Sheet

Лабораторная работа №1

"Распознавание активности человека на основе данных с мобильных сенсоров"

Работа выполнена: Осининой Татьяной, студенткой R3237, факультет Систем Управления и Робототехники

Задание:

Необходимо по данным с мобильных сенсоров при помощи прикладных алгоритмов машинного обучения предсказать активность человека по шести классам движений:

- Движется по прямой
- Движется вверх (например, движение по лестнице вверх)
- Движется вниз (например, движение по лестнице вниз)
- Сидит
- Стоит
- Лежит

Сведения о наборе данных

Набор данных содержит записи датчиков со сматфонов (акселерометр и гироскоп с частотой дискретизации 50 Гц) от 30 участников, выполняющих следующие действия: ходьба, ходьба по лестнице, ходьба по лестнице, сидение, стояние и лежание. Данные были предварительно обработаны при помощи фильтров шума. Набор данных представлен Хорхе Л. Рейес-Ортисом.

Признаки были извлечены из 3-х осевых необработанных сигналов акселерометра и гироскопа tAcc-XYZ и tGyro-XYZ. Эти сигналы были сняты с постоянной частотой 50 Гц. Затем были отфильтрованны с помощью медианного фильтра и низкочастотного фильтра Баттерворта 3-го порядка с частотой 20 Гц для удаления шумов. Аналогичным образом сигнал ускорения был разделен на сигналы ускорения тела и гравитации (`tBodyAcc-XYZ` и `tGravityAcc-XYZ`) с помощью другого низкочастотного фильтра Баттерворта с угловой частотой 0,3 Гц. Линейное ускорение тела и угловая скорость были использованы для получения сигналов "рывка" — (`tBodyAccJerk-XYZ` и `tBodyGyroJerk-XYZ`). Также величина этих трехмерных сигналов была расчитана с использованием евклидовой нормы — (`tBodyAccMag` , `tGravityAccMag` , `tBodyAccJerkMag` , `tBodyGyroMag` , `tBodyGyroJerkMag`).

Наконец, к некоторым из этих сигналов было применено быстрое преобразование Фурье (БПФ), в результате чего получились `fBodyAcc-XYZ` , `fBodyAccJerk-XYZ` , `fBodyGyro-XYZ` , `fBodyAccJerkMag` , `fBodyGyroMag` , `fBodyGyroJerkMag` . (Обратите внимание на "f" для обозначения сигналов в частотной области).

Набор переменных, которые были оценены по этим сигналам, следующий:

- mean(): Среднее значение
- std(): Стандартное отклонение
- mad(): Среднее абсолютное отклонение
- max(): Наибольшее значение в массиве
- min(): Наименьшее значение в массиве
- sma(): Область величины сигнала

- energy(): Мера энергии. Сумма квадратов, деленная на количество значений.
- iqr(): Интерквартильный размах
- entropy(): Энтропия сигнала
- arCoeff(): Коэффициенты авторегрессии с порядком Burg, равным 4
- correlation(): коэффициент корреляции между двумя сигналами
- maxInds(): индекс частотной составляющей с наибольшей величиной
- meanFreq(): средневзвешенное значение частотных компонент для получения средней частоты
- skewness(): перекос сигнала в частотной области
- kurtosis(): эксцесс сигнала в частотной области
- bandsEnergy(): Энергия частотного интервала в пределах 64 бинов БПФ каждого окна.
- angle(): Угол между векторами.

Импорт библиотек

Первым делом импортируем необходимые библиотеки для работы с данными:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
```

Считываем набор данных

В прикладных задачах машинного обучения очень важен процесс извлечения признаков (feature extraction), в ходе которого данные интерпретируются в информативные признаки. Также этот процесс может называться проектирование признаков (feature engineering), это весьма трудоемкая и творческая задача. В рамках работы мы опустим эту часть и поспользуемся предобработанными данными.

```
def read_data(path, filename):
    return pd.read_csv(os.path.join(path, filename))

df = read_data('/data/notebook_files', 'train.csv')
df.head()
```

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJe kurtosis()
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.710304
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	-0.861499
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	-0.760104
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	-0.482845
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	-0.699205

5 rows × 563 columns

Теперь, загрузим полный набор данных и сохранить его под следующими четырьмя переменными:

- `train_X` : признаки, используемые для обучения модели
- `train_y` : метки, используемые для обучения модели
- `test_X` : признаки, используемые для проверки модели
- `test_y` : метки, используемые для проверки модели

```
def load_dataset(label_dict):
    train_X = read_data('/data/notebook_files', 'train.csv').values[:,:-2]
    train_y = read_data('/data/notebook_files', 'train.csv')['Activity']
    train_y = train_y.map(label_dict).values
    test_X = read_data('/data/notebook_files', 'test.csv').values[:,:-2]
    test_y = read_data('/data/notebook_files', 'test.csv')
    test_y = test_y['Activity'].map(label_dict).values
    return(train_X, train_y, test_X, test_y)
label_dict = {'WALKING':0, 'WALKING_UPSTAIRS':1, 'WALKING_DOWNSTAIRS':2, 'SITTING':3, 'STANDING':4, '
train_X, train_y, test_X, test_y = load_dataset(label_dict)
```

Выбор модели

Чтобы выбрать модель поработаем с несколькими, начнем с метода k-ближайших соседей, этот метод подходит, когда категорий больше 2, как у нас.

Метод k-ближайших соседей

```
#импортирует модель KNN и функцию вычисляющую accuracy
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=8)#создаем модель с гиперпараметром k=8
model.fit(train_X,train_y) #обучаем модель на тренировочных данных
```

```
KNeighborsClassifier(n_neighbors=8)
```

```
prediction=model.predict(test_X)#используя метод k-ближайших соседей прогнозируем активность движени
prediction
```

```
np.array(test_y)
```

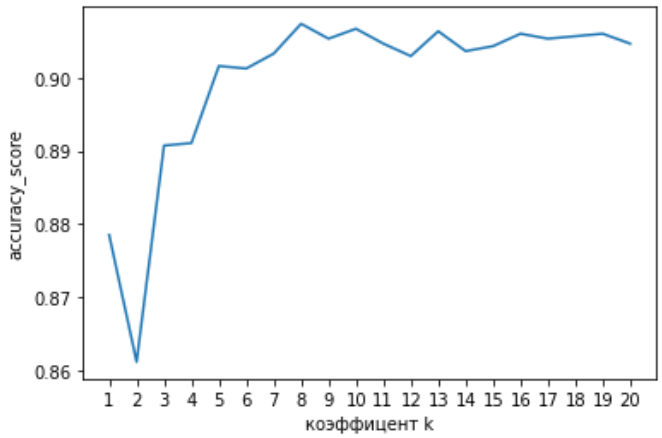
```
accuracy_score (prediction,test_y) #сравниваем предсказания (predition) с базовой истиной (test_y)
```

```
0.9073634204275535
```

Точность модели довольно высокая, но есть к чему стремиться, чтобы проработать модель до конца, переберем коэффицент k (количество ближайших соседей).

```
values_score=[]
#строим модель для разного гиперпараматера k
for k in range (1,21):
    model=KNeighborsClassifier(n_neighbors=k)
    model.fit(train_X,train_y)
    predicted=model.predict(test_X)
    acc=accuracy_score(predicted,test_y)
    values_score.append(acc)
```

```
#строим график зависимости коэффицента k от accuracy_score
plt.plot(list(range(1,21)),values_score)
plt.xticks(list(range(1,21)))
plt.xlabel("коэффицент k")
plt.ylabel("accuracy_score")
plt.show()
```



Если выберем использовать метод k-ближайших соседей, то будем использовать k=8, так при таком значении k accuracy имеет наивысшее значение.

Логическая регрессия

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
model_l = LogisticRegression(solver='liblinear', random_state=0)
model_l.fit(train_X,train_y)#обучаем модель
```

```
LogisticRegression(random_state=0, solver='liblinear')
```

```
prediction_l = model_l.predict(test_X)#используя модель логической регрессии, прогнозируем активность
```

```
np.array(test_y)
```

```
accuracy_score (prediction_l,test_y)#сравниваем предсказания (predition) с базовой истиной (test_y)
```

```
0.9619952494061758
```

SVM

Дальше разберем Метод Опорных Векторов (Support Vector Machines) - один из популярных метов, применяемых на пратике. Алгоритм создает плоскость или линию, котоаря разделяет данные на классы.

```
from sklearn.svm import SVC
#для классификатора (ядра) задаем линейный параметр
model_svc = SVC(kernel='linear',C=1)
model_svc.fit(train_X,train_y)#обучим модель выборкой
```

```
SVC(C=1, kernel='linear')
```

```
prediction = model_svc.predict(test_X)#используя модель SVM прогнозируем активность движений
```

```
np.array(test_y)
```

```
accuracy_score (prediction,test_y)#сравниваем предсказания (predition) с базовой истиной (test_y)
```

```
0.9640312181879878
```

```
#Попробуем подобрать "хороший" параметр
val_score=[]
for n in range (2,12,2):
    model_svc = SVC(kernel='linear',C=n/10,gamma="scale")
    model_svc.fit(train_X,train_y)#обучим модель выборкой
    prediction = model_svc.predict(test_X)
    accurancy_SVC=accuracy_score(prediction,test_y)
    val_score.append(accurancy_SVC)
print(val_score)
```

```
[0.9623345775364778, 0.9633525619273838, 0.9630132337970818, 0.9633525619273838, 0.9640312181879878]
```

Таким образом, из всех классификаторов нам подойдет SVM, так как точность у него получилось самой высокой (0,964), не рассматривали линейную регрессию, так как данный классификатор эффективен для двух категорий , у нас же 6 разных классов. Мы выяснили, что при C=1, точность самая высокая.

Обучение модели

Обучаем модель, используя признаки из обучающего набора (`train_X`) и метки в качестве базовой истины (`train_y`).

```
from sklearn.svm import SVC
#для классификатора (ядра) задаем линейный параметр
model = SVC(kernel='linear',C=1)
model.fit(train_X,train_y)
```

```
SVC(C=1, kernel='linear')
```

Оценка модели

Обученную модель используем для прогнозирования активности движения, используя признаки из тестового набора (`test_X`). В 'yhat' сохраняем прогнозы (до этого записывали в 'prediction')

```
yhat = model.predict(test_X)
yhat
```

Сравнив предсказания (`yhat`) с базовой истиной (`test_y`), получаем точность(accuracy) 0,964, точность показывает, что это модель далеко не плоха, именно поэтому остановимся на SVM модели.

```
accuracy_score (yhat,test_y) #сравниваем предсказания (yhat) с базовой истиной (test_y)
```

```
0.9640312181879878
```

Мы сравнивали модели по точности, но это не единственная метрика, есть еще precision, recall и F-мера. Ниже представлен код по нахождению этих метрик. Хочется пояснить, за что отвечает каждая метрика. Precision - показывает, как хорошо отличает этот класс от других кассов, отвечает за долю объектов,которые являются положительными и которые классификатор определил положительными. A recall показывает, как хорошо вообще обнаруживает этот класс, а также показывает какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм. Наконец, существует мера, которая объединяет precision и recall - это F-мера. В F-мере мы можем учитывать важность precision в определенной ситуации, так как в формуле есть коэффициент, регулируя который мы можем изменять приоритет метрики precision. F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю.

```
from sklearn.metrics import classification_report
target_names = ['Walking', 'Walking Upstairs', 'Walking Downstairs', 'Sitting', 'Standing', 'Laying']

print(classification_report(test_y, yhat, target_names=target_names))
```

```
                    precision    recall  f1-score   support

           Walking       0.96      0.99      0.97       496
  Walking Upstairs       0.98      0.96      0.97       471
Walking Downstairs       0.99      0.98      0.98       420
           Sitting       0.96      0.89      0.92       491
          Standing       0.91      0.97      0.94       532
            Laying       1.00      1.00      1.00       537

          accuracy                           0.96      2947
         macro avg       0.97      0.96      0.96      2947
      weighted avg       0.96      0.96      0.96      2947
```