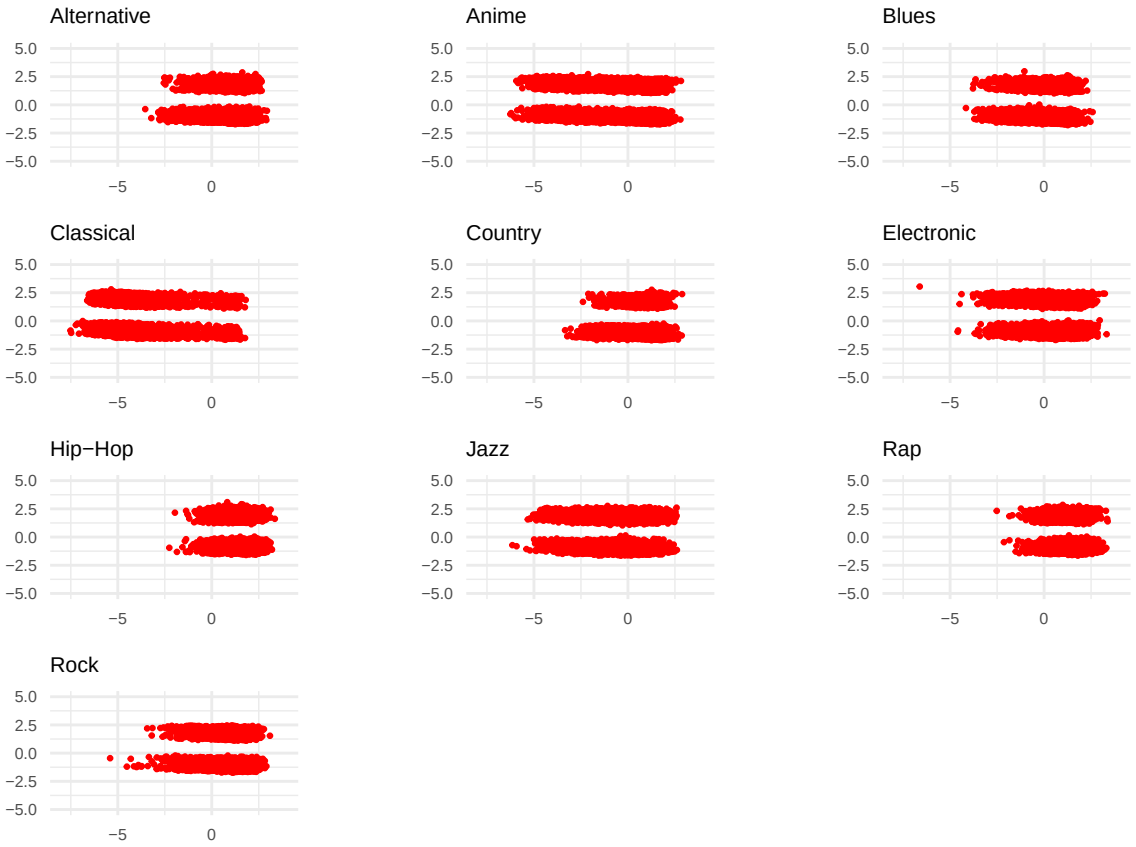
Machine Learning (732A99) Lab Block 2

Christoforos Spyretos & Marc Braun

2022-01-20

Description of the Data

The data used for this lab is a dataset from Kaggle (https://www.kaggle.com/vicsuperman/prediction-of-music-genre) containing information about different songs. The dataset contains 50005 rows and 18 columns representing a unique ID of each song, the artist name, the track name, the key, the mode, the date when the data was obtained and numerical measures of the popularity, acousticness, danceability, duration, energy, instrumentalness, liveness, loudness, speechiness, tempo, and valence. The last column contains the music genre which is divided in the ten categories Electronic, Anime, Jazz, Alternative, Country, Rap, Blues, Rock, Classical and Hip-Hop.
The objective of this report is to predict the music genre of a song based on the aforementioned dataset using a random forest model and a neural network and compare the prediction quality of the two models.
The dataset contains five rows with missing values in each feature and missing values for the tempo in 4980 rows. Furthermore, in 4939 rows the value for the duration is set to -1 (or 4460 rows if one subtracts the rows that have a missing value for tempo).
The coordinates of the data in the first two principal components (PCs), where the x-axis correspondents to the first PC and the y-axis to the second PC, looks this:



It can be concluded that at least from observing the first two PCs it might not be easy to differentiate between the music genres from the given features.

Descriptions of Models and Experimental Design

General Methodology

To predict the music genre of a song based on the features given in the aforementioned dataset, the following steps were conducted.

- *Preselection of models*: First of all, different approaches to predict the target variable have to be selected.
- *Data preparation*: Missing values need to be discarded and only relevant columns kept.
- *Feature selection*: Not every feature in the dataset is suitable to predict the target variable. Therefore, a selection about which features to use has to be made.
- *Model training and hyperparameter tuning*: The models have to be trained with the training data and the hyperparameters of the models be tuned to achieve good prediction results.
- *Comparison of the models*: Based on the trained and tuned models, the best approach to predict the target variable can be selected.

Preselection of Models and Description

The models used in this lab to predict the target variable are random forest and neural network. Both of the models are briefly described hereafter.

Random Forest

A random forest is an ensemble method where multiple decision trees are constructed with bootstraped datasets to reduce the correlation between the different trees. However, the data is not only bootstraped but also whenever the model is about to split at a node in the training process, not all the possible input variables $x_1, x_2, ...x_p$ are considered but only a random subset consisting of $q \leq p$ inputs. This increases the variance of each individual tree. Experience has shown, however, that the decrease in correlation between the trees has a stronger influence on the overall prediction quality than the increase of σ^2.
Random forests have different hyperparameters. The number of trees to be generated does not lead to overfitting because the data for training the tree is bootstraped. Hence, the only reason for choosing a small number for the number of trees is computation time (which increases appr. linear with the number of trees). Another parameter is q, where the rule of thumb $q = \sqrt{p}$ can be applied. A third parameter is the minimal node size which indirectly regulates the depth of the trees.

Neural Networks

Neural Networks form the base of Deep Learning, a subfield of Machine Learning. A neural network is a sequence of algorithms that attempts to recognise underlying relationships of a set of data through a procedure that its structure is inspired by the human brain. According to the definition, neural networks illustrates layers of neurons. The main idea is to extract linear combinations of the inputs as derived features and then model the target value as a nonlinear function of these features.

The components of a neural network are the input layers, which receive the data, the hidden layers, which perform the computations required by our network, and the output layer, which provide the predicted results. More specifically, the m features of the input layers,$x_1, x_2, ..., x_m$, are connected with the n neurons of the hidden layers, each connection is assigned by a weight,$w_1, w_2, ..., w_m$, and the inputs are multiplied by the corresponding weight,$\sum_{i=1}^{n}(x_i w_i)$, which is send as an input to the neurons of the hidden layers. Afterwards, each of these neurons is associated with the bias,$b_1, b_2, ..., b_m$, which is added to the multiplication,$\sum_{i=1}^{m}((x_i w_i) + b_i)$. Then, this value is passed to an activation function, $f(\sum_{i=1}^{m}((x_i w_i) + b_i))$, which decides whether a neuron should be activated or not, and its purpose is to introduce non-linearity into the output of a neuron. Next, the activated neurons transmit data to the next hidden layer, and this step is called forward propagation. The above process is repeated depending on the number of hidden layers the neural network has, and it provides the results to the output layer. However, a neural network model might make a wrong prediction, which is why it needs to be trained. The predicted output is compared to the actual output to realise the error. Based on the error rate, the weights are adjusted in a process called gradient descent until the network makes sufficient predictions.

Results

Random Forest

Data Preparation

As a first step, the data is read and the missing values and the rows containing described in section Description of the Data are removed. After this clean-up of the data, it consists of 40560 rows.

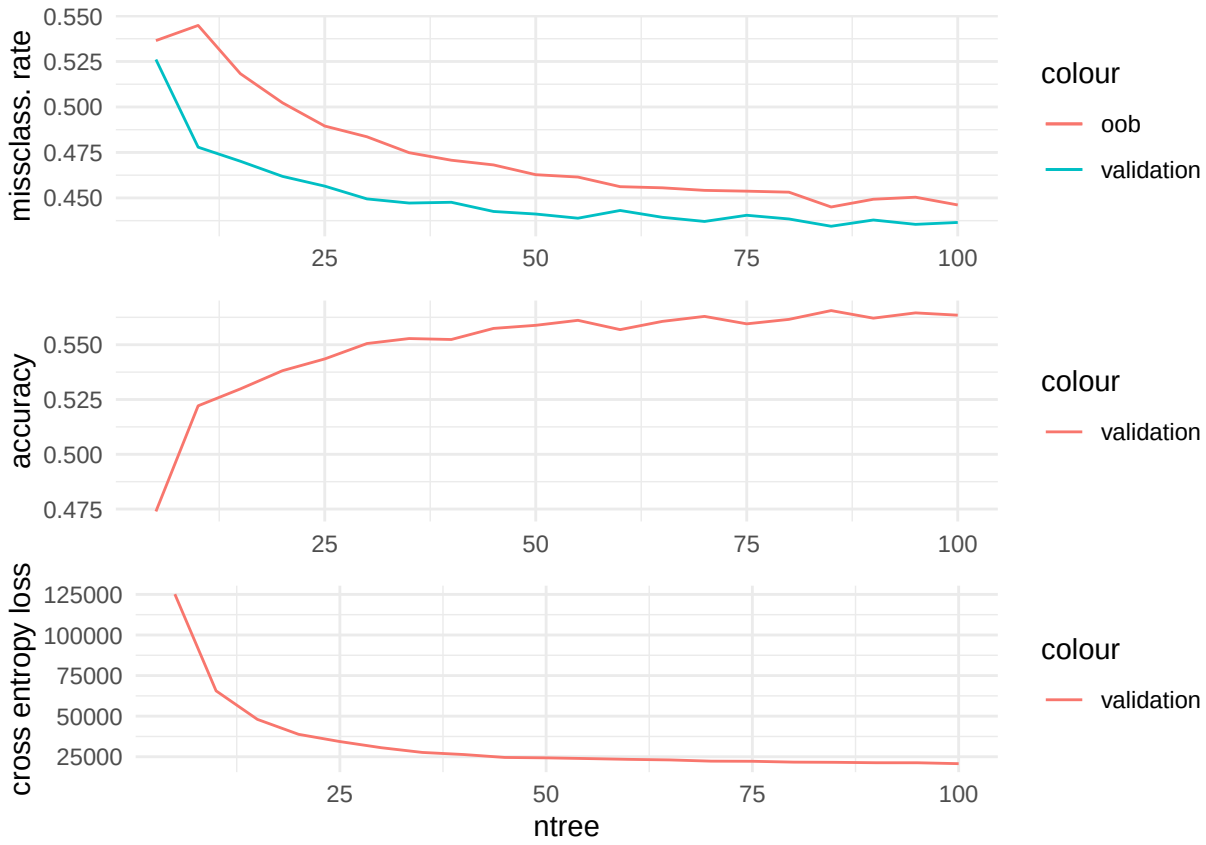
Feature Selection

The columns ID and Song can be removed as they are unique to each song (rows in the data) and therefore to do not contain information that can be used to predict the genre of a song. Furthermore, there are 6363 different artists in the dataset which is too much to use for prediction. Most R implementations of prediction models can not handle that many features and dummy encode all of them would result in 6362 additional columns which is also too much to handle for most models. Therefore, the artist column is also removed. The last column which is removed is the date column containing the date when the data was extracted. As there is no expected connection between the date of extraction and the music genre the column can be discarded.
The result is a dataframe with dimension 40560 x 14.
The data is then split into 40% training, 40% validation and 20% test data randomly.

Model Training and Hyperparameter Tuning

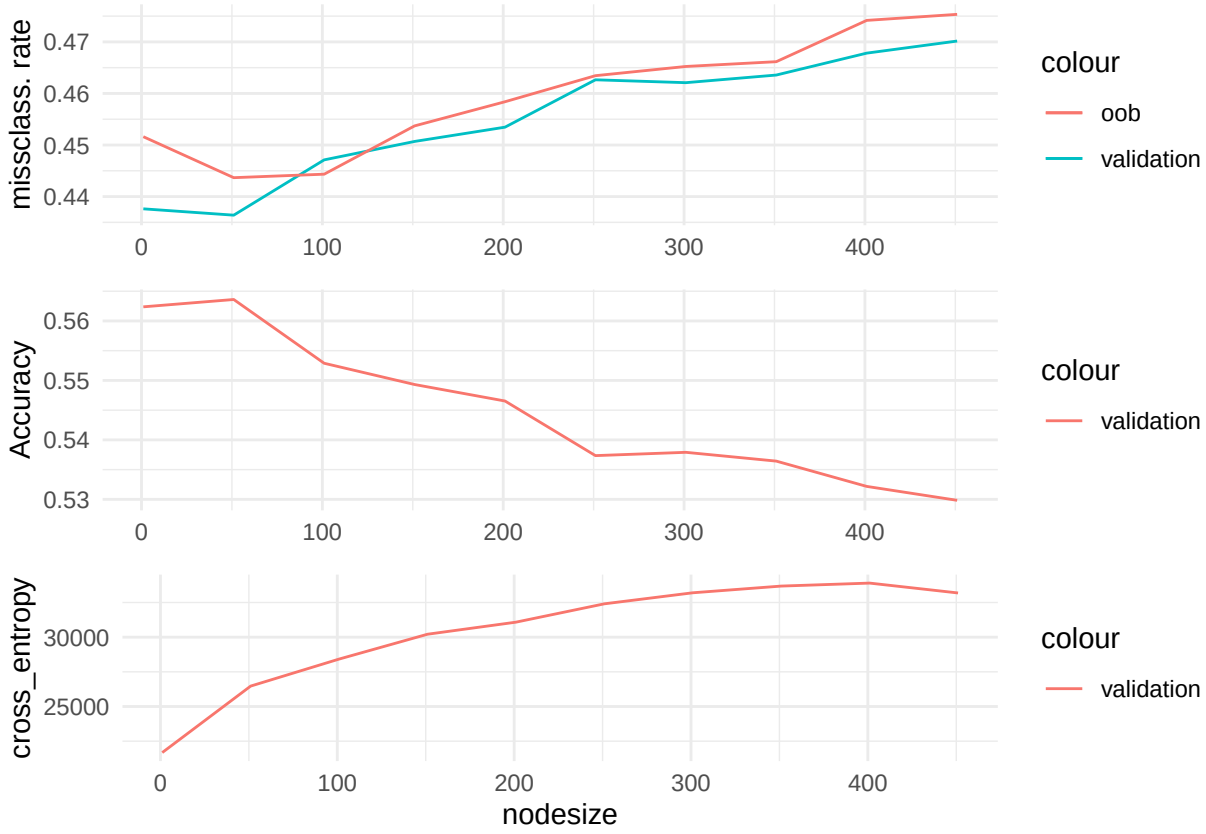
For training the random forest model, the `randomForest` packages `randomForest` function is used. The parameters mentioned in section Random Forest are called ntree (for the number of trees), nodesize (for the minimum nodesize) and mtry (for q, the number of features randomly sampled from all possible features).
For evaluating the parameters, out-of-bag misclassification rate, the validation misclassification rate, the validation accuracy and the validation cross entropy are considered.
In the first step, all three parameters are first examined individually assuming the remaining two parameters are fixed and in the second step a grid search is performed along intervals for the parameters estimated in the first step.

The first parameter to be selected is ntree. The standard value is set to 500 and choosing large values does not lead to overfitting but reducing the number leads to shorter computation time and a large number of trees might not lead to a significant improvement in prediction quality.



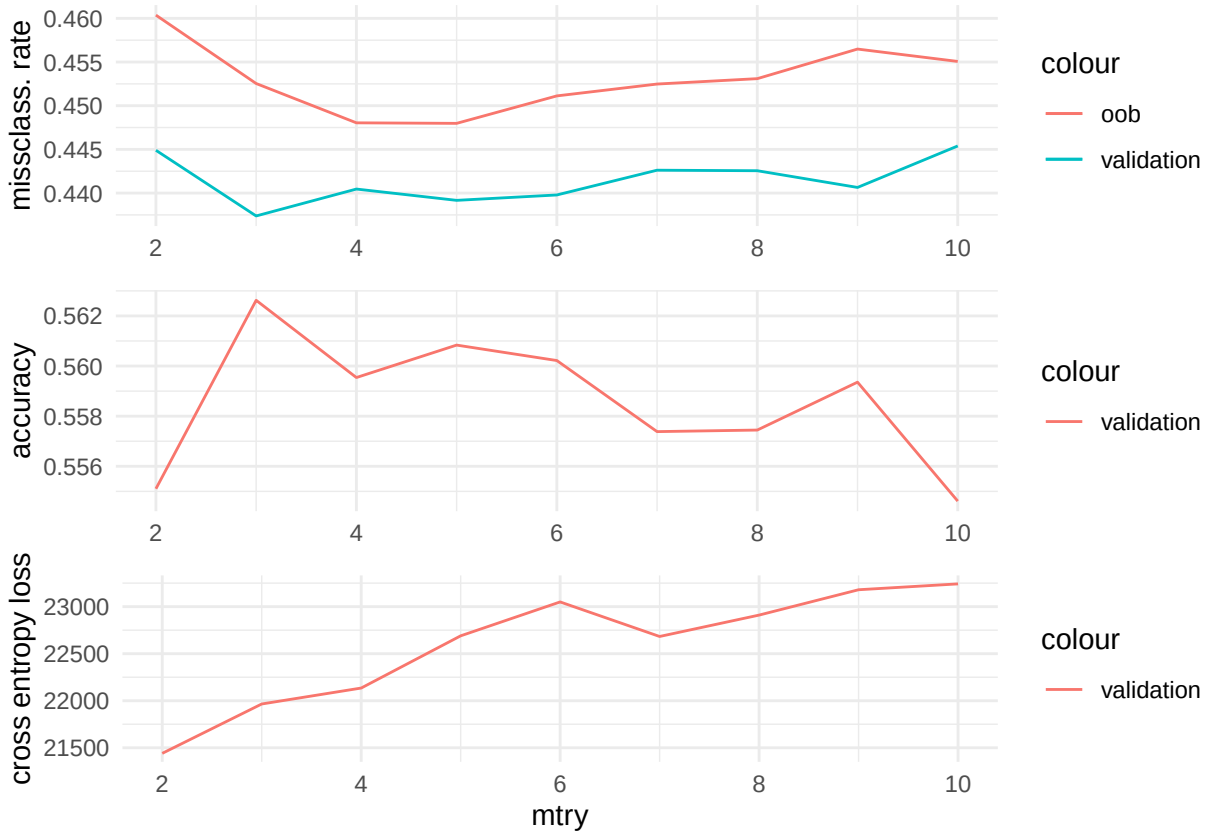
From the plots it can be seen that the misclassification rate decreases, the accuracy increases and the cross-entropy-loss decreases with an increasing number of ntree, as it is expected. However, after about ntree = 80 the prediction does not seem to become significantly better which is why ntree = 80 is chosen. The cross entropy loss does not change significantly even after ntree = 40 so ntree = 80 is a conservative estimation.

The next parameter to be optimized is the nodesize. The default nodesize of the `randomForest` function is 1 for classification, meaning that the depth of the trees is not restricted. In the following plot, values for nodesize between 1 and 500 are tested with ntree = 80.



From the plots it can be seen that not giving a lower bound to the nodesize (as the nodesize parameter defines the minimum number of terminal nodes) results in the best prediction in terms of a minimal cross entropy. The misclassification rate seems to go slightly down (about 0.1 percent points for the validation data and only 0.7 percent points for the oob estimation) from nodesizes of 1 to nodesize of 50 and then up again for a nodesize of 100. Therefore, it can be assumed that the a very good nodesize is in the interval between 1 and 100.

Last, q is examined. The rule of thumb says that for classification problems q should be around $\sqrt{p}$ where p is the number of features used for prediction. As in this lab 13 features are used, a good value is expected to be around 3 (`floor(sqrt(p))` is also the standard value of the `randomForest` function). In the following plots, the prediction quality metrics for mtry between 2 and 10 are shown. For this plot, ntree = 80 and nodesize = 1 is used.



It can be seen that a good value for mtry seems to be three. Both the validation accuracy and the validation misclassification rate are optimal for mtry = 3. The cross-entropy-loss is minimised for mtry = 2 and the out-of-bag misclassification rate for mtry = 4, leading to the conclusion that a good value for mtry is between 2 and 4.

Up until now, the parameters were only examined independently of each other (in the sense that all parameters were held constant except for one and then changes in the remaining parameter examined). Using grid search for the deducted intervals of mtry and nodesize, the final parameter values can be deducted. For the grid search, the number of trees ntree is not considered anymore in order to save cumputing time. The parameter ntree is set to 80 which was already a conservative estimation for the parameter, so considering larger values for ntree would probably not improve the target metrics of the model even if mtry and nodesize change in value.

mtry	nodesize	miscl_valid	miscl_oob	accuracy_valid	ce_valid
2	1	0.4448964	0.4603674	0.5551036	21440.61
2	25	0.4366371	0.4463141	0.5633629	22163.01
2	50	0.4419995	0.4437253	0.5580005	24494.59
2	75	0.4434788	0.4517382	0.5565212	25890.63
2	100	0.4410133	0.4546967	0.5589867	26558.44
3	1	0.4372535	0.4546967	0.5627465	21551.94
3	25	0.4370069	0.4435404	0.5629931	23593.33
3	50	0.4364522	0.4424926	0.5635478	26719.76
3	75	0.4423077	0.4477318	0.5576923	27478.15
3	100	0.4442184	0.4500123	0.5557816	29043.54
4	1	0.4371918	0.4529093	0.5628082	21453.23
4	25	0.4328772	0.4415064	0.5671228	24977.86

mtry	nodesize	miscl_valid	miscl_oob	accuracy_valid	ce_valid
4	50	0.4369453	0.4468688	0.5630547	27211.74
4	75	0.4433555	0.4415680	0.5566445	29725.33
4	100	0.4439719	0.4469921	0.5560281	30454.63

From the table it can be seen that mtry = 4 and nodesize = 25 produce the lowest misclassification rate and mtry = 2 and nodesize = 1 produce the lowest cross entropy loss. The misclassification rate is only 1.2 percentage points worse in the latter case and the cross entropy loss decreased by over 14% so the final parameter values mtry = 2, nodesize = 1 and ntree = 80 are chosen.

Hereafter the result of the metrics for the deducted model and the results for the `randomForest` model without specifying the parameters is shown. Both models are trained using the train data and the metrics calculated using the test data.

	misclass_rate	accuracy	cross_entropy_loss
plain	0.433	0.567	9548.677
proposed	0.447	0.553	11281.812

It can be seen that the plain model seems to perform slightly better, which seems to be caused by the higher number of trees trained. However, the training time for the proposed model is highly reduced and results in almost as good results.

Neural Network

Data Preparation

As a first step, the data is read and the missing values and the rows containing described in section Description of the Data are removed. After this clean-up of the data, it consists of 40560 rows. Moreover, the data is scaled for the implementation of the Neural Network model.

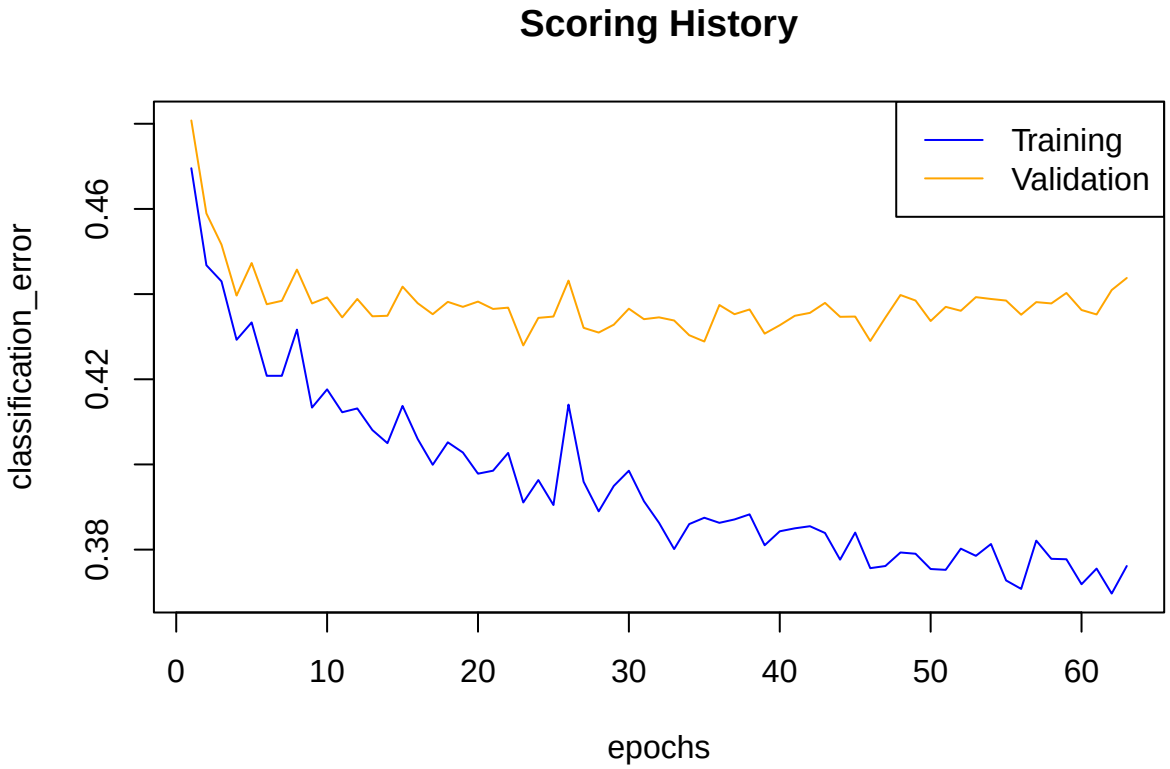
Feature Selection

The Neural Network model uses the same features as the Random Forest, which are described above.

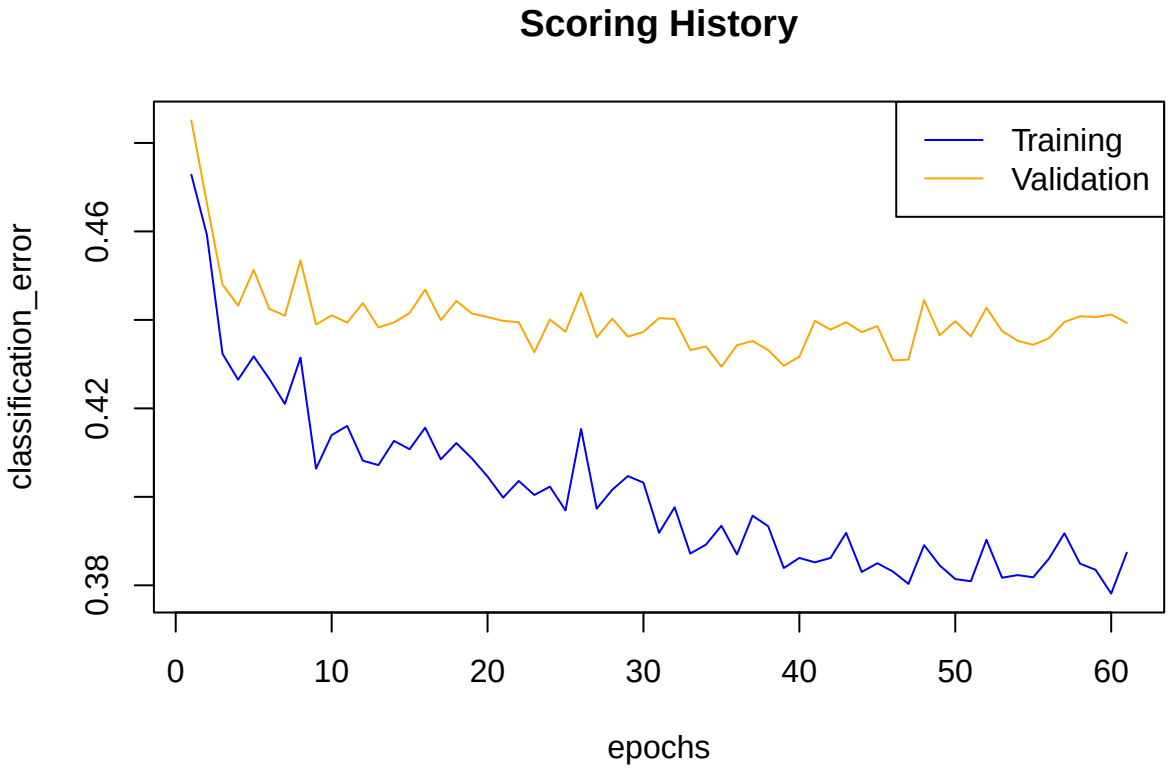
Model Training and Hyperparameter Tuning

The `h2o` package is used to train the Neural Network model.

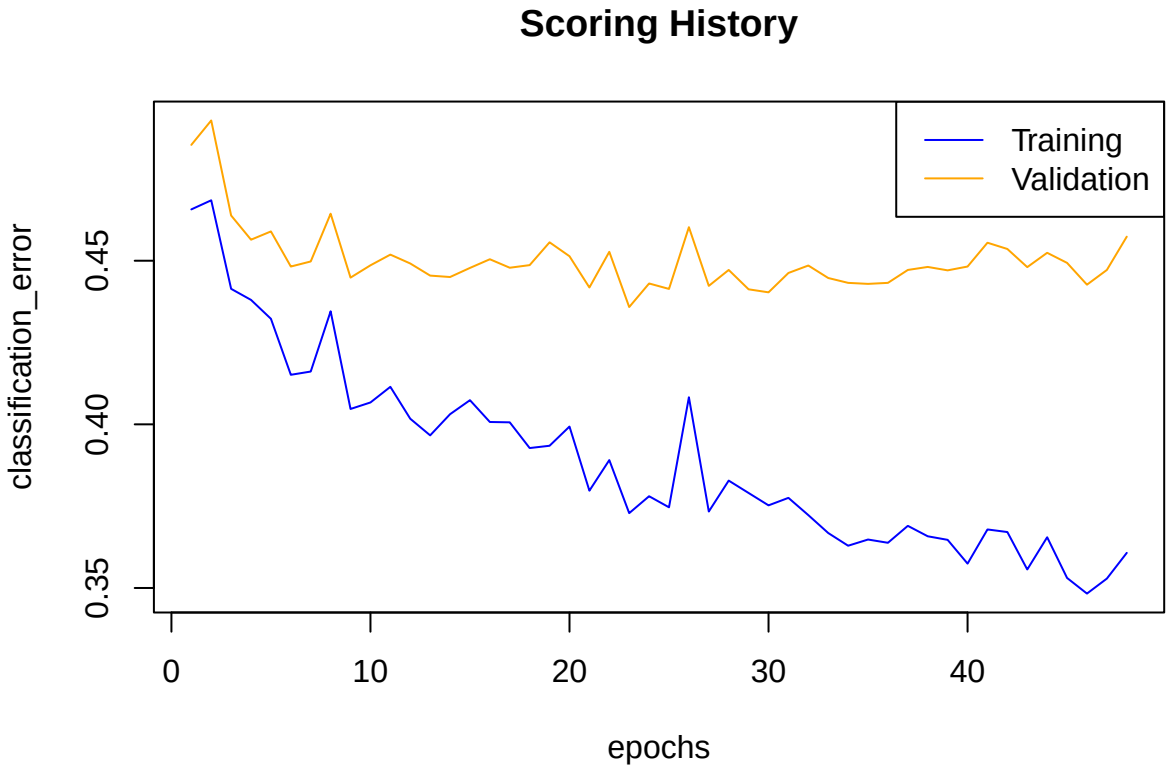
The h2o.deeplearning() function is used with the following arguments:
1. *y = music genre*, specifies the column to use as the dependent variable.
2. *training_frame = train_h2o*, specifies the dataset used to build the model.
3. *validation_frame = valid_h2o*, specifies the dataset used to evaluate the accuracy of the model.
4. *activation = "..."*, specifies the activation function. *Tanh*, *Rectifier* & *Maxout* were used.
5. *hidden = c(30,30,30)*, which refers to the hidden layers that the model used. With too few hidden units, the model might not have enough flexibility to capture the nonlinearities in the data. The extra weights can be shrunk toward zero with too many hidden units. Typically the number of hidden units is somewhere in the range of 5 to 100.
6. *epochs = 150*, the number of iterations that the neural network would be run, could increase the model's performance.
7. *nfolds = 5*, specify the number of folds for cross-validation.
8. *stopping_metric = "misclassification"*, specifies the metric for early stopping.
9. *stopping_rounds = 20*, stops training when the option selected for stopping_metric does not improve for the specified number of training rounds, based on a simple moving average. The metric is computed on the validation data; otherwise, training data is used.
10. *seed = 12345*, specify the random number generator.
11. *reproducible = TRUE*, not getting different outputs when the model is re-run.



The above graph describes the scoring history of the Neural Network model using the Tanh function (Hyperbolic Tangent), $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, as an activation function. Both the training and validation misclassification errors follow almost the same pattern. The training misclassification error dropped to 0.38 in the 45th epoch and remained around 0.38 until the model stopped training. The validation's data misclassification error decreased to 0.43 in the approximately 25th epoch and stayed around 0.43 between the 26th and the last epoch. Additionally, the model stopped training in the 65th epoch because of the stopping metric, 20 stopping rounds. Choosing this value seemed reasonable because after experimenting with multiple values for the stopping rounds, the models would return the following results. On the one hand, training the model with less than 20 stopping rounds stopped too early because the learning rate was insufficient. On the other hand, training the model with values greater than 20 stopping rounds started to overfit after the 65th epoch; for instance, the validation misclassification error increased compared to the training misclassification error, which remained around 0.38.



The above graph describes the scoring history of the Neural Network model using the ReLU function (Rectified Linear Unit), $f(x) = max(0, x)$, as an activation function. Both the training and validation misclassification errors follow almost the same pattern. The training misclassification error dropped to 0.38 in the 47th epoch and remained around 0.38 until the model stopped training. The validation's data misclassification error decreased to 0.43 in the 35th epoch and stayed around 0.43 for the rest epochs. Additionally, the model stopped training in the 63rd epoch because of the stopping metric, 20 stopping rounds. On the one hand, training the model with less than 20 stopping rounds stopped too early because the learning rate was insufficient. On the other hand, training the model with values greater than 20 stopping rounds started to overfit after the 63rd epoch. As a result, the validation misclassification error increased compared to the training misclassification error, which remained around 0.38. Thus, choosing 20 stopping rounds seemed the most appropriate for this model.



The above graph describes the scoring history of Neural Network model using the Maxout function (Maxout Unit), $f(x) = max(W_1x+b_1, W_2x+b_2, ..., W_nx+b_n)$, as an activation function. Both the training and validation misclassification errors follow almost the same pattern. The training misclassification error dropped to 0.35 in the approximately 48th epoch and slightly increased to 0.36 until the last epoch. The validation's data misclassification error decreased from approximately 0.48 to 0.45 in the 8th epoch and stayed around 0.45 for the rest epochs. An interesting feature is that the model stopped training in the 50th epoch, much earlier than the previous models; the reason for this is the stopping metric, 20 stopping rounds. On the one hand, training the model with values less than 20 stopping rounds stopped too early because the learning rate was insufficient. On the other hand, training the model with values greater than 20 stopping rounds started to overfit after the 50th epoch. The validation misclassification error increased compared to the training misclassification error, which remained around 0.35.

The table below illustrates each model's accuracy and cross-entropy. It is evident that the Neural Networks using the Tanh and the ReLU functions had similar performance; the Neural Network using the ReLU performed slightly better. That might happen because the main advantage of the ReLU function is that it does not activate all the neurons simultaneously; the neurons would only be deactivated if the output of the linear transformation is less than 0. This could be useful because the deactivated neurons could result in suboptimal performance of the network. However, a main disadvantage of the ReLU is that all the negative input values become zero immediately, which decreases the model's ability to fit or train from the data correctly (dying ReLU problem). The Tanh helps in centering the data to zero; hence it could easily specify if the values are strongly negative, neutral, or strongly positive. A drawback of this function is that it normalises the neuron's output to a range between -1 and 1, and when the neuron reaches the minimum or maximum value of its range, that respectively correspond to -1 and 1, its derivative is equal to 0. The Maxout function generalises the ReLU and the Leaky ReLU activation functions. The advantage of this function is that Leaky ReLU solves the dying ReLU problem. However, the predictions might not be consistent for negative input values, which is why the accuracy of the neural network might be worse compared to the other neural networks.

Table 3: Validation Data Metrics

	Tanh	ReLU	Maxout
Accuracy	0.556	0.561	0.543
Cross Entropy	18870.617	18802.762	20254.884

The best-predicted genres for the first two Neural Network models are *Anime* and *Classical* as the confusion matrices illustrate below. On the other hand, the worst predicted genre for the first two Neural Network models is *Rap* and *Alternative*. Most of *Rap* songs are classified as *Hip-Hop* songs; that might happen because hip-hop and rap share some common characteristics; for example, both have wordplay lyrics with assonance and rhyming stanzas and deal with social matters such as wealth, drug use, poverty, luxury and politics. More or less, this was expected because in the PCA plots aforementioned in the section Description of the Data, the hip-hop and rap PCA plots look similar. The *Alternative* genre is a type of music, which is produced by performers who are outside the musical mainstream and is regarded as more eclectic, original, or challenging than most popular music (such as rock, pop, or country), and this might be the main reason that it could be predicted as another genre.

Table 4: Confusion Matrix of the Neural Network using Tanh

	Alternative	Anime	Blues	Classical	Country	Electronic	Hip-Hop	Jazz	Rap	Rock	Error	Rate
Alternative	408	22	33	3	103	63	129	38	39	172	0.596	602 / 1,010
Anime	16	847	79	22	15	29	0	15	0	5	0.176	181 / 1,028
Blues	41	78	662	6	59	36	5	77	5	46	0.348	353 / 1,015
Classical	17	82	32	807	1	13	0	62	0	6	0.209	213 / 1,020
Country	98	22	68	1	568	20	27	43	6	133	0.424	418 / 986
Electronic	40	54	70	3	34	653	31	74	7	27	0.342	340 / 993
Hip-Hop	24	0	0	0	10	14	746	12	152	28	0.243	240 / 986
Jazz	33	12	135	41	37	95	27	591	1	21	0.405	402 / 993
Rap	24	0	1	0	7	13	595	4	256	89	0.741	733 / 989
Rock	81	5	6	2	64	8	67	15	51	733	0.290	299 / 1,032
Totals	782	1122	1086	885	898	944	1627	931	517	1260	0.376	3,781 / 10,052

Table 5: Confusion Matrix of the Neural Network using ReLU

	Alternative	Anime	Blues	Classical	Country	Electronic	Hip-Hop	Jazz	Rap	Rock	Error	Rate
Alternative	444	32	12	6	106	28	165	24	39	154	0.560	566 / 1,010
Anime	27	847	50	56	14	17	0	15	0	2	0.176	181 / 1,028
Blues	67	119	497	17	89	44	7	129	3	43	0.510	518 / 1,015
Classical	23	47	12	871	1	7	0	55	0	4	0.146	149 / 1,020
Country	102	27	43	1	593	10	30	31	10	139	0.399	393 / 986
Electronic	87	63	34	12	39	612	42	77	10	17	0.384	381 / 993
Hip-Hop	23	0	0	0	8	3	796	3	125	28	0.193	190 / 986
Jazz	86	16	51	43	48	82	44	594	2	27	0.402	399 / 993
Rap	29	0	0	0	4	2	654	2	215	83	0.783	774 / 989
Rock	121	7	3	2	63	3	60	8	76	689	0.332	343 / 1,032
Totals	1009	1158	702	1008	965	808	1798	938	480	1186	0.387	3,894 / 10,052

However, the third Neural Network model, which uses the Maxout function as an activation function, provides a slightly different confusion matrix. The best-predicted genres are *Anime*, *Classical* and *Rock*, with the lowest error rate. On the other hand, the worst predicted genre is *Alternative*, and it has a higher error rate than the above confusion matrices. An interesting result is that the third Neural Network model handles the classification of *Rap* songs better than the first two Neural Network models.

Table 6: Confusion Matrix of the Neural Network using Maxout

	Alternative	Anime	Blues	Classical	Country	Electronic	Hip-Hop	Jazz	Rap	Rock	Error	Rate
Alternative	287	48	58	9	78	48	122	75	81	204	0.716	723 / 1,010
Anime	4	902	56	27	5	17	0	11	0	6	0.123	126 / 1,028
Blues	17	97	701	7	33	32	5	69	3	51	0.309	314 / 1,015
Classical	10	75	32	841	0	6	0	48	0	8	0.175	179 / 1,020
Country	53	43	99	5	465	24	22	64	20	191	0.528	521 / 986
Electronic	31	66	62	6	21	619	28	109	15	36	0.377	374 / 993
Hip-Hop	10	1	0	0	2	3	558	10	366	36	0.434	428 / 986
Jazz	24	12	126	36	15	51	25	672	7	25	0.323	321 / 993
Rap	10	0	0	1	1	2	324	6	550	95	0.444	439 / 989
Rock	25	9	5	2	14	4	43	23	76	831	0.195	201 / 1,032
Totals	471	1253	1139	934	634	806	1127	1087	1118	1483	0.361	3,626 / 10,052

For tuning the model, the same arguments were used as the trained models with additional *hyperparameters* and *search criteria*, which are described below.

The *hyperparameters* that were used are:
1. *activation = c("Rectifier","Tanh","Maxout")*, the Rectified Linear Unit (ReLU), the Hyperbolic Tangent (Tanh) and the Maxout Unit (Maxout) activation functions were used. 2. Two penalty parameters *l1* & *l2*, both with values *0, 0.00001, 0.0001, 0.001 and 0.01*. The *l1* lets only strong weights survive, and the *l2* prevents any single weight from getting too big.

The *search criteria* that were used are:
1.*strategy = "RandomDiscrete"*, which generates random discrete values.
2.*max_runtime_secs = 600*, refers to the runtime that the tuned model was ran.
3.*stopping_metric = "misclassification"*, specifies the metric to use for early stopping.
4.*stopping_rounds = 20*, they automatically find the optimal number of epochs and stop the model early.

The table illustrates which models performed better after the grid search. The accuracy of the model is calculated using the validation data.

activation	l1	l2	model_ids	accuracy
Tanh	0e+00	1e-04	nn_grid_model_13	0.5654586
Tanh	1e-05	1e-04	nn_grid_model_3	0.5653969
Tanh	0e+00	1e-05	nn_grid_model_8	0.5648422
Rectifier	1e-05	0e+00	nn_grid_model_15	0.5627465
Rectifier	1e-04	1e-03	nn_grid_model_19	0.5625000
Rectifier	1e-04	1e-04	nn_grid_model_11	0.5621302
Rectifier	1e-03	0e+00	nn_grid_model_4	0.5621302
Tanh	1e-05	1e-03	nn_grid_model_18	0.5590483
Tanh	1e-04	1e-03	nn_grid_model_6	0.5574458
Maxout	1e-04	0e+00	nn_grid_model_12	0.5563979
Maxout	0e+00	1e-02	nn_grid_model_10	0.5521450
Maxout	0e+00	1e-04	nn_grid_model_9	0.5515286
Maxout	1e-04	1e-02	nn_grid_model_5	0.5480769
Maxout	1e-02	1e-05	nn_grid_model_1	0.5354413
Tanh	1e-03	1e-02	nn_grid_model_7	0.5120192
Rectifier	1e-02	1e-04	nn_grid_model_16	0.5106016
Rectifier	1e-02	1e-02	nn_grid_model_14	0.4961169
Tanh	1e-02	0e+00	nn_grid_model_17	0.4826183
Tanh	1e-02	1e-04	nn_grid_model_2	0.4751603
Tanh	1e-02	1e-03	nn_grid_model_21	0.4750986
Tanh	1e-02	1e-02	nn_grid_model_20	0.4002096

The table below demonstrates the result of the metrics of the tuned Neural Network model and the Neural Network model without the *l1* and *l2* hyperparameters. Both models are trained using the train data, the same arguments and the activation function that provided the best results on the grid search. Finally, the metrics are calculated using the test data.

```
##   |                                                                      |
##   |                                                                      |
```

	Accuracy	Cross.Entropy
Tuned Model	0.571	9110.971
Plain Model	0.556	9451.491

It could be seen that the tuned model performed slightly better, which might be caused by the *l1* & *l2* penalty parameters and the *runtime* of the model. However, the plain Neural Network model provided almost as good results.

Discussion and Conclusion

Discussion

The objective of this report was to predict the music genre of a song based on the aforementioned dataset using a random forest model and a neural network and compare the prediction quality of the two models. The neural network and the random forest models were trained with the same training data, validated with the same validation data and test on the same test data. The models were trained to predict the music genre of a song based on 13 features.
The proposed random forest model had accuracy of 0.564 on the test data, whereas the tuned neural network had a slightly better accuracy on the test data and can be seen in the table below.

	accuracy
Random Forest	0.553
Neural Network	0.571

One can see that the performance of the neural network is better than the performance of the random forest. The reason for the better performance might be that the neural network takes more complex dependencies in the data into account, whereas the random forest model seems to be too simple to predict the outcome of the target variable as reliable as the neural network. This benefit comes for the cost of higher computation time of the neural network compared to the random forest. Even though neural networks have the tendency to overfit the data, the results of this report show that the neural network performs better on the test data compared to the random forest.

Conclusion

It can be concluded that for predicting the music genre of a song based on the features described in this report, training a neural network with the proposed parameters results in a higher predictive quality compared to the proposed random forest model. There are other reasons why one might consider using the a random forest for this task, for example because of the easy interpretability of the results of a random forest compared to a neural network. As the objective of this report was, however, to compare the predictive quality, a further discussion about this topic will not be conducted here.
Furthermore, it can be stated that the higher predictive quality of the neural network compared to the random forest in this report can not be generalised to other use cases, as their quality was only assessed for one use case (predicting music genre) and one dataset.
Further research might investigate whether there are use cases for which a random forest or a neural network tends to be more suitable.

Appendix

```
library(formatR)
library(h2o)
h2o.init(nthreads = -1)
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(tidy.opts = list(width.cutoff = 80), tidy = TRUE)
# reading the data
data <- read.csv("music_genre.csv")
data <- data[, -c(1, 2, 3, 16)]  # remove ID, Artist, Song Title and Date

# remove NAs or missing values
data <- data[-which(is.na(data$popularity)), ]
data <- data[-which(is.na(as.numeric(data$tempo))), ]
data <- data[-which(data$duration_ms == -1), ]
data$tempo <- as.numeric(data$tempo)

data <- as.data.frame(unclass(data), stringsAsFactors = TRUE)
# scaling the data
library(fastDummies)
data <- dummy_cols(data, select_columns = c("key", "mode"))
data <- data[, -which(colnames(data) == "key" | colnames(data) == "mode")]
scaled_data <- scale(data[, -which(colnames(data) == "music_genre")])

X <- scaled_data
Y <- data$music_genre

res <- princomp(X)

Z <- as.matrix(X) %*% res$loadings

library(ggplot2)
library(gridExtra)
plot_data <- data.frame(P1 = Z[, 1], P2 = Z[, 2], Genre = Y)
plots <- list()
plot_counter <- 1
for (genre_cat in levels(Y)) {
    cur_plot <- ggplot(subset(plot_data, Genre %in% genre_cat), aes(P1, P2)) + geom_point(color = "r
        size = 0.5) + theme_minimal() + labs(subtitle = genre_cat) + xlim(-8, 4) +
        ylim(-5, 5) + coord_fixed() + theme(plot.subtitle = element_text(size = 8),
        aspect.ratio = 0.5, axis.title = element_blank(), axis.text = element_text(size = 6))
    plots[[plot_counter]] <- cur_plot
    plot_counter <- plot_counter + 1
}
grid.arrange(grobs = plots, ncol = 3)
# reading the data
data <- read.csv("music_genre.csv")
data <- data[, -c(1, 2, 3, 16)]  # remove ID, Artist, Song Title and Date

# remove NAs or missing values
data <- data[-which(is.na(data$popularity)), ]
data <- data[-which(is.na(as.numeric(data$tempo))), ]
data <- data[-which(data$duration_ms == -1), ]
data$tempo <- as.numeric(data$tempo)

data <- as.data.frame(unclass(data), stringsAsFactors = TRUE)

# Train-Test-Validation Split
```

```r
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n * 0.4))
train = data[id, ]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n * 0.4))
valid = data[id2, ]
id3 = setdiff(id1, id2)
test = data[id3, ]
library(randomForest)
library(caret)
library(ggplot2)
library(gridExtra)
# random forest

# cost functions
cross_entropy <- function(p, actual) {
    x <- 0
    for (i in 1:length(actual)) {
        if (p[i, which(colnames(p) == actual[i])] == 0)
            p[i, which(colnames(p) == actual[i])] <- 10^(-15)
        x <- x + (log(p[i, which(colnames(p) == actual[i])]))
    }
    return(-x)
}

misclass <- function(actual, predicted) {
    return(length(which(actual != predicted))/length(actual))
}


validation_ntree <- function(nrange = seq(5, 100, 5)) {
    accuracy_valid <- c()
    accuracy_train <- c()
    ce_valid <- c()
    ce_train <- c()
    miscl_valid <- c()
    miscl_oob_train <- c()
    ntree <- c()
    for (n in nrange) {
        forest_model <- randomForest(music_genre ~ ., train, ntree = n)

        predictions <- predict(forest_model, valid)
        perc_pred <- predict(forest_model, valid, type = "prob")
        confMat <- confusionMatrix(predictions, valid$music_genre)
        accuracy_valid <- append(accuracy_valid, confMat$overall[1])
        ce_valid <- append(ce_valid, cross_entropy(perc_pred, valid$music_genre))
        miscl_valid <- append(miscl_valid, misclass(valid$music_genre, predictions))
        miscl_oob_train <- append(miscl_oob_train, misclass(train$music_genre, forest_model$predicte

        predictions <- predict(forest_model, train)
        perc_pred <- predict(forest_model, train, type = "prob")
        confMat <- confusionMatrix(predictions, train$music_genre)
        accuracy_train <- append(accuracy_train, confMat$overall[1])
        ce_train <- append(ce_train, cross_entropy(perc_pred, train$music_genre))
```

```
        ntree <- append(ntree, n)
    }
    miscl_plot <- ggplot(data.frame(ntree = ntree, valid.misscl.rate = miscl_valid,
        oob.misscl.rate = miscl_oob_train)) + geom_line(mapping = aes(ntree, valid.misscl.rate,
        color = "validation")) + geom_line(mapping = aes(ntree, oob.misscl.rate,
        color = "oob")) + theme_minimal() + labs(y = "missclass. rate") + theme(axis.title.x = eleme
        axis.ticks.x = element_blank())

    acc_plot <- ggplot(data.frame(ntree = ntree, Accuracy_valid = accuracy_valid,
        Accuracy_train = accuracy_train)) + geom_line(mapping = aes(ntree, Accuracy_valid,
        color = "validation")) + #geom_line(mapping=aes(ntree, Accuracy_train, color='train')) + col
        color = "validation")) + #geom_line(mapping=aes(ntree, Accuracy_train, color='train')) + =
        color = "validation")) + #geom_line(mapping=aes(ntree, Accuracy_train, color='train')) + "va
        color = "validation")) + #geom_line(mapping=aes(ntree, Accuracy_train, color='train')) + +
        color = "validation")) + #geom_line(mapping=aes(ntree, Accuracy_train, color='train')) + #ge
        color = "validation")) + #geom_line(mapping=aes(ntree, Accuracy_train, color='train')) + Acc
        color = "validation")) + #geom_line(mapping=aes(ntree, Accuracy_train, color='train')) + col
        color = "validation")) + #geom_line(mapping=aes(ntree, Accuracy_train, color='train')) + +
    theme_minimal() + labs(y = "accuracy") + theme(axis.title.x = element_blank(),
        axis.ticks.x = element_blank())

    ce_plot <- ggplot(data.frame(ntree = ntree, Ce_valid = ce_valid, Ce_train = ce_train)) +
        geom_line(mapping = aes(ntree, Ce_valid, color = "validation")) + #geom_line(mapping=aes(ntr
        geom_line(mapping = aes(ntree, Ce_valid, color = "validation")) + #geom_line(mapping=aes(ntr
        geom_line(mapping = aes(ntree, Ce_valid, color = "validation")) + #geom_line(mapping=aes(ntr
        geom_line(mapping = aes(ntree, Ce_valid, color = "validation")) + #geom_line(mapping=aes(ntr
        geom_line(mapping = aes(ntree, Ce_valid, color = "validation")) + #geom_line(mapping=aes(ntr
        geom_line(mapping = aes(ntree, Ce_valid, color = "validation")) + #geom_line(mapping=aes(ntr
        geom_line(mapping = aes(ntree, Ce_valid, color = "validation")) + #geom_line(mapping=aes(ntr
        geom_line(mapping = aes(ntree, Ce_valid, color = "validation")) + #geom_line(mapping=aes(ntr
        geom_line(mapping = aes(ntree, Ce_valid, color = "validation")) + #geom_line(mapping=aes(ntr
        geom_line(mapping = aes(ntree, Ce_valid, color = "validation")) + #geom_line(mapping=aes(ntr
        geom_line(mapping = aes(ntree, Ce_valid, color = "validation")) + #geom_line(mapping=aes(ntr
        geom_line(mapping = aes(ntree, Ce_valid, color = "validation")) + #geom_line(mapping=aes(ntr
    theme_minimal() + labs(y = "cross entropy loss")

    return(grid.arrange(miscl_plot, acc_plot, ce_plot, nrow = 3, ncol = 1))
}
set.seed(12345)
validation_ntree()
validation_nodesize <- function(nrange = seq(1, 500, 50)) {
    accuracy <- c()
    ce <- c()
    nodesize <- c()
    miscl_valid <- c()
    miscl_oob <- c()
    for (n in nrange) {
        forest_model <- randomForest(music_genre ~ ., train, nodesize = n, ntree = 80)
        predictions <- predict(forest_model, valid)
        perc_pred <- predict(forest_model, valid, type = "prob")
        confMat <- confusionMatrix(predictions, valid$music_genre)
        accuracy <- append(accuracy, confMat$overall[1])
        ce <- append(ce, cross_entropy(perc_pred, valid$music_genre))
        nodesize <- append(nodesize, n)
        miscl_valid <- append(miscl_valid, misclass(valid$music_genre, predictions))
        miscl_oob <- append(miscl_oob, misclass(train$music_genre, forest_model$predicted))
    }
```

```
    miscl_plot <- ggplot(data.frame(nodesize = nodesize, valid.misscl.rate = miscl_valid,
        oob.misscl.rate = miscl_oob)) + geom_line(mapping = aes(nodesize, valid.misscl.rate,
        color = "validation")) + geom_line(mapping = aes(nodesize, oob.misscl.rate,
        color = "oob")) + theme_minimal() + labs(y = "missclass. rate") + theme(axis.title.x = eleme
        axis.ticks.x = element_blank())
    acc_plot <- ggplot(data.frame(nodesize = nodesize, Accuracy = accuracy), aes(nodesize,
        Accuracy)) + geom_line(aes(color = "validation")) + theme_minimal() + theme(axis.title.x = e
        axis.ticks.x = element_blank())
    ce_plot <- ggplot(data.frame(nodesize = nodesize, cross_entropy = ce), aes(nodesize,
        cross_entropy)) + geom_line(aes(color = "validation")) + theme_minimal()
    grid.arrange(miscl_plot, acc_plot, ce_plot, nrow = 3, ncol = 1)
}
set.seed(12345)
validation_nodesize()
# rule of thumb for mtry = sqrt(n) where n is the number of features used for
# prediction we use 13 features so mtry is about 3
validation_mtry <- function(nrange = 2:10) {
    accuracy_valid <- c()
    accuracy_train <- c()
    ce_valid <- c()
    ce_train <- c()
    mtry <- c()
    miscl_valid <- c()
    miscl_oob <- c()
    for (n in nrange) {
        forest_model <- randomForest(music_genre ~ ., train, ntree = 80, nodesize = 1,
            mtry = n)

        predictions <- predict(forest_model, valid)
        confMat <- confusionMatrix(predictions, valid$music_genre)
        perc_pred <- predict(forest_model, valid, type = "prob")
        accuracy_valid <- append(accuracy_valid, confMat$overall[1])
        ce_valid <- append(ce_valid, cross_entropy(perc_pred, valid$music_genre))

        miscl_valid <- append(miscl_valid, misclass(valid$music_genre, predictions))
        miscl_oob <- append(miscl_oob, misclass(train$music_genre, forest_model$predicted))

        predictions <- predict(forest_model, train)
        perc_pred <- predict(forest_model, train, type = "prob")
        confMat <- confusionMatrix(predictions, train$music_genre)
        accuracy_train <- append(accuracy_train, confMat$overall[1])
        ce_train <- append(ce_train, cross_entropy(perc_pred, train$music_genre))

        mtry <- append(mtry, n)
    }
    miscl_plot <- ggplot(data.frame(mtry = mtry, valid.misscl.rate = miscl_valid,
        oob.misscl.rate = miscl_oob)) + geom_line(mapping = aes(mtry, valid.misscl.rate,
        color = "validation")) + geom_line(mapping = aes(mtry, oob.misscl.rate, color = "oob")) +
        theme_minimal() + labs(y = "missclass. rate") + theme(axis.title.x = element_blank(),
        axis.ticks.x = element_blank())
    acc_plot <- ggplot(data.frame(mtry = mtry, Accuracy_valid = accuracy_valid, Accuracy_train = acc
        geom_line(mapping = aes(mtry, Accuracy_valid, color = "validation")) + theme_minimal() +
        labs(y = "accuracy") + theme(axis.title.x = element_blank(), axis.ticks.x = element_blank())
    ce_plot <- ggplot(data.frame(mtry = mtry, Cross_entropy_valid = ce_valid, Cross_entropy_train =
        geom_line(mapping = aes(mtry, Cross_entropy_valid, color = "validation")) +
        theme_minimal() + labs(y = "cross entropy loss")
    return(grid.arrange(miscl_plot, acc_plot, ce_plot, nrow = 3, ncol = 1))
```

```
}
set.seed(12345)
validation_mtry()
validation_grid <- function(mtry = 2:4, nodesize = c(1, 25, 50, 75, 100), ntree = 80) {
    accuracy_valid <- c()
    accuracy_train <- c()
    ce_valid <- c()
    ce_train <- c()
    params <- matrix(nrow = length(mtry) * length(nodesize), ncol = 2)
    colnames(params) <- c("mtry", "nodesize")
    miscl_valid <- c()
    miscl_oob <- c()
    counter <- 1
    for (m in mtry) {
        for (n in nodesize) {
            print(paste("counter =", counter))
            print(paste("mtry =", m))
            print(paste("nodesize =", n))

            forest_model <- randomForest(music_genre ~ ., train, ntree = 80, nodesize = n,
                mtry = m)

            predictions <- predict(forest_model, valid)
            confMat <- confusionMatrix(predictions, valid$music_genre)
            perc_pred <- predict(forest_model, valid, type = "prob")
            accuracy_valid <- append(accuracy_valid, confMat$overall[1])
            ce_valid <- append(ce_valid, cross_entropy(perc_pred, valid$music_genre))

            miscl_valid <- append(miscl_valid, misclass(valid$music_genre, predictions))
            miscl_oob <- append(miscl_oob, misclass(train$music_genre, forest_model$predicted))

            predictions <- predict(forest_model, train)
            perc_pred <- predict(forest_model, train, type = "prob")
            confMat <- confusionMatrix(predictions, train$music_genre)
            accuracy_train <- append(accuracy_train, confMat$overall[1])
            ce_train <- append(ce_train, cross_entropy(perc_pred, train$music_genre))

            params[counter, 1] <- m
            params[counter, 2] <- n
            counter <- counter + 1
        }
    }
    res <- data.frame(params, miscl_valid, miscl_oob, accuracy_train, accuracy_valid,
        ce_train, ce_valid)
    return(res)
}
set.seed(12345)
grid_res <- validation_grid()
knitr::kable(subset(grid_res, select = c("mtry", "nodesize", "miscl_valid", "miscl_oob",
    "accuracy_valid", "ce_valid")))
set.seed(12345)
forest_model <- randomForest(music_genre ~ ., train, nodesize = 1, ntree = 80, mtry = 2)
predictions <- predict(forest_model, test)
confMat <- confusionMatrix(predictions, test$music_genre)
perc <- predict(forest_model, test, type = "prob")
accuracy <- confMat$overall[1]
ce <- cross_entropy(perc, test$music_genre)
```

```r
miscl <- misclass(test$music_genre, predictions)

plain_forest_model <- randomForest(music_genre ~ ., train)
plain_predictions <- predict(plain_forest_model, test)
plain_confMat <- confusionMatrix(plain_predictions, test$music_genre)
plain_perc <- predict(plain_forest_model, test, type = "prob")
plain_accuracy <- plain_confMat$overall[1]
plain_ce <- cross_entropy(plain_perc, test$music_genre)
plain_miscl <- misclass(test$music_genre, plain_predictions)

metrics <- data.frame(misclass_rate = round(c(plain_miscl, miscl), 3), accuracy = round(c(plain_accu
    accuracy), 3), cross_entropy_loss = round(c(plain_ce, ce), 3))

rownames(metrics) <- c("plain", "proposed")
final_accuracy_rf <- metrics$accuracy[2]

knitr::kable(metrics)
library(h2o)
h2o.init(nthreads = -1)

# reading the data
data <- read.csv("music_genre.csv")
data <- data[, -c(1, 2, 3, 16)]  # remove ID, Artist, Song Title and Date

# remove NAs or missing values
data <- data[-which(is.na(data$popularity)), ]
data <- data[-which(is.na(as.numeric(data$tempo))), ]
data <- data[-which(data$duration_ms == -1), ]
data$tempo <- as.numeric(data$tempo)

data <- as.data.frame(unclass(data), stringsAsFactors = TRUE)

# Train-Test-Validation Split
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n * 0.4))
train = data[id, ]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n * 0.4))
valid = data[id2, ]
id3 = setdiff(id1, id2)
test = data[id3, ]

data[c(1:6, 8, 9, 11:13)] <- scale(data[c(1:6, 8, 9, 11:13)])

# preparing the data for the h2o package
train_h2o <- as.h2o(train)

valid_h2o <- as.h2o(valid)

test_h2o <- as.h2o(test)
set.seed(12345)

nn1 <- h2o.deeplearning(y = "music_genre", training_frame = train_h2o, validation_frame = valid_h2o,
    activation = "Tanh", epochs = 150, hidden = c(30, 30, 30), nfolds = 5, stopping_rounds = 20,
    stopping_metric = "misclassification", seed = 12345, reproducible = TRUE)
```

```
set.seed(12345)

nn2 <- h2o.deeplearning(y = "music_genre", training_frame = train_h2o, validation_frame = valid_h2o,
    model_id = "nn2", activation = "Rectifier", epochs = 150, hidden = c(30, 30,
        30), nfolds = 5, stopping_rounds = 20, stopping_metric = "misclassification",
    seed = 12345, reproducible = TRUE)
set.seed(12345)

nn3 <- h2o.deeplearning(y = "music_genre", training_frame = train_h2o, validation_frame = valid_h2o,
    model_id = "nn3", activation = "Maxout", epochs = 150, hidden = c(30, 30, 30),
    nfolds = 5, stopping_rounds = 20, stopping_metric = "misclassification", seed = 12345,
    reproducible = TRUE)
plot(nn1)
plot(nn2)
plot(nn3)
set.seed(12345)

cross_entropy <- function(p, actual) {
    x <- 0
    for (i in 1:length(actual)) {
        if (p[i, which(colnames(p) == actual[i])] == 0)
            p[i, which(colnames(p) == actual[i])] <- 10^(-15)
        x <- x + (log(p[i, which(colnames(p) == actual[i])]))
    }
    return(-x)
}

perf_valid1 <- h2o.performance(nn1, newdata = valid_h2o)
cm_valid1 <- as.data.frame(perf_valid1@metrics$cm$table)
cm_valid1 <- cm_valid1[, -c(11, 12)]
cm_valid1 <- cm_valid1[-11, ]
cm_valid1 <- sapply(cm_valid1, as.numeric)
accuracy_valid1 <- sum(diag(cm_valid1))/sum(cm_valid1)
misclass_rate_valid1 <- 1 - accuracy_valid1

probs_valid1 <- as.data.frame(h2o.predict(nn1, valid_h2o, type = "prob")[, -1])
colnames(probs_valid1) <- c("Alternative", "Anime", "Blues", "Classical", "Country",
    "Electronic", "Hip-Hop", "Jazz", "Rap", "Rock")

ce_valid1 <- cross_entropy(probs_valid1, valid$music_genre)
set.seed(12345)

perf_valid2 <- h2o.performance(nn2, newdata = valid_h2o)
cm_valid2 <- as.data.frame(perf_valid2@metrics$cm$table)
cm_valid2 <- cm_valid2[, -c(11, 12)]
cm_valid2 <- cm_valid2[-11, ]
cm_valid2 <- sapply(cm_valid2, as.numeric)
accuracy_valid2 <- sum(diag(cm_valid2))/sum(cm_valid2)
misclass_rate_valid2 <- 1 - accuracy_valid2

probs_valid2 <- as.data.frame(h2o.predict(nn2, valid_h2o, type = "prob")[, -1])
colnames(probs_valid2) <- c("Alternative", "Anime", "Blues", "Classical", "Country",
    "Electronic", "Hip-Hop", "Jazz", "Rap", "Rock")

ce_valid2 <- cross_entropy(probs_valid2, valid$music_genre)
set.seed(12345)
```

```r
perf_valid3 <- h2o.performance(nn3, newdata = valid_h2o)
cm_valid3 <- as.data.frame(perf_valid3@metrics$cm$table)
cm_valid3 <- cm_valid3[, -c(11, 12)]
cm_valid3 <- cm_valid3[-11, ]
cm_valid3 <- sapply(cm_valid3, as.numeric)
accuracy_valid3 <- sum(diag(cm_valid3))/sum(cm_valid3)
misclass_rate_valid3 <- 1 - accuracy_valid3

probs_valid3 <- as.data.frame(h2o.predict(nn3, valid_h2o, type = "prob")[, -1])
colnames(probs_valid3) <- c("Alternative", "Anime", "Blues", "Classical", "Country",
    "Electronic", "Hip-Hop", "Jazz", "Rap", "Rock")

ce_valid3 <- cross_entropy(probs_valid3, valid$music_genre)
df_valid <- data.frame(Tanh = round(c(accuracy_valid1, ce_valid1), 3), ReLU = round(c(accuracy_valid
    ce_valid2), 3), Maxout = round(c(accuracy_valid3, ce_valid3), 3))

rownames(df_valid) <- c("Accuracy", "Cross Entropy")

knitr::kable(df_valid, caption = "Validation Data Metrics")
set.seed(12345)

cm1 <- h2o.confusionMatrix(nn1)
cm1$Error <- round(cm1$Error, 3)

knitr::kable(cm1, caption = "Confusion Matrix of the Neural Network using Tanh")
set.seed(12345)

cm2 <- h2o.confusionMatrix(nn2)
cm2$Error <- round(cm2$Error, 3)

knitr::kable(cm2, caption = "Confusion Matrix of the Neural Network using ReLU")
set.seed(12345)

cm3 <- h2o.confusionMatrix(nn3)
cm3$Error <- round(cm3$Error, 3)

knitr::kable(cm3, caption = "Confusion Matrix of the Neural Network using Maxout")
hyper_params <- list(activation = c("Rectifier", "Tanh", "Maxout"), l1 = c(0, 1e-05,
    1e-04, 0.001, 0.01), l2 = c(0, 1e-05, 1e-04, 0.001, 0.01))

search_criteria <- list(strategy = "RandomDiscrete", max_runtime_secs = 600, stopping_metric = "misc
    stopping_rounds = 20)

# If the below error is received: Illegal argument: training_frame of function:
# grid: Cannot append new models to a grid with different training # input
# Shutdown the h2o with the command h2o.shutdown & run everything from the
# start.

set.seed(12345)

nn_grid <- h2o.grid("deeplearning", y = "music_genre", grid_id = "nn_grid", training_frame = train_h
    validation_frame = valid_h2o, hidden = c(30, 30, 30), epochs = 150, score_interval = 5,
    hyper_params = hyper_params, search_criteria = search_criteria, seed = 12345,
    reproducible = TRUE)
set.seed(12345)
nn_gridperf <- h2o.getGrid(grid_id = "nn_grid", sort_by = "accuracy", decreasing = TRUE)
```

```
knitr::kable(nn_gridperf@summary_table)
set.seed(12345)

best_nn_model_id <- nn_gridperf@model_ids[[1]]
best_nn <- h2o.getModel(best_nn_model_id)

best_nn_perf_test <- h2o.performance(model = best_nn, newdata = test_h2o)

cm_tuned_test <- as.data.frame(best_nn_perf_test@metrics$cm$table)
cm_tuned_test <- cm_tuned_test[, -c(11, 12)]
cm_tuned_test <- cm_tuned_test[-11, ]
cm_tuned_test <- sapply(cm_tuned_test, as.numeric)
accuracy_tuned_test <- sum(diag(cm_tuned_test))/sum(cm_tuned_test)
misclass_rate_tuned_test <- 1 - accuracy_tuned_test

probs_tuned_test <- as.data.frame(h2o.predict(best_nn, test_h2o, type = "prob")[,
    -1])
colnames(probs_tuned_test) <- c("Alternative", "Anime", "Blues", "Classical", "Country",
    "Electronic", "Hip-Hop", "Jazz", "Rap", "Rock")

ce_tuned_test <- cross_entropy(probs_tuned_test, test$music_genre)

nn4 <- h2o.deeplearning(y = "music_genre", training_frame = train_h2o, validation_frame = valid_h2o,
    model_id = "nn4", activation = nn_gridperf@summary_table[["activation"]][1],
    epochs = 150, hidden = c(30, 30, 30), nfolds = 5, stopping_rounds = 20, stopping_metric = "miscl
    seed = 12345, reproducible = TRUE)

perf_test4 <- h2o.performance(nn4, newdata = test_h2o)

cm_test4 <- as.data.frame(perf_test4@metrics$cm$table)
cm_test4 <- cm_test4[, -c(11, 12)]
cm_test4 <- cm_test4[-11, ]
cm_test4 <- sapply(cm_test4, as.numeric)
accuracy_test4 <- sum(diag(cm_test4))/sum(cm_test4)
misclass_rate_test4 <- 1 - accuracy_test4

probs_test4 <- as.data.frame(h2o.predict(nn4, test_h2o, type = "prob")[, -1])
colnames(probs_test4) <- c("Alternative", "Anime", "Blues", "Classical", "Country",
    "Electronic", "Hip-Hop", "Jazz", "Rap", "Rock")

ce_test4 <- cross_entropy(probs_test4, test$music_genre)

df_test <- data.frame(Accuracy = round(c(accuracy_tuned_test, accuracy_test4), 3),
    `Cross Entropy` = round(c(ce_tuned_test, ce_test4), 3))

rownames(df_test) <- c("Tuned Model", "Plain Model")

knitr::kable(df_test, caption = "Test Data Metrics")
metrics <- data.frame(accuracy = round(c(final_accuracy_rf, accuracy_tuned_test),
    3))

rownames(metrics) <- c("Random Forest", "Neural Network")

knitr::kable(metrics)
```