initial_rf_testing

February 28, 2024

[ ]:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from numpy import random
```

[ ]:

```
random.seed(42)

data = pd.read_csv('data/Features_For_Traditional_ML_Techniques.csv')
#remove all meta-data features as we are comparing structure (except hashtags)
#compare to original data performance on baseline models
meta_features = ["followers_count",␣
 ↪"friends_count","favourites_count","statuses_count","listed_count","following","embeddings"
#filter data to only include non-meta features
data = data.drop(meta_features, axis=1)
data = data.drop('majority_target', axis=1)


grouped = data.groupby('statement')
train_data = pd.DataFrame()
test_data = pd.DataFrame()
i=0

for group_name, group_df in grouped:
    if random.random() <= 0.8:
        train_data = pd.concat([train_data, group_df])
    else:
        test_data = pd.concat([test_data, group_df])
    i += 1
    # if i > 100:
    #     break

# Step 2: Split each group into training and test sets

# for group_name, group_df in grouped:
```

```
#     if len(group_df) > 1:  # Ensure at least 2 samples for splitting
#         train_group, test_group = train_test_split(group_df, test_size=0.2,␣
 ↪random_state=42)
#         train_data = pd.concat([train_data, train_group])
#         test_data = pd.concat([test_data, test_group])


# #TEMP drop tweets + statements
# train_data = train_data.drop(["tweet", "statement"], axis=1)
# test_data = test_data.drop(["tweet", "statement"], axis=1)

#split into training_testing data with 80% statements in training and 20% in␣
 ↪testing

# data.head()
```

[ ]:

```
from sklearn.preprocessing import LabelEncoder

# Handle string values using label encoding
label_encoder = LabelEncoder()
# Identify columns with string values (assuming dtype is 'str' or 'object')
string_columns = np.array([np.issubdtype(type(col), np.str_) or np.
 ↪issubdtype(type(col), np.object) for col in train_data.iloc[0]])
# Apply label encoding to string columns
for col_index in np.where(string_columns)[0]:
    train_data.iloc[:, col_index] = label_encoder.fit_transform(train_data.
 ↪iloc[:, col_index].astype(str))

features_train = train_data.astype(np.float32)
```

```
C:\Users\roryb\AppData\Local\Temp\ipykernel_18452\3536426476.py:6:
DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`.
To silence this warning, use `object` by itself. Doing this will not modify any
behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  string_columns = np.array([np.issubdtype(type(col), np.str_) or
np.issubdtype(type(col), np.object) for col in train_data.iloc[0]])
```

[ ]:

```
from sklearn.preprocessing import LabelEncoder

# Handle string values using label encoding
label_encoder = LabelEncoder()
# Identify columns with string values (assuming dtype is 'str' or 'object')
string_columns = np.array([np.issubdtype(type(col), np.str_) or np.
 ↪issubdtype(type(col), np.object) for col in test_data.iloc[0]])
# Apply label encoding to string columns
```

```
for col_index in np.where(string_columns)[0]:
    test_data.iloc[:, col_index] = label_encoder.fit_transform(test_data.iloc[:
 ↪, col_index].astype(str))

features_test = test_data.astype(np.float32)
```

```
C:\Users\roryb\AppData\Local\Temp\ipykernel_18452\4269653626.py:6:
DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`.
To silence this warning, use `object` by itself. Doing this will not modify any
behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  string_columns = np.array([np.issubdtype(type(col), np.str_) or
np.issubdtype(type(col), np.object) for col in test_data.iloc[0]])
```

```
[ ]: len(train_data), len(test_data)
```

```
[ ]: (106733, 27465)
```

```
[ ]: print(train_data.columns, train_data.iloc[0].shape)
```

```
Index(['Unnamed: 0', 'statement', 'BinaryNumTarget', 'tweet', 'hashtags',
       'unique_count', 'total_count', 'ORG_percentage', 'NORP_percentage',
       'GPE_percentage', 'PERSON_percentage', 'MONEY_percentage',
       'DATE_percentage', 'CARDINAL_percentage', 'PERCENT_percentage',
       'ORDINAL_percentage', 'FAC_percentage', 'LAW_percentage',
       'PRODUCT_percentage', 'EVENT_percentage', 'TIME_percentage',
       'LOC_percentage', 'WORK_OF_ART_percentage', 'QUANTITY_percentage',
       'LANGUAGE_percentage', 'Word count', 'Max word length',
       'Min word length', 'Average word length', 'present_verbs', 'past_verbs',
       'adjectives', 'adverbs', 'adpositions', 'pronouns', 'TOs',
       'determiners', 'conjunctions', 'dots', 'exclamation', 'questions',
       'ampersand', 'capitals', 'digits', 'long_word_freq', 'short_word_freq'],
      dtype='object') (46,)
```

```
[ ]: train_targets= train_data['BinaryNumTarget']
     train_data_notarget = train_data.drop(['BinaryNumTarget'], axis=1)

     test_targets= test_data['BinaryNumTarget']
     test_data_notarget = test_data.drop(['BinaryNumTarget'], axis=1)

     print(train_targets.value_counts())
```

```
BinaryNumTarget
1.0    55988
0.0    50745
Name: count, dtype: int64
```

[ ]:

```
#fit random forest model
rf = RandomForestClassifier(n_estimators=100, random_state=0)
rf.fit(train_data_notarget, train_targets)
```

[ ]:

```
RandomForestClassifier(random_state=0)
```

[ ]:

```
rf.predict(test_data_notarget)
print(f1_score(test_targets, rf.predict(test_data_notarget)))
print(accuracy_score(test_targets, rf.predict(test_data_notarget)))
```

```
0.8661886893594212
0.8545785545239396
```

[ ]:

```

```