Disclaimer: There is a parameter n_jobs=15 in some of the functions. That corresponds to number of threads on your pc. Change it depending on your system or remove it altogether

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
import os
from sklearn.feature_selection import RFECV
from sklearn.feature_selection import RFE
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import chi2
from sklearn.ensemble import ExtraTreesClassifier
```

In [2]:
```
song_data = pd.read_csv("dataset.csv", sep = ',')
song_data = song_data[song_data['Genre'] != 'n-a']
song_data = song_data[song_data.iloc[:, 0].str.contains("Global") == True]
le = preprocessing.LabelEncoder()
xcols = ['Genre_new','Artist','Explicit','danceability','energy','key','loudne
X = song_data[xcols]
X = X.apply(lambda col: le.fit_transform(col.astype(str)), axis=0, result_type
y = song_data[['Genre_new']]
y = y.apply(lambda col: le.fit_transform(col.astype(str)), axis=0, result_type
print(X.shape)
print(y.shape)
print(type(X))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:34
44: DtypeWarning: Columns (7,8,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25) h
ave mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
(5423, 37)
(5423, 1)
<class 'pandas.core.frame.DataFrame'>
```

In [3]:
```
X.drop(X.index[X['Genre_new'] == 0], inplace=True)
y.drop(y.index[y['Genre_new'] == 0], inplace=True)
print(X.shape)
print(y.shape)
X.drop('Genre_new',axis=1, inplace=True)
print(X.shape)
```

```
(5421, 37)
(5421, 1)
(5421, 36)
```

In [4]:
```
xcols = ['Artist','Explicit','danceability','energy','key','loudness','mode',
normalized_X = X.values
std_scalor = preprocessing.MinMaxScaler()
X_scaled = std_scalor.fit_transform(normalized_X)
normalized_X = pd.DataFrame(X_scaled, columns=xcols)
print(normalized_X.shape)
print(type(normalized_X))
```

```
(5421, 36)
<class 'pandas.core.frame.DataFrame'>
```

In [5]:
```
normalized_X.head()
```

Out[5]:

	Artist	Explicit	danceability	energy	key	loudness	mode	speechiness	acoustics	instru
0	0.734448	0.0	0.717330	0.671233	0.090909	0.327564	1.0	0.835965	0.716433	
1	0.888025	0.0	0.440341	0.448319	0.727273	0.902964	0.0	0.357895	0.472784	
2	0.469285	1.0	0.321023	0.458281	0.727273	0.408340	1.0	0.440351	0.625713	
3	0.312208	0.0	0.359375	0.930262	0.090909	0.776292	1.0	0.150877	0.004147	
4	0.545879	0.0	0.470170	0.674969	0.363636	0.627590	0.0	0.035965	0.065319	

5 rows × 36 columns

In [6]:
```
X_train, X_test, Y_train, Y_test = train_test_split(normalized_X, y, test_size
print(X_train.shape)
print(X_test.shape)
```

```
(4336, 36)
(1085, 36)
```

Performance without Feature Selection

In [7]:

```
%%time
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=20)

clf_dt = DecisionTreeClassifier()
clf_rf = RandomForestClassifier()
clf_knn = KNeighborsClassifier()
clf_lrsag = LogisticRegression(C=0.1, solver="sag", penalty="l2", max_iter = 1
clf_svm = svm.SVC(C=0.1, kernel='linear')

clf_dt.fit(X_train, Y_train.values.ravel())
clf_rf.fit(X_train, Y_train.values.ravel())
clf_knn.fit(X_train, Y_train.values.ravel())
clf_lrsag.fit(X_train, Y_train.values.ravel())
clf_svm.fit(X_train, Y_train.values.ravel())

tracdt = cross_val_score(clf_dt, X_train, Y_train.values.ravel(), cv=cv, n_jo
teacdt = clf_dt.score(X_test, Y_test.values.ravel())

tracrf = cross_val_score(clf_rf, X_train, Y_train.values.ravel(), cv=cv, n_jo
teacrf = clf_rf.score(X_test, Y_test.values.ravel())

tracknn = cross_val_score(clf_knn, X_train, Y_train.values.ravel(), cv=cv, n_
teacknn = clf_knn.score(X_test, Y_test.values.ravel())

traclrsag = cross_val_score(clf_lrsag, X_train, Y_train.values.ravel(), cv=cv,
teaclrsag = clf_lrsag.score(X_test, Y_test.values.ravel())

tracsvm = cross_val_score(clf_svm, X_train, Y_train.values.ravel(), cv=cv, n_
teacsvm = clf_svm.score(X_test, Y_test.values.ravel())

classifiers = ['Decision Trees','Random Forests','K nearest neighbors', 'Logis
train_error = [1-tracdt, 1-tracrf, 1-tracknn, 1-traclrsag, 1-tracsvm]
test_error = [1-teacdt, 1-teacrf, 1-teacknn, 1-teaclrsag, 1-teacsvm]
print(classifiers)
print(train_error)
print(test_error)
```

```
['Decision Trees', 'Random Forests', 'K nearest neighbors', 'Logistic Regressi
on', 'SVM']
[0.36938511722948886, 0.382690850459233, 0.5378937005778994, 0.484847942231351
3, 0.5016593586700866]
[0.3723502304147466, 0.3907834101382488, 0.5253456221198156, 0.492165898617511
5, 0.5179723502304148]
Wall time: 1min 16s
```

Tree Based Feature Selection

In [8]:

```
%%time
rf_clf = ExtraTreesClassifier()
rf_clf = rf_clf.fit(X_train, Y_train.values.ravel())
model = SelectFromModel(rf_clf, prefit=True)
rf_X_train = X_train.loc[:, model.get_support()]
rf_X_test = X_test.loc[:, model.get_support()]
print(rf_X_train.shape)
print(rf_X_test.shape)
```

```
(4336, 13)
(1085, 13)
Wall time: 692 ms
```

In [9]:

```
%%time
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=20)

clf_dt = DecisionTreeClassifier()
clf_rf = RandomForestClassifier()
clf_knn = KNeighborsClassifier()
clf_lrsag = LogisticRegression(C=0.1, solver="sag", penalty="l2", max_iter = 1
clf_svm = svm.SVC(C=0.1, kernel='linear')

clf_dt.fit(rf_X_train, Y_train.values.ravel())
clf_rf.fit(rf_X_train, Y_train.values.ravel())
clf_knn.fit(rf_X_train, Y_train.values.ravel())
clf_lrsag.fit(rf_X_train, Y_train.values.ravel())
clf_svm.fit(rf_X_train, Y_train.values.ravel())

tracdt = cross_val_score(clf_dt, rf_X_train, Y_train.values.ravel(), cv=cv, n_
teacdt = clf_dt.score(rf_X_test, Y_test.values.ravel())

tracrf = cross_val_score(clf_rf, rf_X_train, Y_train.values.ravel(), cv=cv, n_
teacrf = clf_rf.score(rf_X_test, Y_test.values.ravel())

tracknn = cross_val_score(clf_knn, rf_X_train, Y_train.values.ravel(), cv=cv,
teacknn = clf_knn.score(rf_X_test, Y_test.values.ravel())

traclrsag = cross_val_score(clf_lrsag, rf_X_train, Y_train.values.ravel(), cv=
teaclrsag = clf_lrsag.score(rf_X_test, Y_test.values.ravel())

tracsvm = cross_val_score(clf_svm, rf_X_train, Y_train.values.ravel(), cv=cv,
teacsvm = clf_svm.score(rf_X_test, Y_test.values.ravel())

classifiers = ['Decision Trees','Random Forests','K nearest neighbors', 'Logis
tree_based_train_error = [1-tracdt, 1-tracrf, 1-tracknn, 1-traclrsag, 1-tracsv
tree_based_test_error = [1-teacdt, 1-teacrf, 1-teacknn, 1-teaclrsag, 1-teacsvm
print(classifiers)
print(tree_based_train_error)
print(tree_based_test_error)
```

```
['Decision Trees', 'Random Forests', 'K nearest neighbors', 'Logistic Regressi
on', 'SVM']
[0.3650829067379019, 0.3665122763699833, 0.5362317610497973, 0.509190701461244
6, 0.525496668830685]
[0.35668202764976964, 0.36589861751152075, 0.536405529953917, 0.51428571428571
42, 0.5493087557603686]
Wall time: 41.1 s
```

Recursive Feature Elimination

In [10]:

```
%%time
rfecv_clf = svm.SVC(C=0.1, kernel='linear')
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=20)
rfecv = RFECV(estimator = rfecv_clf, step = 1, cv = cv, scoring='accuracy', n_
rfetrain=rfecv.fit(X_train, Y_train.values.ravel())
print('Optimal number of features :', rfecv.n_features_)

rfe = RFE(estimator=rfecv_clf, n_features_to_select=rfecv.n_features_, step=1)
rfe = rfe.fit(X_train, Y_train.values.ravel())

rfe_X_train = X_train.loc[:, rfe.get_support()]
rfe_X_test = X_test.loc[:, rfe.get_support()]
print(rfe_X_train.shape)
print(rfe_X_test.shape)
```

```
Optimal number of features : 35
(4336, 35)
(1085, 35)
Wall time: 11min 35s
```

In [11]:

```
%%time
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=20)

clf_dt = DecisionTreeClassifier()
clf_rf = RandomForestClassifier()
clf_knn = KNeighborsClassifier()
clf_lrsag = LogisticRegression(C=0.1, solver="sag", penalty="l2", max_iter = 1
clf_svm = svm.SVC(C=0.1, kernel='linear')

clf_dt.fit(rfe_X_train, Y_train.values.ravel())
clf_rf.fit(rfe_X_train, Y_train.values.ravel())
clf_knn.fit(rfe_X_train, Y_train.values.ravel())
clf_lrsag.fit(rfe_X_train, Y_train.values.ravel())
clf_svm.fit(rfe_X_train, Y_train.values.ravel())

tracdt = cross_val_score(clf_dt, rfe_X_train, Y_train.values.ravel(), cv=cv, n
teacdt = clf_dt.score(rfe_X_test, Y_test.values.ravel())

tracrf = cross_val_score(clf_rf, rfe_X_train, Y_train.values.ravel(), cv=cv, n
teacrf = clf_rf.score(rfe_X_test, Y_test.values.ravel())

tracknn = cross_val_score(clf_knn, rfe_X_train, Y_train.values.ravel(), cv=cv,
teacknn = clf_knn.score(rfe_X_test, Y_test.values.ravel())

traclrsag = cross_val_score(clf_lrsag, rfe_X_train, Y_train.values.ravel(), cv
teaclrsag = clf_lrsag.score(rfe_X_test, Y_test.values.ravel())

tracsvm = cross_val_score(clf_svm, rfe_X_train, Y_train.values.ravel(), cv=cv,
teacsvm = clf_svm.score(rfe_X_test, Y_test.values.ravel())

classifiers = ['Decision Trees','Random Forests','K nearest neighbors', 'Logis
RFE_train_error = [1-tracdt, 1-tracrf, 1-tracknn, 1-traclrsag, 1-tracsvm]
RFE_test_error = [1-teacdt, 1-teacrf, 1-teacknn, 1-teaclrsag, 1-teacsvm]
print(classifiers)
print(RFE_train_error)
print(RFE_test_error)
```

```
['Decision Trees', 'Random Forests', 'K nearest neighbors', 'Logistic Regressi
on', 'SVM']
[0.37491704004853077, 0.3873864688540991, 0.5395482966337097, 0.48519537361245
624, 0.5016266323261779]
[0.36589861751152075, 0.39262672811059907, 0.519815668202765, 0.49216589861751
15, 0.5179723502304148]
Wall time: 1min 23s
```

In [12]:

```
for index, rank in enumerate(rfe.ranking_):
    print("Rank " + str(rank)+ " feature: " + xcols[index])
```

```
Rank 1 feature: Artist
Rank 1 feature: Explicit
Rank 1 feature: danceability
Rank 1 feature: energy
Rank 1 feature: key
Rank 1 feature: loudness
Rank 1 feature: mode
Rank 1 feature: speechiness
Rank 1 feature: acoustics
Rank 1 feature: instrumentalness
Rank 1 feature: liveliness
Rank 1 feature: valence
Rank 1 feature: tempo
Rank 1 feature: duration_ms
Rank 2 feature: time_signature
Rank 1 feature: syuzhet_norm
Rank 1 feature: bing_norm
Rank 1 feature: afinn_norm
Rank 1 feature: nrc_norm
Rank 1 feature: n_words
Rank 1 feature: anger_norm2
Rank 1 feature: anticipation_norm2
Rank 1 feature: disgust_norm2
Rank 1 feature: fear_norm2
Rank 1 feature: joy_norm2
Rank 1 feature: sadness_norm2
Rank 1 feature: surprise_norm2
Rank 1 feature: trust_norm2
Rank 1 feature: negative_norm2
Rank 1 feature: positive_norm2
Rank 1 feature: negative_bog_jr
Rank 1 feature: positive_bog_jr
Rank 1 feature: Bayes
Rank 1 feature: LDA_Topic
Rank 1 feature: Top10_dummy
Rank 1 feature: Top50_dummy
```

ANOVA F measure

https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/

In [13]:

```
%%time

num_features = []
dt1,rf1,knn1,lr1,svm1 = [],[],[],[],[]
dt2,rf2,knn2,lr2,svm2 = [],[],[],[],[]
xcols = np.array(xcols)
for count in range(1,37):
    print(count)
    clf_dt = DecisionTreeClassifier()
    clf_rf = RandomForestClassifier()
    clf_knn = KNeighborsClassifier()
    clf_lrsag = LogisticRegression(C=0.1, solver="sag", penalty="l2", max_ite
    clf_svm = svm.SVC(C=0.1, kernel='linear')
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=20)

    test = SelectKBest(score_func=f_classif, k = count)
    X_F_measure = test.fit_transform(normalized_X,y.values.ravel())
    f_xcols = xcols[test.get_support()]
    X_F_measure = pd.DataFrame(X_F_measure, columns=f_xcols)
    X_train2, X_test2, Y_train2, Y_test2 = train_test_split(X_F_measure, y, te

    clf_dt.fit(X_train2, Y_train2.values.ravel())
    dt1.append(cross_val_score(clf_dt, X_train2, Y_train2.values.ravel(), cv=c
    dt2.append(clf_dt.score(X_test2, Y_test2.values.ravel()))

    clf_rf.fit(X_train2, Y_train2.values.ravel())
    rf1.append(cross_val_score(clf_rf, X_train2, Y_train2.values.ravel(), cv=c
    rf2.append(clf_rf.score(X_test2, Y_test2.values.ravel()))

    clf_knn.fit(X_train2, Y_train2.values.ravel())
    knn1.append(cross_val_score(clf_knn, X_train2, Y_train2.values.ravel(), cv
    knn2.append(clf_knn.score(X_test2, Y_test2.values.ravel()))

    clf_lrsag.fit(X_train2, Y_train2.values.ravel())
    lr1.append(cross_val_score(clf_lrsag, X_train2, Y_train2.values.ravel(), c
    lr2.append(clf_lrsag.score(X_test2, Y_test2.values.ravel()))

    clf_svm.fit(X_train2, Y_train2.values.ravel())
    svm1.append(cross_val_score(clf_svm, X_train2, Y_train2.values.ravel(), cv
    svm2.append(clf_svm.score(X_test2, Y_test2.values.ravel()))

    num_features.append(count)

classifiers = ['Decision Trees','Random Forests','K nearest neighbors', 'Logis
train_acc_F_measure = []
test_acc_F_measure = []
best_num_feats = []

test_acc_F_measure.append(max(dt2))
best_num_feats.append(num_features[dt2.index(max(dt2))])
train_acc_F_measure.append(dt1[dt2.index(max(dt2))])

test_acc_F_measure.append(max(rf2))
best_num_feats.append(num_features[rf2.index(max(rf2))])
train_acc_F_measure.append(rf1[rf2.index(max(rf2))])

test_acc_F_measure.append(max(knn2))
best_num_feats.append(num_features[knn2.index(max(knn2))])
```

```
train_acc_F_measure.append(knn1[knn2.index(max(knn2))])

test_acc_F_measure.append(max(lr2))
best_num_feats.append(num_features[lr2.index(max(lr2))])
train_acc_F_measure.append(lr1[lr2.index(max(lr2))])

test_acc_F_measure.append(max(svm2))
best_num_feats.append(num_features[svm2.index(max(svm2))])
train_acc_F_measure.append(svm1[svm2.index(max(svm2))])

test_err_F = [(1-i) for i in test_acc_F_measure]
train_err_F = [(1-j) for j in train_acc_F_measure]

print(classifiers)
print(train_err_F)
print(test_err_F)
print(best_num_feats)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
['Decision Trees', 'Random Forests', 'K nearest neighbors', 'Logistic Regressi
on', 'SVM']
[0.3403055256968317, 0.3932220548951161, 0.5345126967571652, 0.490877864220261
6, 0.5072422866934154]
[0.3262672811059908, 0.3493087557603687, 0.5179723502304148, 0.462672811059907
```

```
9, 0.48202764976958523]
[20, 27, 31, 35, 31]
Wall time: 35min 43s
```