final_project

Author Name

December 11, 2020

1 Overview

In this term project, you will deploy Deep Learning models to build a classification model using RapidMiner to predict the sentiment of consumers towards US airlines based on their reviews expressed in the form of tweets. If you strongly prefer to use some other DL-based software/frameworks instead of RapidMiner, such as TensorFlow or PyTorch, let me know before starting the work. This is a group project, and you should work on it in the groups that you have formed already.

[1]:
```python
# fetch data
import requests
from io import StringIO

# core
import numpy as np
import pandas as pd

# preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

# baseline algorithms
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# deep learning
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout
from tensorflow.keras.losses import mean_absolute_error as tf_mae
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

# evaluation
from sklearn.model_selection import cross_validate
from sklearn.metrics import mean_absolute_error as skl_mae
```

2 1. Fetch Data

The data is provided to you in two versions:

1. The original version of the tweets (and their sentiments) is located at https://drive.google.com/file/d/1atyRH5Yz7TU-2ziyZknfd7ib2LLwYeuv/view?usp=sharing
2. The preprocessed version of the tweets is located at https://drive.google.com/file/d/1c96crlNZr7XiF3-9lmZ1nEJaY3MHTTz5/view?usp=sharing, where text preprocessing and pre-training of the text embeddings of the tweets using autoencoders have already been done to make your life simpler. This preprocessed version contains the sentiments about the tweets in column 1 of the spreadsheet (either positive (1) or negative (0)) and the 8-dimenisonal pre-trained embeddings of the tweets (in columns 2 – 9 of the spreadsheet).

I recommend that you use the preprocessed version of the tweets since it will save you a lot of preprocessing work to build these embeddings that is non-trivial. However, if you like challenges, you can do preprocessing and building the embeddings using autoencoders yourself and, therefore, work directly with the "raw" tweets. As a "reward" for this extra work, you will be awarded 10 extra points (the max score of this project is 100) if you preprocess tweets yourself.

[2]:
```python
# url of dataset
google_drive_file_url = 'https://drivesds.google.com/file/d/
↪1c96crlNZr7XiF3-9lmZ1nEJaY3MHTTz5/view?usp=sharing'

def fetch_google_drive_csv(google_drive_file_url):

    file_id = google_drive_file_url.split('/')[-2]
    download_url = 'https://drive.google.com/uc?export=download&id=' + file_id
    url = requests.get(download_url)
    csv_raw = StringIO(url.text)
    return pd.read_csv(csv_raw)

data = fetch_google_drive_csv(google_drive_file_url)
```

[3]:
```python
data.head()
```

[3]:
```
   sentiment  dimension1  dimension2  dimension3  dimension4  dimension5  \
0          1   -0.400418    0.293417   -0.572702    0.125659    0.471714
1          1   -0.454608   -0.194998   -0.497063    0.242207    0.209621
2          0   -0.515892   -0.120781   -0.106512   -0.260192    0.197666
3          1    0.047770   -0.230509    0.132355    0.174913    0.242040
4          0   -0.574353   -0.132517   -0.091610    0.466463    0.510980

   dimension6  dimension7  dimension8
0   -0.034476    0.042176   -0.429317
1    0.064868    0.072154    0.629457
2   -0.155029   -0.306803    0.694974
3   -0.229259   -0.835945    0.294148
4   -0.338480    0.202040   -0.100443
```

[4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55524 entries, 0 to 55523
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   sentiment   55524 non-null  int64
 1   dimension1  55524 non-null  float64
 2   dimension2  55524 non-null  float64
 3   dimension3  55524 non-null  float64
 4   dimension4  55524 non-null  float64
 5   dimension5  55524 non-null  float64
 6   dimension6  55524 non-null  float64
 7   dimension7  55524 non-null  float64
 8   dimension8  55524 non-null  float64
dtypes: float64(8), int64(1)
memory usage: 3.8 MB
```

[5]:
```
data.shape
```

[5]:
```
(55524, 9)
```

3 2. Preprocess Data

Your task is to predict the score of the sentiment (positive or negative) between 0 and 1 based on the embeddings of the tweets specified in columns 2 – 9 of the pre-possessed spreadsheet (or the original tweets if you decided to work with the raw tweeting data). To evaluate the performance of your model, please split the dataset into the train set and the test set in the 0.8:0.2 ratio and use cross-validation to calculate the prediction performance.

3.1 2.1 Train/Test Split

[6]:
```
target = 'sentiment'

X_train, X_test, y_train, y_test = train_test_split(
    data.drop(target, axis=1), # predictors
    data[target], # target
    test_size=0.2,
    random_state=90
)
```

[7]:
```
X_train.shape
```

[7]:
```
(44419, 8)
```

[8]:
```
X_test.shape
```

[8]: (11105, 8)

3.2 2.2 Cross-Validation

[9]:
```
cv_splits = KFold(n_splits=5, random_state=90, shuffle=True)
```

[10]:
```
fold_num = 1
for train, test in cv_splits.split(X_train, y_train):
    print(f'Fold #{fold_num}: Train shape: {X_train.iloc[train].shape}, Test␣
 ↪shape: {X_train.iloc[test].shape}')
    fold_num += 1
```

```
Fold #1: Train shape: (35535, 8), Test shape: (8884, 8)
Fold #2: Train shape: (35535, 8), Test shape: (8884, 8)
Fold #3: Train shape: (35535, 8), Test shape: (8884, 8)
Fold #4: Train shape: (35535, 8), Test shape: (8884, 8)
Fold #5: Train shape: (35536, 8), Test shape: (8883, 8)
```

[11]:
```
def cross_val_score_keras(X_train, y_train, cv_spliter, model, batch_size,␣
 ↪num_epochs):

    # model configuration
    loss_function = tf_mae
    optimizer = Adam()
    callback = EarlyStopping(monitor='loss', patience=10)

    # cross-validation scores
    cv_results = list()

    # K-fold Cross Validation model evaluation
    for train, test in cv_spliter.split(X_train, y_train):

        # compitle the model
        model.compile(loss=loss_function,
                      optimizer=optimizer)

        # fit data to model
        history = model.fit(X_train.iloc[train], y_train.iloc[train],
                            batch_size=batch_size,
                            epochs=num_epochs,
                            callbacks=[callback],
                            verbose=0)

        # generate generalization metrics
```

```
        score = model.evaluate(X_train.iloc[test], y_train.iloc[test],␣
↪verbose=0)
        cv_results.append(score)

    return np.array(cv_results)
```

4 3. Neural Networks

You can use any neural network model you like for this classification task. In particular, you may start with a simple single fully connected network as a "baseline" and then try to use more complex models, including CNN and RNN based models, to achieve better performance results than this simple baseline model. Your goal is to reach the mean absolute error of at least 0.48, which should not be too difficult. If you want to be more ambitious, you can try to reach the mean absolute error of 0.47 (medium difficulty), or even 0.46 (this is difficult). The higher accuracy you get, the more points you will be awarded.

In addition to the simple NN baseline mentioned above, you should also build another basic baseline, such as a logistic regression model (similar to the one we used in the RapidMiner Lab done in the class) and compare the performance results of your DL-based model with that baseline. The expectation is that the more sophisticated DL-model should outperform simple baselines.

4.1 3.1 Baseline Model Performance

4.1.1 3.1.1 Logistic Regression

[12]:
```
%%time

model_lr = LogisticRegression()
cv_results_lr = cross_validate(model_lr, X_train, y_train,␣
↪scoring='neg_mean_absolute_error', cv=cv_splits)
cv_lr_mu = -cv_results_lr['test_score'].mean()
cv_lr_sd = cv_results_lr['test_score'].std()

print(f'Logistic Regression MAE: {cv_lr_mu:.4f} +/- {cv_lr_sd:.4f}\n')
```

```
Logistic Regression MAE: 0.4366 +/- 0.0041

CPU times: user 551 ms, sys: 79 ms, total: 630 ms
Wall time: 173 ms
```

4.1.2 4.1.2 Random Forest

[13]:
```
%%time

model_rf = RandomForestClassifier(n_estimators=500)
cv_results_rf = cross_validate(model_rf, X_train, y_train,␣
↪scoring='neg_mean_absolute_error', cv=cv_splits)
```

```
cv_rf_mu = -cv_results_rf['test_score'].mean()
cv_rf_sd = cv_results_rf['test_score'].std()

print(f'Random Forest MAE: {cv_rf_mu:.4f} +/- {cv_rf_sd:.4f}\n')
```

```
Random Forest MAE: 0.4186 +/- 0.0067

CPU times: user 3min 21s, sys: 687 ms, total: 3min 22s
Wall time: 3min 22s
```

4.1.3 4.1.3 Simple Neural Network

[14]:
```
%%time

# define the model architecture
model_nn_0 = Sequential([
    Dense(32, activation='relu', input_dim=8),
    Dense(1, activation='sigmoid')
])

# cross-validation
cv_results_nn_0 = cross_val_score_keras(
    X_train, y_train,
    cv_splits,
    model_nn_0,
    batch_size=50,
    num_epochs=100
)

# summary statistics
cv_nn_0_mu = cv_results_nn_0.mean()
cv_nn_0_sd = cv_results_nn_0.std()
print(f'Simlpe NN Baseline Model #0 MAE: {cv_nn_0_mu:.4f} +/- {cv_nn_0_sd:.
 ↪4f}\n')
```

```
Simlpe NN Baseline Model #0 MAE: 0.4703 +/- 0.0046

CPU times: user 27.7 s, sys: 5.04 s, total: 32.7 s
Wall time: 18.1 s
```

4.2 4.2 Deep Learning

4.2.1 4.2.1 DL Model #1

[30]:
```
%%time

# define the model architecture
model_nn_1 = Sequential([
```

```
    Dense(8, activation='relu', input_dim=8),
    Dense(8, activation='relu'),
    Dense(8, activation='relu'),
    Dense(8, activation='relu'),
    Dense(8, activation='relu'),
    Dense(8, activation='relu'),
    Dense(8, activation='relu'),
    Dense(8, activation='relu'),
    Dense(1, activation='sigmoid')
])

# cross-validation
cv_results_nn_1 = cross_val_score_keras(
    X_train, y_train,
    cv_splits,
    model_nn_1,
    batch_size=250,
    num_epochs=100
)

# summary statistics
cv_nn_1_mu = cv_results_nn_1.mean()
cv_nn_1_sd = cv_results_nn_1.std()
print(f'Deep Learning Model #1 MAE: {cv_nn_1_mu:.4f} +/- {cv_nn_1_sd:.4f}\n')
```

```
Deep Learning Model #1 MAE: 0.4703 +/- 0.0046

CPU times: user 12.6 s, sys: 2.2 s, total: 14.8 s
Wall time: 7.88 s
```

[31]:
```
# model paramters
batch_size = 250
num_epochs = 100
loss_function = tf_mae
optimizer = Adam()
callback = EarlyStopping(monitor='loss', patience=10)

# architecture
model_nn_1 = Sequential([
    Dense(8, activation='relu', input_dim=8),
    Dense(8, activation='relu'),
    Dense(8, activation='relu'),
    Dense(8, activation='relu'),
    Dense(8, activation='relu'),
    Dense(8, activation='relu'),
    Dense(8, activation='relu'),
    Dense(8, activation='relu'),
```

```
    Dense(1, activation='sigmoid')
])

# compitle the model
model_nn_1.compile(loss=loss_function,
                   optimizer=optimizer)

# fit data to model
history_nn1 = model_nn_1.fit(X_train, y_train,
                             batch_size=batch_size,
                             epochs=num_epochs,
                             callbacks=[callback],
                             verbose=0)
```

4.2.2 4.2.1 DL Model #2

[29]:
```
%%time

# define the model architecture
model_nn_2 = Sequential([
    Dense(96, activation='relu', input_dim=8),
    BatchNormalization(),
    Dense(64, activation='relu'),
    BatchNormalization(),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

# cross-validation
cv_results_nn_2 = cross_val_score_keras(
    X_train, y_train,
    cv_splits,
    model_nn_2,
    batch_size=500,
    num_epochs=100
)

# summary statistics
cv_nn_2_mu = cv_results_nn_2.mean()
cv_nn_2_sd = cv_results_nn_2.std()
print(f'Deep Learning Model #1 MAE: {cv_nn_2_mu:.4f} +/- {cv_nn_2_sd:.4f}\n')
```

```
Deep Learning Model #1 MAE: 0.4035 +/- 0.0073

CPU times: user 54.9 s, sys: 11.8 s, total: 1min 6s
Wall time: 27.4 s
```

[ ]:
```
# model paramters
batch_size = 250
num_epochs = 100
loss_function = tf_mae
optimizer = Adam()
callback = EarlyStopping(monitor='loss', patience=10)

# compitle the model
model_nn_2.compile(loss=loss_function,
                   optimizer=optimizer)

# fit data to model
history_nn2 = model_nn_2.fit(X_train, y_train,
                             batch_size=batch_size,
                             epochs=num_epochs,
                             callbacks=[callback],
                             verbose=0)
```

5 4. Evaluation

After you build your neural network, apply the trained deep learning model to the test set and evaluate its performance using the accuracy measures.

[ ]:
```
```