Berkant Deniz Aktaş
Esen Erdemgil

FINAL RESULTS

In this part, we will conclude our project by building a model and testing it with ML techniques.

If you are using Spotify, you know that there is a playlist called "Discover weekly". This playlist suggests user some songs every week depending on what user is listening. We have interested in this process and tried to create a similar model.
In the previous part of this project, we saw that song attributes matter. Every song has some attributes which show the meta info about the corresponding song.

Imagine you created a playlist with some songs you like. Can we suggest you a song which you might like?

Let's test with the techniques we have learned.

```
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
import seaborn as sns
import graphviz
import pydotplus
import io
from scipy import misc
%matplotlib inline
```

We will use sklearn to make classification and plan is creating decision tree / random forest (decision tree set).

```
data = pd.read_csv('data.csv')
data.describe()
```

SPOTIFIED

```
In [2]: data = pd.read_csv('data.csv')
```

```
In [3]: data.describe()
```

Out[3]:

	Unnamed: 0	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness
count	2017.000000	2017.000000	2017.000000	2.017000e+03	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000
mean	1008.000000	0.187590	0.618422	2.463062e+05	0.681577	0.133286	5.342588	0.190844	-7.085624	0.612295	0.092664
std	582.402066	0.259989	0.161029	8.198181e+04	0.210273	0.273162	3.648240	0.155453	3.761684	0.487347	0.089931
min	0.000000	0.000003	0.122000	1.604200e+04	0.014800	0.000000	0.000000	0.018800	-33.097000	0.000000	0.023100
25%	504.000000	0.009630	0.514000	2.000150e+05	0.563000	0.000000	2.000000	0.092300	-8.394000	0.000000	0.037500
50%	1008.000000	0.063300	0.631000	2.292610e+05	0.715000	0.000076	6.000000	0.127000	-6.248000	1.000000	0.054900
75%	1512.000000	0.265000	0.738000	2.703330e+05	0.846000	0.054000	9.000000	0.247000	-4.746000	1.000000	0.108000
max	2016.000000	0.995000	0.984000	1.004627e+06	0.998000	0.976000	11.000000	0.969000	-0.307000	1.000000	0.816000

```
In [4]: data.head()
```

Out[4]:

	Unnamed: 0	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	time_signature	valence
0	0	0.0102	0.833	204600	0.434	0.021900	2	0.1650	-8.795	1	0.4310	150.062	4.0	0.286
1	1	0.1990	0.743	326933	0.359	0.006110	1	0.1370	-10.401	1	0.0794	160.083	4.0	0.588
2	2	0.0344	0.838	185707	0.412	0.000234	2	0.1590	-7.148	1	0.2890	75.044	4.0	0.173
3	3	0.6040	0.494	199413	0.338	0.510000	5	0.0922	-15.236	1	0.0261	86.468	4.0	0.230
4	4	0.1800	0.678	392893	0.561	0.512000	5	0.4390	-11.648	0	0.0694	174.004	4.0	0.904

So basically, we have a dataset which has songs that we like and we did not like. Thanks to Spotify API, you can reach this data (After authentication).

We have 2017 songs on the list but let's check if there are null values and meta info about our df.

```
data.info()
data.isnull().any().sum()
```

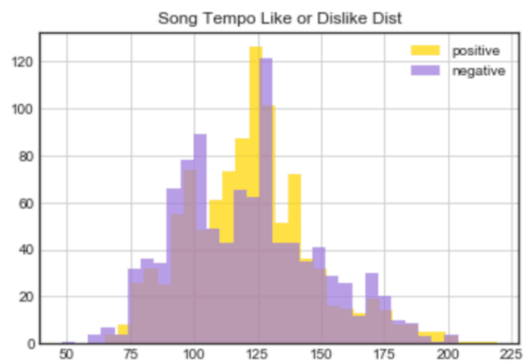
isnull returned 0, so we do not have null data.
For data info we have there are 17 columns which means 17 attributes for a song:

```
RangeIndex: 2017 entries, 0 to 2016
Data columns (total 17 columns):
Unnamed: 0      2017 non-null int64
acousticness         2017 non-null float64
danceability         2017 non-null float64
duration_ms          2017 non-null int64
energy               2017 non-null float64
instrumentalness     2017 non-null float64
key                  2017 non-null int64
liveness             2017 non-null float64
loudness             2017 non-null float64
mode                 2017 non-null int64
speechiness          2017 non-null float64
tempo                2017 non-null float64
time_signature       2017 non-null float64
valence              2017 non-null float64
target               2017 non-null int64
song_title           2017 non-null object
artist               2017 non-null object
dtypes: float64(10), int64(5), object(2)
memory usage: 268.0+ KB
```

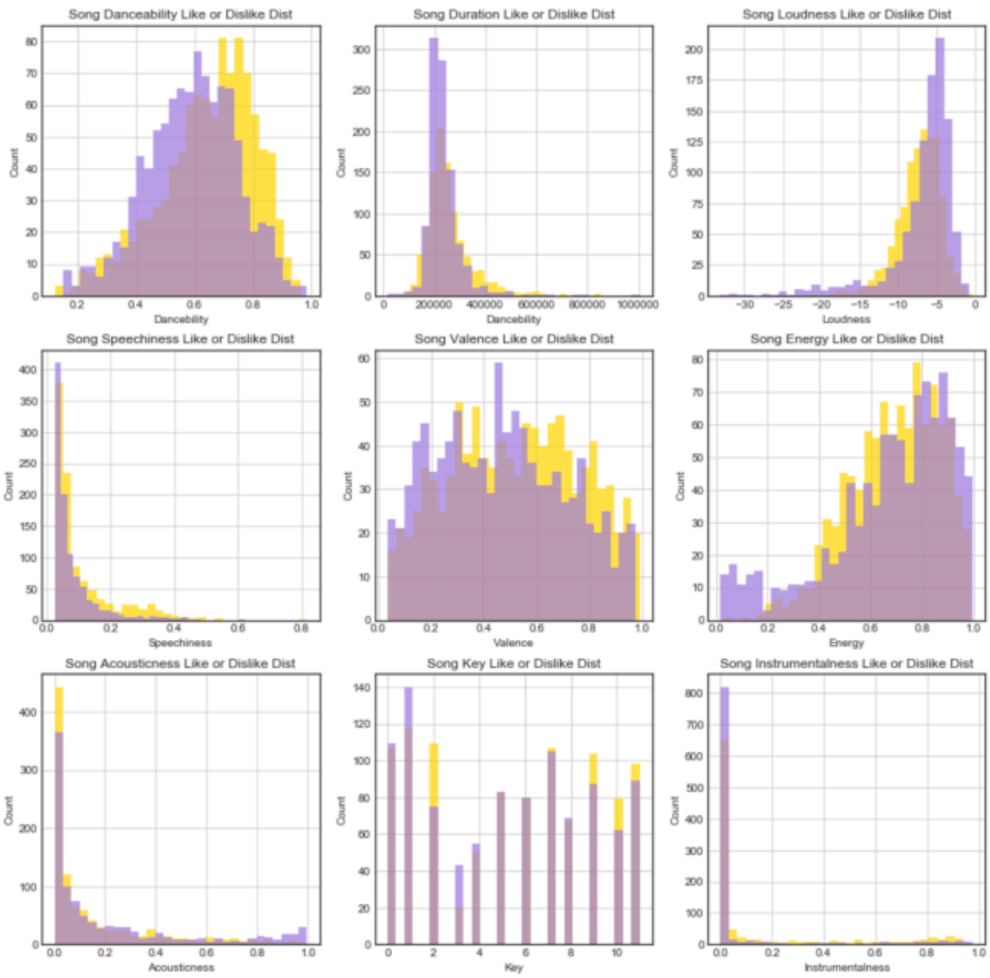
Visualisation of Attributes

So, we need to create histograms to see the attributes. Since 'target' attribute is for our like/dislike we can divide this dataset and show each attribute on a graph based on the target.

An example histogram of all song's tempo is:



Yellow for the songs that have like purple for the songs that disliked. Based on this approach we can create histograms of all attributes.



As you can see from the graphs, some attributes create bias e.g. For some attribute X, the user tends to like a song if X is large enough.

Guessing a new song

Now, sklearn's time to shine.

We need to create a training data and make it learn some stuff.

```
train, test = train_test_split(data, test_size = 0.15,random_state=42)
print("Training size: {}; Test size {}".format(len(train), len(test)))
```

Training size: 1714; Test size 303

So basically, we will train %85 of the data and depend on what they learn we will test remaining %15. Since we know the real results for %15, we can see if our data predict right. And after that we can see how accurate it is!

1) Decision Tree

Decision tree is the first ML technique we will use to test our training data.

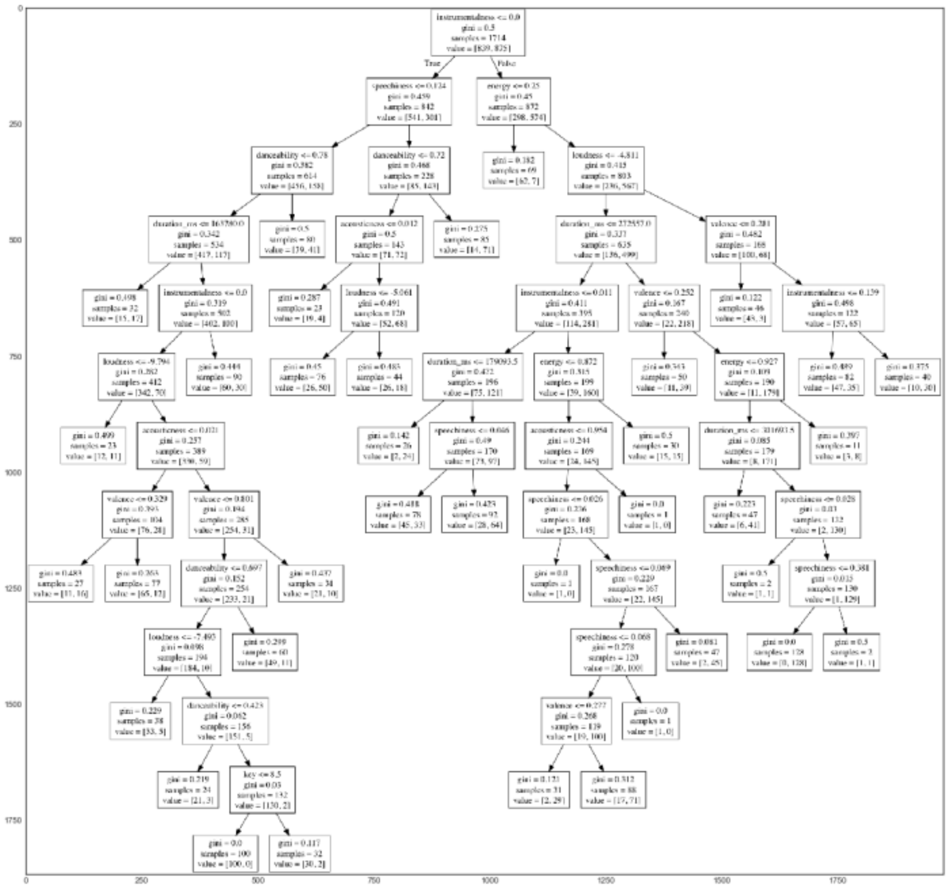
Let's create a classifier.

```
c = DecisionTreeClassifier(min_samples_split=100,random_state=42)
```

Note: 100 sounds a lot but, after trying lower values we decided that 100 is good.

```
features = ["danceability","loudness", "valence", "energy","instrumentalness",
"acousticness","key","speechiness","duration_ms"]
X_train = train[features]
y_train = train["target"]
X_test = test[features]
y_test = test["target"]
dt = c.fit(X_train, y_train)
```

Now we created our decision tree, we need to visualize it and we need to find how accurate is it. Used graphviz for visualisation. (Bigger image: https://imgur.com/a/8eioP8K)



Let's test our Decision tree

```
y_pred = c.predict(X_test)
score = accuracy_score(y_test,y_pred)*100
print("Accuracy using decision Tree", round(score,1),"% ")
```

Accuracy using decision Tree 69.0 %

We have got almost 70 % success rate!

2) Random Forest

Not let's try more than one Decision Trees.

We need to import it from sklearn.

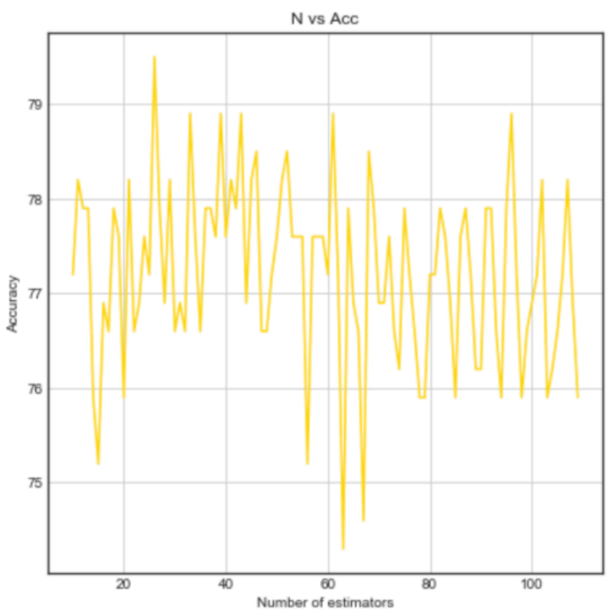
```
from sklearn.ensemble import RandomForestClassifier.
```

And let's bound number of estimators and compare it with each other.

```
score_pairs={}
for n in range(10,110,1):
    clf = RandomForestClassifier(n_estimators = 100)
    rfc = clf.fit(X_train, y_train)
    forest_y_pred = clf.predict(X_test)
    score = accuracy_score(y_test, forest_y_pred) * 100
    rounded_score = round(score, 1)
    score_pairs[n]=rounded_score
```

Now we tested the number of estimators between 10-100.

Let's check the accuracy of all of them.



As you can see global maxima is between 25-30 with max % 80 success rate.

Conclusion

As you can see from previous part, we created a decision tree than used random forest to create more than one.
We realized that using a random forest with 80 estimators creates best decision tree with avg %80 success rate comparing with %69.