Predicting user activity from smartphone's sensory data

CS-C3240 Machine Learning D Project stage 2

Content:

1.Introduction

Smartphones have advanced sensory systems to navigate their physical movement and thus measure the smartphone's motions in three-dimensional space with respect to time. Using machine learning, the sensory information can be applied to predict the current activity of the person who may be carrying the smartphone with them. Such information can be used for instance in apps that track user activity levels in order to provide guidance on how the user could improve their health over time.

The paper is structured as follow: section 2 discusses problem formulation and features of the dataset; section 3 discusses data preparation, model selection and model validation; section 4 discusses the results of the study in this report; and lastly, section 5 concludes the report and reflect on the limitations of the methods with possible ways to improve them.

2.Problem Formulation

With means of machine learning, this report tries to predict the activity of the person carrying a smartphone based on the smartphone's sensory information. Each data point consists of data that smartphone sensors have recorded as a function of time that are then associated with some activity. In total there are 561 features in the dataset. Therefore feature selection is applied to choose 6 features that are used in the prediction. The feature selection process and features are explained in the methods section.

The type of data of the features is continuous as each datapoint gets different numerical values from sensor readings. The labels for each datapoint are the different activities associated with the sensory input. The possible activities in the dataset are "laying", "sitting", "standing", "walking", "walking downstairs" and "walking upstairs". Therefore, the type of data for labels is "categorized". As each datapoint has the features associated with a label "activity" in the training phase, the machine learning task in question is a supervised task.

3.Methods

3. 1. Preparing data

The data to study this issue was provided by a dataset associated with the research centers of University of Genova in Italy and Universitat Politècnica de Catalunya in Spain. The dataset was accessed through Kaggle [1]. It contains extensive data on different activities labelled to data collected from smartphones. The dataset contains in total 3609 datapoints from which 2526 are selected to be the training set. This leaves the size of the validation dataset to be 541 and the final test set to be 542. This design choice enables the model to effectively learn the correlations in the dataset while also having a large validation set to reflect the results. The provided code shows the importing of relevant libraries as well as the correlation calculations.

The chosen methods for this report to study are logistic regression and decision tree. The choices are based on the fact that both logistic regression and decision tree are effective in classifying tasks when the type of output is not continuous. In this case the labels are categorized and the logistic regression and decision tree are suitable for the task.

The model can also be used to map a confusion matrix on the validation set to show which activities most typically may be confused with other types of activities based on the sensory data.

For the loss function, the logistic loss is naturally chosen because it is provided as a library for logistic regression. It is beneficial as it assesses the performance of the model by comparing the probabilities of the answers with the true labels in the validation phase.

In the training, machine learning task will use columns tBodyAccJerk.entropy.X , tBodyAccJerk.entropy.Y, tBodyAcc.sma, tBodyAcc.mean.Z, tGravityAcc.energy.X and tGravityAcc.mean.Z as features to predict the value of “activity”. The features tBodyAccJerk.entropy.X and tBodyAccJerk.entropy.Y correspond to the irregularity of movement on X and Y axis. This is useful in predicting movements between walking upstairs, walking or standing for example. tBodyAcc.sma gives the magnitude of acceleration parameter on all axis holistically. It can further be used to predict differences between active and static activities such as walking and standing. Lastly, tBodyAcc.mean.Z gives the mean of the Z axis acceleration. The feature can be used to distinguish between laying vs. sitting or walking upstairs vs. walking downstairs. tGravityAcc.energy.X corresponds to the energy of gravity acceleration along the X-axis. Finally, tGravityAcc.mean.Z is the average gravity acceleration along the Z-axis. As there are 563 columns in the dataset in total, many features are excluded. Thus, the activities are predicted based on only the six features of the smartphone sensor.

The tBodyAccJerk.entropy features were chosen based on the strong correlation between the column value and the activity label. Although the other features had weaker correlation to activities, the additional choices were made to supplement each other in distinctions that might be harder from based on only the previous values. tGravityAcc.energy.X and tGravityAcc.mean.Z are chosen based on principal component analysis, for which the code is provided in the appendix.

3. 2. Model selection

3.2.1. Logistic Regression

The first choice for the machine learning method in this task is the logistic regression. The model uses a linear hypothesis space. The choice is based on the fact that logistic regression is effective in classifying tasks when the type of output is not continuous. In this case the labels are categorized and the logistic regression is suitable for the task. The model can also be used to map a confusion matrix on the validation set to show which activities most typically may be confused with other types of activities based on the sensory data.

For the loss function, the logistic loss is naturally chosen because it is provided as a library for logistic regression. It is beneficial as it assesses the performance of the model by comparing the probabilities of the answers with the true labels in the validation phase. Logistic loss provides a good way to study how confident the model is in its predictions

3.2.2. Decision tree

The second machine learning method evaluated for this task is the decision tree. It is chosen as the decision tree is suitable for categorical and numerical features which is good for this classification task. The model works by splitting the data to smaller and smaller subsets based

on the feature that results in the largest gini index reduction. Decision tree's advantage is that it can model non-linear relationships in the data between feature values.

For the loss function, a usual method used for decision trees is gini impurity. It measures the quality of each split individually. The loss function for decision tree is therefore harder to utilize in this report, because each split is evaluated individually in gini impurity and the whole training task requires numerous splits.

3.3. Model validation

The data is split into training set to train the models and validation sets to assess the performance of the models. The data is split with sklearn library's train_test_split -function into training and validation sets for the features and labels. The validation set is 15% of the size of the whole set whereas 70% of the set is used for the training. This sizing is used as it provides the model sufficient data to learn the relationships between features and labels while maintaining a large enough validation set to evaluate the performance. The main validation method for both models is the accuracy of the prediction, i. e. whether the prediction was the correct activity or not, provided by the sklearn library's accuracy_score -method. This is simple yet effective way of measuring the performance of the models in classifying tasks such as the activity prediction.

4.Results

The test results suggest that both models are rather suitable for predicting the activities from the features. Logistic regression achieved an accuracy of 74.7% and the decision tree had an accuracy score of 77.8%. Therefore both models had very similar performances. There were 6 possible activities that the models were predicting yielding an accuracy score of 16.6% by random guess. The models therefore demonstrate rather good predictive power for predicting the activities.

For the training data, the logistic regression achieved an accuracy score of 78.2% which is similar to that of the validation data. However, the decision tree had achieved an accuracy score of 99% for the training set, which suggests that there was overfitting happening for the decision tree model in the data.

Furthermore, log loss is also applicable for the decision tree as log loss is useful for classification tasks. Log loss for logistic regression is 0.497 for the training set and 0.534 for the validation set. Conversely, log loss for the decision tree is near zero for the training set and 0.534 for the validation set. Therefore both methods have rather low values for log loss.

These findings suggest that the logistic regression is chosen as the final machine learning method for this task. Both methods yielded rather high accuracy scores and log loss outcomes suggested good confidence in the predictions as well. However, the decision tree seemed to suffer some amount from overfitting. Hence, the although the decision tree had slightly higher accuracy, the logistic regression was chosen based on likely better generalizability and its practically equal log loss outcome to the decision tree.

As discussed previously, the test set is constructed from the last 15% of the remaining data leaving a sufficient split consisting of 542 data points. With regards to the test error, the logistic regression shows an accuracy of 77.6% correct predictions for the activities in the test set with log loss value of 0.534.

5.Conclusion

In this report an accuracy of 77.6% of correctly predicted activities was achieved when using the logistic regression machine learning method. This result demonstrates that machine learning can be used to differentiate between different activities based on smartphone's sensory data. All in all, while the decision tree achieved slightly higher accuracy, the indications of overfitting to the decision that the logistic regression was the final method.

Even though both models discussed in the report showed a great predictive power, there were clear signs of overfitting on the decision tree model. This suggests that the decision tree "memorized" the training data instead of learning the general patterns in the data. To tackle this problem, pruning could be applied to the decision tree in order to reduce overfitting. For the logistic regression, generalizability could be improved by collecting more data that the model could learn.

References

[1] Dataset: Simplified human activity recognition using smartphones. Available at: https://www.kaggle.com/datasets/mboaglio/simplifiedhuarus

Appendix

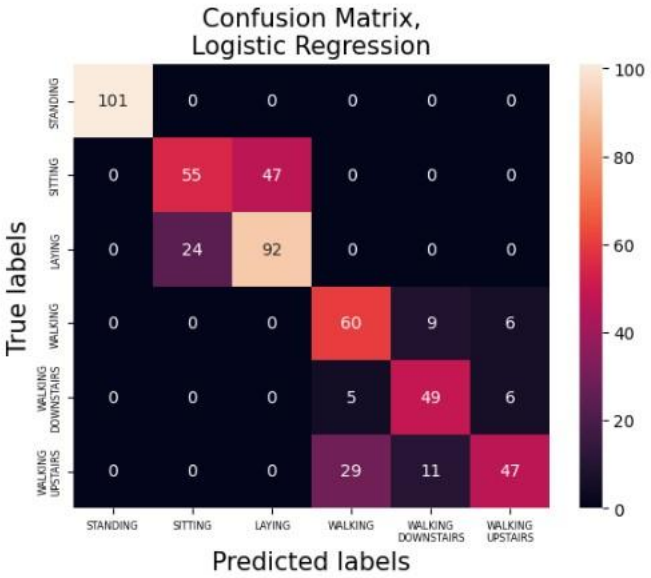

Figure 1: Confusion matrix for the logistic regression on the validation set

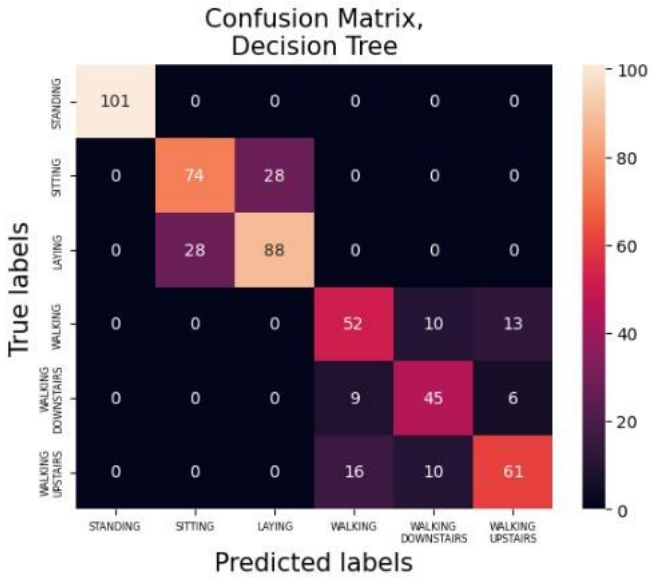

Figure 2: Confusion matrix for the decision tree on the validation set

code

October 9, 2024

[ ]:
```
%config Completer.use_jedi = False  # enable code auto-completion
import warnings

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns  #data visualization library
from sklearn.linear_model import LogisticRegression
from sklearn.tree import export_text, plot_tree, DecisionTreeClassifier, export_graphviz
from sklearn.metrics import accuracy_score, confusion_matrix, log_loss  # evaluation metrics

from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder

# Load the dataset to inspect the number of data points and features
data = pd.read_csv('train.csv')

X = data.drop(columns=['activity'])  # All columns except 'activity' are features
y = data['activity']  # 'activity' column is the label
X.shape
```

Choose features:

[ ]:
```
# Showing the correlations between the activity and the column value
label_encoder = LabelEncoder()
data2=data
data2['activity'] = label_encoder.fit_transform(data['activity'])
correlation_matrix = data2.corr()
activity_correlation = correlation_matrix['activity'].drop('activity')
sorted_correlation = activity_correlation.abs().sort_values(ascending=False)
sorted_correlation.head(10)
```

[ ]:
```
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA to get 8 most important components
pca = PCA(n_components=8)
X_reduced = pca.fit_transform(X_scaled)

# Get the most important features in each principal component
importance = pca.components_

# Summing the absolute values of feature importances across all principal components
feature_importance = pd.DataFrame(importance.T, index=X.columns, columns=[f'PC{i+1}' for i in range(8)])
feature_importance['Total Importance'] = feature_importance.abs().sum(axis=1)

# Sorting by total importance and getting the top 8 most important features
top_8_features = feature_importance['Total Importance'].sort_values(ascending=False).head(8)
top_8_features.head(8)
```

[ ]:
```
selected_columns = ['activity','tGravityAcc.max.X', 'tGravityAcc.mean.X', 'tGravityAcc.min.X',
                    'tGravityAcc.energy.X', 'tGravityAcc.std.X', 'tGravityAcc.mad.X', 'angle.X.gravityMean', 'tGravityAcc.iqr.X'] #Show

reducedData = dataTrain[selected_columns]

reducedData.sample(10)
```

```
# Create the feature matrix X, make sure X.shape==(m,1)
# and create label vector y for the features

selected_features3 = ['tBodyAccJerk.entropy.X', 'tBodyAccJerk.entropy.Y', 'tBodyAcc.sma', 'tBodyAcc.mean.Z', 'tGravityAcc.energy.X', 't
X = dataTrain[selected_features3]
y = dataTrain['activity']
X_train, X_remaining, y_train, y_remaining = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_remaining, y_remaining, test_size=0.5, random_state=42)
```

```
#Model1 Logistic Regression:
log_clf = LogisticRegression(max_iter=1000)    # initialise a LogisticRegression classifier, use default value for all arguments
log_clf.fit(X_train,y_train)       # fit cfl_1 to data
y_pred = log_clf.predict(X_train)   # compute predicted labels for training data
accuracy = accuracy_score(y_train,y_pred) # compute accuracy on the training set
print("accuracy of LogReg : ", accuracy)
```

```
# Model 2: Decision Tree
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)  # No need for scaling with decision trees
y_pred_tree = tree_clf.predict(X_train)
accuracytree = accuracy_score(y_train,y_pred_tree) # compute accuracy on the training set
print("accuracy of DecisionTree : ", accuracytree)
```

```

def generate_confusion_matrix(title, y_true, y_pred):
    # Visualize the confusion matrix you computed
    ax= plt.subplot()

    sns.heatmap(confusion_matrix(y_true, y_pred), annot=True, fmt='g', ax=ax)

    ax.set_xlabel('Predicted labels',fontsize=15)
    ax.set_ylabel('True labels',fontsize=15)
    ax.set_title(title,fontsize=15)
    ax.xaxis.set_ticklabels(['STANDING', 'SITTING', 'LAYING', 'WALKING', 'WALKING\nDOWNSTAIRS','WALKING\nUPSTAIRS'],fontsize=6)
    ax.yaxis.set_ticklabels(['STANDING', 'SITTING', 'LAYING', 'WALKING', 'WALKING\nDOWNSTAIRS','WALKING\nUPSTAIRS'],fontsize=6)

y_pred_log = log_clf.predict(X_val)
accuracy_log = accuracy_score(y_val,y_pred_log)
generate_confusion_matrix('Confusion Matrix,\nLogistic Regression',y_val, y_pred_log)
plt.show()
print(accuracy_log)
```

```
y_pred_tree = tree_clf.predict(X_val)
accuracy_tree = accuracy_score(y_val,y_pred_tree)
generate_confusion_matrix('Confusion Matrix,\nDecision Tree',y_val, y_pred_tree)
plt.show()
print(accuracy_tree)
```

```
# Compute the log loss for logistic regression
y_prob_train_log = log_clf.predict_proba(X_train)  # Get predicted probabilities
y_prob_val_log = log_clf.predict_proba(X_val)    # Get predicted probabilities
train_log_loss = log_loss(y_train, y_prob_train_log)
val_log_loss = log_loss(y_val, y_prob_val_log)
print("Logistic Regression - Training Log Loss: ", train_log_loss)
print("Logistic Regression - Validation Log Loss: ", val_log_loss)
```

```
# Compute the log loss for the decision tree
y_prob_train_tree = tree_clf.predict_proba(X_train)  # Get predicted probabilities
y_prob_test_tree = tree_clf.predict_proba(X_val)    # Get predicted probabilities
tree_train_log_loss = log_loss(y_train, y_prob_train_tree)
tree_val_log_loss = log_loss(y_val, y_prob_val_log)
print("Decision tree - Training Log Loss: ", tree_train_log_loss)
print("Decision tree - Validation Log Loss: ", tree_val_log_loss)
```

```
# Test set: decision tree
y_pred_log_test = log_clf.predict(X_test)
accuracy_log_test = accuracy_score(y_test,y_pred_log_test)
print("accuracy of Logistic regression : ", accuracy_log_test)
# Compute the log loss for the decision tree in test set
y_prob_test_log = log_clf.predict_proba(X_test)    # Get predicted probabilities
log_loss_test = log_loss(y_test, y_prob_test_log)
print("Logistic regression - Test set Log Loss: ", log_loss_test)
```