Statistical Machine Learning Mini Project

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this project, we attempted to find best model for classification of the 'song_to_classify'. Within this context, we tried different algorithms such as 'Logistic Regresssion', 'KNN', 'Random Forest', 'Bagging Classifer' and different data processing like feature selection, feature scaling, dimension reduction and normalization. Among these, 'Bagging classifier' with feature selection and scaling provided best model for us with the accuracy of 0.81.
Number of group members: 3

1 Introduction

We are interested in making a model that can predict songs which are to the taste of a user using the 'Training Data Set' of the user. Such prediction models are used by all big productions and media services providers like Spotify, Netflix and even Facebook. Training Data set is the data of all the songs listened to, by the user and are liked or disliked depending on different features of the songs. Different Machine learning algorithms are applied to train the model and these algorithms use different features like song duration, loudness, liveness, speechess, time signature, instrumentalness and some other features to make a reliable predictive model. To get the best model, we tried using Logistic Regression, Random Forest, Boosting and KNN methods which are discussed below in detail.

2 Models

2.1 Logistic Regression

Simple classifier: We tried it by using logistic regression as it is the simplest statistical method without any tunning of the data or the method and we got all "Like" as output And it gave an accuracy of 0.62.

Then we tried to study the dataset to figure out which features have much effect on the classes value 0,1 . Feature selection: Three filter feature selection methods were used to evaluate the best fit for the model with selected features. Because we have categorical features and one of the features has negative values that's why we can't choose CHi2 method also because our data is Quantitative that's why we can calculate the value between each feature and the target using the f_classif method F_classif: this method commute the ANOVA value between features and target Mutual_info: this method can measure the dependency between features and output the information about features and the target values And With the help of selectkbest we could find the highest scoring features.

Selected features with highest score = ["acousticness", "danceability","energy", "liveness","loudness", "speechiness", "tempo", "Valence"]

Features scaling: In our model we considered features scaling. as it increases the weights of the important features. We choose the standardscaler method. It transforms the data distribution to have mean=0 and standard deviation =1

$$y = \frac{1}{1+e^{-(w_o + w_1 x)}}$$

Figure 1: This figure indicates the Logistic regression math equation.

Logistic regression is used for classification problems as an alternative for the linear regression method. The method practices sigmoid function on linear regression to calculate the output. Output is to be restricted to [0,1] range in our case it's like/dislike which is estimated probability. The mathematical form for the model is

There are many types of logistic regression and the one we use was the binary logistic regression which has only 2 possible outputs/classification categories. We used sklearn.linear model for applying logistic regression and adjusted the class weight to 'balanced' which uses "y" to adjust the value of the weights. Then we use fit() method to train the model with our training data by giving the model "the n samples and the n classes(0/1)." Our model is based on regularized logistic regression with 2 classes only; as a result we chose the 'liblinear' library for the solver [5]. In addition, we raised the iterations to 1000. We tried most of the methods and the linear regression gave us the least accurate results among other methods with 0.68.

2.2 K-Nearest Neighbors (KNN)

The k-nearest neighbors (KNN) postulates that similar things should be near to each other (close proximity) and predict the outcome based on distance and k-value. It classifies the object by means of the majority voting of its neighbors [1]. The significant part of this method is the selection of the most suitable k for the problem as well as the increasing accuracy, and preventing it from overfitting. For this reason, for the application of the KNN, a series steps were driven such that feature selection, feature scaling, normalization, dimension reduction (in this case PCA) and their combinations were utilized. Firstly, the dataset was divided into training and test subset as 80 and 20 percent respectively. In order to increase the accuracy, feature selection was done in a way that as shown in the Figure 2, the top 9 effective features (from 'speechiness' to 'tempo') were selected. In order to improve classification, as stated in [4], feature scaling-a method for the normalization of features, can be additionally utilized for this purpose. Data standardization and subsequent data normalization were operated and then data was scaled into the range in between 0 and 1 [2]. In this case, the test accuracy was obtained as 0.84 when K = 8. After that, K-Fold cross validation was utilized for evaluation of the model in terms of whether there is an overfitting issue or not. In K-fold cross validation, data is divided into "k different subgroups." K-1 subsets are used to train the model and the last one is, however, for test. The obtained average error value indicates the validity of the model [6]. As shown in the below Figure 2, the suitable k of the preprocessed data (the dataset after feature selection, standardization and normalization) is at k = 25 with the accuracy of 0.808. This model then was used for estimation of the test data set and it is resulting in 0.775 accuracy.

2.3 Random Forest

Another method we explored is Random Forest Classifier, also known as Bootstrap Aggregation. How it worked? In simple words it worked on combining multiple models to produce the result. This technique is called Ensembles Technique. Random Forest use multiple decision trees to get the output. From our training data, multiple decision trees were created randomly. As this method works on randomness it reduces the variance and produces more accurate model as compared to some other methods. Each decision tree then produces an output and by taking the mean of the output of each decision tree, we get the final output of the Random Forest. This process is called

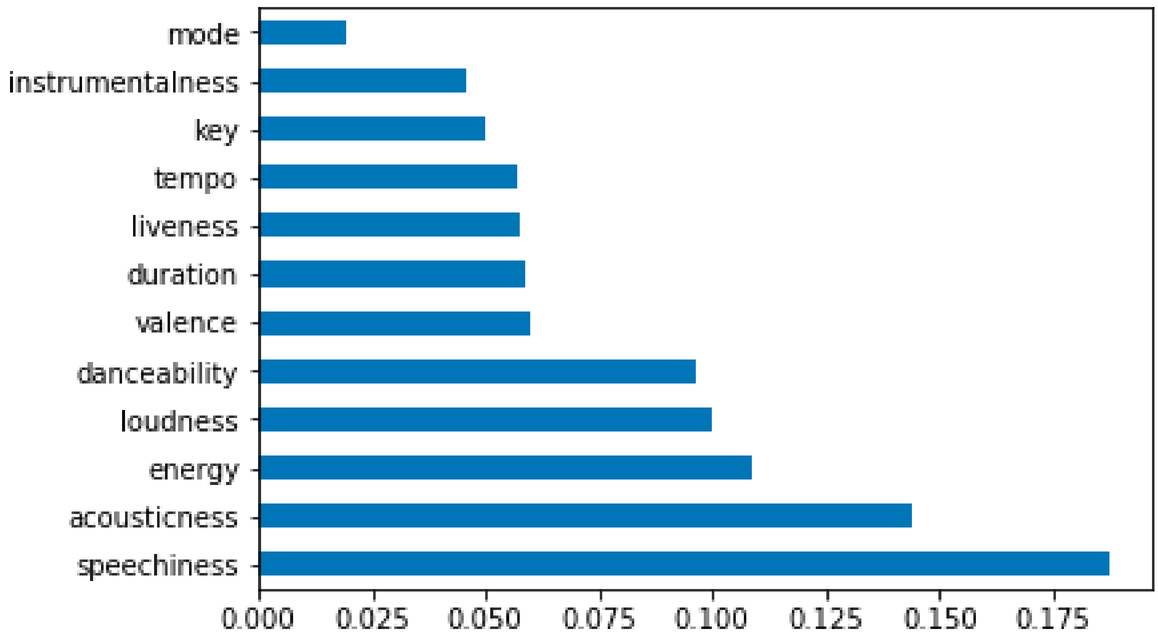


Figure 2: This figure indicates the effect of the attributes on the outcome.

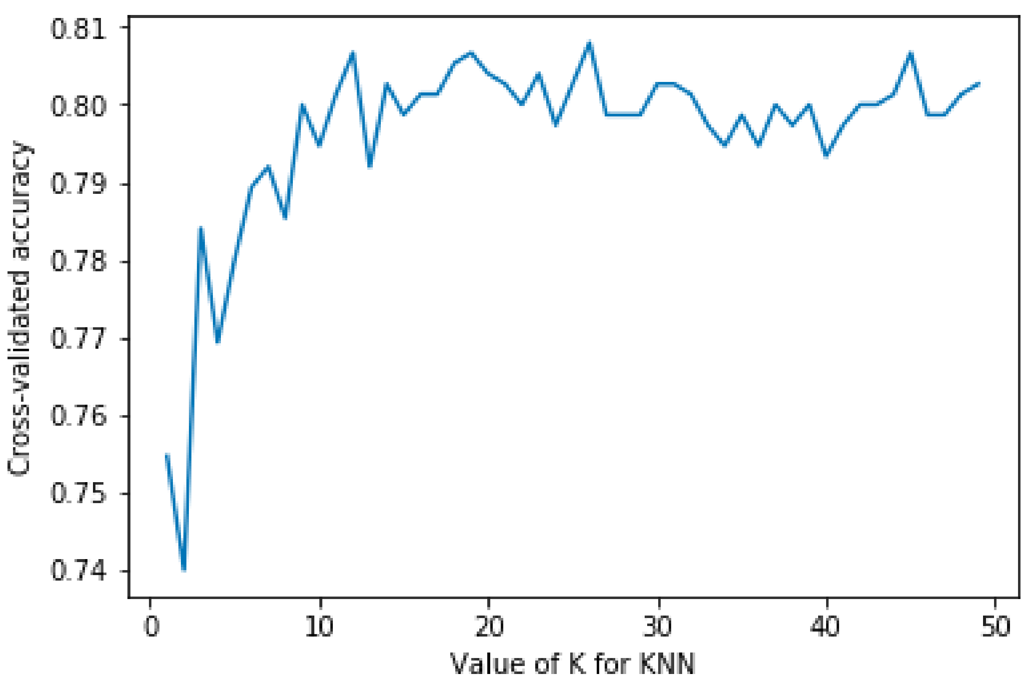


Figure 3: This figure indicates the cross-validated accuracy with respect to k for KNN.

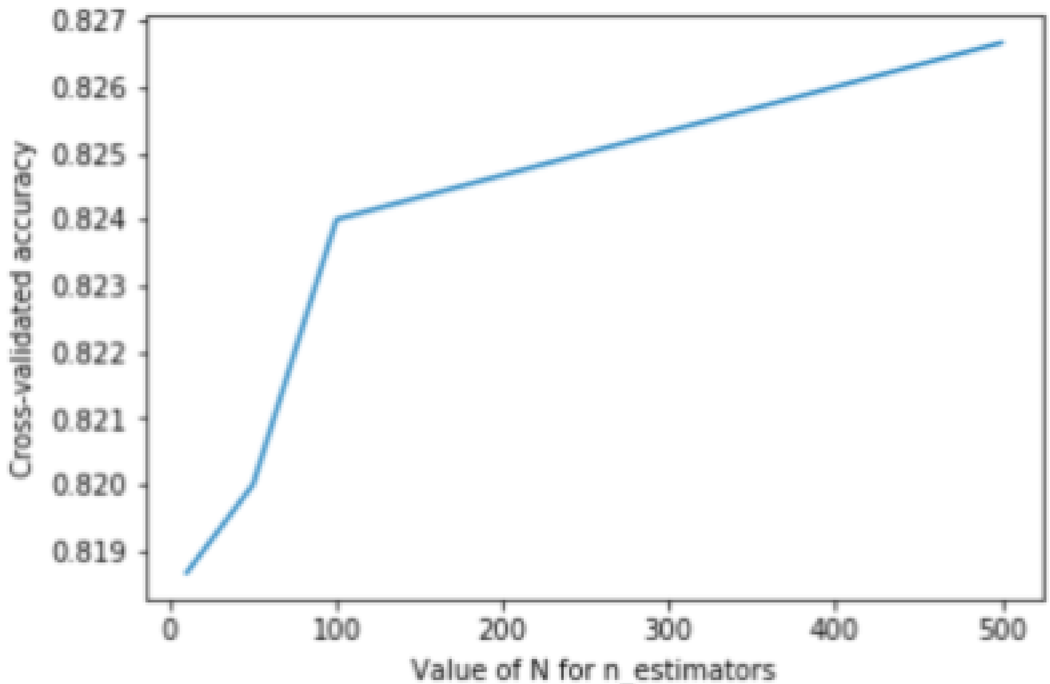


Figure 4: This figure indicates the best value of the n_estimators.

aggregation. Please note that here we are using Random Forest for classification. We can also control the number of decision trees by passing the argument "n estimators" to the algorithm used. In our case we created 100 decisions trees; since it was giving us the most accurate model for the training set data as compared to 99 or 101 decisions trees. We use SelectKBest method to choose all those features which contribute more to our model using Random Forest. This function uses two important arguments: chi2 and K. K is the number columns and chi2 (chi-squared) which uses some statistics and then informs us about the variables which is essential to the model. The higher statistical value means better variable for model. Using this method, we get the highest accuracy of 0.832 accuracy with feature section and without features selection, it reduces the accuracy to 0.82. The visual representation of the first tree has been shared in appendix for reference.

2.4 Bagging Classifier

The method creates bags of data (random subsets of the training data) and uses the bags with different data sets and train different models. We then use the same inputs for all models and get the mean output. The method gave the highest accuracy score by choosing the DecisionTreeClassifier() as base estimator, by trying different number on estimators [figure 4] and comparing the accuracy using cross validation in each value we get the best value by setting n_estimators = 500 and random_state = 8.

3 Production and Result

We used the bagging classifier method with the selected features, the bagging method outperformed the random forest method by 0.01.as it reduces variance, in other words it limits the data overfitting which gives accurate results. Also, It considers all the features in each node of the tree. The bagging method with selected features got 0.81 accuracy on leader board

	Random Forest	Bagging	KNN	Logistic Regression	Simple Classifier
With Features Selection	0.832	0.814666667	0.808	0.602666667	0
Without Features Selection (Simple Classifier)	0.82	0.8	0.638	0.68	0.62
leaderboard	0.785	0.81	0.775	0	0

Figure 5: This figure indicates the comparison between all classifiers accuracy.

4 Reflection Task

Machine Learning has an impact on the people, in both legal and ethical aspects, when it is utilized for making decisions about fields like insurance. Machine learning engineers must have legal responsibilities and be accountable for their products. Firstly, the product should provide two vital thin- which are fairness and explain-ability, in order to satisfy the trustworthiness. However, this trustworthiness can be damaged by bias, because bias means prejudice due to personal innate attributes such as age, gender, race. Explainable system is one that reasonably satisfy its user. For example, it should be able to explain the decision of why some people are not selected for credit or loan. In this case, for this reason, the dataset should be purified or minimized from bias in order to give better result. To do this, the machine learning engineer should explain the situation to the customers and educate them as well. The idea behind this is that minimization bias could be done by many methods, but the most effective is getting feedback from the customers. Besides, this case also rubs shoulders with tort law because this law compensates any victims for loses and provides product liability. In this case, the company might not make an insurance policy to someone who deserves (true negative). For this reason, engineers should provide bias information to the customers [3]. Also, there is also IEEE standards for the AI in bias case. It is stated that bias risk should be minimized or eliminated. The way of the elimination or minimization of the risk of the bias is made the customer be aware of bias and get continuously feedback from them to train model to get better results.

Sometimes we don't need to inform our customers about every small details or problems. Firstly, we know models are working on the given data. So, it is possible that the model did not work well on the future unpredicted data. This can be kept as a secret from customers as we don't know about their future data and the model that machine learning engineer design worked fine on the provided data and showed no biases. Considering GDPR rules and the company policies, we may not inform the customers to avoid violations of the secret terms of the company. Secondly, as we know no model is perfect, but some are useful. There will always be some small technical problems which may cause biases, and these are not the issues we faced while testing but the ones we know may occur in the future. ML algorithms work differently on different data. One algorithm might work fine on the given data set, but the other might not. As the complexity of the data increases, the model might show some biases.

5 Conclusion

In this project we looked at the data set of the songs liked or disliked by Andreas Lindholm. We used different methods to make a model which can predict his future preferences of the songs. First, we tired Logistic Regression and trained our model, but the accuracy of the model was not good enough. Then we used KNN method to train the model using same data-the accuracy improved from 0.602 to 0.808 but it was not what we expected. We also used Random forest with selected features, and it gave us a more accurate model. This time the accuracy was 0.832 better than the previous ones. But the method which gives us the highest accuracy is Boosting. Please check the following table for values comparison between different models.

References

[1] NS Altman. The american statistician. *An introduction to kernel and nearest-neighbor nonparametric regression*, 46(3):175–185, 1992.

[2] Piotr Juszczak, D Tax, and Robert PW Duin. Feature scaling in support vector data description. In *Proc. asci*, pages 95–102. Citeseer, 2002.

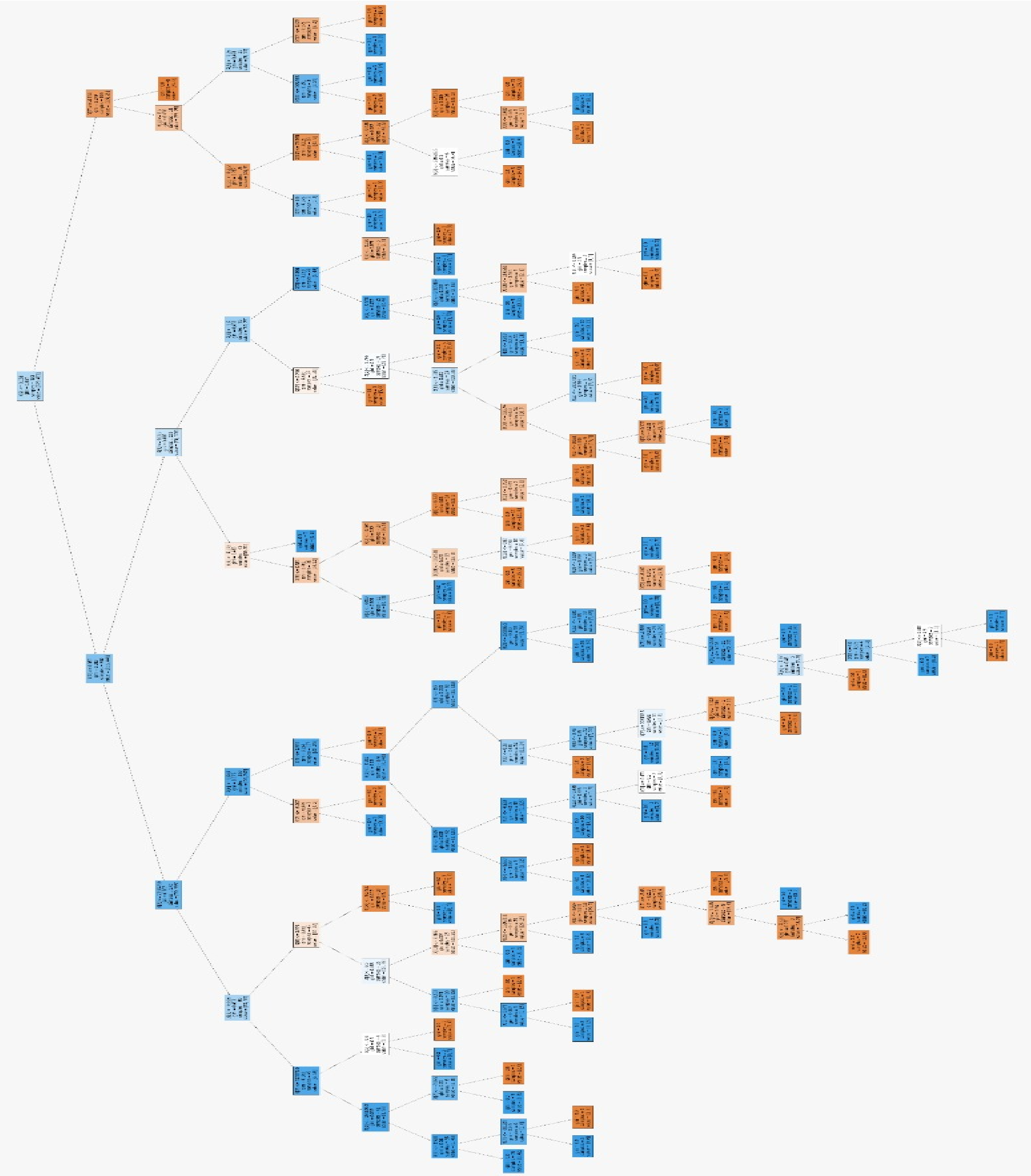
[3] Jon Kleinberg, Jens Ludwig, Sendhil Mullainathan, and Cass R Sunstein. Discrimination in the age of algorithms. *Journal of Legal Analysis*, 10, 2018.

[4] F Nigsch and A Bender. Bb van, j. tissen, e. nigsch and jb mitchell. *J. Chem. Inf. Model*, 46:2412–2422, 2006.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):111–133, 1974.

Appendix



In [202]:

```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.linear_model       import LogisticRegression
import sklearn.discriminant_analysis as skl_da
import sklearn.neighbors as skl_nb
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.model_selection    import train_test_split
from sklearn.preprocessing      import StandardScaler
from sklearn.metrics            import classification_report
from matplotlib                 import pyplot as plt
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.feature_selection import f_classif,mutual_info_classif, SelectKBest,f_regres
sion
from sklearn.metrics import log_loss
from sklearn import metrics
import sklearn.model_selection as skl_ms
import sklearn.preprocessing as skl_pre
from sklearn.decomposition import PCA
from sklearn.feature_selection import chi2
from sklearn.tree import export_graphviz
from sklearn import tree
from IPython.display import display, Image


np.random.seed(100)


#preparing data
music_data = pd.read_csv('training_data.csv')
music_test = pd.read_csv('songs_to_classify.csv')
```

In [203]:

```python
#Necessary Function Declarations
def prepare_file_for_train_test(file):
    outcome_column_name = "label";
    X = file.drop(columns =[outcome_column_name])
    y = file[outcome_column_name]
    return X,y


def cross_validate(X, y):
    k_range = range(1,50)
    k_scores = []
    for k in k_range:
        knn_model = skl_nb.KNeighborsClassifier(n_neighbors=k)
        scores = skl_ms.cross_val_score(knn_model, X, y, cv = cv, scoring='accuracy')
        k_scores.append(scores.mean())

    max, index = find_max_and_index(k_scores)
    return max, index;

def pca(X, dimension):
    pca = PCA(n_components=dimension)
    pca.fit(X)
    return pca.transform(X)

def find_max_and_index(arr):
    max = np.amax(arr)
    index = np.where(arr == np.amax(arr))

    return max, index
```

```python
def trim_white_space(arr):
    st=""
    for index in range(len(arr)):
        st+=str(arr[index])

    return st;

      152
def scale_data(X):
    ss = StandardScaler();
    df = pd.DataFrame(X)
    ss.fit(df)
    return ss.transform(df)
```

In [204]:

```python
features = [
    "acousticness",
    "danceability",
    "duration",
    "instrumentalness",
    "energy",
    "key",
    "liveness",
    "loudness",
    "mode",
    "speechiness",
    "tempo",
    "time_signature",
    "valence" ]

m_data = music_data[features].corr()
plt.subplots(figsize=(12,9))
sns.heatmap(m_data,cmap="RdYlGn", annot=True,vmax=0.9, square=True)
```

Out[204]:

```
<matplotlib.axes._subplots.AxesSubplot object at 0x7fc642130ed0>
```

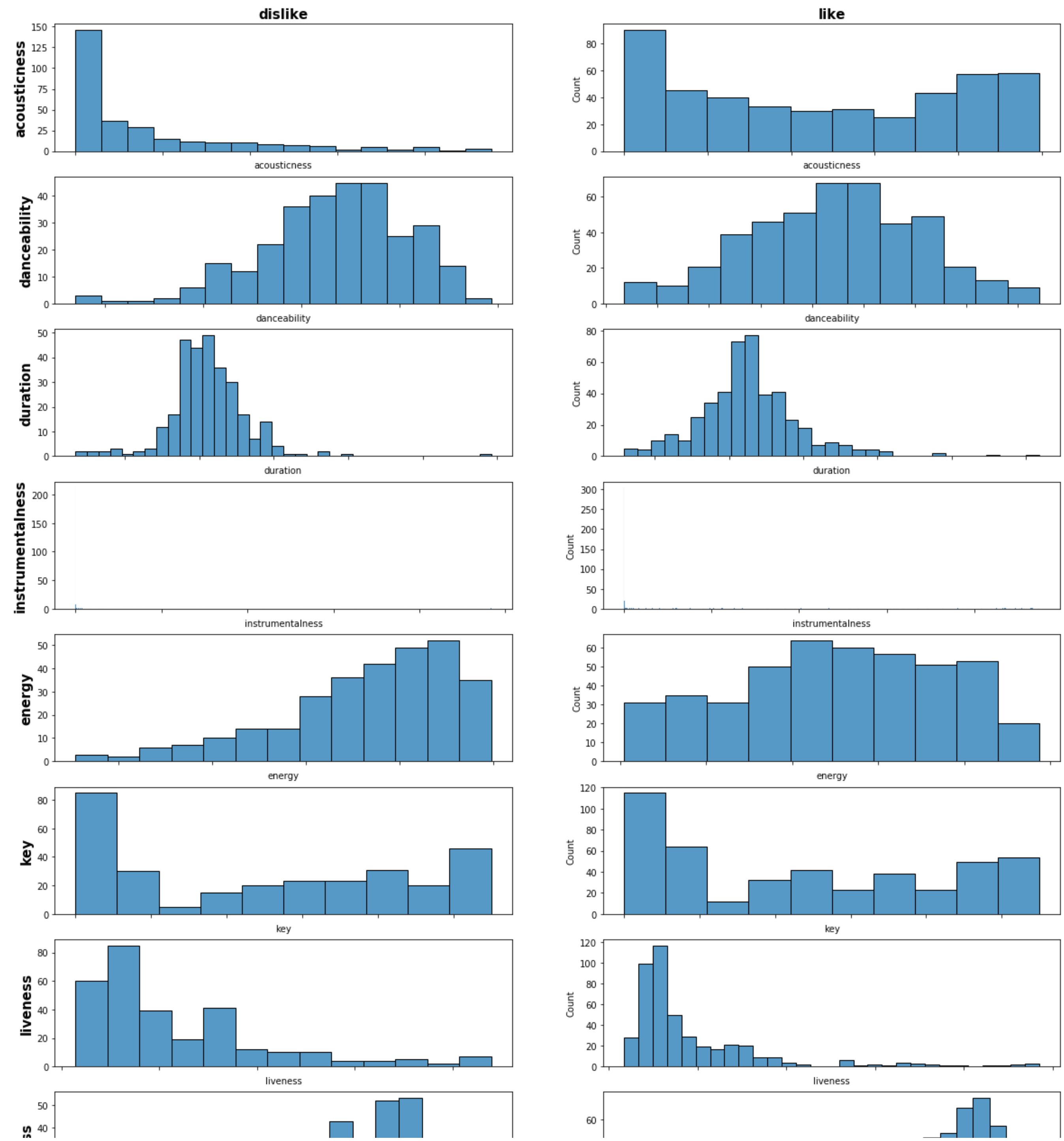
In [205]:

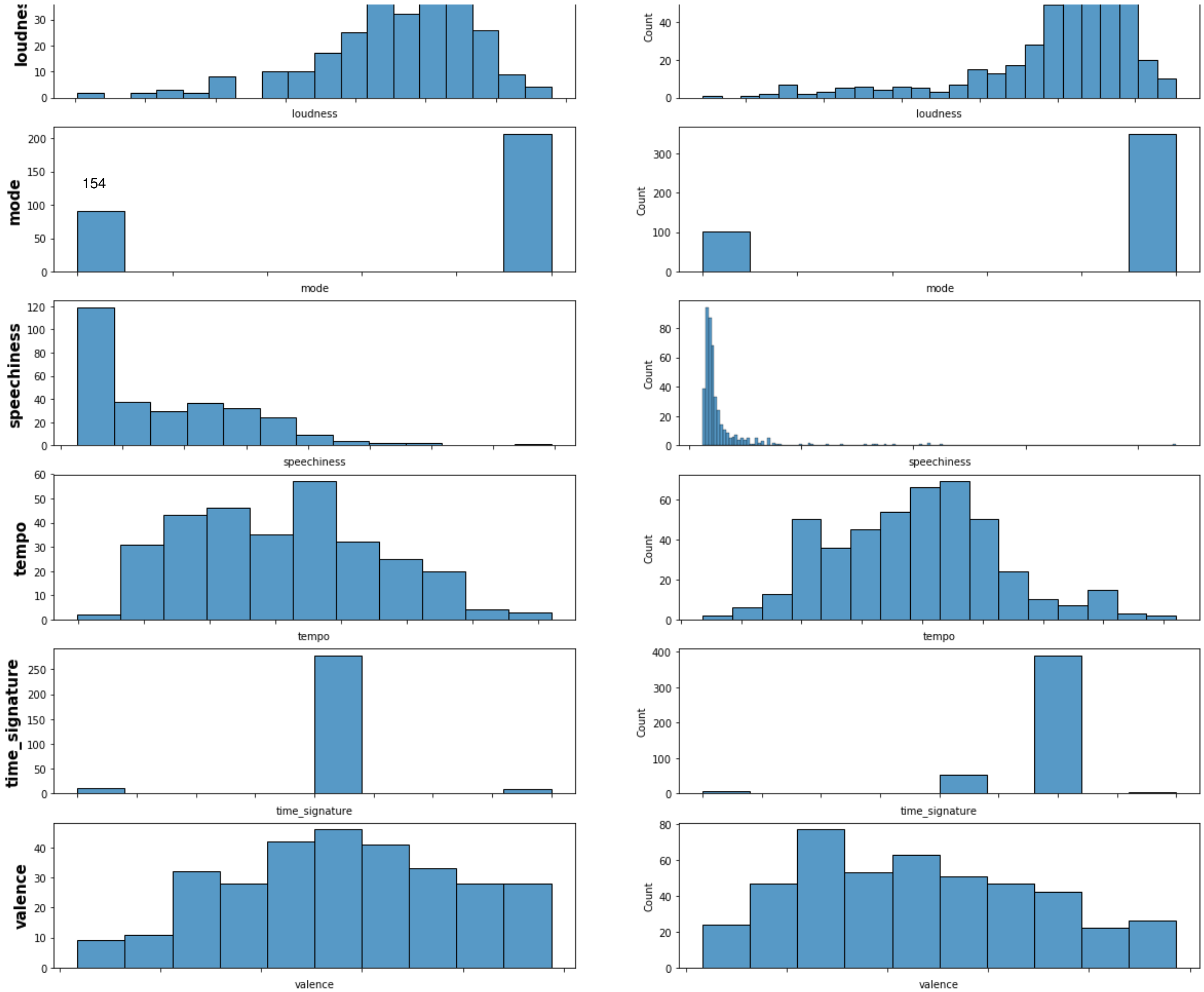
```
#separate like and dislike
dislike = music_data.loc[music_data['label'] == 0].copy()
like = music_data.loc[music_data['label'] == 1].copy()

fig, axs = plt.subplots(len(features), 2, figsize=(20, 40))

# Plotting histogram for eavh feature
axs[0,0].set_title('dislike', fontweight="bold", size=15)
axs[0,1].set_title('like', fontweight="bold", size=15)
for index, col in enumerate(features):
    axs[index,0].set_ylabel(col, fontweight="bold", fontsize=15)
    sns.histplot(dislike[col], ax=axs[index,0])
    sns.histplot(like[col], ax=axs[index,1])
    axs[index,0].set_xticklabels([])
    axs[index,1].set_xticklabels([])

plt.show()
```





We noticed that some features has changes in the data distribution in case of like and dislike and some dosn't have noticable change example of features that changed:

1. **acousticness**
2. **danceability**
3. **energy**
4. **liveness**
5. **oudness**
6. **speechiness**
7. **tempo**
8. **valence**
9. **duration**

WE will try to find a method for selecting the most important features for training the model

In [206]:

```
X, y = prepare_file_for_train_test(music_data)
```

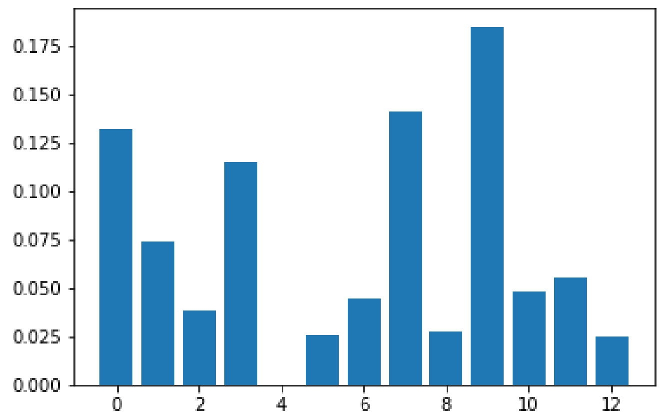
In [207]:

```
sel_f = SelectKBest(mutual_info_classif, k=9)
X_train_f = sel_f.fit(X, y)
print(sel_f.get_support())

for i in range(len(sel_f.scores_)):
 print('%s: %f' % (features[i], sel_f.scores_[i]))
# plot the scores
plt.bar([i for i in range(len(sel_f.scores_))], sel_f.scores_)
plt.show()
```

```
np.shape(X_train_f)
```

```
[ True  True  True  True False False  True  True False  True  True  True
 False]
acousticness: 0.132118
danceability: 0.073618
duration: 0.038445
instrumentalness: 0.114977
energy: 0.000000
key: 0.025767
liveness: 0.044364
loudness: 0.140876
mode: 0.027320
speechiness: 0.185026
tempo: 0.048318
time_signature: 0.054901
valence: 0.024419
```



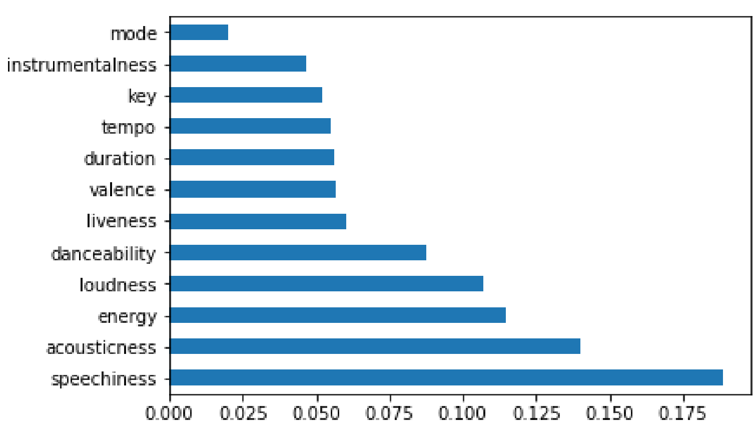
Out[207]:

```
()
```

In [208]:

```
from sklearn.ensemble import ExtraTreesClassifier
extr = ExtraTreesClassifier()
extr.fit(X, y)
print(extr.feature_importances_)
feat_importances = pd.Series(extr.feature_importances_, index=X.columns)
feat_importances.nlargest(12).plot(kind='barh')
plt.show()
```

```
[0.14028947 0.08747589 0.05621527 0.11488037 0.04651137 0.05191509
 0.06020034 0.10721684 0.02019557 0.18855259 0.05481158 0.01472643
 0.05700919]
```



In [209]:

```
selected_features = ["acousticness",
    "danceability",
```

```
    "energy",
    "liveness",
    "loudness",
    "speechiness",
    "tempo",
    "valence"]
```

156

In [210]:

```
w = X[selected_features]
all_features = music_data[features]
m_labels = music_data['label'].copy()
ss = StandardScaler();
scal=pd.DataFrame(ss.fit_transform(w), columns=[selected_features])
```

In [211]:

```
m_features_train, m_features_test, m_labels_train, m_labels_test = train_test_split(w, m
_labels, test_size=0.2)
baggingModel = BaggingClassifier()
baggingModel.fit(w, m_labels)
prediction = baggingModel.predict(music_test[selected_features])

print(skl_ms.cross_val_score(baggingModel, X , y , cv=10, scoring='accuracy').mean())

print(trim_white_space(prediction))
```

```
0.828
1110010101111001100010111111100111001111010110000000110100010001100001110010000101111110
0111100011110100010101100110110111000010110111111011111101010000000001001010000101111111110
11101010101001111111000
```

In [212]:

```
selected_features_new = ["acousticness",
    "danceability",
    "energy",
    "liveness",
    "loudness",
    "speechiness",
    "tempo",
    "valence",
    "liveness"]
```

In [213]:

```
#KNN Trials
Xnew = music_test
cv = skl_ms.KFold(n_splits=10, random_state=2, shuffle=True)

# Raw
print("RAW")
X_train, X_test, y_train, y_test = skl_ms.train_test_split(X, y, random_state=2)
test(X_train, y_train, X_test, y_test)
print(cross_validate(X,y))
print("\n\n")

# selected
print("Selected")
X_selected = X[selected_features_new]
X_selected_train, X_selected_test, y_selected_train, y_selected_test = skl_ms.train_test
_split(X_selected, y, random_state=2)
test(X_selected_train, y_selected_train, X_selected_test, y_selected_test)
print(cross_validate(X_selected,y))

print("\n\n")

# normalized
print("Normalized")
X_normalized = skl_pre.normalize(X)
X_normalized_train, X_normalized_test, y_normalized_train, y_normalized_test = skl_ms.tra
```

```
in_test_split(X_normalized, y, random_state=2)
test(X_normalized_train, y_normalized_train, X_normalized_test, y_normalized_test)
print(cross_validate(X_normalized,y))

print("\n\n")

# selected and normalized
print("selected and normalized")
X_selected_normalized = skl_pre.normalize(X_selected)
X_selected_normalized_train, X_selected_normalized_test, y_selected_normalized_train, y_s
elected_normalized_test = skl_ms.train_test_split(X_selected_normalized, y, random_state
=2)
test(X_selected_normalized_train, y_selected_normalized_train, X_selected_normalized_test
, y_selected_normalized_test)
print(cross_validate(X_selected_normalized,y))
print("\n\n")


# scaled
print("scaled")
X_original_scaled = scale_data(X)
X_original_scaled_train, X_original_scaled_test, y_original_scaled_train, y_original_scal
ed_test = skl_ms.train_test_split(X_original_scaled, y, random_state=2)
test(X_original_scaled_train,y_original_scaled_train, X_original_scaled_test, y_original_
scaled_test)
print(cross_validate(X_original_scaled,y))
print("\n\n")


# selected and scaled
print("selected and scaled")
X_selected_scaled = scale_data(X_selected)
X_selected_scaled_train, X_selected_scaled_test, y_selected_scaled_train, y_scaled_scaled
_test = skl_ms.train_test_split(X_selected_scaled, y, random_state=2)
test(X_selected_scaled_train, y_selected_scaled_train, X_selected_scaled_test, y_scaled_s
caled_test)
print(cross_validate(X_selected_scaled,y))
print("\n\n")


# selected and scaled and normalized
print("selected and scaled and normalized")
X_selected_scaled_normalized = skl_pre.normalize(X_selected_scaled)
X_selected_scaled_normalized_train, X_selected_scaled_normalized_test, y_selected_scaled_
normalized_train, y_selected_scaled_normalized_test = skl_ms.train_test_split(X_selected_
scaled_normalized, y, random_state=2)
test(X_selected_scaled_normalized_train, y_selected_scaled_normalized_train, X_selected_s
caled_normalized_test, y_selected_scaled_normalized_test)
print(cross_validate(X_selected_scaled_normalized,y))
print("\n\n")


#PCA
# original
print("pca - orginal")
X_pca = pca(X,6)
X_pca_train, X_pca_test, y_pca_train, y_pca_test = skl_ms.train_test_split(X_pca, y, ran
dom_state=2)
test(X_pca_train, y_pca_train, X_pca_test, y_pca_test)
print(cross_validate(X_pca,y))
print("\n\n")

# scaled
print("pca - scaled")
X_scaled_pca = pca(X_original_scaled, 6)
X_pca_scaled_train, X_pca_scaled_test, y_pca_scaled_train, y_pca_scaled_test = skl_ms.tra
in_test_split(X_scaled_pca, y, random_state=2)
test(X_pca_scaled_train, y_pca_scaled_train, X_pca_scaled_test, y_pca_scaled_test)
print(cross_validate(X_scaled_pca,y))
print("\n\n")

# selected
```

```python
print("pca - scaled")
X_selected_pca = pca(X_selected, 6)
X_pca_selected_train, X_pca_selected_test, y_pca_selected_train, y_pca_selected_test = skl_ms.train_test_split(X_selected_pca, y, random_state=2)
test(X_pca_selected_train, y_pca_selected_train, X_pca_selected_test, y_pca_selected_test)
print(cross_validate(X_selected_pca,y))
print("\n\n")


# selected and normalized
print("pca - selected - normalized")
X_selected_normalized_pca = pca(X_selected_normalized, 6)
X_pca_selected_norm_train, X_pca_selected_norm_test, y_pca_selected_norm_train, y_pca_selected_norm_test = skl_ms.train_test_split(X_selected_normalized_pca, y, random_state=2)
test(X_pca_selected_norm_train, y_pca_selected_norm_train, X_pca_selected_norm_test, y_pca_selected_norm_test)
print(cross_validate(X_selected_normalized_pca,y))
print("\n\n")

# selected and scaled
print("pca - selected - scaled")
X_selected_scaled_pca = pca(X_selected_scaled, 6)
X_pca_selected_scaled_train, X_pca_selected_scaled_test, y_pca_selected_scaled_train, y_pca_selected_scaled_test = skl_ms.train_test_split(X_selected_scaled_pca, y, random_state=2)
test(X_pca_selected_scaled_train, y_pca_selected_scaled_train, X_pca_selected_scaled_test, y_pca_selected_scaled_test)
print(cross_validate(X_selected_scaled_pca,y))
print("\n\n")
```

```
RAW
Greatest accuracy =  0.6223404255319149 when K= 34
(0.6386666666666667, (array([16]),))



Selected
Greatest accuracy =  0.75 when K= 18
(0.7120000000000001, (array([25]),))



Normalized
Greatest accuracy =  0.6968085106382979 when K= 19
(0.6533333333333334, (array([10]),))



selected and normalized
Greatest accuracy =  0.7712765957446809 when K= 22
(0.7733333333333334, (array([8]),))



scaled
Greatest accuracy =  0.824468085106383 when K= 4
(0.8133333333333332, (array([21]),))



selected and scaled
Greatest accuracy =  0.8297872340425532 when K= 9
(0.8066666666666669, (array([7]),))



selected and scaled and normalized
Greatest accuracy =  0.8404255319148937 when K= 8
(0.808, (array([25]),))
```

```
pca - orginal
Greatest accuracy =  0.6223404255319149 when K= 34
(0.6386666666666667, (array([16]),))



pca - scaled
Greatest accuracy =  0.8404255319148937 when K= 35
(0.796, (array([19]),))



pca - scaled
Greatest accuracy =  0.75 when K= 18
(0.7120000000000001, (array([25]),))



pca - selected - normalized
Greatest accuracy =  0.8031914893617021 when K= 10
(0.7493333333333333, (array([10]),))



pca - selected - scaled
Greatest accuracy =  0.8085106382978723 when K= 16
(0.8039999999999999, (array([ 2, 39, 41]),))
```

In [214]:

```
X_new = music_test
X_new_selected = X_new[selected_features_new]
X_new_selected_scaled = scale_data(X_new_selected)
X_new_selected_scaled_normalized = skl_pre.normalize(X_new_selected_scaled)

knnModel = skl_nb.KNeighborsClassifier(n_neighbors=25).fit(X_selected_scaled_normalized,
y)
predict = knnModel.predict(X_new_selected_scaled_normalized)
trim_white_space(predict)
```

Out[214]:

```
'1110000101111001110010111111101111011110110011000001011111011000110110001111000010111111
0011110001111010011110110011001111110011011011011111111111101111001101111010011101111101
011101010101011111000'
```

In [215]:

```
X_new_scaled = scale_data(X_new)
knnModel = skl_nb.KNeighborsClassifier(n_neighbors=21).fit(X_original_scaled, y)
predict = knnModel.predict(X_new_scaled)
trim_white_space(predict)
```

Out[215]:

```
'1110111101111101100110111111101111011111111011101001111111011000011110101101001011111111
1111100101111010001111110111101111111011011011011111111110101111001101111010010101111101
011101010101111111111100'
```

In [216]:

```
#ALL LIKE
df_features_train, df_features_test, df_labels_train, df_labels_test = train_test_split(
X, y, test_size=0.2, random_state=2)
lg = LogisticRegression()
lg.fit(df_features_train, df_labels_train)

prediction = lg.predict(df_features_test)
```

```
print(metrics.accuracy_score(df_labels_test, prediction))
loss = log_loss(df_labels_test, prediction, eps=1e-15, normalize=True, sample_weight=None
, labels=None)
prediction
```

```
0.62
```
160

Out[216]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [217]:

```
Xtrain = music_data.copy().drop(columns = ['label'])
Ytrain = music_data['label']
Xtrain1 = Xtrain.copy().drop(columns=['loudness'])
Xtrain1
```

Out[217]:

	acousticness	danceability	duration	energy	instrumentalness	key	liveness	mode	speechiness	tempo	time_signatur
0	0.88500	0.366	352000	0.1390	0.913000	7	0.0725	1	0.0390	139.478	
1	0.12400	0.863	236293	0.5760	0.000000	5	0.1430	0	0.2390	132.054	
2	0.18400	0.631	219160	0.6990	0.000000	9	0.1080	0	0.0284	128.433	
3	0.01080	0.800	201840	0.8940	0.437000	6	0.0285	0	0.0400	138.480	
4	0.00440	0.788	228000	0.6730	0.000005	9	0.0755	1	0.1990	99.979	
...	...	...	...	...	...	...	...	...	...	...	.
745	0.82300	0.635	89067	0.3380	0.000434	9	0.2210	1	0.5120	168.163	
746	0.03250	0.544	238493	0.5000	0.000004	11	0.1090	1	0.0260	93.621	
747	0.99200	0.525	226293	0.0633	0.905000	9	0.1050	1	0.0497	71.855	
748	0.54500	0.365	237267	0.5200	0.000000	9	0.1110	1	0.0331	106.152	
749	0.00513	0.834	312820	0.7300	0.000000	8	0.1240	1	0.2220	155.008	

750 rows × 12 columns

In [218]:

```
best_features = SelectKBest(score_func=chi2, k=9)
fitti = best_features.fit(Xtrain1, Ytrain)
D_scores = pd.DataFrame(fitti.scores_)
D_col = pd.DataFrame(Xtrain1.columns)
```

In [219]:

```
features_scores = pd.concat([D_col, D_scores], axis=1)
features_scores.columns = ['Specs', 'Score']
features_scores
```

Out[219]:

	Specs	Score
0	acousticness	49.695755
1	danceability	4.849639
2	duration	197661.171385

	Specs	Score
3	energy	15.169557
4	instrumentalness	9.315685
5	key	3.798891
6	liveness	2.217660
7	161 mode	1.536297
8	speechiness	20.656177
9	tempo	25.650998
10	time_signature	0.949704
11	valence	3.195746

In [220]:

```
print(features_scores.nlargest(10, 'Score'))
```

```
               Specs          Score
2           duration  197661.171385
0       acousticness      49.695755
9              tempo      25.650998
8        speechiness      20.656177
3             energy      15.169557
4   instrumentalness       9.315685
1       danceability       4.849639
5                key       3.798891
11           valence       3.195746
6           liveness       2.217660
```

In [221]:

```
randomModel = RandomForestClassifier()
#n_estimator is used to make number of trees .
randomModel.fit(Xtrain,Ytrain)

y_test = randomModel.predict(Xnew)
```

In [222]:

```
scores = skl_ms.cross_val_score(randomModel,Xtrain, Ytrain, cv=10, scoring='accuracy')
print(scores.mean())
```

```
0.8226666666666667
```

In [223]:

```
Xtrain1 = Xtrain.copy().drop(columns=['time_signature','liveness', 'mode'])
Xtrain1
```

Out[223]:

	acousticness	danceability	duration	energy	instrumentalness	key	loudness	speechiness	tempo	valence
0	0.88500	0.366	352000	0.1390	0.913000	7	-19.978	0.0390	139.478	0.310
1	0.12400	0.863	236293	0.5760	0.000000	5	-5.687	0.2390	132.054	0.832
2	0.18400	0.631	219160	0.6990	0.000000	9	-7.625	0.0284	128.433	0.707
3	0.01080	0.800	201840	0.8940	0.437000	6	-7.346	0.0400	138.480	0.967
4	0.00440	0.788	228000	0.6730	0.000005	9	-9.232	0.1990	99.979	0.478
...	...	...	...	...	...	...	...	...	...	...
745	0.82300	0.635	89067	0.3380	0.000434	9	-8.078	0.5120	168.163	0.736
746	0.03250	0.544	238493	0.5000	0.000004	11	-8.253	0.0260	93.621	0.177
747	0.99200	0.525	226293	0.0633	0.905000	9	-23.072	0.0497	71.855	0.297
748	0.54500	0.365	237267	0.5200	0.000000	9	-6.520	0.0331	106.152	0.400
749	0.00513	0.834	312820	0.7300	0.000000	8	-3.714	0.2220	155.008	0.446

	acousticness	danceability	duration	energy	instrumentalness	key	loudness	speechiness	tempo	valence
749	0.00513	0.834	312820	0.7300	0.000000	8	-3.714	0.2220	155.008	0.446

750 rows × 10 columns

In [224]:


```
randomModel = RandomForestClassifier(n_estimators=100)
randomModel.fit(Xtrain1, Ytrain)

y_test = randomModel.predict(Xnew.copy().drop(columns = ['time_signature', 'liveness','m
ode']))
scores = skl_ms.cross_val_score(randomModel,Xtrain1, Ytrain, cv=10, scoring='accuracy')
print(scores.mean())
```

```
0.8333333333333334
```

In [225]:

```
trim_white_space(y_test)
```

Out[225]:

```
'1110000101111001100010111111100111011110101011000000011010001110010010011101000010111111
0111110011111010011010111011011011110001011011011101011110101010101001111010010101111010101
01111101010101111111010'
```

In [226]:

```
x_grph = music_data.copy().drop(columns = ['label'])
y_grph = music_data['label']
```

In [227]:

```
model_G = RandomForestClassifier(n_estimators=100)
model_G = model_G.fit(x_grph,y_grph)
```

In [228]:

```
len(model_G.estimators_)
```

Out[228]:

```
100
```

In [ ]:

```
plt.figure(figsize=(400,300))
tree.plot_tree(model_G.estimators_[1], filled = True)
```

Out[ ]:

```
[Text(9876.6,15771.2,'X[9] <= 0.055\ngini = 0.473\nsamples = 501\nvalue = [288, 462]'),
 Text(2899.41,14683.5,'X[1] <= 0.523\ngini = 0.265\nsamples = 274\nvalue = [65, 349]'),
 Text(992.311,13595.8,'X[4] <= 0.935\ngini = 0.023\nsamples = 110\nvalue = [2, 168]'),
 Text(496.156,12508.2,'X[1] <= 0.493\ngini = 0.012\nsamples = 108\nvalue = [1, 167]'),
 Text(248.078,11420.5,'gini = 0.0\nsamples = 92\nvalue = [0, 144]'),
 Text(744.233,11420.5,'X[3] <= 0.446\ngini = 0.08\nsamples = 16\nvalue = [1, 23]'),
 Text(496.156,10332.8,'X[4] <= 0.033\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(248.078,9245.17,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(744.233,9245.17,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(992.311,10332.8,'gini = 0.0\nsamples = 14\nvalue = [0, 22]'),
 Text(1488.47,12508.2,'X[2] <= 214393.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(1240.39,11420.5,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(1736.54,11420.5,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(4806.51,13595.8,'X[8] <= 0.5\ngini = 0.383\nsamples = 164\nvalue = [63, 181]'),
 Text(2728.86,12508.2,'X[7] <= -3.625\ngini = 0.219\nsamples = 42\nvalue = [8, 56]'),
 Text(2232.7,11420.5,'X[6] <= 0.058\ngini = 0.126\nsamples = 38\nvalue = [4, 55]'),
 Text(1984.62,10332.8,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(2480.78,10332.8,'X[7] <= -5.207\ngini = 0.098\nsamples = 37\nvalue = [3, 55]'),
 Text(1984.62,9245.17,'X[1] <= 0.75\ngini = 0.039\nsamples = 32\nvalue = [1, 49]'),
 Text(1736.54,8157.5,'gini = 0.0\nsamples = 26\nvalue = [0, 43]'),
 Text(2232.7,8157.5,'X[12] <= 0.85\ngini = 0.245\nsamples = 6\nvalue = [1, 6]'),
 Text(1984.62,7069.83,'gini = 0.0\nsamples = 5\nvalue = [0, 6]'),
```

```
Text(2480.78,7069.83,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(2976.93,9245.17,'X[5] <= 8.5\ngini = 0.375\nsamples = 5\nvalue = [2, 6]'),
Text(2728.86,8157.5,'gini = 0.0\nsamples = 3\nvalue = [0, 6]'),
Text(3225.01,8157.5,'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(3225.01,11420.5,'X[0] <= 0.093\ngini = 0.32\nsamples = 4\nvalue = [4, 1]'),
Text(2976.93,10332.8,'gini = 0.0\nsamples = 3\nvalue = [4, 0]'),
Text(3473.09,10332.8,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(6884.16,12508.2,'X[7] <= -4.282\ngini = 0.424\nsamples = 122\nvalue = [55, 125]'),
Text(5829.83,11420.5,'X[6] <= 0.236\ngini = 0.35\nsamples = 107\nvalue = [35, 120]'),
Text(4465.4,10332.8,'X[6] <= 0.062\ngini = 0.259\nsamples = 83\nvalue = [18, 100]'),
Text(3969.24,9245.17,'X[2] <= 301654.5\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(3721.17,8157.5,'X[1] <= 0.737\ngini = 0.32\nsamples = 5\nvalue = [4, 1]'),
Text(3473.09,7069.83,'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
Text(3969.24,7069.83,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(4217.32,8157.5,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(4961.56,9245.17,'X[1] <= 0.524\ngini = 0.219\nsamples = 77\nvalue = [14, 98]'),
Text(4713.48,8157.5,'gini = 0.0\nsamples = 2\nvalue = [3, 0]'),
Text(5209.63,8157.5,'X[3] <= 0.945\ngini = 0.181\nsamples = 75\nvalue = [11, 98]'),
Text(4961.56,7069.83,'X[5] <= 10.5\ngini = 0.154\nsamples = 74\nvalue = [9, 98]'),
Text(4465.4,5982.17,'X[6] <= 0.101\ngini = 0.113\nsamples = 70\nvalue = [6, 94]'),
Text(4217.32,4894.5,'X[0] <= 0.275\ngini = 0.255\nsamples = 25\nvalue = [6, 34]'),
Text(3721.17,3806.83,'X[12] <= 0.343\ngini = 0.087\nsamples = 14\nvalue = [1, 21]'),
Text(3473.09,2719.17,'X[6] <= 0.079\ngini = 0.444\nsamples = 2\nvalue = [1, 2]'),
Text(3225.01,1631.5,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(3721.17,1631.5,'gini = 0.0\nsamples = 1\nvalue = [0, 2]'),
Text(3969.24,2719.17,'gini = 0.0\nsamples = 12\nvalue = [0, 19]'),
Text(4713.48,3806.83,'X[7] <= -7.715\ngini = 0.401\nsamples = 11\nvalue = [5, 13]'),
Text(4465.4,2719.17,'X[3] <= 0.311\ngini = 0.305\nsamples = 9\nvalue = [3, 13]'),
Text(4217.32,1631.5,'X[3] <= 0.266\ngini = 0.5\nsamples = 3\nvalue = [3, 3]'),
Text(3969.24,543.833,'gini = 0.0\nsamples = 2\nvalue = [0, 3]'),
Text(4465.4,543.833,'gini = 0.0\nsamples = 1\nvalue = [3, 0]'),
Text(4713.48,1631.5,'gini = 0.0\nsamples = 6\nvalue = [0, 10]'),
Text(4961.56,2719.17,'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(4713.48,4894.5,'gini = 0.0\nsamples = 45\nvalue = [0, 60]'),
Text(5457.71,5982.17,'X[7] <= -8.714\ngini = 0.49\nsamples = 4\nvalue = [3, 4]'),
Text(5209.63,4894.5,'gini = 0.0\nsamples = 2\nvalue = [3, 0]'),
Text(5705.79,4894.5,'gini = 0.0\nsamples = 2\nvalue = [0, 4]'),
Text(5457.71,7069.83,'gini = 0.0\nsamples = 1\nvalue = [2, 0]'),
Text(7194.26,10332.8,'X[7] <= -4.987\ngini = 0.497\nsamples = 24\nvalue = [17, 20]'),
Text(6946.18,9245.17,'X[4] <= 0.0\ngini = 0.495\nsamples = 21\nvalue = [17, 14]'),
Text(6450.02,8157.5,'X[0] <= 0.631\ngini = 0.332\nsamples = 14\nvalue = [15, 4]'),
Text(6201.94,7069.83,'X[9] <= 0.026\ngini = 0.208\nsamples = 12\nvalue = [15, 2]'),
Text(5953.87,5982.17,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(6450.02,5982.17,'X[3] <= 0.563\ngini = 0.117\nsamples = 11\nvalue = [15, 1]'),
Text(6201.94,4894.5,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(6698.1,4894.5,'gini = 0.0\nsamples = 10\nvalue = [15, 0]'),
Text(6698.1,7069.83,'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(7442.33,8157.5,'X[1] <= 0.773\ngini = 0.278\nsamples = 7\nvalue = [2, 10]'),
Text(7194.26,7069.83,'gini = 0.0\nsamples = 6\nvalue = [0, 10]'),
Text(7690.41,7069.83,'gini = 0.0\nsamples = 1\nvalue = [2, 0]'),
Text(7442.33,9245.17,'gini = 0.0\nsamples = 3\nvalue = [0, 6]'),
Text(7938.49,11420.5,'X[1] <= 0.565\ngini = 0.32\nsamples = 15\nvalue = [20, 5]'),
Text(7690.41,10332.8,'gini = 0.0\nsamples = 2\nvalue = [0, 4]'),
Text(8186.57,10332.8,'X[0] <= 0.011\ngini = 0.091\nsamples = 13\nvalue = [20, 1]'),
Text(7938.49,9245.17,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(8434.64,9245.17,'gini = 0.0\nsamples = 12\nvalue = [20, 0]'),
Text(16853.8,14683.5,'X[9] <= 0.152\ngini = 0.446\nsamples = 227\nvalue = [223, 113]'),
Text(14171.4,13595.8,'X[0] <= 0.453\ngini = 0.5\nsamples = 124\nvalue = [89, 93]'),
Text(12217.8,12508.2,'X[3] <= 0.815\ngini = 0.457\nsamples = 91\nvalue = [84, 46]'),
Text(10171.2,11420.5,'X[1] <= 0.536\ngini = 0.5\nsamples = 49\nvalue = [36, 37]'),
Text(9178.88,10332.8,'X[2] <= 154822.0\ngini = 0.291\nsamples = 12\nvalue = [3, 14]'),
Text(8930.8,9245.17,'gini = 0.0\nsamples = 1\nvalue = [2, 0]'),
Text(9426.96,9245.17,'X[7] <= -11.787\ngini = 0.124\nsamples = 11\nvalue = [1, 14]'),
Text(9178.88,8157.5,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(9675.03,8157.5,'gini = 0.0\nsamples = 10\nvalue = [0, 14]'),
Text(11163.5,10332.8,'X[2] <= 178280.0\ngini = 0.484\nsamples = 37\nvalue = [33, 23]'),
Text(10419.3,9245.17,'X[8] <= 0.5\ngini = 0.165\nsamples = 7\nvalue = [1, 10]'),
Text(10171.2,8157.5,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(10667.3,8157.5,'gini = 0.0\nsamples = 6\nvalue = [0, 10]'),
Text(11907.7,9245.17,'X[4] <= 0.0\ngini = 0.411\nsamples = 30\nvalue = [32, 13]'),
Text(11163.5,8157.5,'X[7] <= -6.31\ngini = 0.245\nsamples = 24\nvalue = [30, 5]'),
Text(10915.4,7069.83,'gini = 0.0\nsamples = 10\nvalue = [17, 0]'),
```

```
Text(11411.6,7069.83,'X[4] <= 0.0\ngini = 0.401\nsamples = 14\nvalue = [13, 5]'),
Text(11163.5,5982.17,'X[10] <= 158.618\ngini = 0.494\nsamples = 8\nvalue = [4, 5]'),
Text(10915.4,4894.5,'X[0] <= 0.006\ngini = 0.408\nsamples = 6\nvalue = [2, 5]'),
Text(10667.3,3806.83,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(11163.5,3806.83,'X[10] <= 138.916\ngini = 0.278\nsamples = 5\nvalue = [1, 5]'),
Text(10915.4,2719.17,'gini = 0.0\nsamples = 3\nvalue = [0, 4]'),
Text(11411.6,2719.17,'X[0] <= 0.257\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(11163.5,1631.5,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(11659.7,1631.5,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(11411.6,4894.5,'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(11659.7,5982.17,'gini = 0.0\nsamples = 6\nvalue = [9, 0]'),
Text(12652,8157.5,'X[2] <= 232046.5\ngini = 0.32\nsamples = 6\nvalue = [2, 8]'),
Text(12403.9,7069.83,'X[10] <= 115.504\ngini = 0.444\nsamples = 2\nvalue = [2, 1]'),
Text(12155.8,5982.17,'gini = 0.0\nsamples = 1\nvalue = [2, 0]'),
Text(12652,5982.17,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(12900,7069.83,'gini = 0.0\nsamples = 4\nvalue = [0, 7]'),
Text(14264.5,11420.5,'X[9] <= 0.064\ngini = 0.266\nsamples = 42\nvalue = [48, 9]'),
Text(14016.4,10332.8,'gini = 0.0\nsamples = 13\nvalue = [19, 0]'),
Text(14512.6,10332.8,'X[7] <= -5.043\ngini = 0.361\nsamples = 29\nvalue = [29, 9]'),
Text(13892.4,9245.17,'X[0] <= 0.099\ngini = 0.496\nsamples = 9\nvalue = [5, 6]'),
Text(13644.3,8157.5,'X[5] <= 2.5\ngini = 0.278\nsamples = 5\nvalue = [5, 1]'),
Text(13396.2,7069.83,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(13892.4,7069.83,'gini = 0.0\nsamples = 4\nvalue = [5, 0]'),
Text(14140.4,8157.5,'gini = 0.0\nsamples = 4\nvalue = [0, 5]'),
Text(15132.7,9245.17,'X[10] <= 97.489\ngini = 0.198\nsamples = 20\nvalue = [24, 3]'),
Text(14636.6,8157.5,'X[12] <= 0.627\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(14388.5,7069.83,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(14884.7,7069.83,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(15628.9,8157.5,'X[12] <= 0.468\ngini = 0.147\nsamples = 18\nvalue = [23, 2]'),
Text(15380.8,7069.83,'X[3] <= 0.896\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(15132.7,5982.17,'X[2] <= 277946.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(14884.7,4894.5,'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(15380.8,4894.5,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(15628.9,5982.17,'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(15877,7069.83,'gini = 0.0\nsamples = 12\nvalue = [19, 0]'),
Text(16125.1,12508.2,'X[6] <= 0.067\ngini = 0.174\nsamples = 33\nvalue = [5, 47]'),
Text(15877,11420.5,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(16373.1,11420.5,'X[5] <= 4.5\ngini = 0.145\nsamples = 32\nvalue = [4, 47]'),
Text(16125.1,10332.8,'gini = 0.0\nsamples = 21\nvalue = [0, 36]'),
Text(16621.2,10332.8,'X[3] <= 0.605\ngini = 0.391\nsamples = 11\nvalue = [4, 11]'),
Text(16373.1,9245.17,'X[2] <= 114213.5\ngini = 0.153\nsamples = 10\nvalue = [1, 11]'),
Text(16125.1,8157.5,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(16621.2,8157.5,'gini = 0.0\nsamples = 9\nvalue = [0, 11]'),
Text(16869.3,9245.17,'gini = 0.0\nsamples = 1\nvalue = [3, 0]'),
Text(19536.1,13595.8,'X[7] <= -8.105\ngini = 0.226\nsamples = 103\nvalue = [134, 20]'),
Text(18357.8,12508.2,'X[4] <= 0.001\ngini = 0.498\nsamples = 20\nvalue = [17, 15]'),
Text(18109.7,11420.5,'X[5] <= 7.5\ngini = 0.488\nsamples = 16\nvalue = [11, 15]'),
Text(17613.5,10332.8,'X[4] <= 0.0\ngini = 0.426\nsamples = 10\nvalue = [9, 4]'),
Text(17365.4,9245.17,'X[2] <= 311373.0\ngini = 0.298\nsamples = 8\nvalue = [9, 2]'),
Text(17117.4,8157.5,'gini = 0.0\nsamples = 6\nvalue = [9, 0]'),
Text(17613.5,8157.5,'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(17861.6,9245.17,'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(18605.8,10332.8,'X[1] <= 0.715\ngini = 0.26\nsamples = 6\nvalue = [2, 11]'),
Text(18357.8,9245.17,'gini = 0.0\nsamples = 3\nvalue = [0, 10]'),
Text(18853.9,9245.17,'X[7] <= -10.299\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(18605.8,8157.5,'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(19102,8157.5,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(18605.8,11420.5,'gini = 0.0\nsamples = 4\nvalue = [6, 0]'),
Text(20714.5,12508.2,'X[5] <= 9.5\ngini = 0.079\nsamples = 83\nvalue = [117, 5]'),
Text(20094.3,11420.5,'X[11] <= 3.5\ngini = 0.038\nsamples = 70\nvalue = [101, 2]'),
Text(19598.1,10332.8,'X[1] <= 0.667\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(19350.1,9245.17,'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(19846.2,9245.17,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(20590.5,10332.8,'X[10] <= 73.565\ngini = 0.02\nsamples = 68\nvalue = [100, 1]'),
Text(20342.4,9245.17,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(20838.5,9245.17,'gini = 0.0\nsamples = 67\nvalue = [100, 0]'),
Text(21334.7,11420.5,'X[2] <= 142650.0\ngini = 0.266\nsamples = 13\nvalue = [16, 3]'),
Text(21086.6,10332.8,'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(21582.8,10332.8,'X[10] <= 108.668\ngini = 0.198\nsamples = 12\nvalue = [16, 2]'),
Text(21334.7,9245.17,'gini = 0.0\nsamples = 6\nvalue = [9, 0]'),
Text(21830.8,9245.17,'X[0] <= 0.006\ngini = 0.346\nsamples = 6\nvalue = [7, 2]'),
Text(21582.8,8157.5,'gini = 0.0\nsamples = 1\nvalue = [0, 2]'),
```

```
Text(22078.9,8157.5,'gini = 0.0\nsamples = 5\nvalue = [7, 0]')]
```