

Branch: master ▾ Myna / Dataset /

Create new file

Find file

History

 王小辉 Updated XGBoost training script

Latest commit 6dba636 on 13 Jul

..

 [README.md](#)

Updated XGBoost training script

5 months ago

 [XGBoostrHAR.py](#)

Updated XGBoost training script

5 months ago

 [README.md](#)

## 用户行为识别

### 下载数据集

可以从下面任何一个地址下载数据集，解压缩到 dataset 目录下：

- [Google Drive](#)
- [Baidu 网盘](#) 密码：dgyf

### Data Structure

数据集中包含下面 6 种行为的传感器数据：

- Walking
- Running

- Bus
- Subway
- Car
- Stationary

由于数据采集是有成本的，为了保证后面测试的灵活性，在测试过程中，使用 100Hz 的频率进行传感器数据采集，在数据处理的时候可以进行降频重采样，测试在不同的较低频率下识别模型的性能。使用到的传感器类型

- 加速度
- 陀螺仪
- 磁场传感器

通过加速度、陀螺仪以及磁场传感器的数据，可以计算出设备从机身坐标系到真实世界坐标系的旋转矩阵，然后通过该旋转矩阵，将机身坐标系的加速度转换到真实世界坐标系，而且消除不同行为下，不同的设备姿态对传感器数据的影响：

```
public static void calculateWorldAcce(SensorData sd){  
    float[] Rotate = new float[16];  
    float[] I = new float[16];  
    float[] currOrientation = new float[3];  
    SensorManager.getRotationMatrix(Rotate, I, sd.gravity, sd.magnetic);  
    SensorManager.getOrientation(Rotate, currOrientation);  
    System.arraycopy(currOrientation, 0, sd.orientation, 0, 3);  
  
    float[] relativeAcc = new float[4];  
    float[] earthAcc = new float[4];  
    float[] inv = new float[16];  
    System.arraycopy(sd.accelerate, 0, relativeAcc, 0, 3);  
    relativeAcc[3] = 0;  
    android.opengl.Matrix.invertM(inv, 0, Rotate, 0);  
    android.opengl.Matrix.multiplyMV(earthAcc, 0, inv, 0, relativeAcc, 0);  
    System.arraycopy(earthAcc, 0, sd.world_accelerometer, 0, 3);  
}
```

## 数据格式

orientationX,orientationY,orientationZ,world\_accelerometerX,world\_accelerometerY,world\_accelerometerZ

## 数据整理

一般人行走的步频是 1-2 步每秒，跑步的频率要更快一些。而不同的行为会表现在携带的设备上，传感器数据会具有周期性。为了保留这种周期信息，进行识别时的时间窗口不能太短，为了识别的及时性，也不能太长，我们取 20Hz 采样频率下获取 128 组数据进行识别，这样每条数据之间时间间隔为 50ms，整组数据的时间窗口长度 6.4s。

在降频重采样的过程中，可以对重复值进行过滤：

```
def get_resample_dataset(file_path):
    re_sampled = []
    with open(file_path, "r") as lines:
        index = 0
        last_value = ""
        for line in lines:
            index += 1
            if index == 5:
                values = line.split(",")
                if len(values) == 6:
                    current_value = "{},{},{},{}".format(values[3], values[4], values[5])
                    if current_value != last_value:
                        re_sampled.append(current_value)
                        last_value = current_value
                        index = 0
                else:
                    index -= 1
            else:
                index -= 1
    print("\tAfter re-sampling, the count of the lines are: {}".format(len(re_sampled)))
    return re_sampled
```

另外，行为识别时，采集到的传感器数据是连续的时间序列数据，为了提高识别的及时性，我们可以通过半重叠的方式对采集到的数据进行处理，使得下一组数据的前半部分和上一组数据的后半部分一样，可以在时间窗口的一半时间给出识别结果：

```
def get_half_overlap_dataset(dataset):
    overlapped = []
    for i in range(0, len(dataset) - batch_size, batch_size / 2):
        overlapped.append(dataset[i: i + batch_size])
    print("\tThe number of the groups after half-overlapping is: {}".format(len(overlapped)))
    return overlapped
```

接下来对数据进行随机分组，70% 的数据用于训练，30% 的数据用于模型测试。另一个做交叉验证的方法是：将数据集（每种行为的数据量相当，也就是数据集基本平衡）针对每个行为分为 10 份，每次有放回地随机抽取 7 份训练，剩余的 3 份用于测试，多次重复后对模型的 precision 和 recall 取平均。

```
def split_train_test(dataset):
    total_dataset = np.array(dataset)
    train_test_split = np.random.rand(len(total_dataset)) < 0.70
    train_dataset = total_dataset[train_test_split]
    test_dataset = total_dataset[~train_test_split]
    print("\t\tCount of train dataset: {}\n\t\tCount of test dataset: {}".format(len(train_dataset), len(test_dataset)))
    return train_dataset.tolist(), test_dataset.tolist()
```



## 特征抽取

传感器数据本身是时域信号数据，除了可以其统计特征外，还可以将其转换到频域进行频域特征的抽取。傅里叶变换后可以得到信号的频率，但丢失了时域信息，而通过小波变换可以更好地保留时域和频域信息。[能不能通俗的讲解下傅立叶分析和小波分析之间的关系？](#) 对此有很直观的分析。

对于每个坐标轴的数据，我们选择下面的 8 个特征进行测试：

- 最小值
- 最大值
- 均值
- 标准差
- 小波变换后 Approximation 的均值
- 小波变换后 Approximation 的标准差
- 快速傅里叶变换将数据转换到频域后最大振幅
- 快速傅里叶变换将数据转换到频域后最大振幅对应的频率

具体的实现方式可以从 `XGBoostrHAR.py` 脚本中查看。

## 模型训练

我们使用在 Kaggle 比赛中非常受欢迎的 XGBoost 算法进行模型训练：

```
def xgTestSelfDataset(train_X, train_Y, test_X, test_Y):  
    import xgboost as xgb  
    import time  
  
    # label need to be 0 to num_class -1  
    xg_train = xgb.DMatrix(train_X, label=train_Y)  
    xg_test = xgb.DMatrix(test_X, label=test_Y)  
    # setup parameters for xgboost  
    param = {'objective': 'multi:softprob',  
            'eta': 0.15,  
            'max_depth': 6,  
            'silent': 1,  
            'num_class': 5,  
            "n_estimators": 1000,  
            "subsample": 0.7,  
            "scale_pos_weight": 0.5,  
            "seed": 32}
```

```
watchlist = [(xg_train, 'train'), (xg_test, 'test')]
num_round = 50

start = time.time()
bst = xgb.train(param, xg_train, num_round, watchlist)
trainDuration = time.time() - start
start = time.time()
yprob = bst.predict(xg_test).reshape(test_Y.shape[0], 5)
testDuration = time.time() - start
ylabel = np.argmax(yprob, axis=1)

if os.path.exists("rhar.model"):
    os.remove("rhar.model")
bst.save_model("rhar.model")
```

在训练完成后，我们将得到的模型保存下来，之后在 Android 代码中加载使用。针对这里的数据集，我们得到了下面的 metrics：

```
Precision 0.909969288145
Recall 0.908256880734
f1_score 0.90816711949
confusion_matrix
[[ 93   0   1   0   0   0]
 [  0 115   1   1   1   0]
 [  0   0 102   3   5   0]
 [  3   0   9  84   6   2]
 [  2   0   7  11  85   1]
 [  0   0   4   3   0 115]]
predicting, classification error=0.091743
```

## 在 Android 程序中进行 inference

XGBoost 的官方 Java 实现需要通过 jni 调用 native 模块，这里我们选择 [komiya-atsushi/xgboost-predictor-java](#) 的纯 Java 实现，作者声称该实现比官方实现要快很多：

Xgboost-predictor-java is about 6,000 to 10,000 times faster than xgboost4j on prediction tasks.

我们的 XGBoost 分类器实现为：

```
public class XGBoostClassifier implements ClassifierInterface {

    private Predictor predictor;
    private double[] features;

    public static final String TYPE = "xgboost";

    public XGBoostClassifier(Context ctx) {
        try {
            InputStream is = ctx.getAssets().open("rhar.model");
            predictor = new Predictor(is);
            is.close();
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }

    /**
     * Extract and select features from the raw sensor data points.
     * These data points are collected with certain sampling frequency and windows.
     * @param sensorData Raw sensor data points.
     * @return Extracted features.
     */
    private double[] prepareFeatures(SensorData[] sensorData, final int sampleFreq, final int sampleCount)

        double[] matrix = new double[SensorFeature.FEATURE_COUNT];
        Feature aFeature = new Feature();
        aFeature.extractFeatures(sensorData, sampleFreq, sampleCount);
        System.arraycopy(aFeature.getFeaturesAsArray(), 0, matrix, 0, SensorFeature.FEATURE_COUNT);
    }
```

```
        return matrix;
    }

    /**
     * Recognize current human activity based on pre-defined rules.
     * @param sensorData Raw sensor data points.
     */
    @Override
    public double[] recognize(SensorData[] sensorData, final int sampleFreq, final int sampleCount) {
        features = prepareFeatures(sensorData, sampleFreq, sampleCount);
        return predict();
    }

    @Override
    public double[] getCurrentFeatures(){
        return features;
    }

    private double[] predict() {
        FVec vector = FVec.Transformer.fromArray(features, true);
        return predictor.predict(vector);
    }
}
```

