AIML421_A3

kakauchad_300212228

2022-10-05

Contents

1 AIML421 Assignment 3

This project attempts to develop a model to predict music genres of unseen data, based on training over 50,000 rows of data, with 19 features, including the class label (i.e. the song's genre).

2 Core: Exploring and understanding the Data

2.1 Initial data analysis

2.1.1 Data features and class label

The training data consists a total of 50,000 rows of data, with 19 features. The target feature is 'genre' and the classes are balanced - 5000 instances of each of the 10 possible classes. The dataset has seven categorical features 'artist_name', 'track_name', 'track_id', 'key', 'mode', 'time_signature', and the class label 'genre'.

Of the categorical features, 'key' and 'mode' refer to musical attributes (i.e. 'key' is the predominant note that the song is played, 'mode' indicates if the song is in the major or minor key) and could provide some indication of musical genre.

The categorical features 'track_id', 'artist_name', and 'track_name' each have repeated values (i.e. same song name or same band name etc.). By concatenating all three features we can create a new feature and use this to check if there are any fully duplicated songs in the list. After checking, there were no duplcated songs.

The dataset has 12 numerical features 'instance_id', 'popularity', 'acousticness', 'danceability', 'duration_ms', 'energy', 'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo', and 'valence'.

Of the numerical features:

- 'instance_id' is an arbitrary value for identification only
- 'popularity' has integer values ranging from 0 - 96, with median 41
- 'acoustic' has float values ranging 0-1, with mean 0.35
- 'danceability' has float values, ranging 0-1, with mean 0.57
- 'duration_ms' has float values, ranging from -1 to 4.8e06, with mean 1.5e05
- 'energy' has float values, ranging from 0-1, with mean 0.62
- 'instrumentalness' has float values, ranging from 0-1, with mean 0.09
- 'liveness' has float values, ranging from 0-1, with mean 0.26
- 'loudness' has float values, ranging from -38.4 to 3.
- 'speechness' has float values, ranging from 0-1, with mean 0.17
- 'valence' has float values, ranging from 0-1, with mean 0.49

2.1.2 Insights from histograms

We can see from the histograms (Figure 1) that 'acousticness', 'duration_ms', 'instrumentalness', and 'speechiness' are heavily weighted toward low values, with more than three-quarters of the instances in the lowest bins. Most of these features provide little discriminatory power because they are essentially the same across all of the different classes, so make good candidates for removal from the model.

'danceability', 'popularity', and 'valence' have distributions that appear normal. 'loudness' and 'energy' are have heavier left tails, and 'liveness' has a heavier right tail. Although these features are individually imbalanced, they may have useful interactions that help predict music genre.

We can also plot histograms of encoded categoric features, although the encoded categoric features ('key', 'mode', 'time_signature') don't appear to offer much additional value.
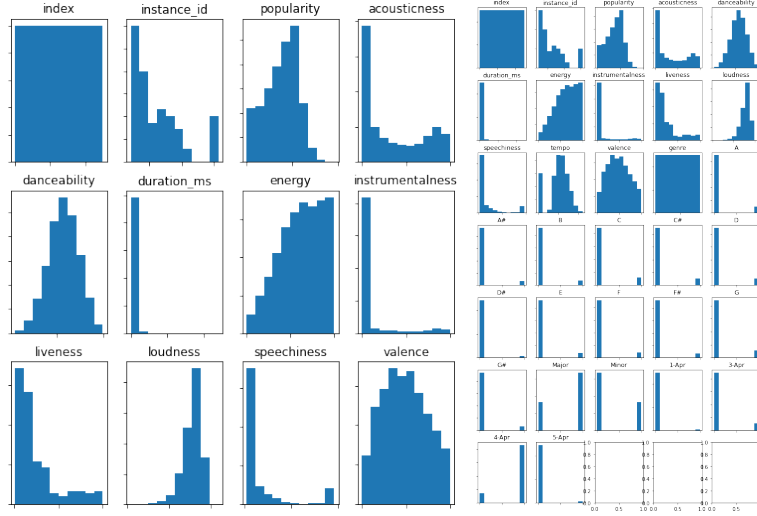
Figure 1: Histograms of raw numeric features and encoded features

2.1.3 Handling missing and invalid values

We see that 'tempo' has a question mark as a value and there are some songs with a 'duration_ms' of '-1 ms', which is an invalid value. We can replace these invalid values or remove the instances that have these invalid values. Checking how many times these values appear to determine how we can treat them, we see that 'tempo' has 7461 occurrences of '?', and 'duration_ms' has around 10,000 occurrences of '-1'.

We have a range of options for treating the invalid values: remove the missing row with the invalid value, impute some value to replace the invalid value, or replace the value with some arbitrary value.

Since 'duration_ms' has a heavily skewed distribution (i.e. mostly very low values) so it is unlikely that this feature will contribute much to the predictive power of the model. If we impute values in this feature it may mask the predictive power of other features, so we will remove those instances with invalid values in 'duration_ms'.

2.1.4 Feature combinations

We can examine pairs of features to check for correlation to identify highly correlated features that may present options for removal . We run scatter plots across all features but highlight only three interesting insights (see Figure 2):
1. 'tempo' feature has strong predictive power for some genres (indicated by strong colour banding in combination with multiple features 2. 'popularity' feature has some weaker predictive power but still exhibits some clustering of genres, when combined with some features
3. some feature pairs appear to have positive correlation (e.g. 'energy' & 'loudness', 'danceability' & 'valence'), some negatively featured (e.g. 'acousticness' & 'loudness', and 'acousticness' & 'energy')

2.1.5 Insights from the initial analysis

The key insights from the initial analysis are that 'tempo' looks to be a good predictor of some music genres; 'popularity' is lesser but still a reasonable predictor of music genre; 'loudness' is positively correlated with 'energy' and negatively correlated with 'acousticness' so is potentially redundant against two other features; most categorical features have heavily unbalanced distributions and are unlikely to offer much predictive power. With a dataset of 50,000 features, we can either remove the instances with invalid features (i.e. 10,000 in 'duration_ms' and 7,500 in 'tempo') and still have sufficient instances to establish a good
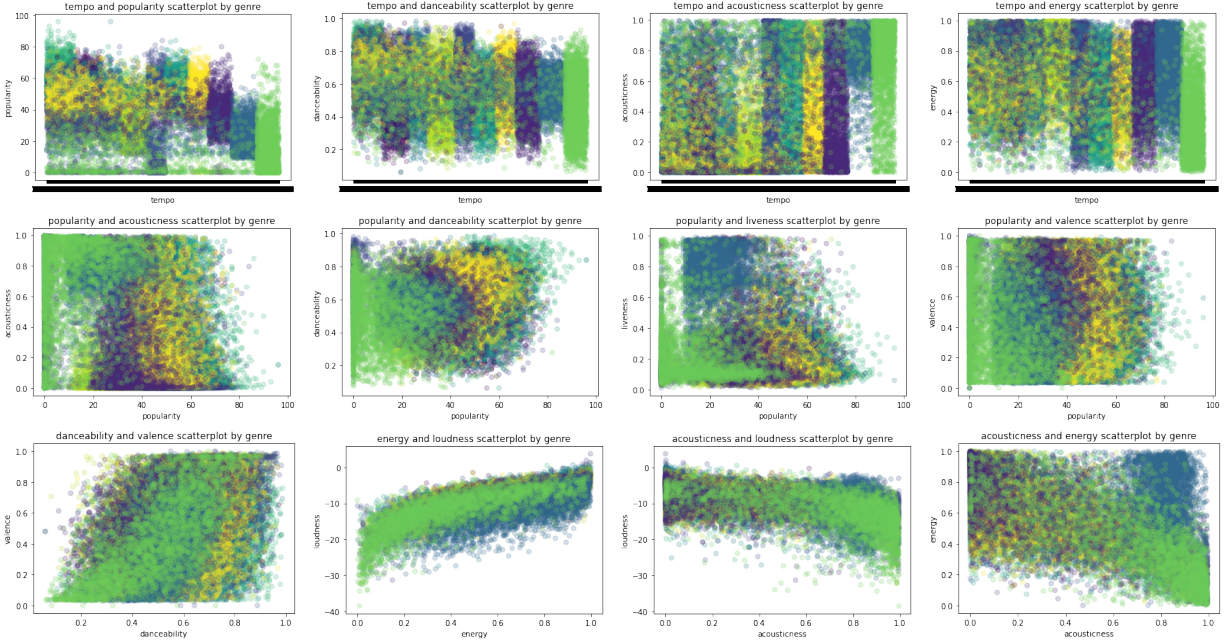
Figure 2: Scatterplots showing clear banding in 'tempo' feature (top row), good clustering by 'popularity' (middle row) and some correlated features (bottom row)

predictive model. Alternatively we could consider removing the 'duration_ms' feature since it has a heavily biased distribution, but we should likely retain 'tempo' because it has shown clear banding by genre, so this is a better candidate for imputing invalid values.

3   Completion: Developing and testing machine learning systems

3.1   General approach

Based on the initial data analysis my general approach was to handle the invalid values (imputation using mean) to allow 'duration_ms' and 'tempo' to remain in case of some feature interaction. Then onehotencode categoric values. Having prepared the data, I wanted to do some feature selection and then try a couple of different algorithms, including a linear model, and a convolutional neural network.

3.2   Preprocessing

Before selecting any particular model, I removed three categoric features ('artist_name', 'track_name', 'track_id') because encoding them made the model... ridiculous (onehotencoded the categoric features - although some categoric features do have ordinality (e.g. key), enforcing that ordinality (e.g 'F' higher than 'E') seemed misleading, because musical notes cycle so an 'E' would be higher than an 'F' from a lower octave.

I handled the invalid values by first removing the instances that had invalid values, and also by imputing average values. Obviously, removing the instances reduces the number of instances a model has to learn but doesn't introduce any distortion. Recognising that the features themselves may be found as not particularly predictive, I also ran the simple imputation by replacing the values with the average of the feature.

I scaled the dataset (excluding 'artist_name', 'track_name', 'track_id', and the label 'genre'), split the data into train and test sets (50:50) and then ran a Logistic Regression model, using 'elasticnet' as the penalty.

Logistic Regression is a model that allows each feature to contribute to the prediction, but I introduced a penalty for more complex models, encouraging a simpler model.

In designing my system I had just finished working up a neural network using the tensorflow.keras library so I decided to try applying that to this system (not a great decision, but more on that in the lessons learnt part).

3.3 Dimensionality reduction

I tried using Principal Component Analysis (PCA), with six components, that resulted in around 40% of the variance explained. I manually removed numerous features (i.e. dummy features from encoded 'key', 'time_signature', 'mode') and retained numeric features that looked to have good variability and clustering from the scatterplots (i.e. 'popularity', 'tempo', 'danceabiilty', 'acousticness', and 'valence'). These features explained around 70% of the variance, with three components. In spite of the improved performance, Figure 3 shows no obvious clustering with either six (left) or three (right) principal components.
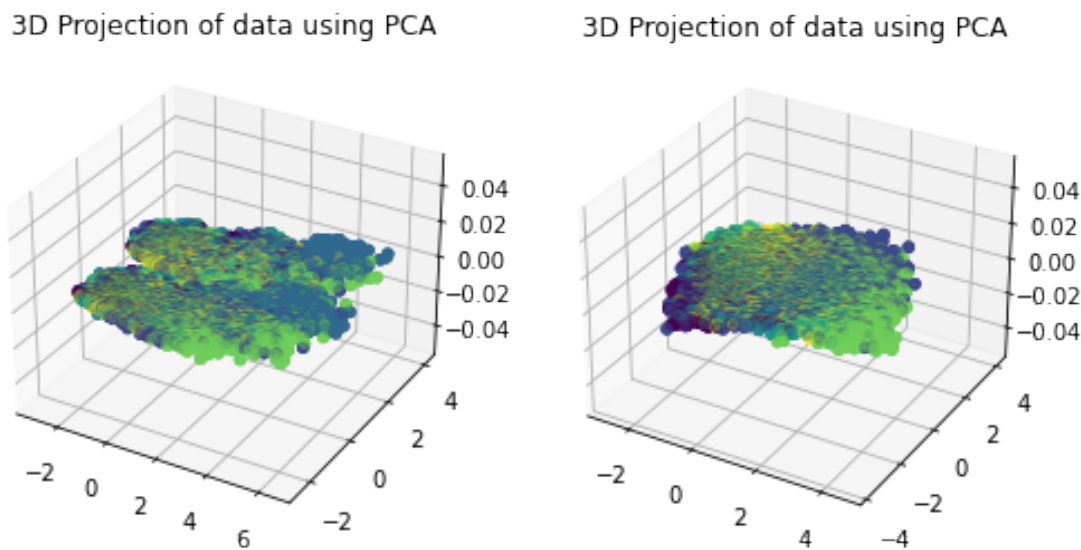

Figure 3: Principal component analysis for six components (left), three components (right)

3.4 Initial model comparison: Logistic Regression, Gradient Boosting, and Random Forest classifiers

I ran a comparison of sklearn models, with default settings and found that of the three models, Random Forest and Gradient Boosting have the same accuracy (61%) and Logistic Regression had 54% accuracy. These models were trained on the full dataset (only missing three categoric features).

I thought 60% accuracy wasn't that great so I tried another way - manually removing features AND trying a different mode. I couldn't get past the scatterplots that showed good clustering and banding for certain numeric features, so I manually selected those features ('popularity', 'tempo', 'danceability', 'acousticness', 'valence', 'energy') and fed those into a convolutional neural network, because I had seen that convolutional neural networks could be really good at classifying audio.

3.4.1 Convolutional neural network

I had a go at implementing a convolutional neural network, comprising six latent layers (3 dense layers of 256, 32, and 256 nodes each) a 1D Convolution layer (with 8 output nodes and a 64 bit kernel window) followed by a latent layer of 128 nodes and finally an output layer of 10 nodes (one for each class). The neural network uses Rectified Linear activation at all latent layers, and obviously softmax activation for classification.

I chose dense layers because they are straight forward to implement (i.e. easy to reshape inputs and outputs to match the data) and had performed well in earlier model comparisons. The number of nodes at each layer was a fairly arbitrary mix, on the assumption that having multiple times more nodes than output or input features provides more differentiation points to account for feature interactions (reflected as different weightings at each node). I was also hoping for some interaction by having different numbers of nodes between layers.

The CONV1D layer runs an 8-bit kernel across every 'pixel' or datapoint in the set and calculates an average output across those bits. Those outputs are then fed to the next dense layer of 128 nodes and on to the classification output layer.

This network achieved around 54% accuracy after around 5 epochs (with a dataset of 12,500 training instances). Which wasn't very good.

3.5 Recursive feature elimination

I tried letting the machine select the best features using Recursive feature elimination with cross validation, with a Decision Tree Classifier. The recursive feature elimination reached maximum classification accuracy (around 47%) with 8 features (see Figure 4). Which made me think, my selection of features was probably fine, because the machine ended up with that many features.
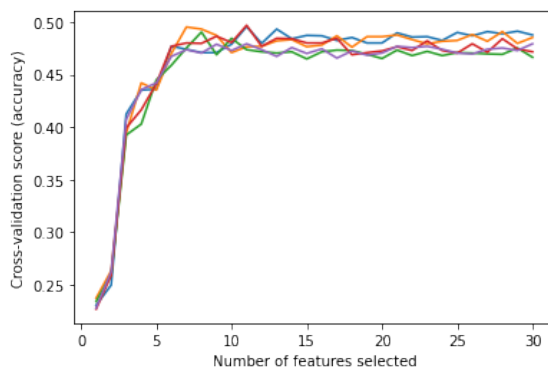

Figure 4: Recursive Feature Elimination with cross validation

3.6 But sci-kit-learn can do all the feature and model selection at once. . .
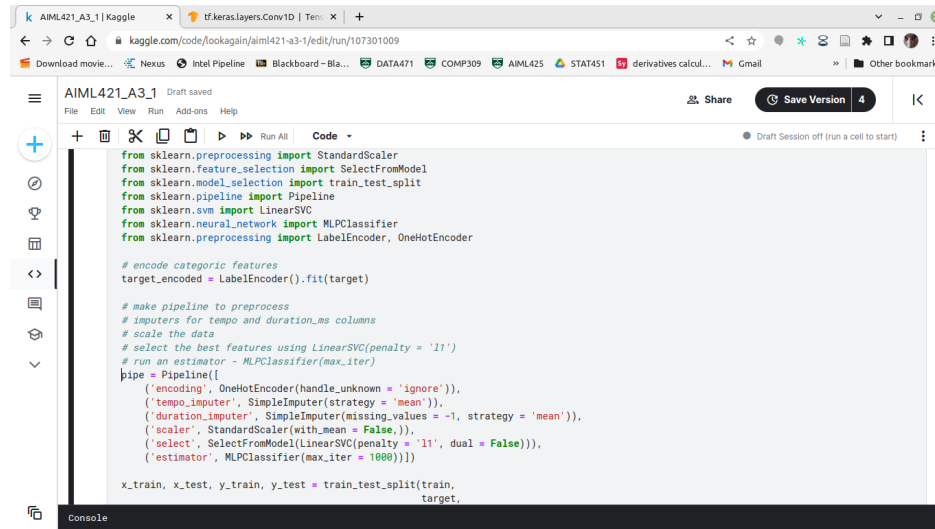
After spending lots of time making dumb, time-eating decisions, I just threw the data into sklearn, let the model to the encoding, imputing missing values, scaling data, handling feature selection, training a default neural network and then running the prediction from that model. It takes several minutes but selects a model that predicts with 63% accuracy.

For feature selection, I used the SelectFromModel() function, with the LinearSVC model (with L1 normalisation) as the feature selector. I chose LinearSVC because it had performed well on other classification tasks in the past. Obviously L1 normalisation penalises complex models, so should lead to a model with the fewest necessary features.

4 Final model selection

In the end, I selected the default neural network implemented as part of a single pipeline that performed all the preprocessing. I chose this for two reasons:

1. I was struggling with time and this model was tidy (see Figure 5) and could be easily modified for a different model if required, and

2. it actually did an ok job of predicting music genre.

Figure 5: Recursive Feature Elimination with cross validation

In toy datasets during previous assignments, I expected classification accuracy in the 90% region, but this data was clearly more noisy than many of the previous, manicured datasets. Having sklearn do the feature selection gave me more confidence in the results, because my own manual feature selection had been driven by my own perceptions so was heavily biased by the coloured scatterplots early in the development process.

5 Challenge: Reflecting on your findings

The final model is fairly easy to interpret, because after pre-processing, it has two key processes: feature selection using LinearSVC, and classification using default multilayer perceptron. The model performs at above 60% accuracy, which feels pretty good considering the banding and clustering in the early scatterplots - these visual cues showed some features and combinations make good predictors, but all of those plots had lots of random noise which makes perfect prediction unlikely.

To improve interprebility, I could perhaps explicitly state the network architecture (i.e. how many nodes in each layer), as well as the objective and activation functions. The final model is very easy to implement (mostly out of the box sklearn functions), running at around 10 lines for building the pipeline, training, validating and testing. After all of the angst in working up the assignment, the final model is very vanilla.

To improve performance, I would have used GridSearchCV, which I have since used in a couple of other implementations and very much regret not returning to sklearn much earlier in the process because of all the built in functionality and modularity (e.g. pipelines for sequencing machine learning tasks, built in metrics etc.). The simplicity with which a preprocessing model can be coupled with a range of classifiers, each with multiple options for cross-validating hyper-parameters to identify the best performing model... mind-blowing. After I spent too much time on the other stuff :(

What did I spend too much time on:
- waiting for computationally expensive scatterplots - because they were plotting colour plots with the full

dataset (rather than a sample of the dataset) - trawling through documentation to try and figure out shape and reshape parameters to get data into convolutional neural networks in order to use functions in that library because ... - ... I didn't appreciate early enough that the CNN functions I was interested in were good at classifying music from arrays of AUDIO data rather than metadata about a song or track - not leaning on (i.e. learning from and using) functions and techniques used in previous assignments by looking at each one as an individual piece of work - partly because each assignment has been focussed on a specific learning objective, but lots of the steps, techniques, skills are similar and reinforcing - it just took too long to allow the cross over - defining complex functions that tried to do too much in one go, rather than creating small, focussed functions (that are easier to develop) and then calling those functions in other functions.

6 Public Kaggle competitions: House Prices - Advanced Regression Techniques

This Kaggle competition requires developing a machine learning regression model to predict the price of a house based on features that a house buyer may find important. The competition instructions require an output file containing Id and SalePrice, and confirms the evaluation metric as Root-mean-squared error (RMSE) between the log of the predicted value and the log of the actual value of the house.

Because there are alot of features I would expect to run an initial data analysis to try and get a better appreciation of the individual features, what they represent and whether they interact. I would then work on handling missing values before encoding categoric features and then running some feature selection process. Then I would try a few different regression machine learning algorithms and choose the best performing model.

I have seen this dataset before so I know there are several features categorical features that are ordinal, so I would have to figure out a way to capture that ordinality (ratings that are sorted alphabetically by default) which will probably require a dedicated step in preprocessing. There are also several missing values so I would use a SimpleImputer to impute those - because it is quick and fairly straightforward.

Once encoded it becomes a pretty massive array so the next step is to scale the data so that very high values don't overpower smaller values etc. Because we are predicting a continuous variable I would test a number of regression models and hyper-parameters using the GridSearchCV cross-validation method from sklearn. Likely methods would be:
- Linear Regression - Decision Tree Regressor - SVR - LinearSVR - GradientBoostingRegressor - MLPRegressor etc.

As part of the pipeline to feed data into the GridSearchCV function I would include a feature selection step and providing three alternative functions: SelectFromModel (because it only requires a single fit per model set-up), in one case, using the model (LinearSVC(penalty = 'l1')) to identify the features with highest coefficients (more influential on the prediction) and in one case using the ExtraTreesClassifier() to identify features that have the best impurity values. The third feature selection would be SelectKBest(chi2, k = 6), because the feature selection is fairly quick.

After identifying the best model (and parameters) using GridSearchCV, I would run that model fitted model to predict house prices on the test dataset.

Comparing the chosen Residential House Price competition with the class competition:
- the class competition has a classification task and fairly good quality data (i.e. invalid values are limited to two features), whereas the Residential House Prices competition is a regression task
- the machine learning models are fundamentally the same (apart from regressor vs classifier variants) allowing for the use of the same libraries (i.e. sklearn for machine learning tasks, matplotlib or seaborn for visualisation etc.) - the datasets are similar (once encoded) although the music dataset will have very high number of dummy features from the 'artist_name', 'track_name' and 'track_id' features.

Overall, these two competitions are very similar in terms of data, tasks and preprocessing.