Human Activity Recognition - Decision Tree

1. 데이터 로드 및 전처리

1-1. 데이터 로드

In [13]:
```
import pandas as pd
import matplotlib.pyplot as plt

feature_name_df = pd.read_csv('./human_activity/features.txt', sep='₩s+', header=None, names=['column_index', 'column_name'])

feature_name = feature_name_df.iloc[:, 1].values.tolist()
print('전체 피처명에서 10개만 추출:', feature_name[:10])
```

```
전체 피처명에서 10개만 추출: ['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']
```

In [14]:
```
feature_name_df
```

Out[14]:

	column_index	column_name
0	1	tBodyAcc-mean()-X
1	2	tBodyAcc-mean()-Y
2	3	tBodyAcc-mean()-Z
3	4	tBodyAcc-std()-X
4	5	tBodyAcc-std()-Y
...	...	...
556	557	angle(tBodyGyroMean,gravityMean)
557	558	angle(tBodyGyroJerkMean,gravityMean)
558	559	angle(X,gravityMean)
559	560	angle(Y,gravityMean)
560	561	angle(Z,gravityMean)

561 rows × 2 columns

In [15]:
```
feature_dup_df = pd.DataFrame(data=feature_name_df.groupby('column_name').cumcount(), columns=['dup_cnt'])
feature_dup_df.head(20)
```

Out[15]:

	dup_cnt
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0

1-2. 중복된 피처명을 확인

In [16]:
```
feature_dup_df = feature_name_df.groupby('column_name').count()
feature_dup_df
```

Out[16]:

column_name	column_index
angle(X,gravityMean)	1
angle(Y,gravityMean)	1
angle(Z,gravityMean)	1
angle(tBodyAccJerkMean),gravityMean)	1
angle(tBodyAccMean,gravity)	1
...	...
tGravityAccMag-max()	1
tGravityAccMag-mean()	1
tGravityAccMag-min()	1
tGravityAccMag-sma()	1
tGravityAccMag-std()	1

477 rows × 1 columns

In [17]:
```
print(feature_dup_df[feature_dup_df['column_index'] > 1].count())
feature_dup_df[feature_dup_df['column_index'] > 1].head()
```

```
column_index    42
dtype: int64
```

Out[17]:

	column_index
column_name	
fBodyAcc-bandsEnergy()-1,16	3
fBodyAcc-bandsEnergy()-1,24	3
fBodyAcc-bandsEnergy()-1,8	3
fBodyAcc-bandsEnergy()-17,24	3
fBodyAcc-bandsEnergy()-17,32	3

1-3. 중복된 피처명에 대해 새로운 피처명을 부여하는 데이터프레임을 반환 함수 만들기

> 중복된 피처명이 있을 경우 모델 학습 등의 과정에서 혼란을 방지하기 위함

In [18]:
```
def get_new_feature_name_df(old_feature_name_df):
    feature_dup_df = pd.DataFrame(data=old_feature_name_df.groupby('column_name').cumcount(), columns=['dup_cnt'])
    print('*'*20)
    print(feature_dup_df)
    feature_dup_df = feature_dup_df.reset_index()
    new_feature_name_df = pd.merge(old_feature_name_df.reset_index(), feature_dup_df, how='outer')
    new_feature_name_df['column_name'] = new_feature_name_df[['column_name', 'dup_cnt']].apply(lambda x : x[0]+'_'+str(x[1]) i

    return new_feature_name_df
```

1-4. human_activity 데이터를 불러오는 함수를 정의한 뒤 해당 데이터를 Train과 Test로 나누어 반환하는 작업을 수행

```
In [19]: def get_human_dataset():
             feature_name_df = pd.read_csv('./human_activity/features.txt', sep='₩s+', header=None, names=['column_index', 'column_name
             print(feature_name_df)

             # 중복된 피처명을 수정하는 get_new_feature_name_df()를 이용, 신규 피처명 DataFrame 생성
             new_feature_name_df = get_new_feature_name_df(feature_name_df)
             feature_name = new_feature_name_df.iloc[:, 1].values.tolist()
             print(new_feature_name_df)

             X_train = pd.read_csv('./human_activity/train/X_train.txt', sep='₩s+', names=feature_name)
             X_test = pd.read_csv('./human_activity/test/X_test.txt', sep='₩s+', names=feature_name)

             y_train = pd.read_csv('./human_activity/train/y_train.txt', sep='₩s+', names=['action'])
             y_test = pd.read_csv('./human_activity/test/y_test.txt', sep='₩s+', names=['action'])

             return X_train, X_test, y_train, y_test

         X_train, X_test, y_train, y_test = get_human_dataset()
```

```
     column_index                        column_name
0               1                  tBodyAcc-mean()-X
1               2                  tBodyAcc-mean()-Y
2               3                  tBodyAcc-mean()-Z
3               4                   tBodyAcc-std()-X
4               5                   tBodyAcc-std()-Y
..            ...                                ...
556           557      angle(tBodyGyroMean,gravityMean)
557           558  angle(tBodyGyroJerkMean,gravityMean)
558           559               angle(X,gravityMean)
559           560               angle(Y,gravityMean)
560           561               angle(Z,gravityMean)

[561 rows x 2 columns]
********************
     dup_cnt
0          0
1          0
2          0
3          0
4          0
..       ...
556        0
557        0
558        0
559        0
560        0

[561 rows x 1 columns]
     index  column_index                        column_name  dup_cnt
0        0             1                  tBodyAcc-mean()-X        0
1        1             2                  tBodyAcc-mean()-Y        0
2        2             3                  tBodyAcc-mean()-Z        0
3        3             4                   tBodyAcc-std()-X        0
4        4             5                   tBodyAcc-std()-Y        0
..     ...           ...                                ...      ...
556    556           557      angle(tBodyGyroMean,gravityMean)        0
557    557           558  angle(tBodyGyroJerkMean,gravityMean)        0
558    558           559               angle(X,gravityMean)        0
559    559           560               angle(Y,gravityMean)        0
560    560           561               angle(Z,gravityMean)        0
```

```
[561 rows x 4 columns]
```

1-5. 결과 확인

In [20]:
```
print('# 학습 피처 데이터셋 info() #')
print(X_train.info())
```

```
# 학습 피처 데이터셋 info() #
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7352 entries, 0 to 7351
Columns: 561 entries, 1 to 561
dtypes: float64(561)
memory usage: 31.5 MB
None
```

In [21]:
```
print(y_train['action'].value_counts())
```

```
action
6    1407
5    1374
4    1286
1    1226
2    1073
3     986
Name: count, dtype: int64
```

2. Basic Machine Learning 정확도 검증

2-1. 라이브러리 import + 모델 생성 및 학습

In [30]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

import warnings
warnings.filterwarnings('ignore')

# 결정트리, Random Forest, 로지스틱 회귀를 위한 사이킷런 Classifier 클래스 생성
dt_clf = DecisionTreeClassifier(random_state=11)
rf_clf = RandomForestClassifier(random_state=11)
lr_clf = LogisticRegression(random_state=11)

# DecisionTreeClassifier 학습/예측/평가
dt_clf.fit(X_train, y_train)
dt_pred = dt_clf.predict(X_test)
print('DecisionTreeClassifier 정확도: {0:.4f}'.format(accuracy_score(y_test, dt_pred)))

# RandomForestClassifer 학습/예측/평가
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)
print('RandomforestClassifier 정확도: {0:.4f}'.format(accuracy_score(y_test, rf_pred)))

# LogisticRegression 학습/예측/평가
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_test)
print('LogisticRegression 정확도: {0:.4f}'.format(accuracy_score(y_test, lr_pred)))
```

```
DecisionTreeClassifier 정확도: 0.8612
RandomforestClassifier 정확도: 0.9260
LogisticRegression 정확도: 0.9576
```

2-2. 결과 확인

In [23]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

dt_clf = DecisionTreeClassifier(random_state=11)

dt_clf.fit(X_train, y_train)
pred = dt_clf.predict(X_test)

print(f'예측 정확도: {accuracy_score(y_test, pred):.4f}')

# DecisionTreeClassifier의 하이퍼 파라미터 추출
print('DecisionTreeClassifier 기본 하이퍼 파라미터: \n', dt_clf.get_params())
```

```
예측 정확도: 0.8612
DecisionTreeClassifier 기본 하이퍼 파라미터:
 {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': Non
e, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_stat
e': 11, 'splitter': 'best'}
```

3. Machine Learning with GridSearchCV (하이퍼 파라미터 튜닝) 정확도 검증

3-1. 라이브러리 import + 모델 생성 및 학습

In [24]:
```
from sklearn.model_selection import GridSearchCV

params = {'max_depth' : [6, 8, 10, 12, 16, 20, 24]}

grid_cv = GridSearchCV(dt_clf, param_grid=params, scoring='accuracy', cv=5, verbose=1)
grid_cv.fit(X_train, y_train)
print(f'GridSearchCV 최고 평균 정확도 수치: {grid_cv.best_score_:.4f}')
print('GridSearchCV 최적 하이퍼 파라미터:', grid_cv.best_params_)
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
GridSearchCV 최고 평균 정확도 수치: 0.8523
GridSearchCV 최적 하이퍼 파라미터: {'max_depth': 8}
```

In [25]:
```
# GridSearchCV 객체의 cv_results_ 속성을 DataFrame으로 생성
cv_results_df = pd.DataFrame(grid_cv.cv_results_)

# max_depth 파라미터 값과 그때의 테스트(Evaluation) set, 학습 데이터 셋의 정확도 수치 추출
cv_results_df[['param_max_depth', 'mean_test_score']]
```

Out[25]:

	param_max_depth	mean_test_score
0	6	0.842221
1	8	0.852294
2	10	0.844269
3	12	0.840871
4	16	0.846175
5	20	0.840188
6	24	0.840053

In [26]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

dtree = DecisionTreeClassifier(random_state=11)
print(id(dtree))

# parameter들을 dictionary 형태로 설정
params = {'max_depth': [8, 9, 10], 'min_samples_split': [9, 10]}
grid_dtree = GridSearchCV(dtree, param_grid=params, cv=3, refit=True)
grid_dtree.fit(X_train, y_train)


print('GridSearchCV 최적 하이퍼 파라미터 :', grid_dtree.best_params_)
print(f'GridSearchCV 최고 정확도: {grid_dtree.best_score_:.4f}')
best_dtree = grid_dtree.best_estimator_
```

```
1278245811088
GridSearchCV 최적 하이퍼 파라미터 : {'max_depth': 8, 'min_samples_split': 10}
GridSearchCV 최고 정확도: 0.8334
```

3-2. params 값 변경 후 확인

In [27]:
```
params = {
    'max_depth': [8, 12, 16, 20],
    'min_samples_split': [16, 24],
}

gird_cv = GridSearchCV(dt_clf, param_grid=params, scoring='accuracy', cv=5, verbose=1)
grid_cv.fit(X_train, y_train)

print(f'GridSearchCV 최고 평균 정확도 수치: {grid_cv.best_score_:.4f}')
print('GridSearchCV 최적 하이퍼 파라미터 :', grid_cv.best_params_)
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
GridSearchCV 최고 평균 정확도 수치: 0.8523
GridSearchCV 최적 하이퍼 파라미터 : {'max_depth': 8}
```

3-3. 최적 하이퍼 파라미터의 결정 트리 예측 정확도 확인

In [28]:
```
best_df_clf = grid_cv.best_estimator_
pred1 = best_df_clf.predict(X_test)
accuracy = accuracy_score(y_test, pred1)
print(f'결정 트리 예측 정확도: {accuracy:.4f}')
```

```
결정 트리 예측 정확도: 0.8717
```

---

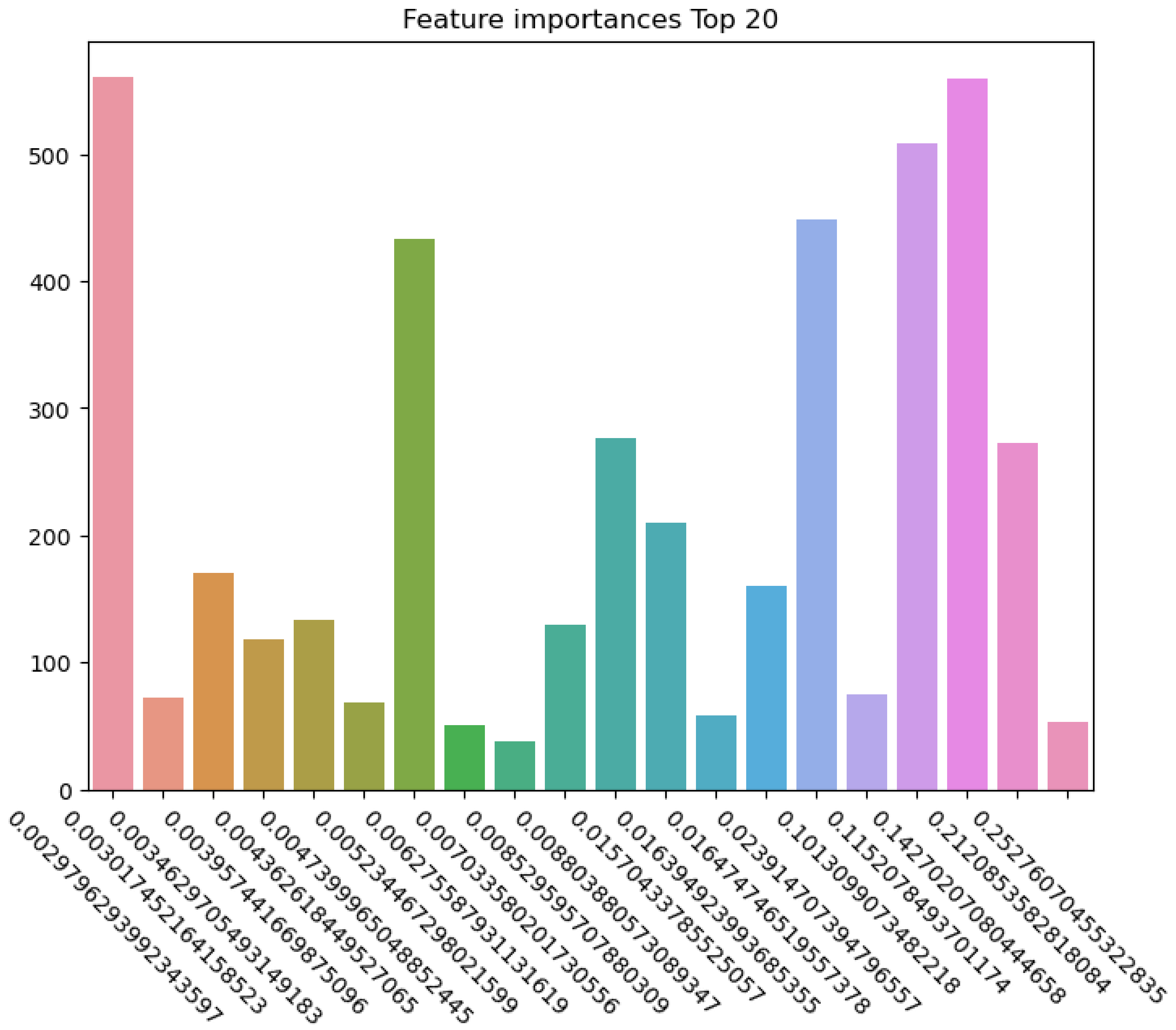
4. Feature 중요도 시각화

In [29]:
```
import seaborn as sns

ftr_importances_values = best_df_clf.feature_importances_
ftr_importances = pd.Series(ftr_importances_values, index=X_train.columns)

ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]

plt.figure(figsize=(8, 6))
plt.title('Feature importances Top 20')
plt.xticks(rotation=-45)
sns.barplot(x=ftr_top20, y=ftr_top20.index)
plt.show()
```


Feature importances Top 20
0
100
200
300
400
500
0.0029796293992343597
0.003017452164158523
0.0034629705493149183
0.003957441669875096
0.004362618449527065
0.0047399965048852445
0.005234467298021599
0.006275587931131619
0.007033580201730556
0.008529595707880309
0.008803880573089347
0.015704337855525057
0.016394923993685355
0.016474746519557378
0.023914707394796557
0.1013099073482218
0.1152078493701174
0.14270207080444658
0.2120853582818084
0.25276070455322835