Machine Learning End of Term Paper

Bora KANDEMİR Batuhan ÇEVİK

2024-05-29

Contents

1-) Problem, Features and Target

Analyzing and predicting the energy levels of songs based on the given data set. The dataset includes various features that provide information about songs.These features include danceability, valence, liveness etc.The target variable in this data set is whether the energy levels of the songs are greater than or less than 50.It is represented by the "energy_%" column, where 1 indicates the energy level of songs are equal or greater than 50 and 0 indicates its less than 50.

2-) The Dataset and Preparation for Analysis

The dataset includes 817 observations, 5 categorical variables and 19 numeric variables.

```
data <- read_csv("spotify-2023.csv")
glimpse(data)
```

```
## Rows: 953
## Columns: 24
## $ track_name           <chr> "Seven (feat. Latto) (Explicit Ver.)", "LALA", "v~
## $ `artist(s)_name`     <chr> "Latto, Jung Kook", "Myke Towers", "Olivia Rodrig~
## $ artist_count         <dbl> 2, 1, 1, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1~
## $ released_year        <dbl> 2023, 2023, 2023, 2019, 2023, 2023, 2023, 2023, 2~
## $ released_month       <dbl> 7, 3, 6, 8, 5, 6, 3, 7, 5, 3, 4, 7, 1, 4, 3, 12, ~
## $ released_day         <dbl> 14, 23, 30, 23, 18, 1, 16, 7, 15, 17, 17, 7, 12, ~
## $ in_spotify_playlists <dbl> 553, 1474, 1397, 7858, 3133, 2186, 3090, 714, 109~
## $ in_spotify_charts    <dbl> 147, 48, 113, 100, 50, 91, 50, 43, 83, 44, 40, 55~
## $ streams              <chr> "141381703", "133716286", "140003974", "800840817~
## $ in_apple_playlists   <dbl> 43, 48, 94, 116, 84, 67, 34, 25, 60, 49, 41, 37, ~
## $ in_apple_charts      <dbl> 263, 126, 207, 207, 133, 213, 222, 89, 210, 110, ~
## $ in_deezer_playlists  <dbl> 45, 58, 91, 125, 87, 88, 43, 30, 48, 66, 54, 21, ~
## $ in_deezer_charts     <dbl> 10, 14, 14, 12, 15, 17, 13, 13, 11, 13, 12, 5, 58~
## $ in_shazam_charts     <dbl> 826, 382, 949, 548, 425, 946, 418, 194, 953, 339,~
## $ bpm                  <dbl> 125, 92, 138, 170, 144, 141, 148, 100, 130, 170, ~
## $ key                  <chr> "B", "C#", "F", "A", "A", "C#", "F", "F", "C#", "~
## $ mode                 <chr> "Major", "Major", "Major", "Major", "Minor", "Maj~
## $ `danceability_%`     <dbl> 80, 71, 51, 55, 65, 92, 67, 67, 85, 81, 57, 78, 7~
## $ `valence_%`          <dbl> 89, 61, 32, 58, 23, 66, 83, 26, 22, 56, 56, 52, 6~
## $ `energy_%`           <dbl> 83, 74, 53, 72, 80, 58, 76, 71, 62, 48, 72, 82, 6~
## $ `acousticness_%`     <dbl> 31, 7, 17, 11, 14, 19, 48, 37, 12, 21, 23, 18, 6,~
## $ `instrumentalness_%` <dbl> 0, 0, 0, 0, 63, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 17,~
## $ `liveness_%`         <dbl> 8, 10, 31, 11, 11, 8, 8, 11, 28, 8, 27, 15, 3, 9,~
## $ `speechiness_%`      <dbl> 4, 4, 6, 15, 6, 24, 3, 4, 9, 33, 5, 7, 7, 3, 6, 4~
```

```
data$`energy_%` <- ifelse(data$`energy_%` >= 50, 1, 0)
```

This R code line converts the energy_% column in the data dataframe into a binary variable. As a result, the values in the energy_% column are recoded to be either 0 or 1. This process transforms the energy_% column into a binary variable, making it suitable for categorical analyses and modeling.

```
data <- data%>%na.omit()
```

```
names(data)[names(data) == "energy_%"] <- "energy"
columns_to_keep <- c("energy","in_deezer_charts", "danceability_%","in_spotify_charts","in_deezer_playli
spotify_data <- data[, columns_to_keep]
colnames(spotify_data)[3] <- "danceability"
colnames(spotify_data)[6] <- "valence"
colnames(spotify_data)[7] <- "acousticness"
colnames(spotify_data)[9] <- "liveness"
colnames(spotify_data)[11] <- "speechiness"
```

This R code block changes the column names in the data dataframe and then selects and renames specific columns to create a new dataframe. In summary, this code block selects certain columns from the data dataframe, changes the names of some columns, and results in the creation of a new dataframe named spotify_data.

2.1-) Imbalancedness Problem

```
set.seed(150)
data_balanced_s <- ovun.sample(energy~., data = spotify_data, method = "over", p=0.5)
data_balanced <- data_balanced_s$data
```

For imbalanced problem, This code block balances the energy variable in the spotify_data dataset and addresses class imbalances within the dataset. This process ensures that the classes have an equal number of examples, which can improve the model's training performance by balancing the dataset and setting randomness configurations.

2.2-) Checking for Best Values of set.seed() and Prop

```
best_results <- list(m_number=NULL, i_number=NULL, best_ball=-Inf, k_number=NULL)

for(i in seq(0.75, 0.9, 0.02)){
  for(m in seq(120,150,2)){
    for(k in seq(120,150,2)){
      set.seed(k)
      data_balanced_s <- ovun.sample(energy~., data = spotify_data, method = "over", p=0.5)
      data_balanced <- data_balanced_s$data
      set.seed(m)
      data_split <- initial_split(data = data_balanced, prop = i)
      data_train <- data_split |> training()
      data_test <-  data_split |> testing()
      lr_model <- glm(energy ~ ., data = data_train, family = "binomial")
      lr_preds <- predict(lr_model, newdata = data_test, type = "response")
      lr_pred_classes <- ifelse(lr_preds > 0.5, 1, 0)
      lr_conf_matrix <- confusionMatrix(factor(lr_pred_classes), factor(data_test$energy))
      ball_ac <- lr_conf_matrix$byClass["Balanced Accuracy"]
      if(ball_ac > best_results$best_ball){
      best_results$m_number <- m
      best_results$i_number <- i
      best_results$best_ball <- ball_ac
      best_results$k_number <- k
      }
    }
  }
}
print(best_results)
```

```
## $m_number
## [1] 146
##
## $i_number
## [1] 0.87
##
## $best_ball
## Balanced Accuracy
##         0.8662791
##
## $k_number
## [1] 150
```

This R code block contains a loop that searches various parameters to get the best balanced accuracy using a

logistic regression model on a dataset. To briefly summarize, the code block tries to find the best model by iterating over three parameters (i, m and k).

3-) Splitting Edited Data

```
set.seed(146)
data_split <- initial_split(data = data_balanced, prop = 0.87)
data_train <- data_split |> training()
data_test <-  data_split |> testing()
```

This code block randomly splits the balanced dataset into 87% training set and 13% test set.

4-) Logistic Regression Model and It's Performance

```
lr_model <- glm(energy ~ ., data = data_train, family = "binomial")
summary(lr_model)
```

```
##
## Call:
## glm(formula = energy ~ ., family = "binomial", data = data_train)
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -3.244e+01  1.480e+01  -2.192 0.028396 *
## in_deezer_charts     -7.782e-03  2.092e-02  -0.372 0.709888
## danceability         -2.097e-02  6.188e-03  -3.389 0.000702 ***
## in_spotify_charts     2.249e-02  6.462e-03   3.481 0.000500 ***
## in_deezer_playlists   1.321e-05  1.388e-04   0.095 0.924174
## valence               4.415e-02  4.065e-03  10.862  < 2e-16 ***
## acousticness         -5.082e-02  3.368e-03 -15.090  < 2e-16 ***
## released_year         1.657e-02  7.346e-03   2.256 0.024053 *
## liveness              1.695e-02  6.434e-03   2.635 0.008406 **
## in_spotify_playlists -1.156e-05  2.211e-05  -0.523 0.601024
## speechiness          -2.765e-02  6.588e-03  -4.196 2.71e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1592.8  on 1148  degrees of freedom
## Residual deviance: 1038.6  on 1138  degrees of freedom
## AIC: 1060.6
##
## Number of Fisher Scoring iterations: 5
```

This R code block creates a logistic regression model and displays the summary of this model. In the summary of the model, the effects of the independent variables on the dependent variable and the overall performance of the model can be evaluated.

Comment: Looking at the coefficients of the independent variables, the coefficients of the variables "valence" and "acousticness" are quite high, which may indicate that these variables have a significant impact on the energy level. The coefficients of the "speechiness" and "danceability" variables are also noteworthy. According to the significance codes, several variables such as "valence", "acousticness", "danceability" and "speechiness"

are highly significant ($p < 0.001$). Other variables are not statistically significant. The difference between null and residual deviation values indicates that the model provides improvement. The AIC value is 1060.6. This value evaluates the fit of the model, while a lower AIC value will indicate that the model fits the data better. The Fisher Scoring iterations value is 5, indicating that the optimization process was repeated five times.

4.1-) Logistic Regression Model's Predictions

```
lr_preds <- predict(lr_model, newdata = data_test, type = "response")
lr_pred_classes <- ifelse(lr_preds > 0.5, 1, 0)
```

This R code block makes predictions on the test dataset using the logistic regression model and converts these predictions into binary classes. In summary, this block of code makes probability predictions on the test dataset using the created logistic regression model and then transforms these predictions into two classes, 0 and 1. This process is used to evaluate the classification performance of the model.

```
head(lr_preds)
```

```
##         1         2         3         4         5         6
## 0.9560074 0.8860477 0.9733243 0.9446315 0.1511248 0.9620404
```

This R code block provides a preview of how the model's predicted probabilities are distributed by displaying the first few predicted probability values in the lr_preds vector.

Comment: These probabilities are classified according to whether they are above or below the equal value (0.5). That is, probabilities greater than 0.5 are assigned to class 1, and probabilities less than 0.5 are assigned to class 0.

```
head(lr_pred_classes)
```

```
## 1 2 3 4 5 6
## 1 1 1 1 0 1
```

This R code block displays the first few class predictions (0 or 1) in the lr_pred_classes vector, allowing you to quickly see which observations the model has assigned to which class.

4.2-) Confusion Matrix of Logistic Regression Model

```
lr_conf_matrix <- confusionMatrix(factor(lr_pred_classes), factor(data_test$energy))
print(lr_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 75 12
##          1 11 74
##
##                Accuracy : 0.8663
##                  95% CI : (0.8061, 0.9133)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7326
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8721
```

```
##             Specificity : 0.8605
##          Pos Pred Value : 0.8621
##          Neg Pred Value : 0.8706
##              Prevalence : 0.5000
##          Detection Rate : 0.4360
##    Detection Prevalence : 0.5058
##       Balanced Accuracy : 0.8663
##
##        'Positive' Class : 0
##
```

This code block creates a confusion matrix to evaluate the performance of the logistic regression model and prints this matrix to the console, allowing you to see the accuracy of the model's predictions and various performance metrics.

Comment: Shows how the model compares actual and predicted classes. The classes on the left represent the real classes, while the classes on the top represent the predicted classes. For example, cell (0,0) represents true negatives (true value 0 and predicted value 0), while cell (1,0) represents false negatives (true value 1 and predicted value 0). Accuracy: Shows the proportion of observations classified as correct. In this case, the accuracy rate is 0.8663, meaning the model has a high ability to predict correctly. Sensitivity: Shows the proportion of true positives correctly classified. In this case, the precision ratio is 0.8721, meaning that the model's ability to correctly identify positive cases is quite high. Specificity: Shows the proportion of true negatives that are correctly classified. In this case, the specificity ratio is 0.8605, meaning the model's ability to correctly identify negative situations is quite high. Kappa Statistic: Measures how good the model is relative to its ability to make random predictions. A value close to 1 indicates that the model is much better than random guessing. Sensitivity (Detection Rate): Shows the detection rate of true positives. In this case, the detection rate is 0.4360, meaning only 43.60% of true positives were correctly detected.

```
rmse <- sqrt(mean((lr_pred_classes-data_test$energy)^2))
rmse
```

```
## [1] 0.3656787
```

This code block calculates the RMSE to measure how much the logistic regression model's predictions differ from the actual values and prints this value. RMSE is a commonly used metric to evaluate the prediction performance of a model. A lower RMSE value indicates better model performance.

Comment: The RMSE value is quite small, indicating that the model has a low margin of error between the actual values and its predictions. Therefore, it can be said that the predictive ability of this model is quite good.

4.3-) Overfitting for Logistic Regression Model

```
cvModel <- train(energy ~ ., data = data_train, method = "glm",family = "binomial", trControl = trainCo
print(cvModel)
```

```
## Generalized Linear Model
##
## 1149 samples
##   10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1034, 1034, 1035, 1034, 1034, 1034, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
```

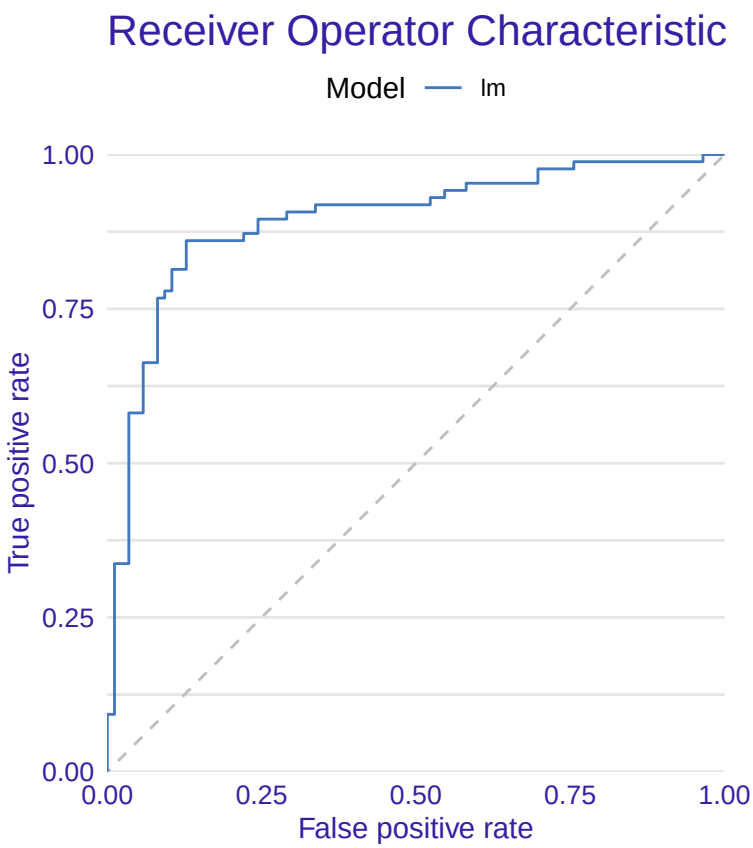
```
##   0.3897397  0.3958881  0.29885
```

This R code block creates a logistic regression model using cross-validation and displays the model summary. This code block trains the logistic regression model using the cross-validation method and evaluates the model's performance. The results, along with the cross-validation results and performance metrics, are printed.

Comment: It shows that the obtained RMSE, R-squared and MAE values are quite low and there is no sign of overfitting.

4.4-) ROC Curve of Logistic Regression Model

```r
explain_lr <- DALEX::explain(model = lr_model,
              data = data_test[, -1],
              y = data_test$energy == "1",
              type = "classification",
              verbose = FALSE)
performance_lr <- DALEX::model_performance(explain_lr)
plot(performance_lr, geom = "roc")
```



This code block plots a ROC curve to evaluate the performance of a logistic regression model. The ROC curve is used to visually assess the model's sensitivity and specificity performance.

Comment: The ROC curve of the model lies well above the diagonal line, indicating that the model has a good ability to distinguish positive and negative classes. The model is able to maintain a low FPR while achieving a high TPR. This shows that the model is effective in distinguishing the two classes. Overall, the ROC curve shows that the model performs well in classifying samples and provides a good balance between sensitivity and specificity.

4.5-) Brier Score of Logistic Regression Model

```
BrierScore(lr_model)
```

```
## [1] 0.1479531
```

This code block calculates the Brier score to evaluate the model's calibration. A lower Brier score indicates that the model's predictions are better aligned with the actual labels, whereas a higher Brier score may indicate poor model calibration.

Comment: The Brier score is a measure that assesses how well the predictions of a model align with the true labels. In this particular case, the Brier score has been calculated as 0.1479531. A lower Brier score indicates that the model's predictions are more in line with the true labels. Therefore, a Brier score of 0.1479531 suggests that the model's predictions are generally acceptably good and indicative of alignment with the true labels.

5-) Decision Tree Model and It's Performance

```
dt_model <- decision_tree() |>
set_engine("rpart") |>
set_mode("classification")
```

This code enables rapid creation of decision tree models for classification problems.

5.1-) Decision Tree Model

```
data_train$energy <- as.factor(data_train$energy)
spotify_dt <- dt_model |>
fit(energy~., data = data_train)
spotify_dt
```

```
## parsnip model object
##
## n= 1149
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 1149 572 1 (0.49782419 0.50217581)
##     2) acousticness>=49.5 403  60 0 (0.85111663 0.14888337)
##       4) acousticness>=69.5 260  16 0 (0.93846154 0.06153846) *
##       5) acousticness< 69.5 143  44 0 (0.69230769 0.30769231)
##        10) valence< 77 129  30 0 (0.76744186 0.23255814) *
##        11) valence>=77 14   0 1 (0.00000000 1.00000000) *
##     3) acousticness< 49.5 746 229 1 (0.30697051 0.69302949)
##       6) valence< 37.5 264 130 1 (0.49242424 0.50757576)
##        12) in_spotify_charts< 18 226 103 0 (0.54424779 0.45575221)
##          24) in_deezer_playlists>=643.5 29   4 0 (0.86206897 0.13793103) *
##          25) in_deezer_playlists< 643.5 197  98 1 (0.49746193 0.50253807)
##            50) in_deezer_playlists< 188 184  86 0 (0.53260870 0.46739130)
##             100) in_spotify_playlists>=1782 96  33 0 (0.65625000 0.34375000)
##               200) danceability>=47 86  24 0 (0.72093023 0.27906977) *
##               201) danceability< 47 10   1 1 (0.10000000 0.90000000) *
##             101) in_spotify_playlists< 1782 88  35 1 (0.39772727 0.60227273)
```

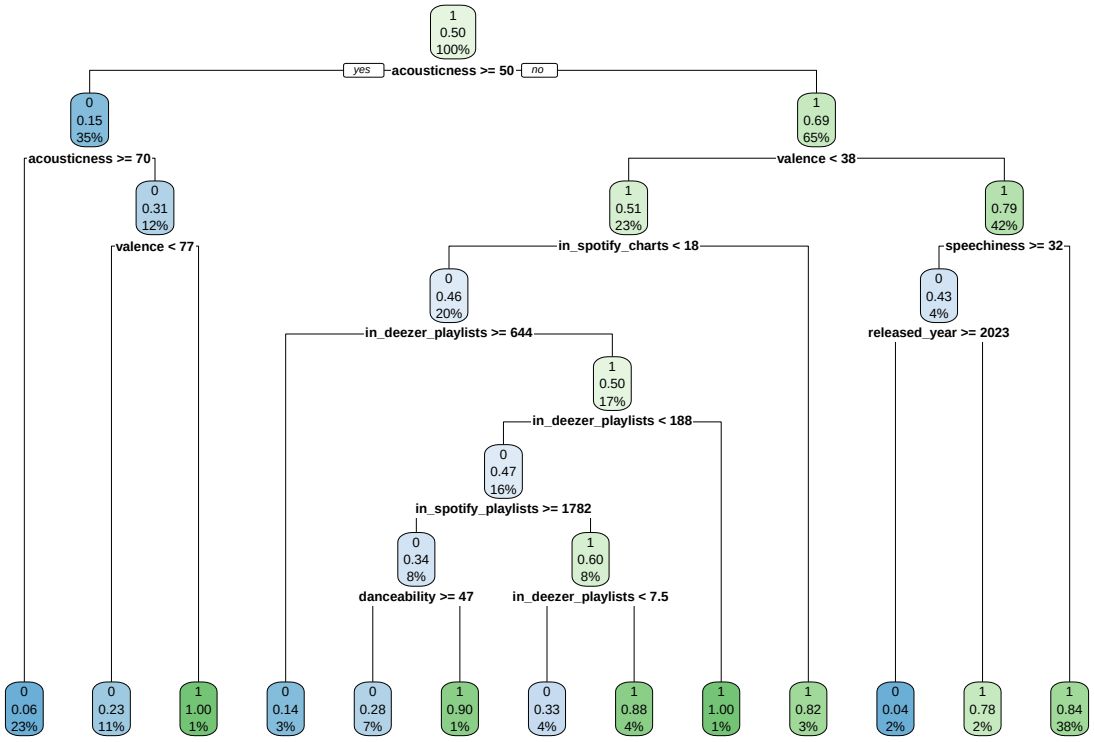
```
##               202) in_deezer_playlists< 7.5 45  15 0 (0.66666667 0.33333333) *
##               203) in_deezer_playlists>=7.5 43   5 1 (0.11627907 0.88372093) *
##            51) in_deezer_playlists>=188 13   0 1 (0.00000000 1.00000000) *
##         13) in_spotify_charts>=18 38   7 1 (0.18421053 0.81578947) *
##       7) valence>=37.5 482  99 1 (0.20539419 0.79460581)
##         14) speechiness>=31.5 51  22 0 (0.56862745 0.43137255)
##           28) released_year>=2022.5 24   1 0 (0.95833333 0.04166667) *
##           29) released_year< 2022.5 27   6 1 (0.22222222 0.77777778) *
##         15) speechiness< 31.5 431  70 1 (0.16241299 0.83758701) *
```

This code creates a decision tree model and then uses the dataset to train that model. The trained model is called spotify_dt and can be used to classify music energy.

Comment : This output describes the decision rule that the decision tree model uses to classify data samples based on certain characteristics. For example, it makes decisions based on properties such as acousticness, valence, danceability, in_spotify_charts, in_deezer_playlists, released_year, and speechlessness. Each decision point checks the value of an attribute and makes a decision based on a certain threshold value. For example, (acousticness>=49.5) checks whether the acousticness property is greater than or equal to 49.5. This determines which subcategory a data point will be assigned to. Ultimately, this output represents the complex decision processes of the model and shows the importance of features in the data set in the classification process.

5.2-) Graph of the Decision Tree Model

```
rpart.plot(spotify_dt$fit)
```



This decision tree chart is a visual representation of a model that predicts whether a song will be popular based on certain musical characteristics. The decision tree makes predictions by splitting the dataset based

on a feature at each node.

Comment: Root Node; acousticness <= 50: This is the first split point of the decision tree. If a song's acoustic value is less than 50, right; If not, we go left. Left Branch; acousticness <= 70: If acousticness is greater than 50 and less than or equal to 70, we continue on this branch. valence < 77: If the valence (the song's positivity level) is less than 77, we continue. Valence < 77: The song is unlikely to be popular (0.6%). Valence >= 77: The song has a high probability of being popular (1.0%). acousticness > 70: If acousticness is greater than 70, the song is more likely to be popular (1.0%). The comments will continue like this.

5.3-) Predictions of Decision Tree Model

```
spotify_predictions <- spotify_dt |>
predict(new_data = data_test)
spotify_predictions
```

```
## # A tibble: 172 x 1
##    .pred_class
##    <fct>
##  1 1
##  2 1
##  3 1
##  4 1
##  5 0
##  6 1
##  7 1
##  8 1
##  9 1
## 10 1
## # i 162 more rows
```

This code is used to evaluate the performance of the model on data that was not used in training the model. Predictions can be used to evaluate how the model performs on new examples in the test dataset.

```
spotify_dt |>
predict(new_data = data_test,
        type = "prob")
```

```
## # A tibble: 172 x 2
##    .pred_0 .pred_1
##      <dbl>   <dbl>
##  1   0.162   0.838
##  2   0.162   0.838
##  3   0.162   0.838
##  4   0.184   0.816
##  5   0.767   0.233
##  6   0.162   0.838
##  7   0.162   0.838
##  8   0.162   0.838
##  9   0.184   0.816
## 10   0.162   0.838
## # i 162 more rows
```

This output will return a matrix containing the classification probabilities of each data point. These probabilities can be used to predict which class the data point belongs to based on the probability of each class.

5.4-) Confusion Matrix of Decision Tree Modeel

```
data_test$energy <- as.factor(data_test$energy)
spotify_results <- tibble(predicted=spotify_predictions$.pred_class,
                          actual=data_test$energy)
spotify_results|> conf_mat(truth = actual, estimate = predicted)
```

```
##           Truth
## Prediction  0  1
##          0 75 13
##          1 11 73
```

This output is called a confusion matrix, which is used to evaluate the performance of a classification model. The confusion matrix shows the relationships between the classes predicted by the model and the actual classes.

5.5-) Performance Metric Values of Decision Tree Model

```
spotify_results |> accuracy(truth = actual, estimate = predicted)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.860
```

In line with this information, the accuracy metric of the model was calculated as 86.05%. This indicates that 86.05% of all predictions of the model were correct. The accuracy metric is used to evaluate the overall performance of the model and in this case we can say that the model performs quite well.

```
spotify_results |> sens(truth = actual, estimate = predicted)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    binary         0.872
```

This result shows that the model can identify examples belonging to the positive class with 87.21% accuracy and does not miss them. This metric shows that the model performs quite well in terms of correctly detecting positive examples.

```
spotify_results |> f_meas(truth = actual, estimate = predicted)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 f_meas  binary         0.862
```

According to this table, the F1 score of the model is calculated as 86.21%. This shows that the model strikes a good balance between precision and recall values. The model is successful in correctly identifying positive classes and minimizing the number of false positives.

5.6-) Overfitting Checking for Decision Tree Model

```
spotifyfit <- dt_model |>
last_fit(as.factor(energy) ~., split = data_split)
spotifyfit |> collect_metrics()
```

```
## # A tibble: 3 x 4
##   .metric     .estimator .estimate .config
##   <chr>       <chr>          <dbl> <chr>
## 1 accuracy    binary         0.860 Preprocessor1_Model1
## 2 roc_auc     binary         0.889 Preprocessor1_Model1
## 3 brier_class binary         0.116 Preprocessor1_Model1
```

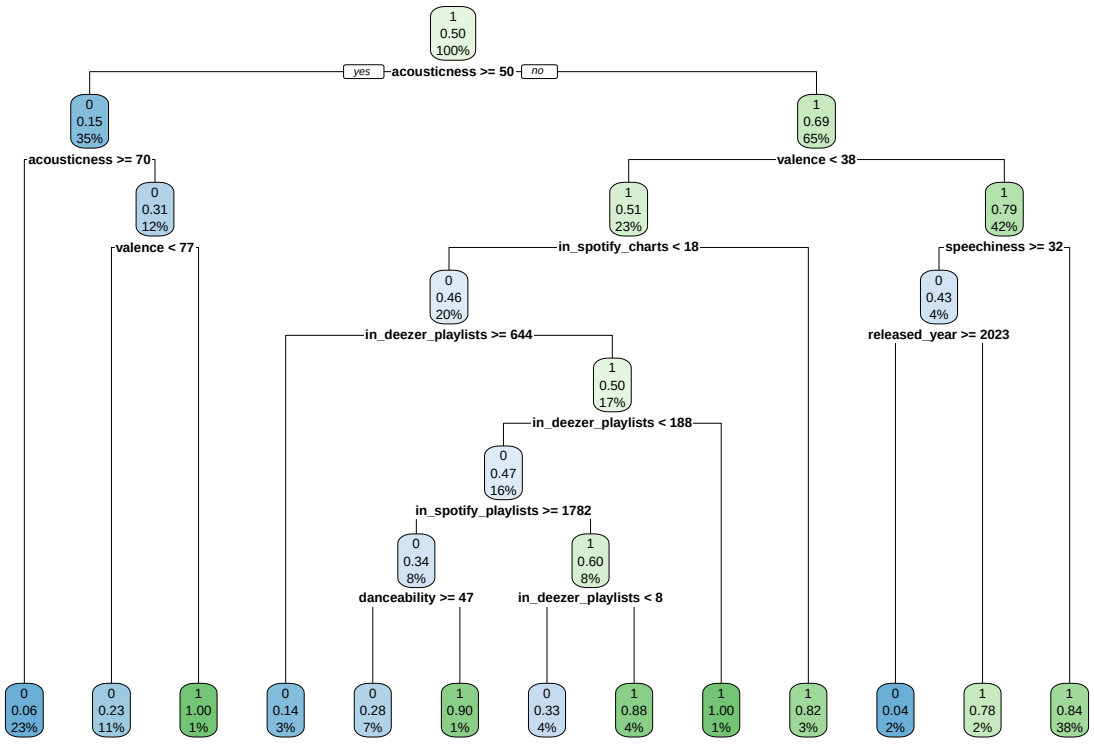
This output is a table summarizing three different metrics that evaluate a model's performance. These metrics include the model's accuracy (accuracy), area under the ROC curve (roc_auc), and Brier score (brier_class). Accuracy: 86.05% of all predictions of the model are correct. This indicates that the overall performance of the model is good. ROC AUC: ROC AUC score indicates that the classification ability of the model is good. The closer it is to 1, the better the model's ability to distinguish positive and negative classes. This score indicates that the model performs quite well with 88.93%. Brier Score: The Brier score measures how accurate the predicted probabilities are. A lower Brier score indicates better performance. In this case, a low score of 0.116 indicates that the model's probability predictions are accurate.

```
spotifyfit |> collect_predictions()
```

```
## # A tibble: 172 x 7
##    .pred_class .pred_0 .pred_1 id               .row `as.factor(energy)` .config
##    <fct>         <dbl>   <dbl> <chr>           <int> <fct>               <chr>
##  1 1             0.162   0.838 train/test spl~    12 1                   Prepro~
##  2 1             0.162   0.838 train/test spl~    14 1                   Prepro~
##  3 1             0.162   0.838 train/test spl~    16 1                   Prepro~
##  4 1             0.184   0.816 train/test spl~    25 1                   Prepro~
##  5 0             0.767   0.233 train/test spl~    58 1                   Prepro~
##  6 1             0.162   0.838 train/test spl~    59 1                   Prepro~
##  7 1             0.162   0.838 train/test spl~    63 1                   Prepro~
##  8 1             0.162   0.838 train/test spl~    71 1                   Prepro~
##  9 1             0.184   0.816 train/test spl~    73 1                   Prepro~
## 10 1             0.162   0.838 train/test spl~    83 1                   Prepro~
## # i 162 more rows
```

This output is a table detailing the predictions from the spotifyfit model. Each row of the table shows predictive information for one data point (sample). Each column in the table represents different information. This information details how confident the model is for which examples and which classes it predicts. It can be used to evaluate the performance of the model and the accuracy of its predictions. Higher probability values of the model indicate that it is more confident in its predictions, which increases the reliability of the model.

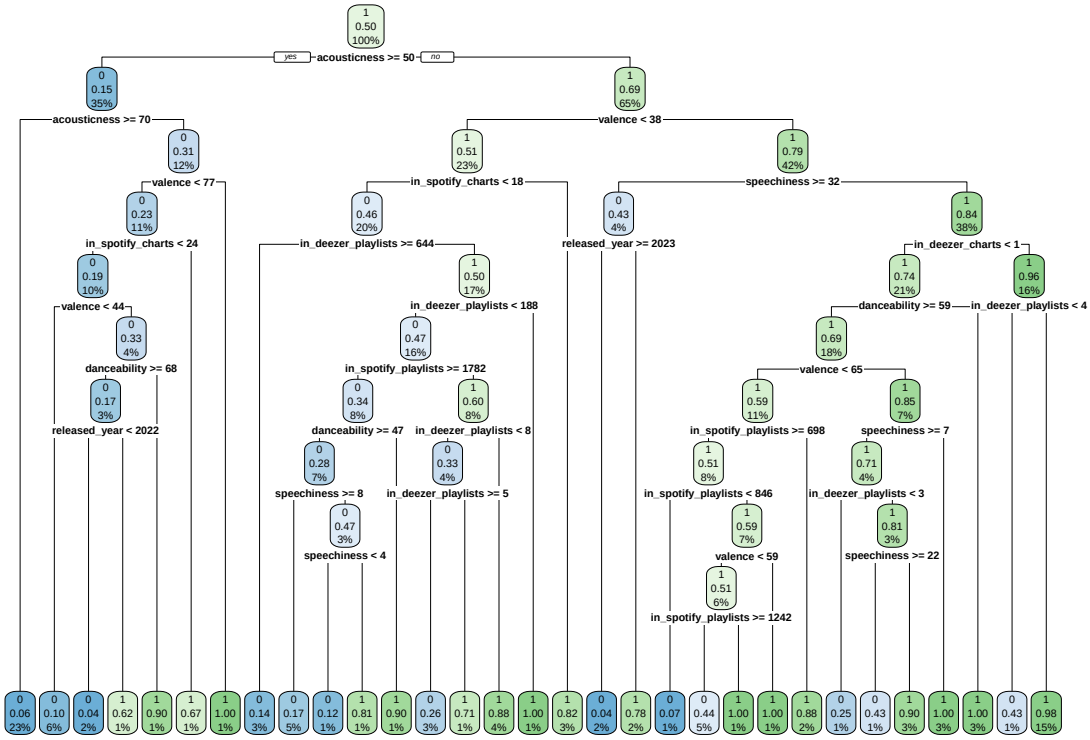
```
spotify_dt1 <- rpart(energy ~ ., data = data_train,
method = "class")
rpart.plot(spotify_dt1)
```



This decision tree uses features such as acoustics, valence, playlist appearances, conversation rate, and release year to predict the popularity of songs. Each leaf node represents the probability that songs belonging to that branch will become popular. In summary, this chart is a visual representation of a model used to predict whether songs will be popular based on musical characteristics.

5.7-) Hyperparameter Tuning for Decision Tree Model

```
new_dt <- rpart(energy ~ ., data = data_train,
method = "class",
maxdepth =10,
cp = 0.001)
rpart.plot(new_dt)
```



This decision tree chart is a visual representation of a more detailed model that predicts whether a song will be popular based on musical features. In each node, a feature and the division decision made based on this feature are shown.

Comment: valence < 38: Songs with valence less than 38: in_spotify_charts < 18: If Spotify charts have less than 18 songs (0.46%). in_deezer_playlists <= 644: If there are 644 or fewer songs in Deezer playlists (0.47%). in_spotify_playlists >= 1782: If Spotify playlists have 1782 or more songs (0.85%). danceability <= 47: Danceability is less than 47 (0.38%). speechiness <= 8: Speechlessness is less than 8 (0.47%). in_deezer_playlists < 188: If there are less than 188 songs in Deezer playlists (0.50%). valence >= 38: Songs with valence 38 or higher: released_year >= 2023: Songs released in 2023 or newer (0.43%). speechlessness >= 32: Speechlessness is 32 or higher (0.84%). in_deezer_charts < 1: Songs that do not appear on Deezer charts at all (0.74%). danceability <= 59: If danceability is less than 59 (0.95%). speechiness <= 7: Speechlessness is less than 7 (0.81%). in_deezer_playlists <= 1: If there is 1 or less song in Deezer playlists (%0.51).

5.8-) Imbalancedness Check for Decision Tree Model

```
table(data_train$energy)/dim(data_train) [1]
```

```
##
##         0         1
## 0.4978242 0.5021758
```

These results show that the class distribution of the energy variable in the data_train dataset is quite balanced. The proportions of both classes (0 and 1) in the dataset are approximately equal, which allows the model to have a more balanced performance in learning and predicting both classes. Balanced class distribution minimizes potential model performance problems that may arise from imbalance between classes.

6-) Bagging Tree Model and It's Performance

```
trained_bt <- ranger(energy ~ .,
                     data = data_train,
                     mtry = 8)
trained_bt
```

```
## Ranger result
##
## Call:
##  ranger(energy ~ ., data = data_train, mtry = 8)
##
## Type:                             Classification
## Number of trees:                  500
## Sample size:                      1149
## Number of independent variables:  10
## Mtry:                             8
## Target node size:                 1
## Variable importance mode:         none
## Splitrule:                        gini
## OOB prediction error:             4.35 %
```

This output contains the summary of the training process of the bagging tree model. Here are some highlights from this output: Model type: A classification model was trained on the given data set. Number of trees: The model consists of a total of 500 trees. Sample size: There are 1149 samples in the training dataset. Number of independent variables: 10 independent variables were used in the model. Mtry: The number of independent variables that trees consider for randomly selected split points is determined as 8. Target node size: set to 1. Variable importance ranking mode: It is stated that the importance order of variables is not determined. Division rule: The "gini" division criterion was used when determining the division points of trees. OOB prediction error: Out-of-Bag (OOB) prediction error is reported as 4.35%. OOB error represents the error rate of predictions made on data points that were not used in training the trees. In this case, 4.35% indicates that approximately 4.35% of these predictions are incorrect.

6.1-) Predictions and Classifications of Bagging Tree Model

```
bt_predicted_classes <- predict(trained_bt, data_test)$predictions
head(bt_predicted_classes)
```

```
## [1] 1 1 1 1 1 1
## Levels: 0 1
```

This output contains some of the predictions made for the test dataset. A class prediction was made for each observation. The predicted classes are categorized as 0 and 1. An initial fraction of predicted classes was reported as 1.

6.2-) Confusion Matrix of Bagging Tree Model

```
conf_matrix <- confusionMatrix(bt_predicted_classes, data_test$energy)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 85  6
```

```
##          1  1 80
##
##                Accuracy : 0.9593
##                  95% CI : (0.9179, 0.9835)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9186
##
##  Mcnemar's Test P-Value : 0.1306
##
##             Sensitivity : 0.9884
##             Specificity : 0.9302
##          Pos Pred Value : 0.9341
##          Neg Pred Value : 0.9877
##              Prevalence : 0.5000
##          Detection Rate : 0.4942
##    Detection Prevalence : 0.5291
##       Balanced Accuracy : 0.9593
##
##        'Positive' Class : 0
##
```

This output shows that the model has a high overall accuracy rate (0.9535), while the specificity value is slightly lower (0.9186) and the Kappa value is slightly lower (0.907).It can be said that the performance of the model is generally quite good and provides a good fit with the data.

7-) Random Forest Model and It's Performance

```
rf_spotify <- randomForest(energy ~ ., data = data_train,
                           type="classification")
rf_spotify
```

```
##
## Call:
##  randomForest(formula = energy ~ ., data = data_train, type = "classification")
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 3.57%
## Confusion matrix:
##     0   1 class.error
## 0 566   6  0.01048951
## 1  35 542  0.06065858
```

Number of trees: The model consists of a total of 500 trees. Number of variables tried at each division: It was stated that 3 variables were tested at each division point. OOB prediction error rate: Out-of-Bag (OOB) prediction error is reported as 3.57%. This is the error rate obtained by predicting a portion of the training data set without being used by each tree. Confusion matrix: A matrix used to evaluate classification performance in more detail. This matrix allows comparison of actual classes with predicted classes. For example, it was stated that there were 566 correct guesses and 6 wrong guesses in class 0. It can be seen that there are 35 wrong guesses and 542 correct guesses in class 1. Class error rates: Error rates are reported separately for each class. For example, the error rate for class 0 is 0.0104%, and for class 1 it is 0.0606%.

This output provides valuable information about the configuration and performance of the model. Since the OOB error rate is low, it can be said that the model generally performs well.

7.1-) Predictions and Classifications of Random Forest Model

```
rf_predicted_classes <- predict(rf_spotify, data_test)
head(rf_predicted_classes)
```

```
## 1 2 3 4 5 6
## 1 1 1 1 0 1
## Levels: 0 1
```

7.2-) Confusion Matrix of Random Forest Model

```
conf_matrix_2 <- confusionMatrix(rf_predicted_classes, data_test$energy)
print(conf_matrix_2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 85  6
##          1  1 80
##
##                Accuracy : 0.9593
##                  95% CI : (0.9179, 0.9835)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9186
##
##  Mcnemar's Test P-Value : 0.1306
##
##             Sensitivity : 0.9884
##             Specificity : 0.9302
##          Pos Pred Value : 0.9341
##          Neg Pred Value : 0.9877
##              Prevalence : 0.5000
##          Detection Rate : 0.4942
##    Detection Prevalence : 0.5291
##       Balanced Accuracy : 0.9593
##
##        'Positive' Class : 0
##
```

According to this output, the model shows a very high accuracy rate (0.9593) and a balanced sensitivity (0.9884) and specificity (0.9302). This indicates that the model performs well overall.