Multiclass classification of songs' genre using one-vs-all and multinomial logistic regression

***Abstract*—The data analysis project, in the scope of Machine Learning Basic Principles course, focuses on multiclass classification to identify songs' genre using custom subset of the Million Song Dataset [1], and the labels obtained from AllMusic.com [2]. During the project, 3 differnt methods were used to optimze and improve the accuracy of results namely, 'K Nearest Neighbours (KNN) with Low Dimensional Embedding', 'Multinomial Logistic Regression' and 'One vs All (OvA) Logistic Regression'. The very first method 'KNN with LDE' employs Principal Component Analysis [5] to extract useful features out of the high-dimensional dataset and then uses KNN algorithm to create target clusters. While Multinomial and OvA method use probablistic models to classify target classes where the main difference occurs in the modeling assumptions behind each of the two models.**

I. Introduction

Music, by no doubt, has become an imperative part of people's life. People prefer listening to their favorite genre of songs for more personalized experience. Because of the availability of tons of songs, listeners crave for more personalized experience while at the same time, they prefer non-static music for mood augmentation.

The project, in particular focuses on finding a pattern to identify the genre of songs. The identification pattern can help to build an online recommendation platform using exisiting databse of users so that listeners can have more customized experience while in search for new songs. During the entire project, we address the question that given a list of songs, how well can we analyze the audio to find a pattern that helps us identiy songs that belong to similar genre of music; which is to say that we construct a predictor h(x) for each genre (Y) using the features (x) of the songs and map it to a probability h(x). Simpler this task may seem for a certain type of songs (e.g. heavy metal vs classical), the classification problem can become difficult for other types (e.g.rock vs blues).

II. Data Analysis

The dataset used for the project is a custom subset of the Million Song Dataset [1], and the labels were obtained from AllMusic.com [2]. For simplicity, each song has been assigned only one label that corresponds to the most representative genre. The data is split into two datasets: a training data set with 4363 songs, and a test set dataset with 6544 songs. Each song has 264 features, and there are 10 possible classes in total, which are:

1) Pop Rock
2) Electronic
3) Rap
4) Jazz
5) Latin
6) RnB
7) International
8) Country
9) Reggae
10) Blues

The Histogram of target labels below provides insights about the training data at hand:

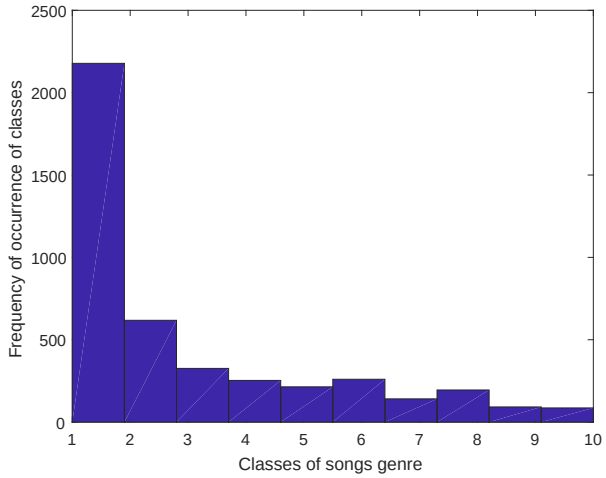


Fig. 1: Histogram of Target Labels

It is evident from the histogram that the training data at hand is imbalanced with a very high representation of class-1 datapoints.

Fig.2 and Fig. 3 represent that layout of a single datapoint (i.e., a song) and strucutral reprensentation of example vector.

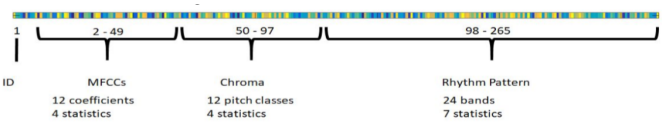


Fig. 2: Layout of Example Vector

The feature vector for each song thus consists of 12 coefficients representing MFFCs (using 4 statistics each), 12 pitch classes representing Chroma (using 4 statistics each), and 24 bands representing Rythm Patterns (using 7 statistics each). The details on the structure of each component of the example vector is as below:

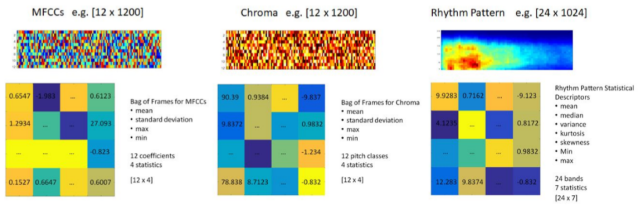


Fig. 3: Structural Representation of Example Vector

III. Methods and Experiments

Multiclass classification problem was explored using three different methods:

- KNN clustering with low dimensional embedding
- Multinomial logistic regression
- One-vs-All (OvA)

A. KNN clustering with low dimensional embedding

Low dimensional embedding with KNN clustering approach was adopted to check the possibility of reducing the high dimensions for performance and accuracy improvement.

The technique employs Principal Component Analysis to extract useful features out of the high-dimensional dataset, essentially projecting the features to a lower dimensional space, by retaining the top n principal components which yield the highest explained variance. Next, it applies KNN clustering on the low-dimensional dataset, with the prior PCA helping to prevent the ML algorithm from succumbing to the curse of dimensionality [7].

In the adopted approach, the dimensions of both our training and test datasets were redyced using PCA, followed by the application of the KNN algorithm to acquire a model that fits the samples to appropriate labels. The model was then applied to the training and test sets, and the accuracy for both applications was measured.

The approach was repeated for different values for dimensionality reduction and K nearest neighbors, with the maximum accuracy finally being found at $K = 22$ and $n = 21$. The final accuracy results were less than satisfactory, although it was evident that the classification task was greatly expedited because of the reduced complexity of the lower dimensional dataset.

Due to the dissatisfactory results, it was decided to not pursue a dimensional reduction approach any further and rather switched to alternate methods for solving the problem.

Before, we jump into the details of Multinomial and OvR Logistic Regresss, first let's try to understand the difference between both. The main difference occurs in the modeling assumptions behind each of the two models. Let's assume that there are K classes. For OVA, the assumption is that there are K independent classification problems, one for each class, i.e., for class i, we learn a logistic (probability) model of the form $\frac{1}{1+exp(-\beta_i^T x)}$ and each of these K problems are independent of the other K-1 logistic regression problems. Whereas for Multinomial model, we assume that there is a multinomial conditional distribution, such that for a class i we have the following logistic model $\frac{exp(-\beta_i^T x)}{\sum_{j=1}^{K} exp(-\beta_j^T x)}$. As can be seen immediately, learning β_i for a particular class i immediately affects the model for other classes j through the joint model.

B. Multinomial logistic regression

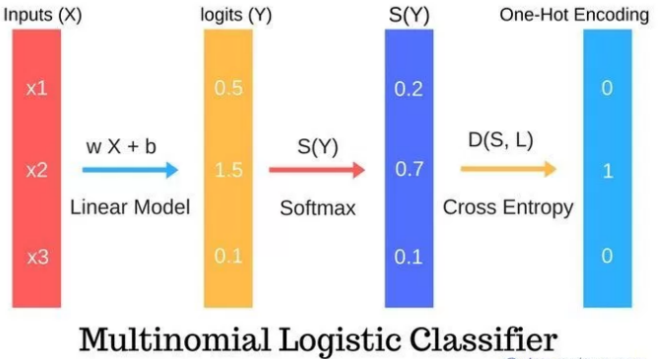
Multinomial Logistic Regression [3] is the linear regression analysis technique used when the dependent variable is nominal with more than two levels. Thus it is an extension of logistic regression, which analyzes dichotomous (binary) dependents. One of the major reasons for choosing multinomial logistic regression is because it works well on big data irrespective of different areas.

For the data analysis project, same technique was used like the logistic regression for binary classification until calculating the probabilities for each target. Once the probabilities were calculated, these were transferred into one hot encoding and cross entropy method was used for calculating the properly optimized weights. The idea is to construct a linear predictor function that constructs a score from a set of weights that are linearly combined with the features of a given observation using a dot product:

$$score(X_i, k) = \beta_k . X_i, \tag{1}$$

where X_i (in our case) is the vector that contains the training data of songs, β_k is a vector of weights (or regression coefficients) corresponding to k=10 outcomes of songs' genre, and $score(X_i, k)$ is the score associated with assigning each observation i to category k.

The following inforgraph explains the idea very well



Just like the sigmoid function in binary logistic regresssion, softmax function was used in multinomial logistic regression. The Softmax function is a probabilistic function which calculates the probabilities for the given score and returns the high probability value for the high scores and fewer probabilities for the remaining scores. Therefore, each observation i in the training dataset (X) was evaluated on k=10 predictors (h(x)) and then, softmax function was used to find the highest probabilitic class to which the observation i belongs.

The last step used in the multinomial logistic regression is the cross-entropy. It is a distance calculation function which takes the calculated probabilities from softmax function and then creates one-hot-encoding matrix to calculate the distance. For the right target class, the distance value is less, and the distance values are larger for the wrong target class.

The major challenge faced with this particular technique was to get the logitic loss to converge. Different number of iterations and different implementations of multinomial logistic regeression were used. $sklearn.linear-model.LogisticRegression$ [2] function from scikit-learn library was also experimented with different number of iterations (upto 500 which takes around 5 to 10 minutes), and different solvers like newton-cg, lbfgs, sag, saga. The best result was obtained using 'newton-cg' solver.

C. One vs All (OvA)

The third approach explored for the project was One-vs-All (also called One-vs-Rest) Classification. One classifier was trained for every class, taking the samples for that particular class as positive, while the remaining samples are treated as negative.

The base classifer must yield a real-valued confidence score for the decision, as opposed to merely a class label, since discrete class labels could become ambiguous if results from multiple classifiers yield different labels for the same sample.

An algorithm describing the procedure is as follows:

- *Inputs to the algorithm:*
  - dataset of samples X and labels y, where $y_i \in \{1, 2, ..., N\}$ is the label for the particular sample X_i
  - L, a binary classification training algorithm
- *Outputs of the algorithm:*
  - a set of classifiers c_n for $n \in \{1, 2, ..., N\}$
- *Mechanism:*
  - For every $n \in \{1, 2, ..., N\}$
    - Build a separate label vector z, with $z_i = 1$ if $y_i = n$ and $z_i = 0$ for all other y_i
    - Apply training algorithm L on samples X and label vector z, to get c_n

Concretely deciding which class an unseen sample x belongs to requires the application of all the obtained classifiers c_n to it, and then choosing the label n whose classifier yields the largest confidence score:

$$\hat{y} = \underset{n \in \{1,2,...,N\}}{argmax} \; c_n(x)$$

Two potential issues with the OVR are approach are:

- The confidence values can be different in terms of scale between each binary classifier
- Despite the training set having a balanced class distribution, the binary classifiers will see it as unbalanced since the number of negative labels will (most likely) be much greater than the number of positive labels.

In the implementation of the OVR approach, the multinomial logistic regression algorithm was modified to generate N separate classifiers. The final label for each is decided based on the classifier that results in the best confidence score.

IV. Results

Table-1 indicates the results for accuracy while table 2 indicates results for LogLoss. Among the many attempts that were made, results for the few are summarized below:

TABLE I: **Accuracy**

Index	Method	Training Accuracy	Validation Accuracy	Kaggle Result
1	KNN with LDE	57.04%	53.03%	NA
2	Multinomial	74.53%	63.14%	62.13%
3	Multinomial	75.8%	63.90%	62.483%
4	OvR	75.9%	63.5%	62.88%
Best	OvR	75.9%	63.5%	62.88%

TABLE II: **LogLoss**

Index	Method	Training_Accuracy	Validation_Accuracy	Kaggle_Result
1	Multinomial	75.8%	63.90%	2.60627
2	OvR	75.8%	63.90%	3.07390
Best	Multinomial	75.8%	63.90%	2.60627

V. Conclusion

The very first method employed, 'K Nearest Neighbours with Low Dimensional Embedding' helped to reduce the dataset to a lower dimensional space and decrease the total run time for data analysis task but as a consequence, majority of dimensions present inside the dataset containing useful information got lost with the reduction of dimensionality that resulted in poor training and validation accuracy. Second method 'Multinomial Logistic Regression' worked well on the provided dataset but did not score more than 62.5% accuracy results. In order to improve results, different parameters were tuned like maximum number of iterations, and solver (e.g., newton-cg, saga,sag) but results only improved to fractional poitns. The major reason seems to be the imbalanced dataset with higher number of representation of class-1 datapoints. Finally, OvA classification method with logistic regression was implemented, yielding comparable results to the Multinomial Logistic Regression. OvA yielded marginally better results for the accuracy metric, whereas MLR yielded better results for the log-loss metric. One particluarly interesting approach, for future purposes, could be to reduce the representation of class-1 datapoints to create a more balanced dataset among all classes and then use Multinomial and OvA logistic regression for target classification, with Low Dimensional Embedding as a pre-processing step for eliminating potential redundancy.

References

[1] , "Million Song Dataset — scaling MIR research", url ="*https://labrosa.ee.columbia.edu/millionsong/*",

[2] , "sklearn.linearmodel.LogisticRegression", url = *"http://scikit-learn.org/stable/modules/generated/sklearn.linear*

[3] "2 ways to implement Multinomial Logistic Regression in Python", url ="*http://dataaspirant.com/2017/05/15/implement-multinomial-logistic-regression-python/*",

[4] "AllMusic", url = *"https://www.allmusic.com/*",

[5] "Using PCA for Dimensionality Reduction", url = *'https://www.analyticsvidhya.com/blog/2015/07/dimension-reduction-methods/ '*,

[6] "Structure preserving embedding", author = "Shaw, B.; Jebara, T." year = "2009" url = *"http://www.cs.columbia.edu/ jebara/papers/spe-icml09.pdf"*

[7] "scikit-learn: Clustering and the curse of dimensionality", url = "http://www.markhneedham.com/blog/2016/08/27/scikit-learn-clustering-and-the-curse-of-dimensionality/".

VI. Appendices

Important functions related to the project and source code is given below. Please feel free to comment for any improvement.

```
def KNN_with_LDE():
""" Load Raw Data """
print("loading data ...")
# Load the Training Data
data_path_x = "train_data.csv";
songs_data = pd.read_csv("train_data.csv", header=
    None);
# Load the True Labels
data_path_y = "train_labels.csv";
songs_labels = pd.read_csv("train_labels.csv",
    header=None);
data_path_z = "test_data.csv";
val_x = pd.read_csv("test_data.csv", header=None);

""" Pre Process Data """
print("preprocessing data ...")
scaler = preprocessing.StandardScaler().fit(
    songs_data)
scaler
scaler.mean_
scaler.scale_
scaler.transform(songs_data)
scaler.transform(val_x)

""" Split training data into training data and test
    data for cross validation """
print("partitioning data ...")
train_x, test_x, train_y, test_y = train_test_split(
    songs_data, songs_labels, train_size=0.8,
    random_state=0);

""" Perform feature reduction"""
print("performing feature reduction ...")
train_x_pca = PCA(21)
train_x_pca.fit(train_x)
train_x_reduced = train_x_pca.transform(train_x)
test_x_reduced = train_x_pca.transform(test_x)
val_x_reduced = train_x_pca.transform(val_x)

""" Build the KNN model"""
print("building model...")
n_neighbors = 22
weights = 'uniform'
clf = neighbors.KNeighborsClassifier(n_neighbors,
    weights)
clf.fit(train_x_reduced, np.asarray(train_y).ravel()
    )

""" Output accuracy results for Train and Test
    partitions"""
print("Train Accuracy :: ", metrics.accuracy_score(
    train_y, clf.predict(train_x_reduced)));
print("Test Accuracy ::  ", metrics.accuracy_score(
    test_y, clf.predict(test_x_reduced)));

""" Predict the Test (unlabeled) Data """
test_data_results = mul_lr.predict(val_x_reduced);
""" Export the Test Data File """
df = pd.DataFrame(test_data_results);
df.to_csv('test_data_results.csv', index=False,
    header=False);
```

Listing 1: KNN Clustering with Low Dimensional Embedding

```
""" logistic_gradient_descent returns w_{opt}"""
def logistic_gradient_descent(x_vec,y_vec,N):
```

```
w = np.zeros((N,1));
w_opt = np.zeros((N,1));
a = 1e-5; """ Learning Rate """
yT = np.transpose(y_vec);
xT = np.transpose(x_vec);
error_margin = 0.0001;
e = 1e20;
_iter = 0;
min_error = e;
#for _iter in range(1000):
while(abs(e) > error_margin):
if (_iter > 1000):
break;
_num = math.exp(-np.dot(yT,np.dot(x_vec,w)));
_den = 1 + math.exp(-np.dot(yT,np.dot(x_vec,w)));
grad_w = -(5/N)*(_num/_den)*np.dot(xT,y_vec);
w = w - a*grad_w;
""" Computer Error """
e = math.log(1 + math.exp(-np.dot(yT,
      np.dot(x_vec,w))));

if (min_error > e):
min_error = e;
w_opt = w;
print (e);
_iter = _iter + 1;
#print (w.shape);
return (w_opt);
```

Listing 2: Logistic Gradient Descent

```
""" Performs Multinomial Logistic Regression """

def mult_nomial_logr():
""" Prepare the Data """
# Load the Training Data
data_path_x = "path/train_data.csv";
songs_data = pd.read_csv
      (data_path_x, header=None);
# Lad the True Labels.
data_path_y = "path/train_labels.csv";
songs_target = pd.read_csv
      (data_path_y, header=None);
data_path_z = "path/test_data.csv";
test_data = pd.read_csv
      (data_path_z, header=None);

""" Split Data into Training and Validation """
train_x, test_x, train_y, test_y =
train_test_split(songs_data, songs_target,
      train_size=0.7, random_state=0);

"""Train multinomial logistic regression model"""
mul_lr = linear_model.LogisticRegression(multi_class
    ='multinomial', max_iter=500, solver='netwon-cg'
    ).fit(train_x, train_y);
print ("Multinomial Logistic regression Train
    Accuracy :: ", metrics.accuracy_score(train_y,
    mul_lr.predict(train_x)));
print ("Multinomial Logistic regression Test
    Accuracy ::  ", metrics.accuracy_score(test_y,
    mul_lr.predict(test_x)));

""" Predict the Test Data """
test_data_results = mul_lr.predict(test_data);
""" Export the Test Data File """
df = pd.DataFrame(test_data_results);
df.to_csv('test_data_results.csv', index=False,
    header=False);
```

Listing 3: Multinomial LogisticRegression

```
""" Performs OvR Logistic Regression """
def mult_nomial_logr():
""" Prepare the Data """
# Load the Training Data
data_path_x = "path/train_data.csv";
songs_data = pd.read_csv(data_path_x, header=None);
# Lad the True Labels.
data_path_y = "path/train_labels.csv";
songs_target = pd.read_csv(data_path_y, header=None)
    ;
data_path_z = "path/test_data.csv";
test_data = pd.read_csv(data_path_z, header=None);

""" Split Data into Training and Validation """
train_x, test_x, train_y, test_y = train_test_split(
    songs_data, songs_target, train_size=0.7,
    random_state=0);

"""Train multinomial logistic regression model"""
mul_lr = linear_model.LogisticRegression(multi_class
    ='ovr', max_iter=500, solver='netwon-cg').fit(
    train_x, train_y);
print ("OvR Logistic regression Train Accuracy :: ",
     metrics.accuracy_score(train_y, mul_lr.predict(
    train_x)));
print ("OvR Logistic regression Test Accuracy ::  ",
     metrics.accuracy_score(test_y, mul_lr.predict(
    test_x)));

""" Predict the Test Data """
test_data_results = mul_lr.predict(test_data);
""" Export the Test Data File """
df = pd.DataFrame(test_data_results);
df.to_csv('test_data_results.csv', index=False,
    header=False);
```

Listing 4: OvR LogisticRegression