UNIVERSIDAD POLITÉCNICA DE VICTORIA

Report - Classifier models

Maestria en ingenieria

Submitted by:
Homero Pérez Mata

Professor:
Dr. Said Polanco Martagón

Course:
MI-20302 Machine learning

Ciudad Victoria, Tamaulipas, February 10 2023.

Index

1 Introduction

In this report the GTZAN Dataset - Music Genre Classification dataset was used in combination with a notebook found on Kaggle containing different models including, Naive Bayes, Stochastic Gradient Descent, KNN, Decision trees, Random Forest, Support Vector Machine, Logic Regression, Neural Nets, Cross Gradient Booster and Cross Gradient Booster (Random Forest). The objective of the report is to obtain the accuracy of these models and then use grid search + cross validation to try to to improve the accuracy of each model.

2 Literature review

2.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a popular optimization algorithm used in machine learning for finding the optimal parameters of a model.

The basic idea behind SGD is to iteratively adjust the parameters of the model by calculating te gradient of the cost function with respect to the parameters, and then updating the parameters in the direction of the negative gradient.

However, instead of using the entire training dataset to calculate the gradient, SGD only uses a small random subset of the data (called a batch) at each iteration. This makes the optimization process much faster and more efficient for large datasets, as the gradient calculation and parameter updates can be performed much more frequently.

SGD also has the benefit of being able to escape from local optima due to its stochastic nature. In contrast, deterministic optimization methods may get stuck in local optima and fail to find the global optimum.

2.2 Cross Gradient Booster

XGB is an iterative algorithm that builds an ensemble of weak prediction models (usually decision trees) in a stage-wise fashion. At each stage, the algorithm fits a new model to the negative gradient of the loss function with respect to the current ensemble's predictions. This new model is then added to the ensemble and the process is repeated until convergence or a specified number of models have been added.

XGB is a powerful and flexible algorithm that can handle a wide range of problems and is particularly useful when there are complex nonlinear relationships between the features and target variable. However, it can be sensitive to overfitting and requires careful tuning of hyperparameters such as the learning rate, number of estimators, and maximum depth of the trees.

3 Methods and materials

The dataset used in this project is known as the GTZAN Dataset - Music Genre Classification dataset[1] which contains a variety of tracks with ten different genres of music. Including blues, classical, country, disco, hiphop, jazz, metal, pop, reggae and roxk. To support the dataset a notebook already created by a member of Kaggle was used[2]. The notebook contains the steps to implement ten different models. Those which are Naive Bayes, Stochastic Gradient Descent, KNN, Decision trees, Random Forest, Support Vector Machine, Logic Regression, Neural Nets, Cross Gradient Booster and Cross Gradient Booster (Random Forest).

In order to run the project an anaconda environment was setup with the following libraries:

- numpy
- matplotlib
- scikit-learn
- seaborn
- pandas

Since the original project is three years old some libraries had to be version specific in order for the source material to run:

- Librosa==0.92
- xgboost==1.3.3

Optional to this a computer with a good CPU and GPU is heavily recommended to improve the time required to process the models.

4 Development

The development of the notebook is divided in three parts. These three parts will be shown one by one with their respective inputs and their resulting outputs.

1. Data cleansing
2. Model creation
3. Optimization

The first step analyze, eliminate the outliers and create a training set and a test set for the model. Generally any data set can be found in a .csv file format. The file was read.

```
data = pd.read_csv(f'{general_path}/features_3_sec.csv')
data = data.iloc[0:, 1:]
data.head()
```

The resulting file should give Python access to the values inside the .csv file. The data set was split into two training and test with a test size of twenty percent.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

The data set training values should now be accessible with the variables $X_train and y_train$. Similar the test values should be accessible with $X_test and y_test$.

The second step consists in selecting the model that will be implemented and soon after testing it to verify its accuracy. The higher the accuracy the better the model will perform.

```
# Naive Bayes
nb = GaussianNB()
model_assess(nb, "Naive Bayes")

# Stochastic Gradient Descent
sgd = SGDClassifier(max_iter=5000, random_state=0)
model_assess(sgd, "Stochastic Gradient Descent")

# KNN
knn = KNeighborsClassifier(n_neighbors=19)
model_assess(knn, "KNN")

# Decission trees
tree = DecisionTreeClassifier()
model_assess(tree, "Decission trees")

# Random Forest
rforest = RandomForestClassifier(n_estimators=1000, max_depth=10, random_state=0)
model_assess(rforest, "Random Forest")

# Support Vector Machine
svm = SVC(decision_function_shape="ovo")
model_assess(svm, "Support Vector Machine")

# Logistic Regression
lg = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial')
model_assess(lg, "Logistic Regression")
```

The first set of models tested are show in figure 1. These models are considered lightweight models because of the low amount of resources and fast time they require to process.

```
Accuracy Naive Bayes : 0.51952

Accuracy Stochastic Gradient Descent : 0.65532

Accuracy KNN : 0.80581

Accuracy Decission trees : 0.64298

Accuracy Random Forest : 0.81415

Accuracy Support Vector Machine : 0.75409

Accuracy Logistic Regression : 0.6977
```

Fig. 1. Light models's accuracy

The respective accuracy of the models is show in figure 2. Most of the models have a low performing accuracy of under 80 percent, but models such as KNN or RandomForest show accuracy above 80 percent which is desirable. As for the second set of models these can be considered heavier models because of the amount of resources and time they take to process.

```
# Neural Nets
nn = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5000, 10), random_state=1)
model_assess(nn, "Neural Nets")
```

```
Accuracy Neural Nets : 0.671
```

```
# Cross Gradient Booster
xgb = XGBClassifier(n_estimators=1000, learning_rate=0.05)
model_assess(xgb, "Cross Gradient Booster")
```

```
[05:05:41] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'm
ulti:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Accuracy Cross Gradient Booster : 0.90224
```

```
# Cross Gradient Booster (Random Forest)
xgbrf = XGBRFClassifier(objective= 'multi:softmax')
model_assess(xgbrf, "Cross Gradient Booster (Random Forest)")
```

```
[05:13:20] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'm
ulti:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Accuracy Cross Gradient Booster (Random Forest) : 0.74575
```

Fig. 2. Heavy models

The final and last step is the model optimization. There are different ways to optimize models in this case two were used GridSearch for all models and differential evolution used specifically for the neural network model. In this case first Cross validation is applied. Cross validation runs the model multiple times randomizing the data set to obtain different results. The results are sum and the mean is obtained. Cross validation makes sure the model didn't have a unlucky run and performed poorly in a single occasion.

```
Model:  Naive Bayes
Cross Validation Scores:  [0.52852853 0.52423852 0.50793651]
Average CV Score:  0.5202345202345202
Model:  Stochastic Gradient Descent
Cross Validation Scores:  [0.61904762 0.63706564 0.63277563]
Average CV Score:  0.6296296296296297
Model:  KNN
Cross Validation Scores:  [0.77091377 0.76104676 0.75375375]
Average CV Score:  0.7619047619047619
Model:  Decission trees
Cross Validation Scores:  [0.5954526  0.6036036  0.59202059]
Average CV Score:  0.597025597025597
Model:  Random Forest
Cross Validation Scores:  [0.79493779 0.79407979 0.77606178]
Average CV Score:  0.7883597883597884
Model:  Support Vector Machine
Cross Validation Scores:  [0.71986272 0.71814672 0.71986272]
Average CV Score:  0.7192907192907194
Model:  Logistic Regression
Cross Validation Scores:  [0.66966967 0.67395967 0.66709567]
Average CV Score:  0.6702416702416704
Model:  Neural Nets
Cross Validation Scores:  [0.6954097  0.65165165 0.65336765]
Average CV Score:  0.6668096668096668
```

Fig. 3. Accuracy with cross validation

Cross validation will reduce the score of any model if the model had a lucky run when it was tested for the first time.

```
Cross Validation Scores:  [0.86100386 0.86872587 0.86958387]
Average CV Score:  0.8664378664378664
```

Fig. 4. Accuracy loss of XGB

The final step is adding GridSearch. GridSearch runs each model once with a combination of the parameters that were specified by the user. Then it compares the accuracy for each model and returns the parameters that performed the best. It could be seen as trial and error.

```
# Naive Bayes - nb
title = 'Naive Bayes'
param_grid = [
        {'var_smoothing': [1e-09, 1e-08, 1e-07, 1e-06, 1e-05]}
    ]
homeros_grid_search(nb,param_grid,title)
```

```
Accuracy Naive Bayes : 0.51952
Best parameters:  {'var_smoothing': 1e-09}
Best score:  nan
```

```
# Stochastic Gradient Descent - sgd
title = 'Stochastic Gradient Descent'
# param_grid = [
#         {'loss': ['hinge', 'log', 'modified_huber', 'squared_hinge'],
#     'penalty': ['l1', 'l2', 'elasticnet'],
#     'alpha': [0.0001, 0.001, 0.01, 0.1],
#     'max_iter': [1000, 5000, 10000]}
#     ]
param_grid = [
        {'loss': ['squared_hinge'],
    'penalty': ['l1', 'l2', 'elasticnet'],
    'alpha': [0.0001],
    'max_iter': [1000]}
    ]


homeros_grid_search(sgd,param_grid,title)
```

```
Accuracy Stochastic Gradient Descent : 0.59059
Best parameters:  {'alpha': 0.0001, 'loss': 'squared_hinge', 'max_iter': 1000, 'penalty': 'l1'}
Best score:  nan
```

```
# KNN - knn
title = 'KNN'
param_grid = [
        {'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]}
    ]
homeros_grid_search(knn,param_grid,title)
```

```
Accuracy KNN : 0.92526
Best parameters:  {'n_neighbors': 3, 'p': 1, 'weights': 'uniform'}
Best score:  nan
```

```
# Decission trees - tree
title = 'Decission trees'
# param_grid = [
#         {'criterion': ['gini', 'entropy'],
#      'max_depth': [None, 3, 5, 7, 9],
#      'min_samples_split': [2, 5, 10],
#      'min_samples_leaf': [1, 2, 4]}
#     ]
param_grid = [
        {'criterion': ['gini', 'entropy'],
    'max_depth': [None, 3, 5],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [2, 4]}
    ]
homeros_grid_search(tree,param_grid,title)
```

```
Accuracy Decission trees : 0.64965
Best parameters:  {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2}
Best score:  nan
```

```
# Random Forest - rforest
title = 'Random Forest'
param_grid = [
        {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
        {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
    ]

homeros_grid_search(rforest,param_grid,title)
```

```
Accuracy Random Forest : 0.56423
Best parameters:  {'max_features': 2, 'n_estimators': 3}
Best score:  nan
```

```
# Support Vector Machine - svm
title = 'Support Vector Machine'
# param_grid = [
#         {'C': [0.1, 1, 10, 100],
#      'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
#      'degree': [2, 3, 4, 5],
#      'gamma': ['scale', 'auto']}
#     ]

param_grid = [
        {'C': [1],
    'kernel': ['linear'],
    'degree': [2, 3, 4, 5],
    'gamma': ['scale', 'auto']}
    ]
homeros_grid_search(svm,param_grid,title)
```

```
Accuracy Support Vector Machine : 0.73073
Best parameters:  {'C': 1, 'degree': 2, 'gamma': 'scale', 'kernel': 'linear'}
Best score:  nan
```

```
# Logistic Regression - lg
title = 'Logistic Regression'
# param_grid = [
#         {'penalty': ['l1', 'l2'],
#      'C': [0.1, 1, 10, 100],
#      'solver': ['liblinear', 'saga'],
#      'max_iter': [100, 200, 500, 1000]}
#     ]
param_grid = [
        {'penalty': ['l1', 'l2'],
    'C': [1, 10],
    'solver': ['saga'],
    'max_iter': [100]}
    ]
homeros_grid_search(lg,param_grid,title)
```

```
Accuracy Logistic Regression : 0.71271
Best parameters:  {'C': 1, 'max_iter': 100, 'penalty': 'l1', 'solver': 'saga'}
Best score:  nan
```

```
# Neural Nets - nn
title = 'Neural Nets'
# param_grid = [
#         {'hidden_layer_sizes': [(5,), (10,), (5,5), (10,10)],
#     'activation': ['logistic', 'tanh', 'relu'],
#     'solver': ['sgd', 'adam'],
#     'alpha': [0.0001, 0.001, 0.01],
#     'max_iter': [100, 200, 500]}
#     ]

param_grid = [
        {'hidden_layer_sizes': [(5,), (10,),(10,10)],
    'activation': ['logistic'],
    'solver': ['sgd'],
    'alpha': [0.01],
    'max_iter': [100]}
    ]

homeros_grid_search(nn,param_grid,title)
```

```
Accuracy Neural Nets : 0.08709
Best parameters:  {'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (5,), 'max_iter': 100, 'solver': 'sgd'}
Best score:  nan
```

GridSearch can improve the accuracy of the model if the user find adequate parameters. In this case the only models that improved were KNN from 0.805 to 0.92526, making it now the top tier performing algorithm and logic regression going from 0.697 to 0.712. As you can see GridSearch can make a seemingly weak model like KNN become the most accurate model of all.

5 Conclusions

To conclude some questions must be answered to find the best solution for this problem:

5.1 Which model has the best accuracy "Accuracy"?

The simple answer is that the accuracy Cross Gradient Booster (XGB) has the biggest accuracy with 0.9022. This is without including any parameter finding technique or tuning such as GridSearch, Differential Evolution, Cross Validation or GridSearch. The accuracy will depend heavily on the model type which will affect depending on the nature of the problem.

5.2 Which model configuration and which data?

The best performing model XGBClassifier by SKlearn is used with a number of estimators of a 1000 and a learning rate of 0.5. Leaving the rest of the parameters to default settings.

5.3 Using GridSearch and Cross Validation results improve?

Yes, and no. The use of GridSearch can be a hit or miss depending on the parameters that are being look for. GridSearch is a technique that will try a series of parameters inputted by the user, then it evaluates the accuracy score and finally returns the best parameters for that model with the dataset. The main problem of GridSearch is the time taken to run all the parameters for the exact model. Models based in RandomForest, Naive Bayes or KNN can be processed quickly, unlike Neural Networks or XGB. In this document only a few parameters were tested due to the time taken to process each of the models.

Certainly there is some improvement in the lower performing models, but in this example there wasn't any big improvement in the top performing algorithm which is generally what is looked after. However, lower performing models are generally faster and require less storage than bigger and better performing models. So, depending on the situation not always the best performing model might be chosen for the task. In this case KNN gained a huge accuracy increase by implementing GridSearch + CrossValidation.

To conclude, each model is extremely time consuming and finding different parameters to tune and experiment is also something has to be taken into account. No efficient way to find parameters to fine tune the model and improve an accuracy was found.

Another problem regarding the testing of the models is the time required to process each model. The general behavior that the models seemed to show was that the more precise the model was the longer it took to process. For example, the model that took the longest was the Accuracy Cross Gradient Booster followed by the SVM. Random Forest was the 5th slowest model and second best performing. So it can be safely concluded that Random Forest is the best model for performance per time.

References

[1] *GTZAN Dataset - Music Genre Classification — Kaggle*. URL: https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification.

[2] *Work w/ Audio Data: Visualise, Classify, Recommend — Kaggle*. URL: https://www.kaggle.com/code/andradaolteanu/work-w-audio-data-visualise-classify-recommend.