CS985MLDAGroup21-GA1-CLASS

February 21, 2021

1 CS985MLDAGroup21

Ian Richardson 202074007

Fraser Bayne 202053049

Slav Ivanov 201645797

Lora Kiosseva 202082329

- A description of the model and solution that you employed for the final set of predictions.

  The model used in the end was a OneVersesRest classifier in code block 27, this also writes out the predictions to the file foo.csv

- A justification for why you choose this architecture and solution including: how you came up with the approach, why you selected or modified input variables, explaining what worked and did not work, and what other models were tried.

  We used a number of different models to test against each other with a validation set of the data as well as optimiser algorithms to set the parameters of these models, all of this is kept in the file (blocks 15-25)

When processing the data, we tried a number of approaches such as: factorizing the years into decades and one hot encoding this, grouping some of the values into quntiles of data, scaling the data and dropping certain columns and using over and under sampling to increase the ammount of training data, although all of these would make our final predictions very off and get worse scores on the Kaggle competition, this code was the best combination of preprocessing and column dropping

We also elected to drop all rows in the training with uncommon genres, our cut-off point was less than 6 occurrences. This will improve our training so that it is not getting thrown off unnecessarily by these uncommon genres

The scores from our own validation set where much higher that that of the Kaggle competition, possibly due to overfitting the data. We found that using over and under sampling of the data we could get very high scores in the validation (upwards of 0.8-0.9) but these would not translate to the Kaggle scoring (getting around 0.07)

[1]:
```
# Essential Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
# Preprocessing
from sklearn.preprocessing import StandardScaler, scale
```

[3]:
```
# Model Selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
```

[4]:
```
# ML Models
from sklearn.svm import SVC, LinearSVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
```

[5]:
```
# Metrics
from sklearn.metrics import f1_score
```

[6]:
```
# Scikit-Optimise
from skopt import gp_minimize
from skopt.utils import cook_initial_point_generator
from skopt.space import Real, Integer
```

[7]:
```
# Read in data
train_set = pd.read_csv("./dataset/CS98XClassificationTrain.csv")
test_set = pd.read_csv("./dataset/CS98XClassificationTest.csv")
```

[8]:
```
id_column = np.array(test_set["Id"])
```

[9]:
```
# Drop all rows with NaNs
train_set = train_set.dropna()
# Drop all irrelevant columns "Id", "title", "artist", "spch", "nrgy", "live",␣
 ↪"dB", "bpm"
train_set = train_set.drop(columns=["Id", "title", "artist", "spch", "nrgy",␣
 ↪"live", "dB", "bpm"])
test_set = test_set.drop(columns=["Id", "title", "artist", "spch", "nrgy",␣
 ↪"live", "dB", "bpm"])
```

[10]:
```
# What we want to predict
predict = "top genre"
train_set = train_set.groupby(predict).filter(lambda x : len(x)>=6)
```

[11]:
```
# Get everything except what we want to predict
X = np.array(train_set.drop([predict], 1)).astype(np.float64)
# Column we want to predict
y = np.array(train_set[predict])
# Actual test dataset as numpy array
X_test = np.array(test_set).astype(np.float64)
```

[12]:
```
# Scaled data
std_scaler = StandardScaler()
X_scaled = std_scaler.fit_transform(X)
X_train_scaled, X_val_scaled, y_train, y_val = train_test_split(X_scaled, y,␣
↪test_size=0.25, random_state=42)
```

[13]:
```
# Non-scaled data
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25,␣
↪random_state=42)
```

[14]:
```
# --------------
# Model Testing
# --------------

models = []
```

[15]:
```
def ovo():
    model = OneVsOneClassifier(SVC(kernel="rbf", degree=2, C=1.1, gamma=0.3,
                                   probability=True, decision_function_shape =␣
↪"ovo",
                                   random_state=42))
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_val_scaled)
    models.append(("ovo", model))
    print("> One Versus One Classifier", model.score(X_val_scaled, y_val),
          "- f1", f1_score(y_val, y_pred, average="weighted"))
```

[16]:
```
def ovr():
    model = OneVsRestClassifier(LinearSVC(dual=False, fit_intercept=False,
                                          max_iter=10000,
                                          random_state=42,))
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_val_scaled)
    models.append(("ovr", model))
    print("> One Versus Rest Classifier", model.score(X_val_scaled, y_val),
          "- f1", f1_score(y_val, y_pred, average="weighted"))
```

[17]:
```
def dec_tree():
    model = DecisionTreeClassifier(max_depth=2, splitter="best",
                                   max_features="auto", criterion="gini",
```

```
                                    random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    models.append(("dec_tree", model))
    print("> Decision Tree Classifier", model.score(X_val, y_val),
          "- f1", f1_score(y_val, y_pred, average="weighted"))
```

[18]:
```
def rnd_for():
    model = RandomForestClassifier(n_estimators=155, max_depth=2,
                                                    bootstrap=True, n_jobs=-1,
                                                    warm_start=True,␣
 ↪random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    models.append(("rnd_for", model))
    print("> Random Forest Classifier", model.score(X_val, y_val),
          "- f1", f1_score(y_val, y_pred, average="weighted"))
```

[19]:
```
def ada():
    model = AdaBoostClassifier(DecisionTreeClassifier(max_depth=3,␣
 ↪max_features="auto",),
                               n_estimators=40,
                               algorithm="SAMME.R",
                               learning_rate=0.2,
                               random_state=42)

    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    models.append(("ada", model))
    print("> AdaBoost Classifier", model.score(X_val, y_val),
          "- f1", f1_score(y_val, y_pred, average="weighted"))
```

[20]:
```
def gbc():
    model = GradientBoostingClassifier(max_depth=4, n_estimators=50,
                                       learning_rate=1.0, random_state=42)

    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    models.append(("gbc", model))
    print("> Gradient Boosting Classifier", model.score(X_val, y_val),
          "- f1", f1_score(y_val, y_pred, average="weighted"))
```

[21]:
```
def mlpc():
    model = MLPClassifier(activation = "logistic", solver="adam",
                          max_iter=500, alpha=0.5,
                          learning_rate_init=0.1, warm_start=True,
                          learning_rate="invscaling", shuffle=True,
```

```
                            random_state=42)
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_val_scaled)
    models.append(("mlpc", model))
    print("> Multi-Layer Perceptron Classifier", model.score(X_val_scaled,␣
 ↪y_val),
          "- f1", f1_score(y_val, y_pred, average="weighted"))
```

[22]:
```
def knnc():
    model = KNeighborsClassifier(n_neighbors=15, weights="uniform",
                                 n_jobs=-1)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    models.append(("knnc", model))
    print("> K-Nearest Neighbors Classifier", model.score(X_val, y_val),
          "- f1", f1_score(y_val, y_pred, average="weighted"))
```

[23]:
```
def voting():
    model = VotingClassifier(estimators=models,
        voting='hard')
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_val_scaled)
    print("> Voting Classifier", model.score(X_val_scaled, y_val),
          "- f1", f1_score(y_val, y_pred, average="weighted"))
```

[24]:
```
print("Using columns", train_set.columns.values[:len(train_set.columns.
 ↪values)-1])
ovo()
ovr()
dec_tree()
rnd_for()
ada()
mlpc()
knnc()
voting()
```

```
Using columns ['year' 'dnce' 'val' 'dur' 'acous' 'pop']
> One Versus One Classifier 0.4875 - f1 0.37784090909090917
> One Versus Rest Classifier 0.4875 - f1 0.3940199335548173
> Decision Tree Classifier 0.4625 - f1 0.35241355569155447
> Random Forest Classifier 0.5 - f1 0.38053168635875395
> AdaBoost Classifier 0.5 - f1 0.3975821718327439
> Multi-Layer Perceptron Classifier 0.5 - f1 0.3825213903743315
> K-Nearest Neighbors Classifier 0.475 - f1 0.37164505233847267
> Voting Classifier 0.5125 - f1 0.39080159705159695
```

[25]:
```
# --------------
# Real Prediction
# --------------
```

[26]:
```
model = OneVsRestClassifier(LinearSVC(dual=False, fit_intercept=False,
                                       max_iter=10000,
                                       random_state=42))

model.fit(X, y)
y_pred = model.predict(X_test)

csv = np.stack((id_column, y_pred), axis=1)
csv = np.vstack((np.array(["Id","top genre"]), csv))
np.savetxt("./foo.csv", csv, fmt='%s', delimiter=",")
```

[27]:
```
# --------------
# Optimisation
# --------------
```

[28]:
```
def f(x):
    model = RandomForestClassifier(n_estimators=155, max_depth=2,
                                   bootstrap=True, n_jobs=-1,
                                   warm_start=True, random_state=42)
    model.fit(X_train_scaled, y_train)
    return -model.score(X_val_scaled, y_val)

def bayesian_opt():
    lhs_maximin = cook_initial_point_generator("lhs", criterion="maximin")

    res = gp_minimize(f,
                      [Integer(1, 100, transform='identity'),
                       Integer(10, 1000, transform='identity'),
                       Real(0.1, 100.0, transform='identity')],
                      x0=[[3, 40, 0.2]],
                      xi=0.000001,
                      #kappa=0.001,
                      acq_func='EI', acq_optimizer='sampling',
                      n_calls=100, n_initial_points=10,␣
 ↪initial_point_generator=lhs_maximin,
                      verbose = False,
                      noise = 1e-10,
                      random_state = 42)

    return (res.x, res.fun)
```

This written out CSV (foo.csv) gets a Kaggle competition score of 0.30357