Bootcamp: Arquiteto(a) de Machine Learning

Desafio Prático

Módulo 4: Aplicações de Aprendizado de Máquina para IoT

Objetivos de Ensino

Exercitar os seguintes conceitos trabalhados no Módulo:

1. Análise exploratória dos dados (EDA - *Exploratory Data Analysis*).
2. Preparação dos dados da IoT.
3. Comparação e ajuste de modelos de classificação.

Enunciado

Neste desafio serão abordados todos os conceitos apresentados durante o módulo Aprendizado de Máquina para IoT e Edge. Para este desafio será utilizado o dataset "Fall Detection" disponível no *Kaggle* (https://www.kaggle.com/pitasr/falldata/version/1#).

Para a construção desse dataset foram utilizados um conjunto de sensores "vestíveis" afixados em seis posições distintas do corpo de um grupo de voluntários. Esses sensores são responsáveis por monitorar diferentes atividades do cotidiano desses voluntários. Atividades como levantar, caminhar, sentar e correr são identificadas e classificadas por meio desses sensores. O artigo "*Detecting Falls with Wearable Sensors Using Machine Learning Techniques*" utiliza esses dados e emprega o aprendizado de máquina a fim de identificar os eventos de queda, principalmente em idosos, uma vez que esses eventos são vistos como uma das principais causas de fraturas e problemas graves em pessoas idosas.

Atividades

Os alunos deverão desempenhar as seguintes atividades:

1. Acessar o Google Colaboratory.
2. Realizar o upload do dataset “fall_detection.csv” presente no link:

   https://drive.google.com/file/d/13GyCVK9tAyMIQHmDgqZ2ITGIN0jJS1qc/view?usp=sharing.
3. Construa um código utilizando como base o trabalho prático, os códigos apresentados neste módulo e o enunciado das questões presentes neste desafio.
4. Para a implementação dos algoritmos utilize as definições abaixo:

Divisão entre treinamento e teste:

```
X_train, X_test, y_train, y_test = train_test_split(entrada_normalizada, dados['ACTIVITY'], test_size = 0.3,random_state=42)
```

Algoritmo Regressão logística:

```
lr = LogisticRegression(max_iter=1000,random_state=42)
```

Algoritmo KNN:

```
clf_KNN = KNeighborsClassifier(n_neighbors=5)
```

Algoritmo Árvore de Decisão:

```
dtc = DecisionTreeClassifier(random_state=42)
```

Algoritmo Floresta Randômica:

```
rfc = RandomForestClassifier(n_estimators = 50,random_state=42)
```

Gradiente Boosting:

```
gb = GradientBoostingClassifier(n_estimators=50, learning_rate = 0.2,
max_features=6, max_depth = 5, random_state = 42)
```

Algoritmo SVM:

```
clf_svm=SVC(gamma='auto',kernel='rbf')
```

Obs.:

1 Utilize a normalização dos dados utilizando o StandardScaler() para todos os algoritmos.

2 Para a divisão dos dados de treinamento e teste dos algoritmos utilize o valor de “random_state=42” e a proporção de 70% para treinamento e 30% para teste.

3 Aplique primeiro a normalização e, depois, aplique a divisão dos dados entre treinamento e teste. Para a aplicação de todos os modelos, utilize essa sequência de passos.

4 Utilize a variável “ACTIVITY” como saída e as demais como entrada do modelo.

5 Para a última questão, considere as chamadas a seguir:

```
from sklearn.metrics import confusion_matrix
matriz_confusao = confusion_matrix(y_test, previsao)
print(matriz_confusao)
seaborn.heatmap(matriz_confusao/np.sum(matriz_confusao), annot=True,
        fmt='.2%', cmap='Blues')
```