Music Genre Classification

Linxia Li

University of Oxford

MSc Statistical Science

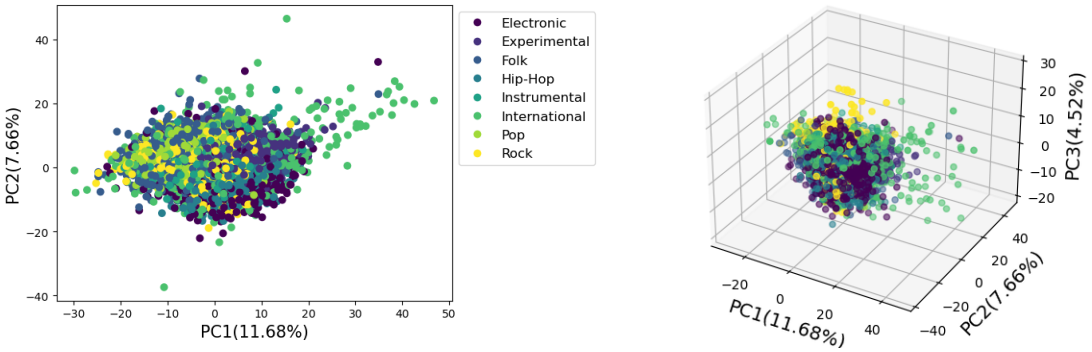
20 March, 2024

1 Introduction

Consisting of $6,000$ song tracks, the training data contains 518 features for each song, which are real-valued and correspond to time-series summary statistics of various musical features. The response variable genres include *Electronic, Experimental, Folk, Hip-Hop, Instrumental, International, Pop, and Rock.* Our data is free of any missing values. Using this data, we aim to investigate a method that can more accurately predict the musical genres of another $2,000$ songs based on their pre-calculated features.

1.1 Data Exploration

To better visualise our data, we first perform principal component analysis to map the scaled training data $(6,000)$ from its original dimension (518) to the 2-D and 3-D principal component spaces. PCA allows us to gain insights into the overall pattern of the data and helps us pre-determine some machine learning methods to use for performing multi-class classification.



(a) Two PCs (19.34% Variance)

(b) Three PCs (23.86% Variance)

Figure 1: Principal Component Analysis

Based on the two scatter-plots in Figure 1, we observe that the boundaries separating each genre are non-linear and not distinct enough for simple models like LDA and QDA to handle. Observe that two PCs can only explain 19.34% of the total variance of the data, and three PCs can only attain 23.86%. Moreover, the cumulative variance explained does not reach 70% until 144 PCs are in use. This result indicates that

- There exists non-linear relationships among features, which principal components created by simple linear combinations are unable to recognise;

- Since dimension reduction fails to retain enough total variation, doing so before model fitting may not be an effective approach for our task.

Therefore, models that can capture complex relationships and handle high dimensions and complicated patterns, such as Random Forest, XGBoost, and Neural Network, are worth exploring.

1.2 Data Preparation

Given that our dataset comprises 8 distinct music genres, it is necessary to transform y_{train} into factors from 0 to 7, which allows the models to treat our problem as a multi-class classification task.

To allow for hyperparameter tuning, model comparison, overfitting detection, and performance evaluation on unseen data, we split our original training dataset into two subsets (70%/30%):

- $X_{training}$, $y_{training}$: 4200 observations for training
- $X_{testing}$, $y_{testing}$: 1800 observations for testing

To address sensitivity to feature scales in certain methods and enhance model performance uniformly, we standardised two subsets of input features $X_{training}$ and $X_{testing}$ to have a mean of 0 and a standard deviation of 1.

2 Methodology

In this section, we will thoroughly present each model we tried. We will introduce our motivations for choosing these methods, show the results of optimal hyperparameter values with their corresponding average cross-validation scores, and discuss their advantages and disadvantages.

2.1 Hyperparameter Tuning and Cross-Validation

For hyperparameter tuning, we first set the ranges of hyperparameters in each model. We then use *GridSearchCV* to identify the optimal hyperparameters for each model that maximise its average cross-validation score. By fitting *GridSearchCV* to the scaled training data, it iterates over all possible combinations of these hyperparameters in the specified ranges, evaluating model performance across the 5-fold cross-validation, and returning the best hyperparameter values.

2.2 Logistic Regression

Our project aims at predicting musical genres of songs from pre-computed features. We choose Logistic Regression as the classification model since it is well-suited for multi-class classification problems.

2.2.1 Model and Results

After conducting hyperparameter tuning, the optimal value of the regularisation parameter **C** is 0.011, with the average cross-validation score being 0.575.

2.2.2 Advantages and Disadvantages:

Advantages: Given the high dimensionality of our feature space, the regularisation parameter helps prevent overfitting. Additionally, model coefficients can reveal the importance of each musical feature in genre prediction, making Logistic Regression a highly interpretable method compared to more complex models. It also requires less computational power and takes less time to train.

Disdvantages: Considering the non-linear relationship between features in our data, Logistic Regression's reliance on linear decision boundaries leads to lower predictive performance. Also, Logistic Regression's performance depends on appropriate feature selection, requiring careful pre-processing and feature engineering to achieve optimal results. Furthermore, while Logistic Regression can be extended to multi-class classification, it inherently models binary outcomes. For multi-class problems, it relies on strategies like one-vs-rest or one-vs-one, which might not be as efficient or effective as native multi-class algorithms.

2.3 Support Vector Machine

We use SVM on our high-dimensional data because it relies on support vectors and margin maximisation process, rather than the dimension of feature space. Considering the underlying non-linear relationship between features, SVM can efficiently handle this by using specific kernel functions like RBF (Radial Basis Function) or polynomial.

2.3.1 Model and Results

After performing the grid search for hyperparameter tuning, we find the optimal type of **kernel** is *"rbf"*, the optimal value of regularisation parameter **C** is 5.69, and the optimal value of **gamma** is *"auto"* (1/number of features). With these hyperparameter values, the best average cross-validation score is 0.606.

2.3.2 Advantages and Disadvantages:

Advantages: The kernel trick enables SVM to effectively handle non-linearly separable data by mapping the inputs into high-dimensional feature spaces, where a segregating hyperplane can be identified. Additionally, SVM mitigates overfitting by including a regularisation term. By finding the segregating boundary primarily using support vectors rather than the entirety of the training dataset, SVM benefits memory usage and computes decision boundary functions easily.

Disadvantages: Choosing the RBF as a kernel in SVM introduces challenges in model interpretability, which arises from the complexity of the decision boundaries in transformed feature spaces. Also, as SVM is based on learning a single hyperplane for binary classification, we need to apply one-vs-one or one-vs-the-rest strategy for multi-class classification. This approach increases computational cost since it requires training multiple binary classifiers.

2.4 Random Forest

Random Forest is an ensemble machine learning method built upon a collection of bagged decision trees, where bootstrap sampling is applied to training data, and features are randomly sampled for each split.

2.4.1 Model and Results

After conducting hyperparameter tuning, the optimal **number of trees** is 840, **minimum number of samples to split** is 4, **minimum number of samples in a leaf** is 2, and **maximum depth of a tree** is 18. With these hyperparameters, we obtain the highest average cross-validation score of 0.564.

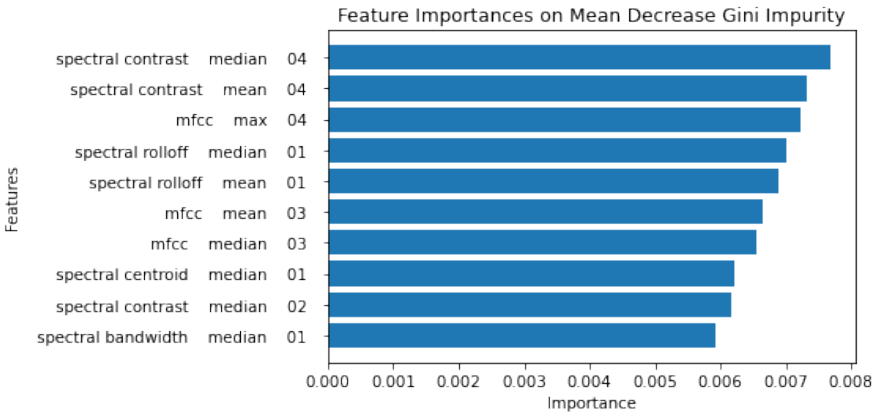


Figure 2: Top 10 Feature Importance

In Figure 2, after training our classifier, we observe the top ten important features included in the decision-making process. The importances are determined by mean decrease in Gini Impurity, which quantifies the average reduction in impurity after introducing a feature to the model.

All of the top three important features are information associated with chroma 4, and four of the rest of six important features contain information about chroma 1. Two mfcc statistics on chroma 3 are also important.

2.4.2 Advantages and Disadvantages:

Advantages: Random Forest is known for its high prediction accuracy and its ability to learn complex, non-linear relationships between a large number of features and the response variable. It does not require us to perform feature selection or dimension reduction prior to model fitting. Moreover, such a tree-based model does not require scaling of variables since it relies on majority-voting to make decisions. Through bagging, Random Forest is also less prone to overfitting as it averages over the predictions given by uncorrelated trees, which helps reduce the variance of the model.

Disadvantages: Random Forest is computationally intensive, which is a trade-off for its accuracy. As we increase the number of trees to improve model performance, it simultaneously results in a higher computational cost. Especially with a large dataset, it requires higher computational power to train, so in practice, it takes longer time to implement compared to other machine learning methods. Furthermore, the interpretability of a forest is significantly low since we have no control over the information contained in each decision tree. Although we can extract feature importance to study the pattern, the split strategy is unknown.

2.5 Extreme Gradient Boosting

Extreme gradient boosting (XGBoost) utilises decision trees as base learners and generates optimisation model by using regularisation, tree induction, and stopping criteria. As our dataset has a high dimension, XGBoost will automatically determine which features are more important and make more accurate predictions.

2.5.1 Model and Results

After conducting hyperparameter tuning, the optimal hyperparameters are:
the optimal **number of trees** is 520, **maximum depth of a tree** is 6, **step size shrinkage** is 0.1, **minimum sum of instance weight needed in a child** is 0, **percentage of rows used per tree building** is 0.6, and **a family of**

parameters for subsampling of columns is 0.9. With these hyperparameters, we obtain the best average cross-validation score being 0.616.

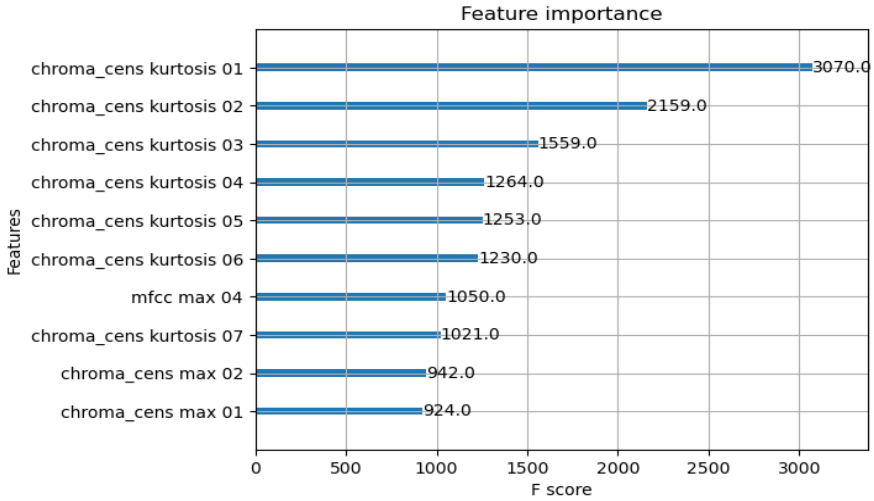


Figure 3: Top 10 Feature Importance

The feature importance in XGBoost is weight, which is the number of times a variable is selected for splitting. The more an attribute is used to make key decisions with decision trees, the higher its relative importance. From Figure 3, chroma_cens kurtosis show a large contribution to the overall model performance, especially chroma_cens kurtosis 01.

2.5.2 Advantages and Disadvantages:

Advantages: XGBoost could capture non-linear, complex relationships between musical features and genres. Its hyperparameters control aspects like tree construction, regularisation, and boosting process, providing flexibility to fine-tune the model. If our primary concern is model performance, XGBoost is effective and valuable. It provides the best result compared to our previous models. Its built-in regularisation parameter **gamma** controls model complexity and reduces overfitting.

Disadvantages: Finding the optimal combination of hyperparameters is challenging and leads to a high computational cost. Additionally, individual trees can be interpreted easily, but with XGBoost, it is hard to interpret and capture the dynamics of weighting trees.

2.6 Neural Network

Neural Network is constructed by artificial neurons stored in fully connected hidden layers. It starts with an input layer of our training data passing into the first hidden layer, where hidden neurons each generate a linear combination of all 518 features with a weight and a bias term unique to each neuron. That linear function is then fed into a non-linear activation function to decide whether the neuron is activated and to be passed onto the next hidden layer. Since we have 8 distinct genres, the output layer has 8 neurons.

2.6.1 Model and Results

In tensorflow, we specify sparse categorical cross-entropy as the loss function since our genre class has 8 different labels. After conducting hyperparameter tuning, the optimal hyperparameters are: 2 **hidden layers**, with first hidden layer having 450 **hidden neurons** and second hidden layer having 32 hidden neurons, **batch size** of 100, **optimiser** Adam, and 20 **epochs**. This combination gives the highest average cross-validation score of 0.596.

2.6.2 Advantages and Disadvantages:

Advantages: Neural Network can handle data with high dimensionality. It learns features on its own and does not require feature engineering before training. Deep learning using neural network has an eminent reputation in image/audio classification earned by its computational power on large, complex dataset.

Disadvantages: Given its "black box" nature, the interpretability of Neural Network is extremely low. We do not know what the model learns from features or why it produces the output. The hyperparameter tuning process also requires intensive computations.

Furthermore, to optimise the power of neural network, it requires a larger amount of data than other techniques do. Although our task is "music genre classification", an area in which neural network is known for achieving superior results, our dataset is summary statistics of features containing discrete information. This explains why models like XGBoost outperform neural network because the size and type of our data are less suitable for neural network to maximise its power.

3 Final Classifier: Soft Voting

Voting is a weighted average ensemble method that combines multiple models and balances out their individual weaknesses. Each model in the ensemble brings its unique approach to capture patterns in data. For instance, Logistic Regression can excel at identifying linear relationships, while SVM and tree-based models are better at capturing non-linear interactions. Neural Network can uncover complex hierarchical patterns. Table 1 shows the test accuracies of models we have tried so far, and we hope to improve the highest accuracy with voting.

Method	**Average CV Score**	**Test Accuracy**
Logistic Regression	0.575	0.561
Support Vector Machine	0.606	0.602
Neural Network	0.596	0.576
Random Forest	0.564	0.559
Extreme Gradient Boosting	0.616	0.606

Table 1: Average Cross-Validation (CV) Score and Test Accuracy of each method

Soft voting computes the average probability of each observation belonging to each class given by base models. It then determines the output class by selecting the one with the highest weighted average probability. We choose Soft Voting since all of the five models are capable of estimating probabilities for each class.

3.1 Base Model Selection

We assign equal weights to the individual model used in our Soft Voting Classifier and construct a Voting Classifier that includes all 5 models to assess the ensemble performance using the testing set. The test accuracy is 0.614.

As fewer models can reduce the computational cost, we would like to see if they can perform better. We remove one of the 5 models and combine the remaining 4 models for the ensemble, and re-evaluate its performance. The test accuracies are 0.609, 0.609, 0.613, 0.599, and 0.613 when we remove Logistic Regression, SVM, Random Forest, XGBoost, and Neural Network, respectively.

We combine 3 of the 5 models and 2 of the 5 models for ensemble and compare all model performances and generalisation ability. Finally, we obtain the final Voting Classifier using SVM and XGBoost as based models, whose test accuracy is 0.616, achieving the best model performance.

3.2 Generalisation Error Under 0-1 Loss

Generalisation error indicates how a model performs on unseen data compared to its performance on its training data. We estimate this by extracting and comparing the gap between the training and test accuracy of our Voting classifier. Under 0-1 loss, the training accuracy is 1, which means that the model learns everything in the training set (4200).

The test accuracy on the hold-out test set (1800) is 0.616, indicating that the model still has a high generalisation error of 38.4% under 0-1 loss, although it already attains the highest test accuracy among all methods we have tried. The model possibly learns the noise and outliers in the training data to perfectly capture everything including information that is not generalised to new data.

3.3 Advantages and Disadvantages:

Advantages: Soft Voting typically achieves higher accuracy than hard voting and individual classifiers since it incorporates the probability estimates of each class and takes into account the confidence of each classifier's prediction. It is also more robust to noise in the data which will increase the total probability of the correct class. Based on our voting classifier with models XGBoost and SVM, the diversity of their underlying algorithms and parameter settings will help capture different aspects of the data and improve ensemble robustness.

Disadvantages: Soft Voting has a relatively high computational cost since it calculates and aggregates probability estimates for each class across all classifiers. Additionally, understanding how the ensemble arrives at a particular decision can be more challenging with soft voting because of the aggregation of probabilistic predictions, which makes it hard to interpret.

4 Conclusions

In this practical, we studied and implemented traditional and advanced machine learning techniques for training a music genre classifier. We split 30% of our training data as our hold-out testing set to be used to calculate generalisation errors at the very end. In the training process, we tuned hyperparameters based on cross-validation scores. For each method, we discussed its pros and cons in terms of functionality, interpretability, and computational cost.

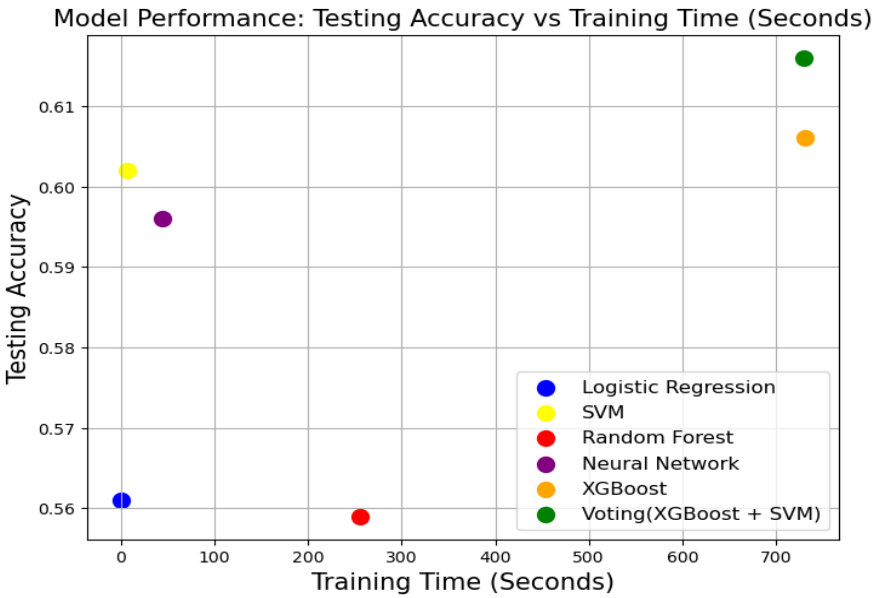


Figure 4: Model Comparison by Computational Cost and Generalisation Error

We can compare model performance by plotting their corresponding test accuracy versus training time in seconds. In Figure 4, observe that:

- Voting classifier exhibits a significant trade-off between its computational cost and accuracy;
- When switching from SVM to XGBoost, there is a huge sacrifice in training time for a very small amount of improvement in model performance;
- SVM outperforms Neural Network for it requires less training time and still achieves better accuracy.

Our final classifier is Soft Voting based on XGBoost and SVM, achieving an estimated 0-1 generalisation error of 38.4% (accuracy of 61.6%). Reflecting on our model training process, we could have improved the model accuracy by assigning different weights to each base model. However, this process requires high computational power, and we are not sure if its trade-off for accuracy is worth the cost. All in all, we expect our output predictions to have a similar accuracy score.

A Appendix: Coding

Attached Python Code

```
############################
#### Import libraries #####
############################

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split,
   cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
# SVM
from sklearn import svm
from sklearn.svm import SVC
# XGBoost
import xgboost as xgb
from xgboost import XGBClassifier
# Voting
from sklearn.ensemble import VotingClassifier

############################
#### Data Preparation #####
############################

# Read in X_train, Y_train and X_test data
X_train = pd.read_csv("X_train.csv", index_col = 0,
          header=[0, 1, 2])
y_train = pd.read_csv("y_train.csv", index_col = 0).
                          squeeze("columns").to_numpy()
X_test = pd.read_csv("X_test.csv", index_col = 0,
         header=[0, 1, 2])

# Factorize the data
factor = pd.factorize(y_train)
y_train = factor[0]

# Split the dataset into training and testing sets
X_training, X_testing, y_training, y_testing =
   train_test_split(X_train, y_train,
    test_size=0.3, random_state=42)
```

```
# Scale the data
scalar = StandardScaler()
X_training_scaled = scalar.fit_transform(X_training)
X_testing_scaled= scalar.fit_transform(X_testing)
X_test_scaled = scalar.fit_transform(X_test)

############################
#### Final Classifier #####
############################

xgboost = XGBClassifier(n_estimators = 520,
                        max_depth = 6,
                        learning_rate = 0.1,
                        min_child_weight = 0,
                        subsample = 0.6,
                        colsample_bytree = 0.9,
                        objective='multi:softprob',
                        eval_metric='mlogloss',
                        use_label_encoder=False,
                        random_state=42)

svm = SVC(kernel = "rbf",C = 5.69,
          gamma = "auto",
          random_state=42, probability=True)

# Voting Classifier
Classifier=VotingClassifier(estimators=[("xgb",xgboost),("svm
   ",svm)], voting="soft")

# Five-fold cross validation
voting_score = cross_val_score(Classifier, X_training_scaled,
    y_training, cv=5, scoring="accuracy")

# Evaluate Model Performance
Classifier.fit(X_training_scaled, y_training)
y_pred = Classifier.predict(X_testing_scaled)
accuracy = accuracy_score(y_testing, y_pred)
```

```
################################
#### Prediction on X_test #####
################################

# Make prediction
y_pred = Classifier.predict(X_test_scaled)

# Change factor to our categorical classes
cat = factor[1]
y_pred = cat[y_pred]

# Export the predictions on the test data in csv format
prediction = pd.DataFrame(y_pred, columns=["Genre"])
prediction.index.name="Id"
prediction.to_csv("myprediction.csv")
```

B Appendix: Hyperparameter Descriptions

This appendix provides an overview of the hyperparameters for the five machine learning models that have been tuned. For each model, we briefly describe the role of various hyperparameters in the model's training process.

- random_state=42: ensure that our experiments are reproducible.

Logistics Regression:

- C: Inverse of regularisation strength. Smaller values of C specify stronger regularisation.

SVM:

- *kernel*: The kernel type to be used in the algorithm.
- C: Inverse of regularisation strength. Smaller values of C specify stronger regularisation.
- *gamma*: Kernel coefficient (for "rbf", "poly", and "sigmoid" kernels).
- *degree*: Degree of the polynomial kernel function (only for "poly" kernels).
- *coef0*: Independent term in kernel function (only significant in "poly" and "sigmoid" kernels).

Random Forests:

- $n\_estimators$: Number of trees in the forest. Increasing $n\_estimators$ results in better model performance but higher computational cost.
- max_depth: The longest path between the root node and a leaf node. Maximum depth of each tree. Increasing max_depth increases the model performance to a point where it overfits the training set.
- $min\_samples\_split$: The minimum number of samples required to split an internal node. Can be used to control the size of the trees and prevent overfitting.
- $min\_samples\_leaf$: The minimum number of samples required to be at a leaf node. Increasing $min\_samples\_leaf$ reduces splitting and thus helps prevent overfitting.

XGBoost:

- *n_estimators*: Number of gradient boosted trees.
- *max_depth*: Maximum depth of a tree.
- *learning_rate*: Step size shrinkage used to prevent overfitting.
- *gamma*: Minimum loss reduction required to make a further partition on a leaf node of the tree.
- *subsample*: Subsample ratio of the training instances.
- *min_child_weight*: Minimum sum of instance weight (hessian) needed in a child.
- *colsample_bytree*: Fraction of features to be randomly sampled for each tree.
- *objective*: Specifies the learning task and the corresponding learning objective.
- *eval_metric*: Evaluation metrics for validation data. It allows the user to specify a metric to be used for model evaluation.

Neural Networks:

- *epochs*: The number of times the learning algorithm working through the entire training dataset.
- *batch_size*: The number of training examples utilised in one iteration.
- *activation*: The activation function to use for the hidden layers.
- *solver*: The optimisation algorithm used to minimise the loss function during training.