Feature Selection

Feature selection is an important next step that will help us to prepare for model selection and hyperparameter tuning. Eliminating irrelevant features will help to make our model faster, more efficient and more accurate.

0. Imports and Data Preparation

Let us start from importing necessary libraries and loading our dataset.

In [ ]:
```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn import preprocessing
import numpy as np
from sklearn import *
import matplotlib.pyplot as plt
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

In [ ]:
```
file_path = '/content/gdrive/MyDrive/Capstone/final_dataset.csv'
df=pd.read_csv(file_path,)
df
```

We are dropping some columns that we do not need.

In [ ]:
```
df2=df.drop(['index', 'Title',  'Artist','track_id'], axis=1)
df2
X = df2.drop(['Popular'], axis=1)
y = df2['Popular']
```

In [ ]:
```
#Let us try with the standard scaler
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(sampling_strategy="not majority")
X_r, y_res = ros.fit_resample(X, y)

stsc=preprocessing.StandardScaler()
X_res = stsc.fit_transform(X_r)

#splitting data into train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
     X_res, y_res, test_size=0.33, random_state=42)

# fitting a logistic regression model
from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression()
logisticRegr.fit(X_train, y_train)
predictions = logisticRegr.predict(X_test)
```

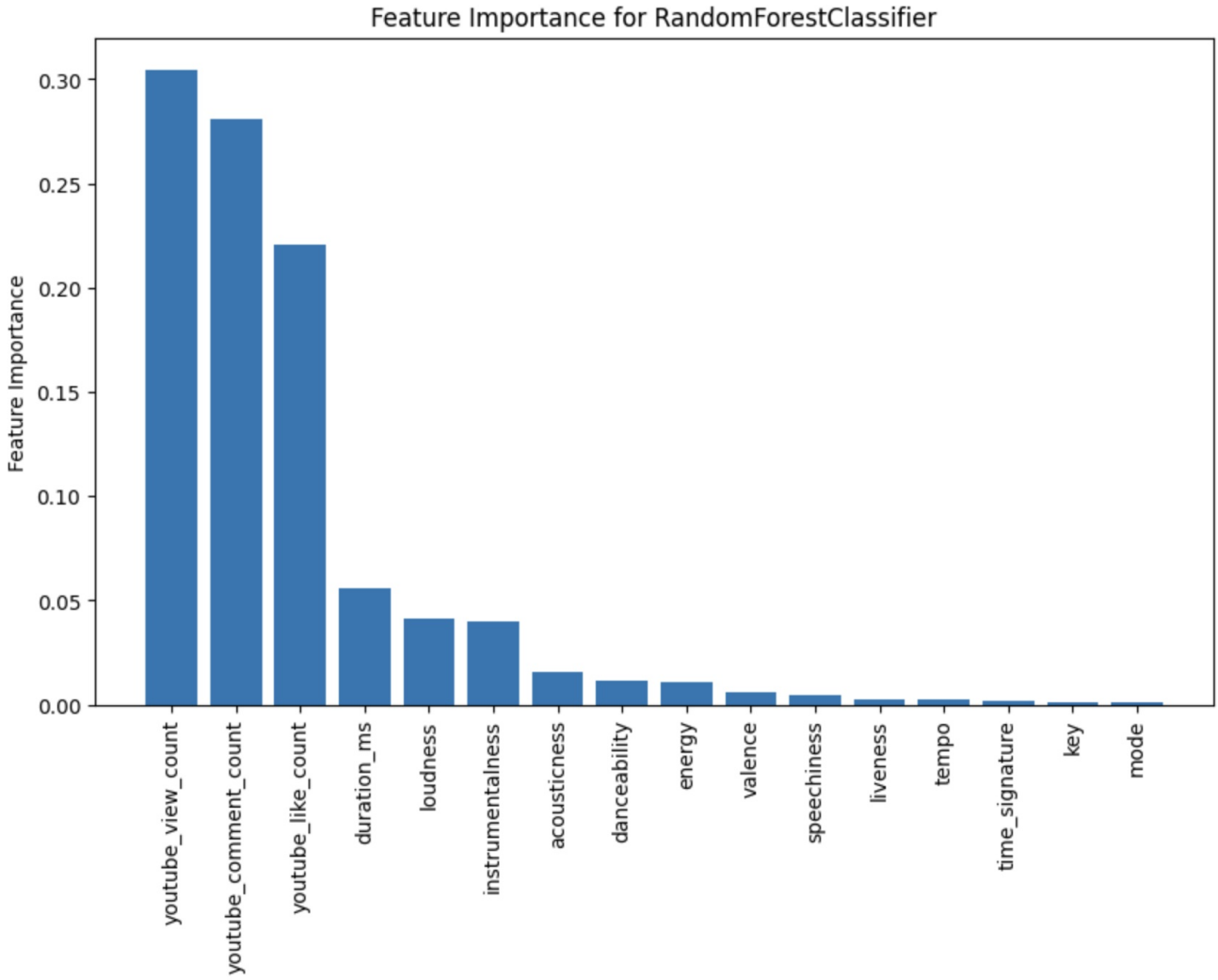
In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

# initiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 500, max_depth = 4, max_features = 3, bootstrap = True, random_state
# train on training data
rf.fit(X_train, y_train)
predictions = rf.predict(X_test)
```

1. Features Importance

This refers to a measure of the contribution or significance of each feature (predictor variable) in a machine learning model(Random Forest here, since feature selection is most appropriate for RF). Feature importance will help us understand which features.

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

# Access feature importance
feature_importance = rf.feature_importances_

# Replace this line with your actual feature names
num_features = X_train.shape[1]
feature_names = X.columns

# Sort features by importance in descending order
sorted_idx = np.argsort(feature_importance)[::-1]
```

```
# Create a bar plot of feature importance
plt.figure(figsize=(10, 6))
plt.bar(range(num_features), feature_importance[sorted_idx])
plt.xticks(range(len(feature_importance)), feature_names[sorted_idx], rotation=90)
plt.xlabel("Feature")
plt.ylabel("Feature Importance")
plt.title("Feature Importance for RandomForestClassifier")
plt.show()
```



We can see in here that the Mode, the Key, the time signature the Tempo and the Liveness, the speechiness and the Valence do not contribute a lot in this model. To make the model better we can remove them from our selected features.

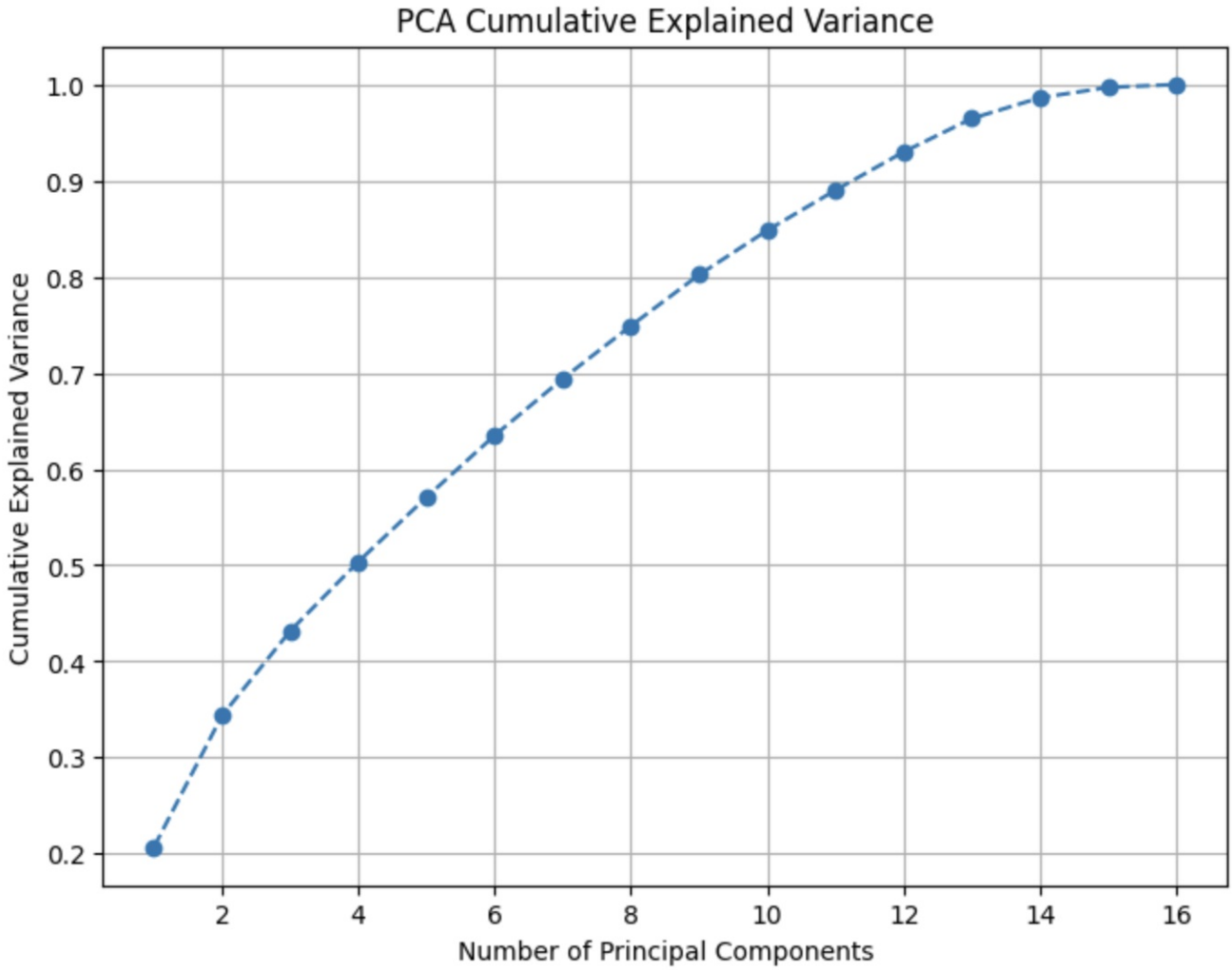
2. PCA (Principal component analysis)

This statistical technique is used to reduce dimensionality.

In [ ]:
```
from sklearn.decomposition import PCA
pca = PCA()
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_std = scaler.fit_transform(X_res)


n_components = 16
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_std)

explained_variance_ratio = pca.explained_variance_ratio_
cumulative_explained_variance = np.cumsum(explained_variance_ratio)

plt.figure(figsize=(8, 6))
```

```
plt.plot(range(1, n_components + 1), cumulative_explained_variance, marker='o', linestyle='--')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('PCA Cumulative Explained Variance')
plt.grid()
plt.show()

k = np.argmax(cumulative_explained_variance >= 0.95) + 1
print(f"Number of components to explain at least 95% variance: {k}")

X_selected = X_pca[:, :k]
```



This means that in our dataset, we need to retain 13 principal components to capture at least 95% of the total variance in the original data.

The cumulative explained variance plot we generated showed that the first 13 principal components collectively account for at least 95% of the variance in the data. This means that these 13 components retain the most important information and structure of our original features. The remaining components contribute very little to the overall variance. By selecting 13 principal components, We are reducing the dimensionality of your dataset. Instead of using all the original audio features, we can now represent your data with a smaller set of 13 features. This can simplify our analysis and potentially reduce noise in the data

Let us look a lot more in the features that are selected from our PCA analysis

In [ ]:
```
n_components = 13

selected_features = set()

top_features = {}

for i in range(n_components):
    feature_contributions = pca.components_[i]
    sorted_features = sorted(zip(X.columns, feature_contributions), key=lambda x: abs(x[1]), reverse=True)
    top_feature = next((feature for feature, contribution in sorted_features if feature not in selected_feature
    if top_feature:
        selected_features.add(top_feature)
        top_features[f'Principal Component {i+1}'] = {'Feature': top_feature, 'Contribution': feature_contribut
top_features_df = pd.DataFrame(top_features).T
top_features_df
```

In here we see the final list of selected features using the PCA. We can also see the Contribution which go in different directions(eg. Negative means negative corolation). The features selected are based on the variance of these features. It is also based on combination

of multiple features.

3. Univariate feature selection

Univariate feature selection is a tool that evaluates the importance of each feature individually based on its relationship with the outcome variable. This is usually assessed through certain statistical measures, such as F-test or chi-squared test.

To perform the univariate feature selection for this classification problem, we will define the number of features through SelectKBest, and we will use ANOVA F-test as our scoring function.

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

In [ ]:
```
#defining model
lr = LogisticRegression()
#keeping track of highest accuracy achieved
max_acc = 0

#iterating over different possible numbers of features
for i in range(1, 17):
    select = SelectKBest(f_classif, k=i)
    select.fit(X_train, y_train)
    X_train_selected = select.transform(X_train)

    #will fit to only those features that got selected
    lr.fit(X_train_selected, y_train)

    #cross-validation to estimate performance in a more stable way
    #using a common stratified k fold cross-validation
    strat_kfold = StratifiedKFold(10, random_state=7, shuffle = True)
    score = cross_val_score(lr, X_train_selected, y_train, scoring='accuracy', cv=strat_kfold)
    print(str(i)+': %.3f (%.3f)' % (np.mean(score), np.std(score)))

    #comparing to identify the best combination of features for the task
    if np.mean(score) > max_acc:
        max_acc = np.mean(score)
        best_n_features = i
        mask = select.get_support(indices=True)

print('\nThe optimal number of features: %i' % best_n_features)
print('The maximum accuracy: %.3f' % max_acc)
print('The features: ', mask)
```

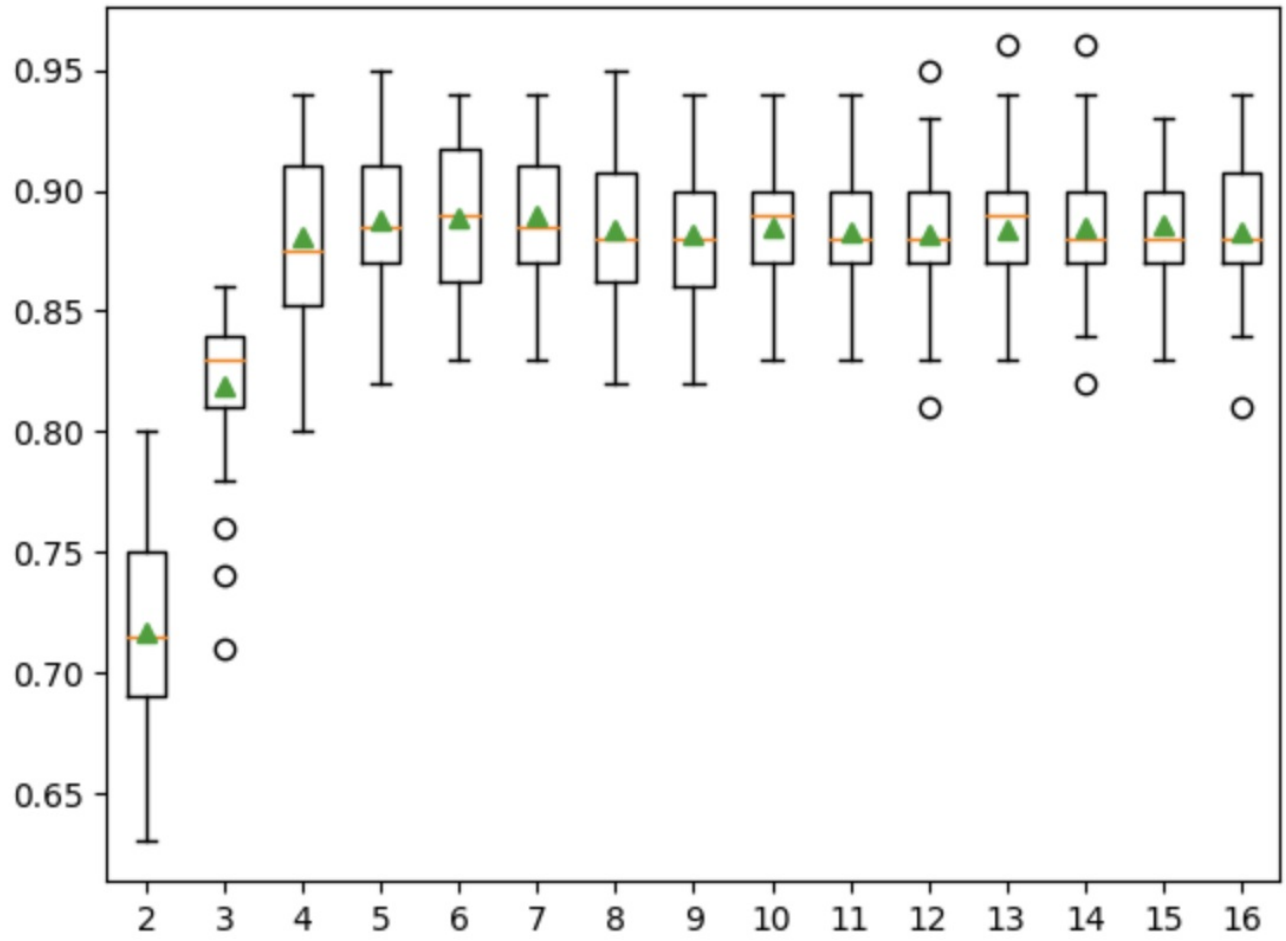
RFE (Recursive Feature Elimination)

Recursive feature elimination is another feature selection technique that fits a model and removes the features (the weakest ones) until the desired number of features is reached. To calculate the feature importance using this method, tree-based models are used, such as Random Forest, Decision Tree Classifier, etc.

In [ ]:
```
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from matplotlib import pyplot
```

In [ ]:
```
#we create synthetic data to use it for feature comparison
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5, random_state=1)

models = dict()

#We go over differnet possible numbers of features (up to 16)
#train the model on those features
for i in range(2, 17):
  rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=i)
  model = DecisionTreeClassifier()
  models[str(i)] = Pipeline(steps=[('s',rfe),('m',model)])

results, names = list(), list()
for name, model in models.items():
  #perform cross-validation to evaluate
  cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
  scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
```

```
  #print the results
  results.append(scores)
  names.append(name)
  print('%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

#plot the graph based on the results
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```



\Now, after knowing how many features are the optimum number, we can actually identify which combination is the best:

In [ ]:
```
rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=7)
model = DecisionTreeClassifier()
rfe = rfe.fit(X, y)


selected = rfe.support_
selected_df = X.loc[:, selected]
print(selected_df.head())
```

Conclusion

In conclusion, we can see that different techniques show us different results for the optimal number and combination of features. Hence, it is hard to come up with the final uniform conclusion.

The best practice would be to use the several best combinations identified by different techniques while training and evaluating models and choosing hyperparameters at our next stage of the project.

The following sources were used:

https://youtu.be/hCwTDTdYirg?si=W8hRWYvvO9BTbWSQ

https://machinelearningmastery.com/rfe-feature-selection-in-python/

https://gist.github.com/pmarcelino/2e4ccd0da0941950cca00bf06b75b396