_stopping, checkpoint]

```
[ ]  f1_1 = f1_m(y_true=y_test,
                 y_pred=predict_keras).numpy()

     print(f"F1 score: {f1_1}")

     F1 score: 0.981712576941288
```





KNeighborsClassifier

TUSUR UNIVERSITY

KNeighbors (K-ближайших соседей): Простой алгоритм, который классифицирует объект на основе голосования ближайших соседей. Чем ближе объекты, тем больше вероятность их схожести.

К ближайших соседей

```python
knn_model = KNeighborsClassifier()

parameters = {
    'n_neighbors': range(1, 10)
}

grid_search_knn = HalvingGridSearchCV(knn_model, parameters, cv=3, n_jobs=-1,
                                      scoring='f1', verbose=False)
grid_search_knn.fit(X_train, y_train)

print(f"Best score: {abs(grid_search_knn.best_score_)}, Best params: {grid_search_knn.best_params_}")

f1_2 = f1_m(y_test, K.round(grid_search_knn.predict_proba(X_test)[:,1])).numpy()
print(f"Test: {f1_2}")
```

Python

```
Best score: 0.658302518473915, Best params: {'n_neighbors': 9}
Test: 0.7090642810321208
```

DecisionTreeClassifier

TUSUR UNIVERSITY

DecisionTree (Дерево решений): Модель, которая принимает решения, разделяя данные на ветви на основе условий. Проста в интерпретации, но склонна к переобучению.

Деревья решений

```python
model_tree = DecisionTreeClassifier(random_state=RANDOM_STATE)

parameters = {
    'min_samples_split': range(2, 4),
    'min_samples_leaf': range(1, 4),
    'max_depth': range(20, 40, 5)
}

grid_search_tree = HalvingGridSearchCV(model_tree, parameters, cv=3, n_jobs=-1,
                                       scoring='f1', verbose=False)
grid_search_tree.fit(X_train, y_train);

print(f"Best score: {abs(grid_search_tree.best_score_)}, Best params: {grid_search_tree.best_params_}")

f1_3 = f1_m(y_test, K.round(grid_search_tree.predict_proba(X_test)[:,1])).numpy()
print(f"Test: {f1_3}")
```

Python

```
Best score: 0.7933024601594761, Best params: {'max_depth': 35, 'min_samples_leaf': 2, 'min_samples_split': 3}
Test: 0.8617933799941364
```

RandomForestClassifier

TUSUR UNIVERSITY

RandomForest (Случайный лес): Ансамбль деревьев решений, который снижает риск переобучения за счет усреднения результатов множества деревьев.

Случайный лес (классификация)

```python
model_forest = RandomForestClassifier(random_state=RANDOM_STATE)

parameters = {
    'min_samples_split': range(6, 9),
    'min_samples_leaf': range(9, 11),
    'max_depth': range(130, 150, 10),
    'max_features':['auto', "sqrt"],
    'n_estimators': [100, 200],
    'class_weight': [None, "balanced"]
}
grid_search_forest = HalvingGridSearchCV(model_forest, parameters, cv=3, n_jobs=-1, scoring='f1', verbose=False)
grid_search_forest.fit(X_train, y_train)

print(f"Best score: {abs(grid_search_forest.best_score_)}, Best params: {grid_search_forest.best_params_}")

f1_4 = f1_m(y_test, K.round(grid_search_forest.predict_proba(X_test)[:,1])).numpy()
print(f"Test: {f1_4}")
```

Python

```
Best score: 0.8382497552315754, Best params: {'class_weight': 'balanced', 'max_depth': 130, 'max_features': 'sqrt', 'r
Test: 0.9019115767750931
```

LGBMClassifier

TUSUR UNIVERSITY

LGBM (LightGBM): Градиентный бустинг на основе деревьев, оптимизированный для скорости и производительности. Хорошо работает с большими данными.

Light Gradient Boosted Machine LGBMClassifier

```python
model_light = LGBMClassifier(random_state=RANDOM_STATE)

parameters = {
    'n_estimators': range(300, 500, 50),
    'max_depth': range(9, 14),
    'learning_rate': [0.5, 0.05]
}


grid_search_light = HalvingGridSearchCV(model_light, parameters, cv=3, n_jobs=-1,
                                        scoring='f1', verbose=False)
grid_search_light.fit(X_train, y_train)

print(f"Best score: {abs(grid_search_light.best_score_)}, Best params: {grid_search_light.best_params_}")

f1_5 = f1_m(y_test, K.round(grid_search_light.predict_proba(X_test)[:,1])).numpy()
print(f"Test: {f1_5}")
```

Python

```
Best score: 0.952911893293984, Best params: {'learning_rate': 0.5, 'max_depth': 12, 'n_estimators': 400}
Test: 0.9777517063955328
```

Балансировка данных с помощью метода SMOTE

TUSUR UNIVERSITY

Сделаю балансировку данных с помощью метода SMOTE для трех моделей, чтобы попробовать улучшить результаты классификации.

SMOTE это алгоритм предварительной обработки данных, используемый для устранения дисбаланса классов в наборе данных. SMOTE позволяет увеличить количество примеров миноритарных классов, избегая при этом чрезмерного обучения. В результате создаются новые синтезированные образцы, близкие к другим точкам (принадлежащим к миноритарному классу) в пространстве признаков.

1. Light Gradient Boosted Machine LGBMClassifier SMOTE

Best score: 0.9608880581293929, Best params: {'light__learning_rate': 0.5, 'light__max_depth': 10, 'light__n_estimators': 450} Test: 0.9772195761201705

2. DecisionTreeClassifier SMOTE

Best score: 0.5658607041006616, Best params: {'tree__max_depth': 35, 'tree__min_samples_leaf': 2, 'tree__min_samples_split': 2} Test: 0.6124323825925387

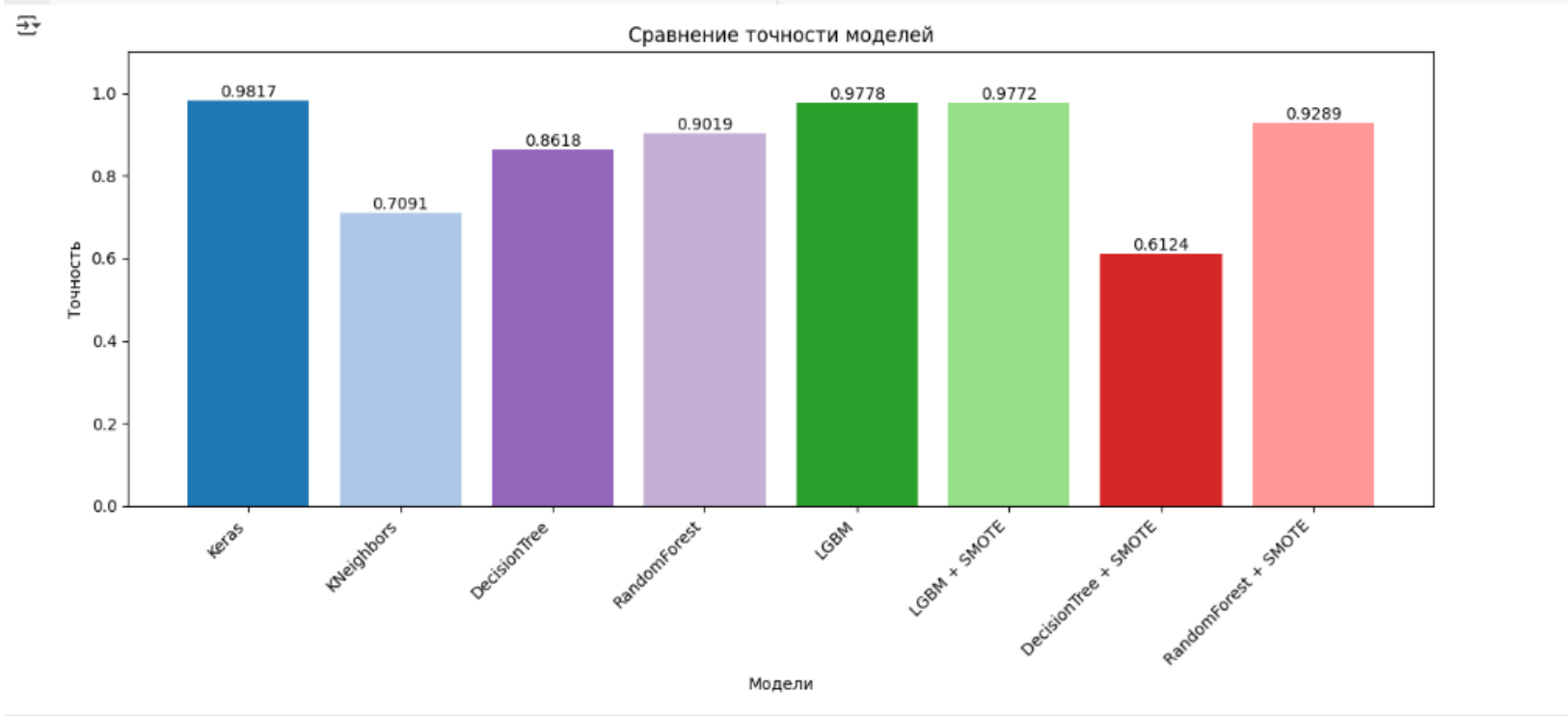
3. RandomForestClassifier SMOTE

Best score: 0.8817099059912076, Best params: {'forest__class_weight': None, 'forest__max_depth': 130, 'forest__max_features': 'sqrt', 'forest__min_samples_leaf': 9, 'forest__min_samples_split': 6, 'forest__n_estimators': 200} Test: 0.9289417006466919

Сравнение результатов классификации.

Наивысшую точность показала модель **Keras** (0.9817), что делает её наиболее эффективной среди всех протестированных моделей.
Модели **LGBM** и **LGBM + SMOTE** также показали высокие результаты (0.9778 и 0.9772 соответственно).
RandomForest и **RandomForest + SMOTE** демонстрируют хорошую точность (0.9019 и 0.9289), что делает их надежными вариантами для решения текущей задачи классификации.
KNeighbors и **DecisionTree** показали более низкие результаты (0.7091 и 0.8618 соответственно), а использование SMOTE для **DecisionTree** значительно снизило точность (0.6124).
В целом, модели на основе нейронных сетей и ансамблевые методы показывают наилучшие результаты, в то время как более простые модели, такие как KNeighbors и DecisionTree, требуют дополнительной настройки или улучшения.

Модель	Точность
Keras	0.9817
KNeighbors	0.7091
DecisionTree	0.8618
RandomForest	0.9019
LGBM	0.9778
LGBM + SMOTE	0.9772
DecisionTree + SMOTE	0.6124
RandomForest + SMOTE	0.9289



Заключение

ТУСУР
TUSUR UNIVERSITY

При решении задачи аутентификации по динамике подписи с применением попарного сравнения образцов на наборе данных с параметрами подписи было проведено исследование с использованием различных классификаторов, а именно: K ближайших соседей (KNeighborsClassifier), Деревья решений(DecisionTreeClassifier), Случайный лес (RandomForestClassifier), LGBMClassifier, а также с применением балансировки данных с помощью SMOTE.

Все классификаторы были протестированы на наборе данных с параметрами подписи.
SMOTE был использован для улучшения качества классификации на несбалансированных данных.
Метрики оценки (Accuracy, Precision, Recall, F1-score, ROC-AUC) показали, что использование балансировки данных и ансамблевых методов (например, RandomForest и LGBM) с помощью SMOTE значительно улучшает качество классификации.

Ансамблевые методы, такие как RandomForest и LGBM, показали себя лучше, чем простые классификаторы, такие как KNN и DecisionTree.

Для задачи аутентификации по динамике подписи важно учитывать дисбаланс классов и использовать методы, которые могут справляться с этим, такие как SMOTE.

В результате проведенного эксперимента по подбору параметров видно, что, исходя из перебранного множества значений можно рекомендовать выбор классификатора на основе LightGBMClassifier (классификатор повышения градиента в машинном обучении, который использует древовидные алгоритмы обучения) обеспечивающий следующие показатели качества работы системы: 0.9777517063955328, а также классификатора RandomForest сбалансированного с помощью метода SMOTE, обеспечивающий следующие показатели качества работы системы: 0.9289417006466919.

ТУСУР
TUSUR UNIVERSITY

Стек технологий:

Языки программирования: Python
Среда разработки: Google Colab
Библиотеки: TensorFlow/Keras, Pandas, NumPy, Matplotlib, Plotly, Seaborn, sklearn
Обученные модели: Keras, KNeighbors, DecisionTree, RandomForest, LGBM, LGBM + SMOTE, DecisionTree + SMOTE, RandomForest + SMOTE.
Балансировка данных: SMOTE
Дополнительные инструменты: Github, OpenAI ChatGPT

ТУСУР
TUSUR UNIVERSITY

Спасибо за внимание!

Мои контакты:

@hakunaaa_matataaaaa
anelia.education@yahoo.com